4D Live Window JavaScript Part 2 – Injection

By Josh Fletcher, Technical Support Engineer, 4D Inc.

Technical Note 07-36

Abstract

Version 1.1 of the 4D Live Window (4DLW) plug-in, which is part of 4D Web 2.0 Pack, added features to support the execution of JavaScript. This Technical Note explores the technique of JavaScript "injection", which allows the developer to add JavaScript to any page displayed in a 4D Live Window plug-in area.

A sample database is provided.

Introduction

The 4DLW plug-in allows the developer to load any HTML content inside a 4D database using the underlying Web browser of the host OS. With the release of 4DLW 1.1, the ability to execute JavaScript functions within the loaded page was added. This feature allows the 4D database to interact directly with the HTML page.

As was shown in Technical Note 07-31, *4D Live Window JavaScript Part 1 – Offscreen Areas*, this feature allows the 4D developer to treat JavaScript functions as code libraries.

This Technical Note will explore the ability to "inject" JavaScript into existing pages. 4DLW 1.1 gives the developer the ability to add JavaScript, any JavaScript, to a page displayed in a 4DLW plug-in area. This can be done even if the page contains no JavaScript at all!

Limitations

Currently the execution of JavaScript is only supported on Windows. The browser layer on Mac OS (Webkit) is simply too unstable when programmatically executing JavaScript and these instabilities can cause the 4D database to crash. As such the demo database will not function correctly on Mac OS. For more information please refer to the 4DLW documentation.

Also the sample database includes the demo version of 4DLW 1.1.2, which expires after 30 minutes of use.

4DLW JavaScript Execution

This Technical Note makes use of the 4DLW function ***Web_JavaScriptExecute***. Here is the documentation for the function:

`Web_JavaScriptExecute (Area; Script) -> error code`

Parameter	Type	Description
Area	Longint	4D Live Window area
Script	Text	JavaScript
Function result	Longint	Error code (0 = No error)

The The command Web_JavaScriptExecute allows the developer to execute JavaScript code. It does not return a result (see Web_JavaScriptReturn for details).

The parameter *Script* can contain the name of an existing JavaScript function or a new JavaScript function, allowing you to "inject" code into displayed web pages.

There is one thing worth clarifying with regards to the documentation: it mentions that the **name** of an existing JavaScript function can be passed. While this is technically true, it may not produce the desired effect. In fact the contents of the *Script* variable should be, literally, JavaScript. Any JavaScript! So, for example, assuming that JavaScript function named *someFunction* exists and needs to be executed, the *Script* parameter should contain "someFunction();" and not "someFunction". This differs from the *Web_JavaScriptReturn* command, which literally takes the function name.

The underlying issue here is that if only the name of the function is passed it is a "reference". The ability to "refer" to a function is critical to the power of JavaScript. This allows a function to be redefined (or overridden), as in:

```
document.prompt = function(){ alert("ha ha!"); };
```

In the above example the "prompt" function (which already exists as a member of the "document" object) is redefined by referring to it. This is completely different than:

```
document.prompt();
```

