SigCaptureWeb SDK Guide

Version 1.0.0.5

Table of Contents

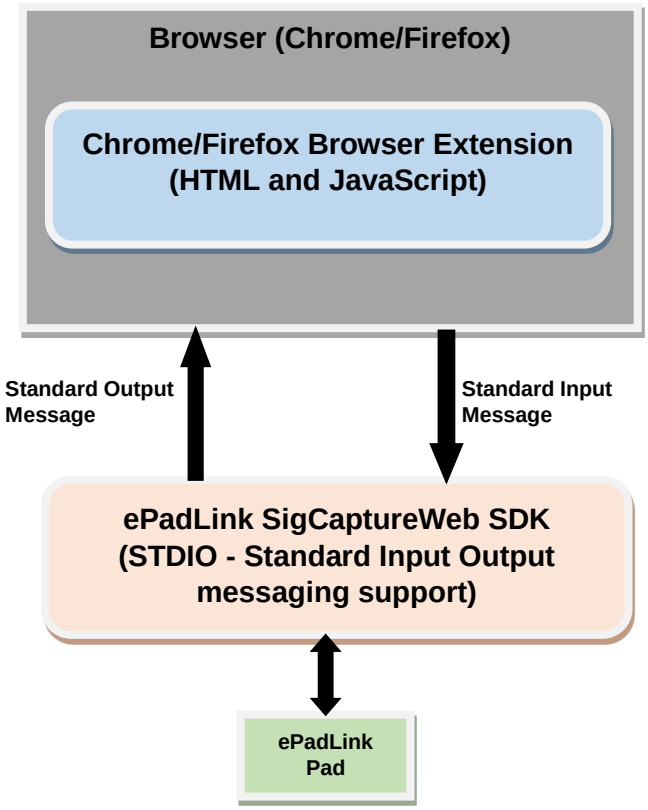
1.0 – Introduction

The ePadLink SigCaptureWeb SDK offers a mechanism and platform for developers and integrators to capture handwritten signatures securely for web applications running in the Chrome and Firefox browsers. The SDK provides capabilities for capturing secured biometric handwritten signatures using electronic signature pads from ePadLink. The SDK exports the images of the captured signatures, as well as the raw biometric signature data in different formats. The images can be used in any application requiring signature images. The biometric signature data for the captured signature is obtained in encrypted form and can be saved in any database for future use.

2.0 – Overview and Architecture

As Google Chrome and Mozilla Firefox are discontinuing support for plug-ins running inside the browsers, neither Java applets nor NPAPI plug-ins can be used for signature capture. Browser extensions are the best option for enabling web pages with the capability to capture signatures using the devices connected to client desktops.

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



2.1 – ePadLink SigCaptureWeb SDK

The ePadLink SigCaptureWeb SDK has been developed as a standard C#.Net application. It has built–in mechanism to capture signatures using ePadLink signature capture devices and also to expose the raw biometric signature data in Emcee or Sig format and the signature as an image (JPG, PNG, etc.). It has all the input and output interfaces implemented as Standard Input and Output streams as required by the Chrome and Firefox Browser Extension frameworks. The SDK processes the input text messages from the Chrome and Firefox browsers and executes the requests asynchronously, and when a task is complete sends back the status or output data as an output text message. It will host all the User Interface functions for capture and display of signatures from devices.

Chrome runs this SDK in a separate process, launches it through Google Connect APIs, and sends a notification back to the Chrome Extension when the application is ended by the user. Firefox runs this SDK as a sub process within Firefox, launches it through SubProcess JavaScript Modules (JSM), and sends notification back to extension when the request is completed.

2.2 – Chrome Extension/Webpage

Chrome Extensions are the HTML, JavaScript, and CSS based code modules that are launched during startup of the browser or launched on demand from web page JavaScript. The extensions use JavaScript based Google Native Messaging APIs to launch and communicate with the ePadLink SigCaptureWeb SDK for signature capture and other relevant features. The extension listens for the output messages from SigCaptureWeb and processes them accordingly. Google Native Messaging has a Connect API to launch the applications (which can process the standard input and output messages) in this SigCaptureWeb SDK and a Disconnect event to let the web page know about termination of the native host application. Using Connect and Disconnect, the life cycle of the native host application can be controlled. Also, Google Native Messaging APIs have mechanisms to send input messages to the SigCaptureWeb SDK and receive output messages from applications.

