**Edit the web files for a custom widget**

Edit the web files for a custom widget in order to develop the content of the widget and link the widget's properties and events.

Before you begin this task, you should be familar with how to develop webpages using HTML5, CSS, and JavaScript. Also, you must have already created the custom widget and added it to your project's library; the associated web files are automatically created in your project folder only after the widget is added to the library.

The web files should be located at:

*<project name>*\Web\Widgets\*<widget name>*\

Each custom widget actually comprises three web files, but only two of the files are user-editable:

* index.html is the webpage itself. It is what is displayed within the widget's frame in the project screen. You may edit the entire body of the webpage (i.e., everything between <body> and </body>).
* custom\_widget.js is the library of JavaScript functions that are associated with the webpage. You need to develop functions that link the widget's properties and events with the actual content and behavior of the webpage.

Do not edit the third file, *<widget name>*.wjson. It contains important settings for the custom widget.

Now, given the almost limitless ways in which you can develop an HTML5-compliant webpage, it is beyond the scope of this documentation to cover every possible step and option in editing these web files. Instead, the rest of this topic will feature a simple example of a custom widget that can load another webpage and then notify your project that the webpage was loaded. The widget — which you created earlier; see [Create a new custom widget](mk:@MSITStore:C:\Program%20Files%20(x86)\Wonderware\InTouch%20Edge%20HMI%20v8.1\Bin\TechRef.CHM::/topics/graphics_customwidgets_newwidget.html#graphics_customwidgets_newwidget) — should have at least one property named URL and one event named PageLoaded.

To edit the web files for a custom widget:

1. Locate the widget's web files in your project folder.

The files should be located at:

*<project name>*\Web\Widgets\*<widget name>*\custom\_widget.js

*<project name>*\Web\Widgets\*<widget name>*\index.html

1. Use a text editor to open index.html.

The default contents of <body> are a simple badge and label.

<!DOCTYPE html>

<html style="overflow: hidden;">

<head>

<script src="../Resources/Apis/Proxy.js"></script>

<script src="./custom\_widget.js"></script>

<title>MyWidget</title>

</head>

<body>

**<div style="width:96vw;height:95vh;background-color:white;text-align:center;vertical-align:middle;line-height:98vh;border:solid;border-width:thin;border-color:#e6e9eb">**

**<div>**

**<img src="../Resources/Images/HTML5.png" style="width:32px;height:32px"/>**

**<p></p>**

**<div style="height: 64px; top: 28px; width: 100%; position: absolute;">MyWidget</div>**

**</div>**

**</div>**

</body>

</html>

1. Delete the default contents of <body>.
2. <!DOCTYPE html>
3. <html style="overflow: hidden;">
4. <head>
5. <script src="../Resources/Apis/Proxy.js"></script>
6. <script src="./custom\_widget.js"></script>
7. <title>MyWidget</title>
8. </head>
9. <body>
10. </body>

</html>

1. Insert your own HTML code into the body of the webpage. The entire body is displayed within the widget's frame in your project screen.

In this example, you are inserting an iframe element that can be used to load other webpages. (The iframe element is a sort of browser window within the browser window, or in this case, a browser window within the custom widget.)

<!DOCTYPE html>

<html style="overflow: hidden;">

<head>

<script src="../Resources/Apis/Proxy.js"></script>

<script src="./custom\_widget.js"></script>

<title>MyWidget</title>

</head>

<body>

**<iframe id="myFrame" style="width: 100vw; height: 100vh;"></iframe>**

</body>

</html>

1. Save and close index.html.
2. Use the text editor to open custom\_widget.js.
3. // Subscribes to receive property changes
4. cwidget.on("PropertyName", function() {
5. // Gets property value
6. console.log(cwidget.PropertyName);
7. // Sets property value
8. cwidget.PropertyName = "value"
9. // Triggers an event
10. cwidget.dispatchEvent("EventName");

});

This is an example of a JavaScript function that can be executed during project run time. cwidget is a JavaScript object that represents your custom widget, and it has various properties and methods associated with it. For example, cwidget.on executes code when the value of a property changes, cwidget.PropertyName (e.g., cwidget.URL) accesses the value of the property itself, cwidget.dispatchEvent notifies your project when an event has occurred, and so on.

You can copy this example as many times as you want, for however many properties and events you have added to your custom widget, and then develop each function to do something different. You can also develop entirely new functions and sub-routines, using your knowledge of JavaScript.

1. Delete the body of the function, so that you can insert your own commands.
2. cwidget.on("PropertyName", function() {

});

1. Replace PropertyName with the name of the property you added to your custom widget.

In this example, you are using the URL property.

cwidget.on("**URL**", function() {

});

Now this function will be executed whenever there is a change in the value of your widget's URL property (or more specifically, whenever there is a change in the value of the project tag associated with the URL property).

1. Develop your function.

In this example, your function will load a webpage and then notify your project that the webpage was loaded.

cwidget.on("URL", function() {

**var myFrame = document.getElementById("myFrame");**

**myFrame.onload = cwidget.dispatchEvent("PageLoaded");**

**myFrame.src = cwidget.URL;**

});

The first line declares a new JavaScript object named myFrame and then links it to the iframe element that you inserted into your HTML file. The second line causes the PageLoaded event to be dispatched when a webpage is loaded. (In other words, the widget in your project is notified that the PageLoaded event occurred.) And the third line sets the source of myFrame to be equal to the widget's URL property, which causes the iframe element to load the specified URL.

Remember, the entire function depends on the cwidget.on method, so whenever there is a change in the value of your widget's URL property, this function will be executed and a new webpage will be loaded.

1. Save and close custom\_widget.js.

Of course there are many other things you can do with these web files, but that is beyond the scope of this documentation. More thorough descriptions and examples will be provided in future releases of this software. In the meantime, if you need help with developing your custom widgets, please contact your InTouch Edge HMI software distributor.

**Parent topic:** [Custom Widget](mk:@MSITStore:C:\Program%20Files%20(x86)\Wonderware\InTouch%20Edge%20HMI%20v8.1\Bin\TechRef.CHM::/topics/graphics_customwidgets.html)

**Previous topic:** [Create a new custom widget](mk:@MSITStore:C:\Program%20Files%20(x86)\Wonderware\InTouch%20Edge%20HMI%20v8.1\Bin\TechRef.CHM::/topics/graphics_customwidgets_newwidget.html)

**Next topic:** [Insert and configure a custom widget](mk:@MSITStore:C:\Program%20Files%20(x86)\Wonderware\InTouch%20Edge%20HMI%20v8.1\Bin\TechRef.CHM::/topics/graphics_customwidgets_insertwidget.html)

**Related concepts**

[Automatic resizing of custom widgets](mk:@MSITStore:C:\Program%20Files%20(x86)\Wonderware\InTouch%20Edge%20HMI%20v8.1\Bin\TechRef.CHM::/topics/graphics_customwidgets_autoresize.html)