SigCaptX - User Guide

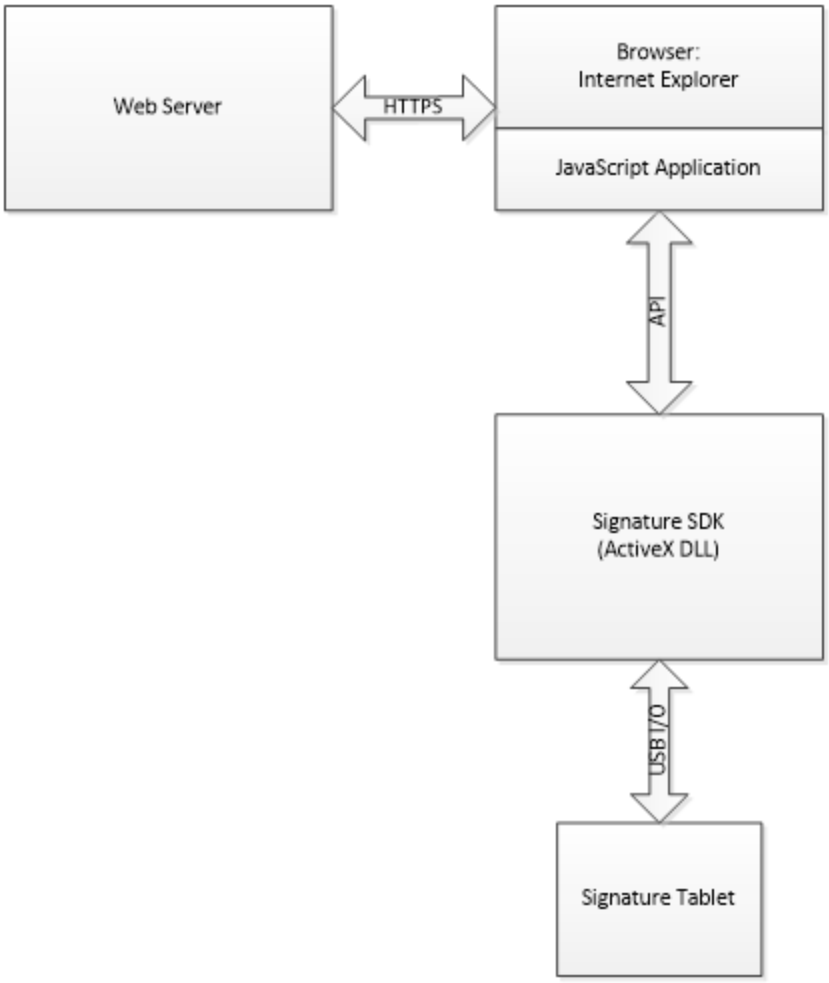
1 Introduction

Support for ActiveX is no longer available in Internet Explorer and is not generally available in alternative browsers such as Firefox and Chrome. SigCaptX has been developed to resolve this issue by providing a cross-browser solution.

1.1 ActiveX method summary

The Signature SDK is supplied as a set of ActiveX controls and these can accessed directly in versions of Internet Explorer which support ActiveX. Versions of Internet Explorer up to IE10 can be used in this way.

To view the method schematically:



The general process is as follows:

- The browser loads an HTML page containing JavaScript application code from the web server.
- To perform signature sdk functions, the application calls directly into the Signature SDK DLL (the ActiveX control installed in Windows).
- The Signature SDK DLL performs the necessary i/o with the signature tablet.
- The Signature SDK DLL renders the signature in its display on the HTML page.