2.3 – Firefox Extension/Webpage

Firefox Extensions are the HTML, JavaScript, and CSS based code modules that are launched during startup of the browser. The Firefox extension uses SubProcess Java Script Modules to launch the ePadLink SigCaptureWeb SDK and send and receive input and output messages. It also has the required callback mechanism to notify the calling module when the launched application is terminated or when some error has occurred.

3.0 – Key Features

The ePadLink SigCaptureWeb 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 PNG/JPG (base64 string)
- Export the raw biometric signature data in eSign Emcee or .Sig formatted encrypted base 64 string.

4.0 – Operating Systems Supported

The ePadLink SigCaptureWeb SDK can be integrated into web pages running in the latest versions of Chrome and Firefox browsers installed on Windows 7/8/8.1/10 32 bit operating systems. For 64 bit Windows operating systems running the 64 bit Chrome and Firefox browsers, the SDK should be run as a 32 bit application.

The samples have been tested in the latest version of Chrome and Firefox browsers. Hence, it is recommended that you install the latest version of the Chrome and Firefox browsers.

Note: Microsoft Visual Studio 2008 with .NET Framework 3.5 is the development IDE used in developing the SigCaptureWeb SDK, and hence .NET Framework 3.5 should be available in the end user Window's computer.

5.0 – Signature Capture Devices

The ePadLink SigCaptureWeb SDK supports capturing signatures using electronic signature pads from ePadLink. Universal Installer based drivers have to be installed for the ePadLink SigCaptureWeb SDK to capture signatures using signature pad devices.