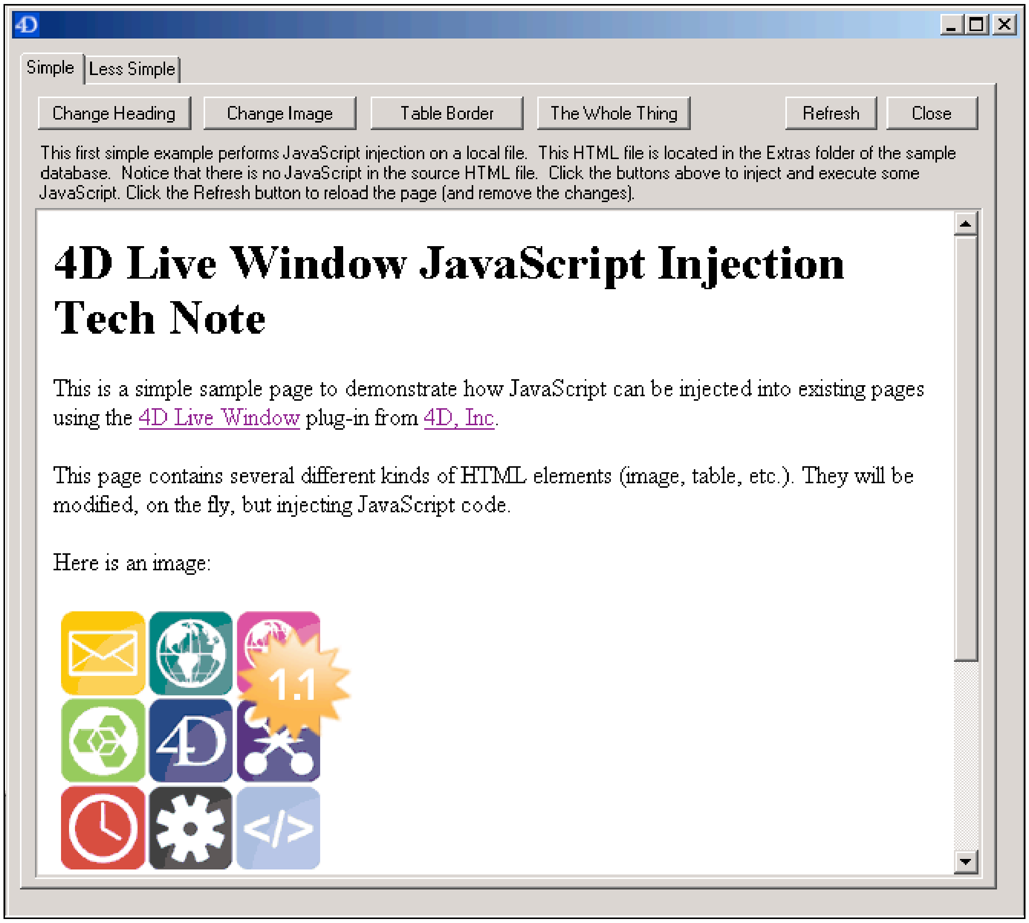
The above example actually executes the JavaScript function as opposed to referring to it.

Demo Database

The demo database included with this Technical Note uses JavaScript injection to modify two different Web pages; one a local HTML file, and the other a "live" Web page (internet connection required).

Using the Demo Database

The demonstration window in the demo database contains a form with 2 pages. Open it from the File menu (select “Open Test Dialog…”):



This first simple example performs JavaScript injection on a local file. This HTML file is located in the Extras folder of the sample database. Notice that there is no JavaScript in the source HTML file. Click the buttons above to inject and execute some JavaScript. Click the Refresh button to reload the page (and remove the changes).

4D Live Window JavaScript Injection Tech Note

This is a simple sample page to demonstrate how JavaScript can be injected into existing pages using the 4D Live Window plug-in from 4D, Inc.

This page contains several different kinds of HTML elements (image, table, etc.). They will be modified, on the fly, but injecting JavaScript code.

Here is an image:

The first page contains a 4DLW plug-in area with a local file loaded. The second page contains a 4DLW plug-in area with a “live” Web page loaded.

At the top of each page are some buttons that can be used to perform certain JavaScript actions. To undo the modifications made by these buttons use the Refresh button to reload the page.

All navigation is disabled for both pages.

The JavaScript snippets themselves are stored in the [JavaScripts] table. These can be edited, from Custom Menus mode, by opening the File menu and selecting “Edit JavaScripts…”. Of course they can also be accessed from User Mode.

JavaScript Injection Demos

Local Page Demo

The local page “4DLW_JS_Inject.html” is located in the “Extras” folder of the demo database. Note that this file contains no JavaScript. There are 4 possible JavaScript injections that can be performed on this file:

Button Title	JS Function Name	Description
Change Heading	changeTitle	Modifies the H1 element “LWJS2_Title” by changing its content (innerHTML) and style (“color: red”).
Change Image	changeImage	Changes the source of the “LWJS2_Image” IMG element to point to a different image
Table Border	changeTableBorder	Adds a thicker border to the “LWJS2_Table” TABLE element.
The Whole Thing	hideEverything	Hides the entire document by setting the style of the “LWJS2_Everything” DIV to “display: none”.