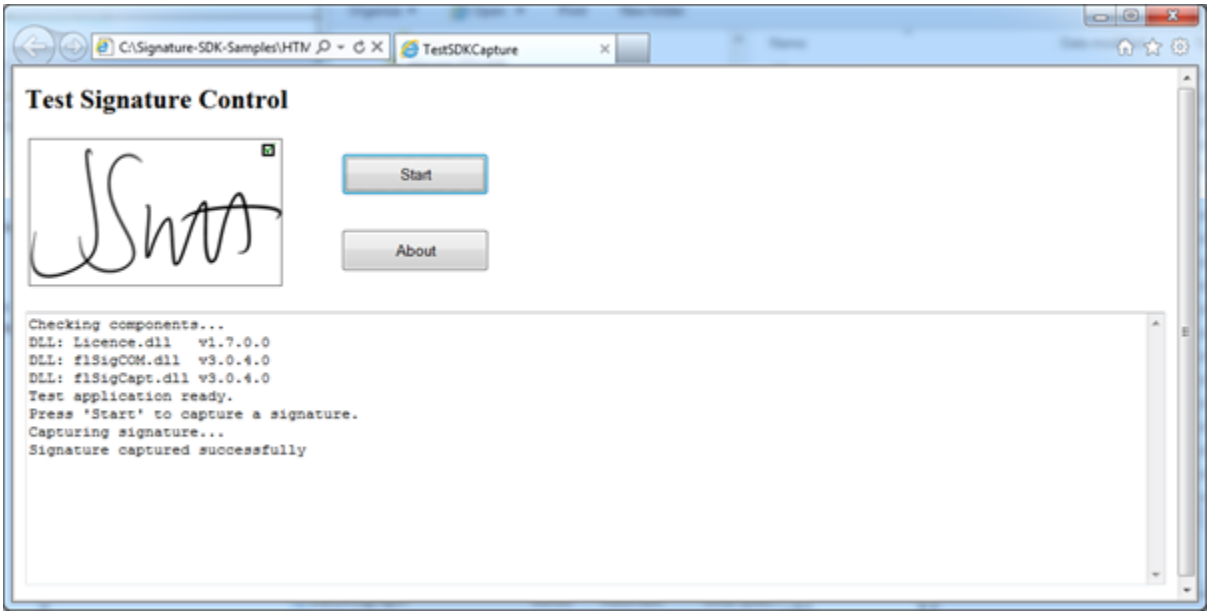
To illustrate, an HTML page creates the ActiveX control:

```
<div>
  <object id="sigCtl1" style="width:60mm;height:35mm"
    type="application/x-florentis-signature">
  </object>
</div>
```

JavaScript application code provides the necessary functionality. For example to capture a signature:

```
  var sigCtl = document.getElementById("sigCtl1");          // get the
signature control
  var dc = new ActiveXObject("Florentis.DynamicCapture");   // create
signature capture
  var rc = dc.Capture(sigCtl, "who", "why");                // call
signature sdk capture
  if(rc == 0 )
    print("Signature captured successfully");               // signature
captured/displayed
```