6.0 – Instructions to Run Sample Applications/Installation Steps

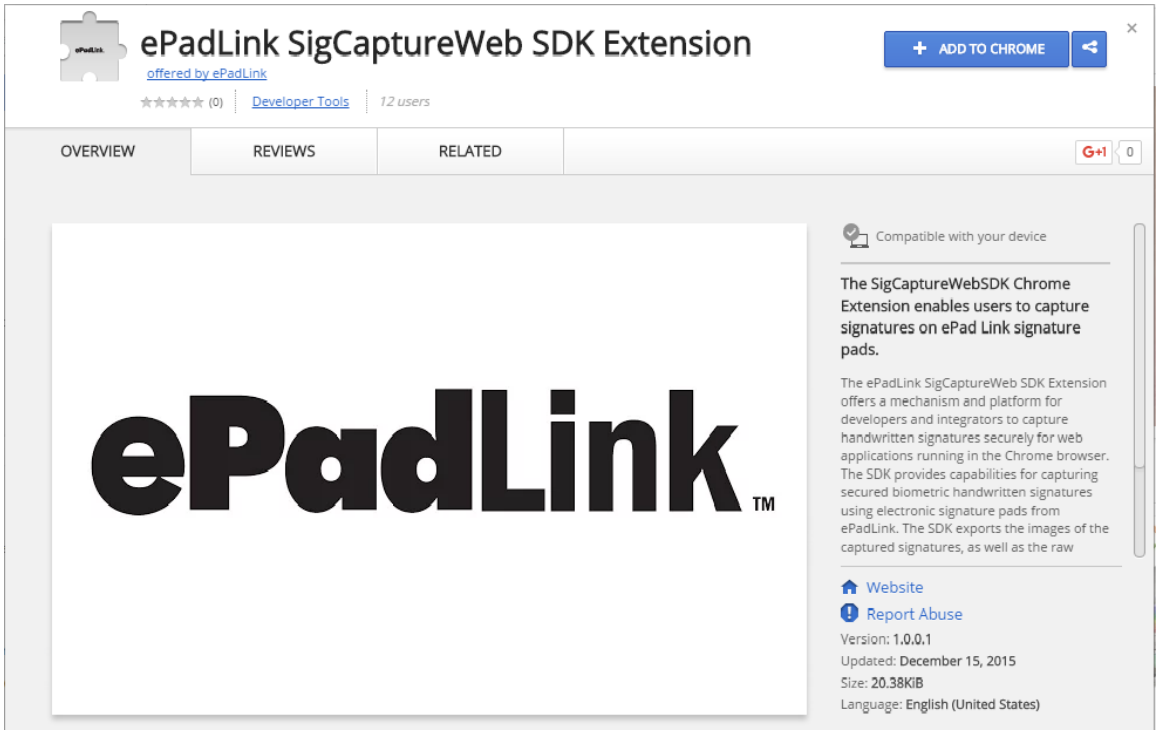
6.1 – Chrome

6.1.1 – ePadLink SigCaptureWeb SDK Installation

Download and run the latest ePadLink SigCaptureWeb SDK installer from www.epadsupport.com.

6.1.2 – Install the ePadLink SigCaptureWeb Chrome Extension

1. Start the Chrome Browser and go to the ePadLink SigCaptureWeb Background Extension page by navigating to the URL: https://chrome.google.com/webstore/detail/epadlink-sigcaptureweb-sd/idldbjenlmipmpigmfamdlfifkkeaplc
2. In the opened page, click on the + ADD TO CHROME button displayed on the top right of the page (see the screenshot below).
3. Click on the Add extension button in the confirmation dialog.



6.1.3 – Run the Sample Webpage

After making sure the ePadLink device drivers are installed and the signature pad is connected to the client desktop, launch Chrome and navigate to the following page: https://www.esignemcee.net/SigCaptureWeb/sign_chrome_ff_sigcapturewebsdk.html. Click on the “Sign” button to capture the signature.