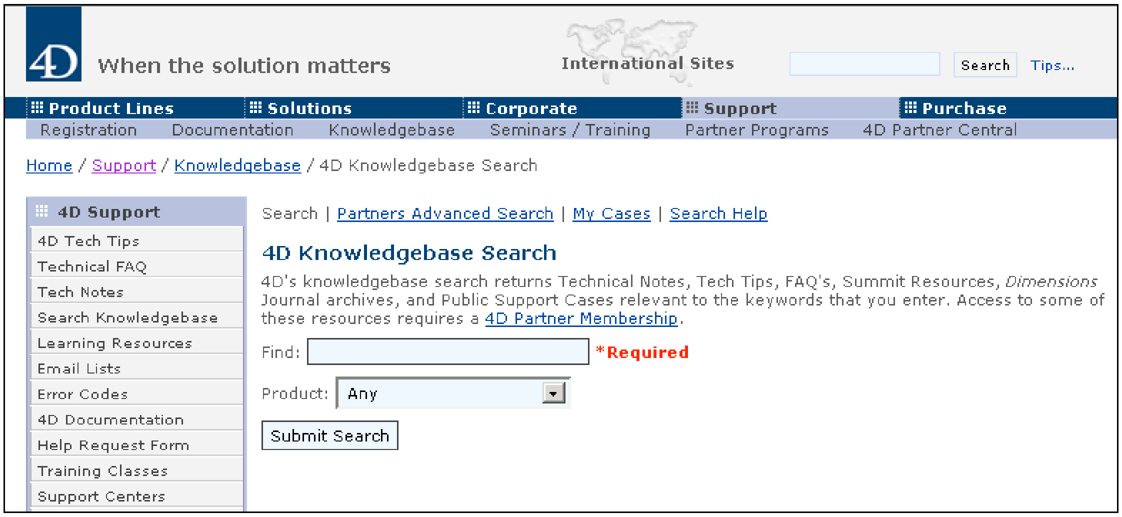
Remote Page Demo

NOTE: The use of and modification of “real” Web pages may not be allowed in a commercial application. For example, the use of Google’s Google Maps API is governed by a license that insists any application that uses the API must be freely available (they have a separate API for commercial apps). Similarly, if you plan to use JavaScript injection to modify a page that you do not own, it would behoove you to contact the page owner and make sure it is allowed.

The remote page is the 4D Knowledgebase Search page, found here:

http://www.4d.com/knowledgebase

Here is a screenshot of the page:



4D Knowledgebase Search

4D's knowledgebase search returns Technical Notes, Tech Tips, FAQ's, Summit Resources, *Dimensions* Journal archives, and Public Support Cases relevant to the keywords that you enter. Access to some of these resources requires a 4D Partner Membership.

Find: *Required

Product: Any

Here are the possible JavaScript injections that can be performed:

Button Title	JS Function Name	Description
Change Colors	changeColors	This is the most complex example. Refer to the JavaScript to see all of the changes. It overrides several styles in the page in order to change the color scheme.
Hide Sidebar	hideSidebar	Hides the block of links that appears on the left of the page.
Hide Breadcrumbs	hideBreadcrumbs	Hides the "breadcrumb trail" at the top of the page.

Inspection of the JavaScripts for the above examples will reveal that most of the changes are accomplished by modifying objects or CSS. One obvious question is how do I figure out the name of the object or CSS class I want to modify? If you are the author of the page it is relatively easy since you probably designed the CSS. If not see the next section.

How to Decide What to Inject

The first demo uses a local page. Deciding how to modify this page was relatively trivial since I designed the page; I already knew the IDs of the DIVs and there is no CSS.

The "live" 4D Knowledgebase search page is another matter entirely. I was not familiar with the design of the page so I needed to be able to identify the names of the objects within the page, as well as the names of the CSS classes I wanted to modify, preferably without manually scouring the files.

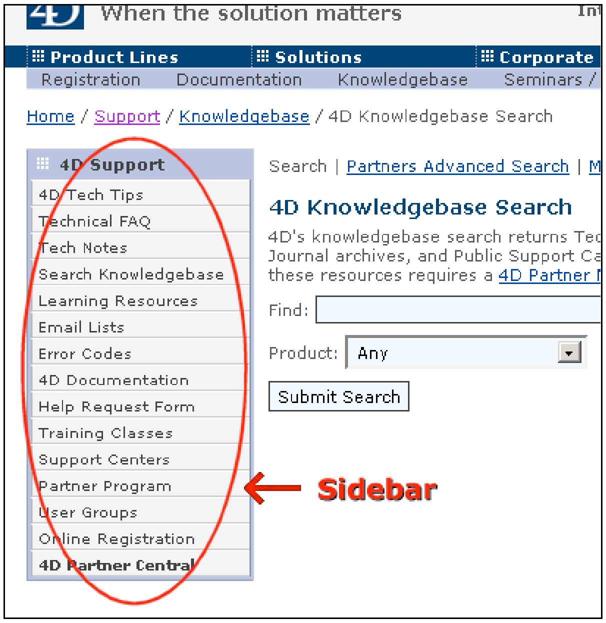
Thankfully there are tools out there to help with this. The most powerful, by far, is Firefox's Firebug extension:

https://addons.mozilla.org/en-US/firefox/addon/1843

With Firebug you can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page. You can also browse the entire DOM tree that represents the current web page.