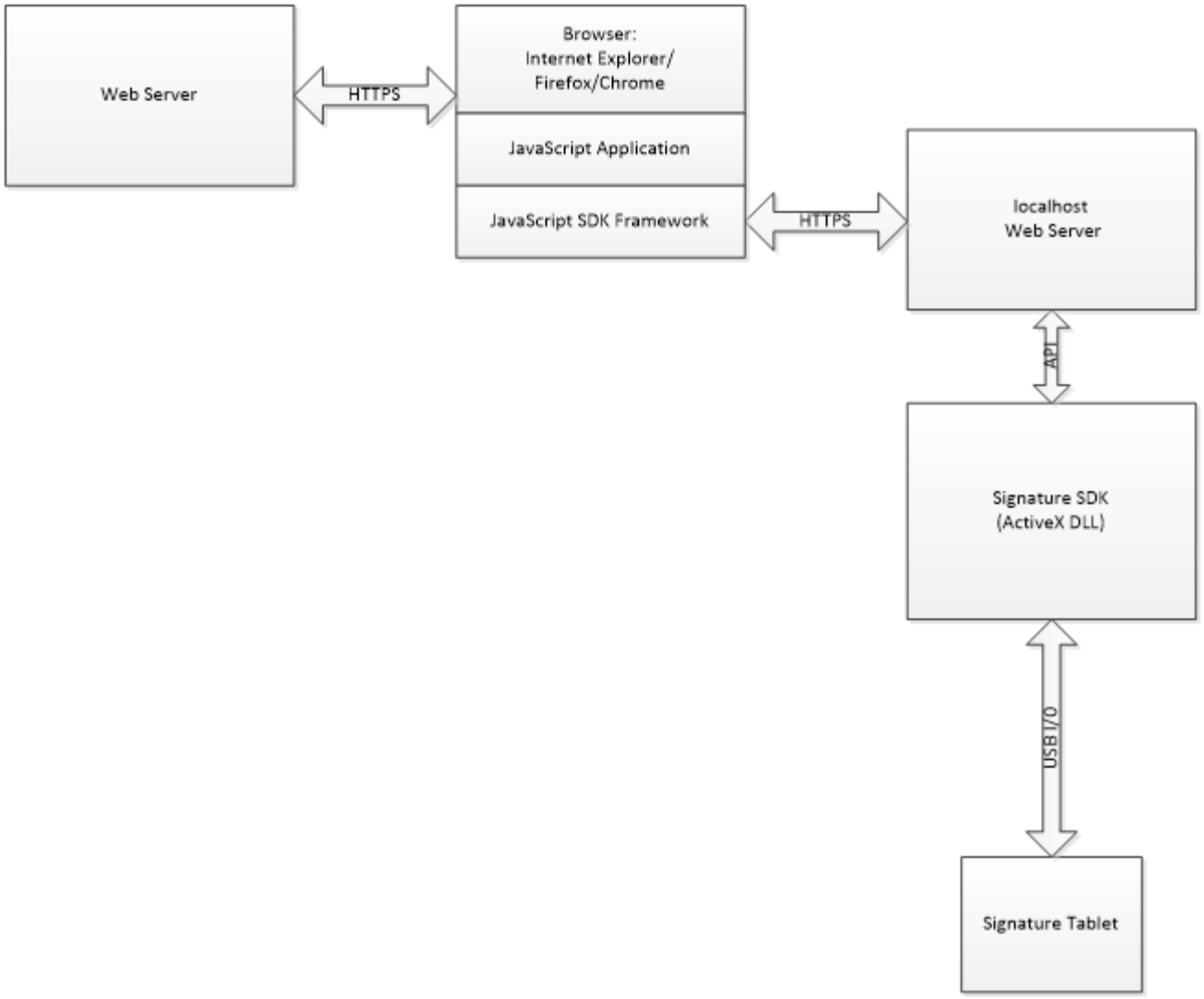
The signature image is displayed automatically in the SigCtl object embedded in the HTML page:



1.2 SigCaptX method summary

In a browser with no ActiveX support it is not possible to access the Signature SDK directly. Instead calls are made indirectly via a localhost web server which is installed as an extension of the Signature SDK. A JavaScript SDK framework is provided to give access to the local web server. The JSONP communication technique is used as the interface to the local web server using HTTPS requests.

To view the solution schematically:



The general process is as follows:

- The web browser loads an HTML page containing JavaScript application code from the web server. The application code includes the SDK framework required to access the localhost web server.
- To perform signature sdk functions, the application code calls the localhost server through the SDK framework. The framework functions each make a JSONP HTTPS request to the localhost server and supply a dedicated callback function. The framework functions return immediately, leaving the server to run independently.
- The server actions the framework function by calling the Signature SDK DLL. For example, in the case of signature capture the DLL performs the necessary i/o with the signature tablet.
- On completion the server uses the JSONP technique to start the callback function which was supplied in the request, passing the relevant return data.
- The callback function retrieves the data and completes the operation.
- For example, in the case of signature capture the HTML application calls the signature capture framework function. Its callback function calls the renderBitmap framework function to request the image of the signature. Its callback function displays the signature image in the html page.