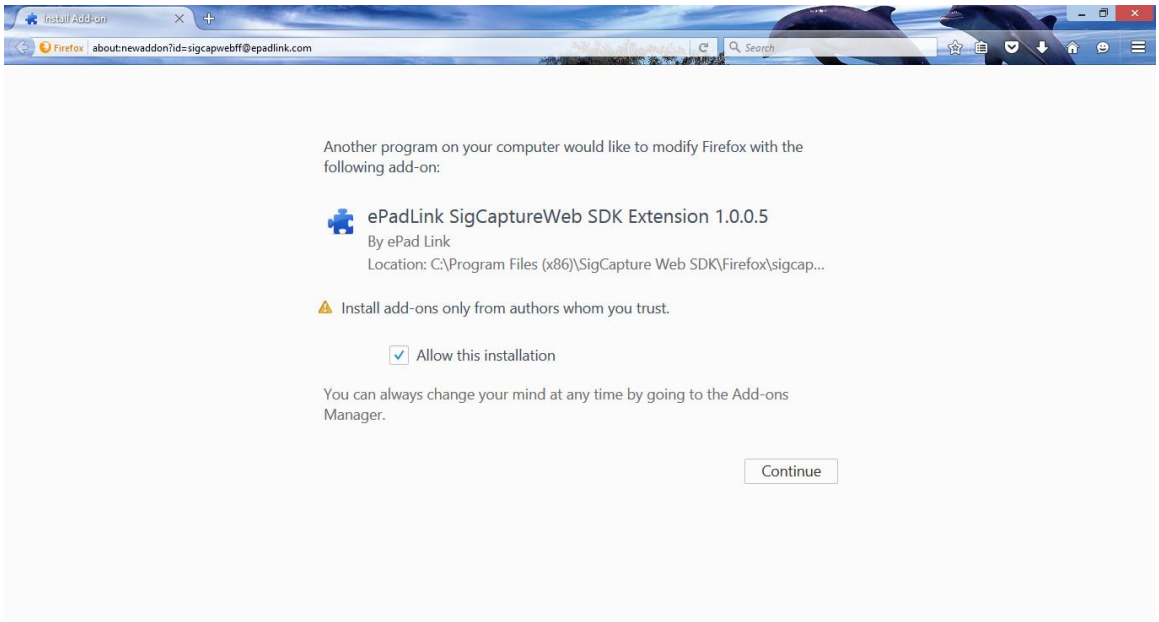
6.2 – Firefox

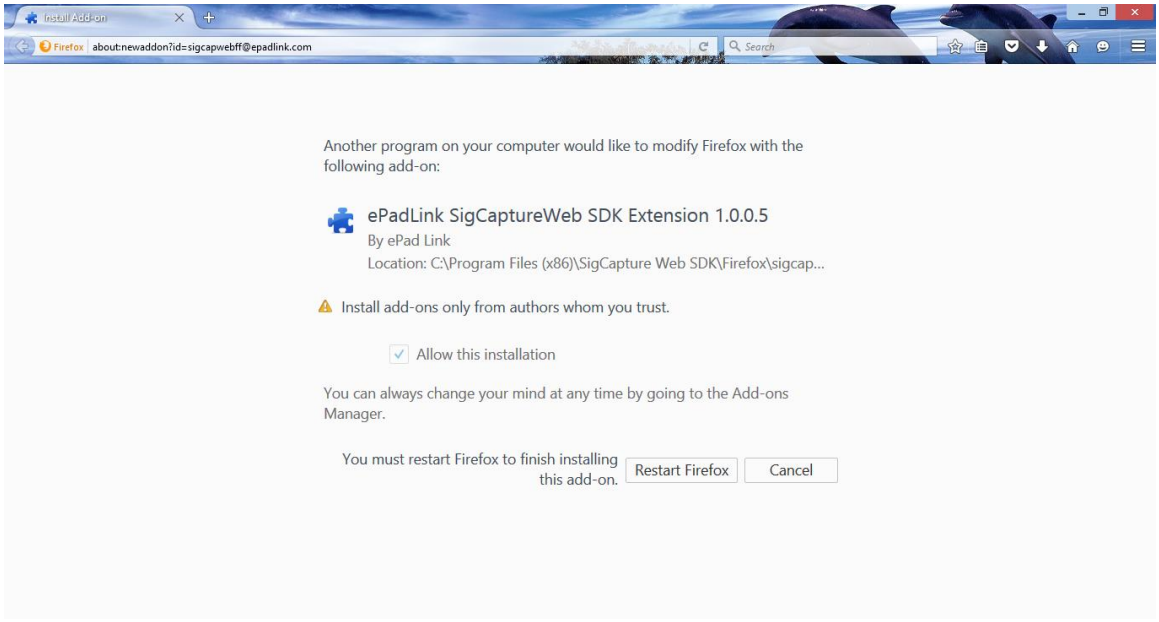
Using SigCaptureWeb in Firefox requires two steps. First, SigCaptureWeb needs to be installed. Then, the SigCaptureWeb extension needs to be set up for Firefox.

6.2.1 – Install SigCaptureWeb

Remove any previous version of the SigCaptureWeb Firefox extension from the Firefox Add-ons manager page, Extension section.

1. Download and run the latest ePadLink SigCaptureWeb SDK installer from www.epadsupport.com.
2. Start the Firefox browser to complete the installation of the Firefox Extension. The browser prompts the user to allow the extension installation. Check the ‘Allow this installation’ option and click the ‘Continue’ button and then the ‘Restart Firefox’ button to complete the installation.





Note:

- Step 2 has to be performed for all the Windows login accounts by logging into Windows and starting the Firefox browser.
- In some cases the Firefox browser may not show the prompt mentioned in step 2; in those cases the extension has to be manually enabled from the Firefox Add-ons manager page by typing in about:addons in the address bar, then going to the Extensions section and clicking on the 'Enable' button next to 'ePadLink SigCaptureWeb SDK Extension' and restarting the browser.

6.2.2 – Run the Sample Webpage

After making sure the ePadLink device drivers are installed and the signature pad is connected to the client desktop, launch Firefox and navigate to the following page: https://www.esignemcee.net/SigCaptureWeb/sign_chrome_ff_sigcapturewebsdk.html. Click on the Sign button to capture the signature.

7.0 – SigCaptureWeb SDK Integration for Signature Capture in Chrome and Firefox

For web pages running in Chrome and Firefox the only required step is to raise and listen for predefined custom HTML events within the web page.

7.1 – Launching the Extensions from a Webpage

The SigCaptureWeb SDK extensions rely on custom HTML events for communication between the web page and the extensions and vice versa.

The Chrome and Firefox Extensions load during browser start up and register a custom HTML event named “SigCaptureWeb_SignStartEvent”. Web pages wishing to capture the signature using the SigCaptureWeb SDK in Chrome and Firefox browsers have to raise the custom HTML event “SigCaptureWeb_SignStartEvent” and send an input message to the SDK as an event attribute.

Once the signature capture task is completed, the SigCaptureWeb extensions raise a custom HTML event named “SigCaptureWeb_SignResponse” and pass the output message as an event attribute. Web pages should register and implement the “SigCaptureWeb_SignResponse” event for processing the output from the extension.

The following code snippet demonstrates raising the custom HTML event “SigCaptureWeb_SignStartEvent” to initiate signature capture and also register and implement the “SigCaptureWeb_SignResponse” event for processing the output from the SDK.

Input and output messages are passed as event attributes. The input to the SDK must be set using the attribute name “SigCaptureWeb_MsgAttribute” and response from the SDK must be read from event attribute “SigCaptureWeb_msgAttri”

```
var message = { "firstName": "", "lastName": "", "eMail": "", "location": "",
"imageFormat": 1, "imageX": imgWidth, "imageY": imgHeight, "imageTransparency":
false, "imageScaling": false, "maxUpScalePercent": 0.0, "rawDataFormat": "ENC",
"minSigPoints": 25 };

document.addEventListener('SigCaptureWeb_SignResponse', SignResponse, false);
var messageData = JSON.stringify(message);
var element = document.createElement("SigCaptureWeb_ExtnDataElem");
element.setAttribute("SigCaptureWeb_MsgAttribute", messageData);
document.documentElement.appendChild(element);
var evt = document.createEvent("Events");
evt.initEvent("SigCaptureWeb_SignStartEvent", true, false);
element.dispatchEvent(evt);
function SignResponse(event)
{
      var str = event.target.getAttribute("SigCaptureWeb_msgAttri");
      var obj = JSON.parse(str);
      SetValues(obj, imgWidth, imgHeight);
 }
```

8.0 – Signature Capture and Data Export

As the SigCaptureWeb SDK is designed to support Standard Input and Output streams for communication between the browser extensions and the SDK, only text data can be exchanged between the applications. The Input message triggers signature capture, and the input message itself contains all the required data as payload.

The Output message payload will contain the signature signed status, the image data (in the specified format), the raw signature data in base 64 format, connected signature pad information, and a parameter to carry the error message in case signing fails.

The Chrome Google native messaging API (through which the Extension sends and receives data to the SDK) mandates the text data to be in JSON format, hence the input and output messages should be in JSON format and for convenience the same JSON format is followed for Firefox as well. The format of the JSON message is

{text: value1, text1: value2}

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

8.1 – Signature Capture

The ePadLink SigCaptureWeb SDK supports only one input message which will trigger the signature capture using the ePadLink signature pads. It gathers all the required inputs (like signature information (First Name, Last Name, Email, Location), the signature image and raw data format (Image: JPG/PNG, RawData: ENC or SIG), width and height of the image, background image transparency, etc.) for export after the capture.