To illustrate, an html page creates the signature image display area:

```
<div id="imageBox" class="boxed" style="height:35mm;width:60mm; border:1px
solid #d3d3d3;">
</div>
```

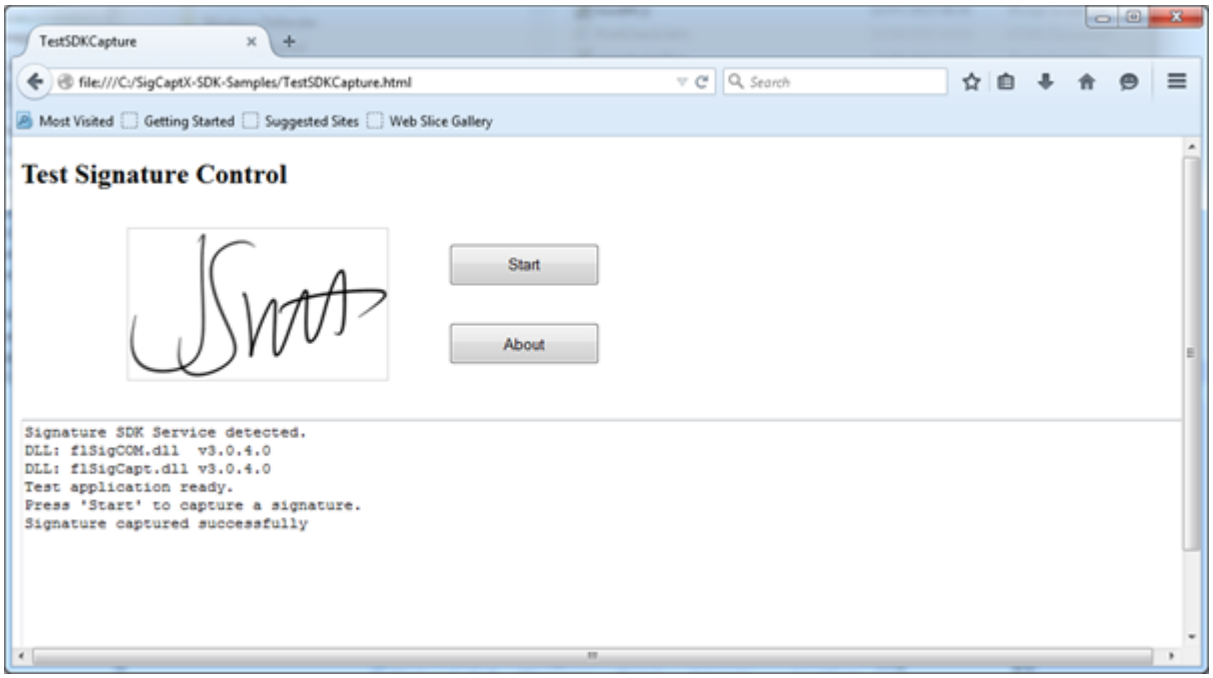
JavaScript application code provides the necessary functionality. For example to capture a signature:

```
function Capture()
{
  // call the SDK framework function with a callback function name
  dynCapt.Capture(sigCtl, "who", "why", null, null, onDynCaptCapture);
  return;


  // the callback function runs when signature capture completes (OK or
Cancel)
  function onDynCaptCapture(dynCaptV, SigObjV, status)
  {
    if(wgssSignatureSDK.DynamicCaptureResult.DynCaptOK != status)
      {
        print("Capture returned: " + status);
      }
      switch( status )
      {
        case wgssSignatureSDK.DynamicCaptureResult.DynCaptOK:
          sigObj = SigObjV;
          print("Signature captured successfully");
          var flags = wgssSignatureSDK.RBFlags.RenderOutputBase64 |
                      wgssSignatureSDK.RBFlags.RenderColor24BPP;
          var imageBox = document.getElementById("imageBox");
          sigObj.RenderBitmap("bmp", imageBox.clientWidth, imageBox.
clientHeight,
               0.7, 0x00000000, 0x00FFFFFF, flags, 0, 0, onRenderBitmap);
          break;
        case wgssSignatureSDK.DynamicCaptureResult.DynCaptCancel:
          print("Signature capture cancelled");
          break;
        default:
          print("Capture Error " + status);
          break;
      }
  }

  function onRenderBitmap(sigObjV, bmpObj, status)
  {
    if(wgssSignatureSDK.ResponseStatus.OK == status)
    {
      var imageBox = document.getElementById("imageBox");
      if(null == imageBox.firstChild)
      {
```

```
          imageBox.appendChild(bmpObj.image);
        }
        else
        {
          imageBox.replaceChild(bmpObj.image, imageBox.firstChild);
        }
      }
      else
      {
        print("Signature Render Bitmap error: " + status);
      }
    }
  }
```