8.1.1 – INPUT Message

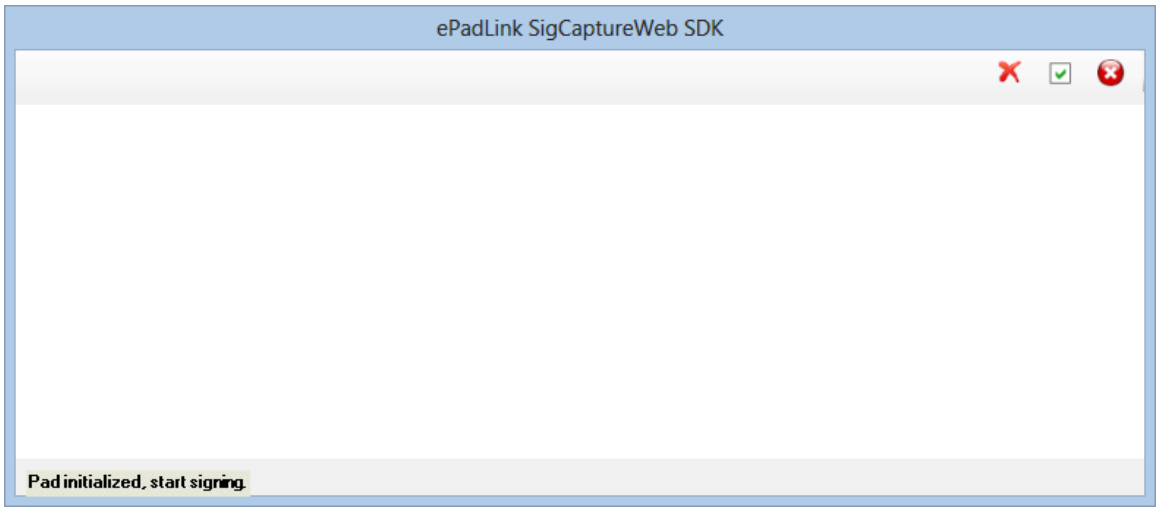
Here is a sample JSON string containing all the supported INPUT.

```
{"firstName": "John" , "lastName": "Doe" , "eMail": "johndoe@demo.com" ,
"location": "LosAngels,CA" , "imageFormat": 2 , "imageX": 300 , "imageY": 150 ,
"imageTransparency": true , "imageScaling": false, "maxUpScalePercent": 0.0,
"rawDataFormat": "ENC" ,  "minSigPoints":25};
```

Parameter	Description
firstName	Signature Detail First Name as String.
lastName	Signature Detail Last Name as String.
eMail	Signature Detail Email as String.
location	The signing location as String, could be a physical location or longitude and latitude.
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 Boolean.
imageScale	Whether to scale the image or not as Boolean.
maxUpScalePercent	Maximum allowed upscale percentage as float (0 to 100).
rawDataFormat	Raw data can be exported after signature capture as string. Use ENC as value for this field for exporting signature raw data in eSign Emcee format and SIG to export it in Topaz .Sig format.
minSigPoints	Minimum number of points required to qualify the signer squiggle as a valid signature as a number.

Once the required JSON input message is formatted, you should pass it to the Chrome or Firefox extension. It should be set as an attribute to the custom html event **SigCaptureWeb_SignStartEvent** raised by a Web page.

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



8.1.2 – OUTPUT Message

The SigCaptureWeb SDK 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/driver related failures.
2. User cancelled signing.
3. User accepted a signature.
4. Signature not captured/accepted.

The output message has a ‘status’ Boolean parameter indicting whether the signature capture is successful or not. Chrome applications can rely on this parameter to identify if a signature is captured or not.

Here is a sample output message:

```
{"isSigned": true , "imgData": "Base64 formatted image data" , "rawSigData":
"Base64 formatted raw signature data in Encrypted eSign Emcee or .Sig format",
“padInfo”: “Connected signature pad information”};
```

Parameter	Description
isSigned	Parameter value specifies whether a signature is captured or not as Boolean. Returns true if user accepted a signature and signature capture is successful. A positive result has succeeding 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 the empty string if the signature is not captured.
rawSigData	Raw Signature data as eSign Emcee Encrypted Base 64 String or as .Sig formatted string. Contains an empty string if the signature is not captured.
padInfo	Contains the information about the signature pad used for signature capture.
errorMsg	In case of any failure or user cancelled signing, this field contains the error message. Client applications can use this info to resolve the errors.

The OUTPUT message is sent back as an event attribute for the event **SigCaptureWeb_SignResponse** raised by the extensions and handled by the web page.

9.0 – End User Deployment

Once the ePadLink SigCaptureWeb SDK integration is completed, the next step is to deploy the required software on end user machines. Follow the steps below:

1. Install the latest ePadLink Universal Installer drivers from www.epadlink.com.
2. For Chrome users, follow the instructions in section 6.1.
3. For Firefox users, follow the instructions in section 6.2.