2 Installation

2.1 Summary

Install SigCaptX by running the installer after installing the Signature SDK components.

The SigCaptX installation includes the:

- Web server
- JavaScript SDK Framework

2.2 Web server

The web server contains four significant parts:

2.2.1 Web Service

js_sig_service.exe is installed as a Windows Service and provides an arbitration service.

At startup it reads a configuration value and attaches to the specified localhost port.

By default the service is attached to https://localhost:8000

The web service function is to arbitrate between different instances of the web server: each user logged in to Windows runs a unique instance of the server.

- The service tells a new server instance which port to attach to.
- The service tells a new browser application which port the server is attached to.

2.2.2 Web Server

js_sig_server.exe is the Server which performs the Signature SDK functionality. The service starts this process when it receives a request JavaScript Framework. It automatically ends when the JavaScript Framework is unloaded from the browser. In cases where browser has crashed or has prevented the close signal from being sent, the server will close after a period of inactivity.

On startup the server gets a port number from the web service and attaches to that port.

By default the first server is attached to https://localhost:8001

The next user's server will attach to https://localhost:8002 and so on.

The server responds to HTTPS requests and carries out the requested function by calling the Signature SDK ActiveX DLLs. Each function in the SDK framework has a corresponding function in the server to perform the operation. The server retains the Signature SDK data and uses references to allow the browser application to read and write data values.

On completion of a function call the server completes the HTTPS request which allows the browser application to resume.

2.2.3 Root Certificate

During SigCaptX installation a unique self-signed root certificate is created and installed in the Windows Certificate Store. The certificate is used by the web server to provide the secure HTTPS access required for JSONP communication.

Following installation the certificate can be viewed in the Windows Certificate Manager as:

Unique Localhost Wacom Certificate Authority

2.2.4 Configuration

During installation registry values are created in:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wacom\SigCaptX]
```

or for a 32-bit server on 64-bit Windows:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Wacom\SigCaptX]
```

The values are generally reserved for internal use but the following value is used to define the web service start port:

```
start_port  REG_DWORD   0x00001f40 (8000)
```

If a web service port other than 8000 should be used it can be defined here.

If required, the upper limit of web server port allocation can also be changed from the default value:

```
end_port    REG_DWORD   0x0000ffff (65535)
```

2.3 JavaScript SDK Framework

2.3.1 Overview

The JavaScript SDK framework is provided in the file signaturesdk.js.

A web application includes the framework script in the normal way:

```
    <script src="signaturesdk.js"></script>
```

The framework provides:

- Wacom Signature SDK functions
- JSONP communication functions

The framework replicates the API of the Wacom Signature SDK. API equivalent Signature SDK objects are created in the framework and control is passed to the web server through the use of JSONP communication. The same object methods and properties are

available through the framework while the data is held in the web server. That is, the framework provides indirect access to the Signature SDK.

The concept of JSONP communication can be illustrated with a simple example:

In a web page containing:

```
<script type="text/javascript">
function my_callback(data)
{
  window.alert("my_callback data: " + data.number);
}
</script>
<script src="scriptfile.js"></script>
```

The contents of scriptfile.js:

```
       my_callback({"number": 10})
```

When the web page is opened in a browser the static file scriptfile.js is loaded and executed, resulting in a popup message displaying “my_callback data: 10”.

The SDK framework uses an extensive elaboration of this technique by requesting script from the local web server. The web server creates the script dynamically and is able to insert the return data and the callback function name supplied in the request.

2.3.2 API

The SDK framework is accessed through its main class WacomGSS_SignatureSDK.

Create objects in your application equivalent to those using the Signature SDK directly:

	WacomGSS_SignatureSDK
	SigCtl
	DynamicCapture
	SigObj
	Hash
	Key
	WizCtl
	InputObj

Refer to the Wacom Signature Components API for details of the object methods and properties.

Note that SigCtl is maintained for API compatibility. The SigCtl properties relate to its appearance as an ActiveX control and are not applicable to the framework implementation.

3 SDK Samples

3.1 PortCheck

Use the PortCheck test page to verify the SigCaptX installation:

The samples can usually be copied to disk and opened in a browser, for example:

file:///C:/SigCaptX-SDK-Samples/PortCheck.htm

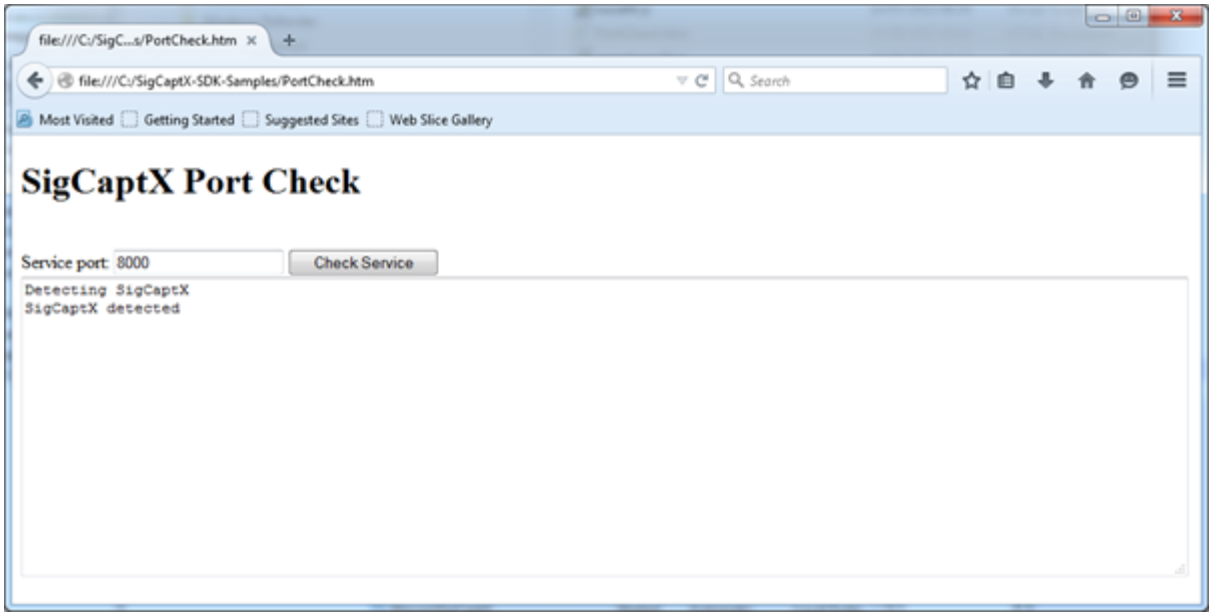
Depending on security settings this can raise a warning, for example:

Allow the blocked content to continue.

If no messages appear in the test page the browser may not be prompting to allow scripts to run. This can be the case when the Windows 10 Edge browser opens a local file.

Security warnings can be avoided by hosting the samples on a secure site, as would be the case in a production system. If required, the test page PortCheck.htm is available on the Wacom GSDT site:

https://gsdt.wacom.eu/support/SigCaptXDemo/PortCheck.htm



Use the PortCheck test page to verify the SigCaptX installation:

Web Service:

If the service is not running or is not accessible the following message appears:

```
Detecting SigCaptX
SigCaptX service not detected
```

Web Server:

If the web service is running but the server has not been started the following message appears:

```
Detecting SigCaptX
SigCaptX service detected, but not the server
```

Root certificate:

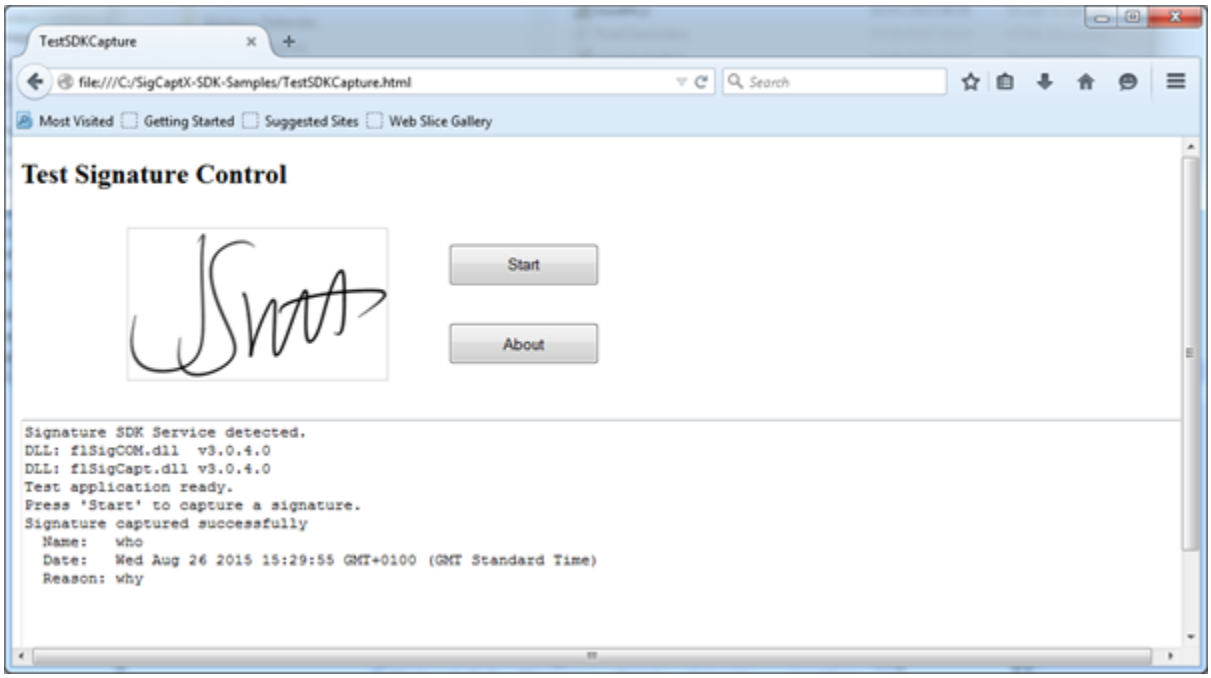
If the certificate has not been successfully installed the browser will display a warning similar to:

3.2 Signature Capture

The SigCaptX SDK samples reproduce the functionality of HTML ActiveX samples supplied with the Signature SDK.

TestSDKCapture.html demonstrates basic signature capture:

Press Start to sign on the signature tablet and display the signature:



Double-click the signature to display information stored in the signature (Name/Date/Reason)