For this Technical Note the most useful feature in Firebug was the ability to "Inspect" a page. This makes identifying individual objects within the page quite easy.

For example, I needed to know the ID of the “sidebar” mentioned previously so that I could hide it:



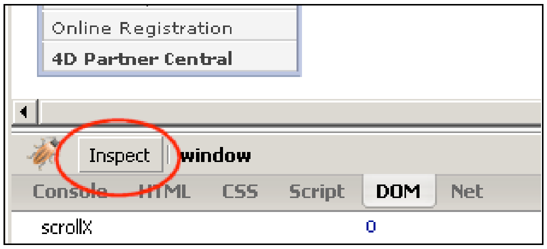
4D Knowledgebase Search

4D's knowledgebase search returns Tec
Journal archives, and Public Support Ca
these resources requires a 4D Partner

Find:

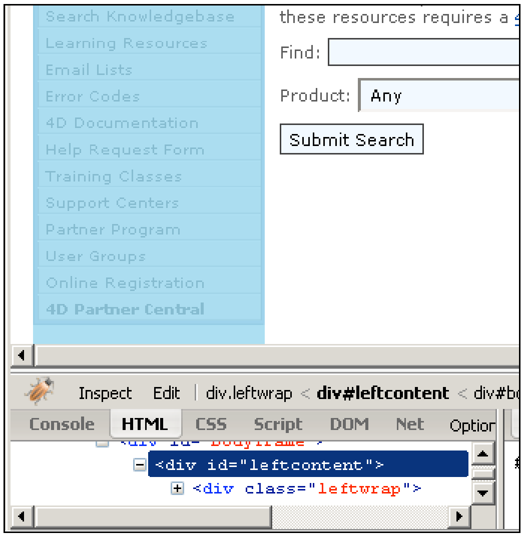
Product: Any

In Firebug, click the “Inspect” button:



scrollX 0

Then simply move the mouse around the page to identify the desired object:



these resources requires a

Find:

Product: Any

```
<div id="leftcontent">
  <div class="leftwrap">
```

In this case the DIV I wanted was "leftcontent" (notice that it is highlighted). Once I had identified the ID of the DIV I wanted, I hid it with the following JavaScript:

```
document.getElementById('leftcontent').style.display='none';
```

Keep in mind however that 4DLW uses Internet Explorer on Windows and Safari on Mac so the structure of the DOM will not necessarily be the same as in Firefox. For example, in Firefox's DOM the style rules are in a Collection called "CSSRules" and any given CSSRule node can contain a style sheet (instead of just a rule). In Internet Explorer style rules are in a Collection called "rules" and there is a separate Collection for style sheets (called "stylesheets").

There are tools out there for debugging Web pages in Internet Explorer and Safari but they are neither as robust as, nor easy to use as, Firebug so I recommend starting there.

Project Method Details

This section makes note of some of the more interesting Project Methods from the example database. Note that all Project Methods in the database contain comments. Please refer to the code for details on methods not mentioned here.

Naming Convention

The Project Methods in the example database use the following naming convention:

*LWJS2_*	These methods are the "important ones".
*LWJS2_M_*	Methods hooked up to menu items.
*LWJS2_P_*	Process methods.
*UTIL_*	Utility methods, completely generic.
*zLWJS2_K_*	Custom "constants".

LWJS2_ExecuteJS

This method installs and executes to indicated JavaScript using *Web_JavaScriptExecute*. The JavaScripts are located in the [JavaScripts] table in this database, but of course they do not have to exist in a table. The parameter to *Web_JavaScriptExecute* is simply a string and should contain valid JavaScript.

This method breaks the process up into two steps:

- First the script to be injected is added to the page (this is the first call to *Web_JavaScriptExecute*).
- Then the script is executed (this is the second call to *Web_JavaScriptExecute*)

Note that this is not necessary; the script could be loaded and executed in the same call. Remember that what you pass to *Web_JavaScriptExecute* is "raw" JavaScript. It is perfectly valid to do something like this:

```
function someNewFunction ()
{
   // Do some stuff…
};

someNewFunction();
```

This both creates and calls "someNewFunction" in the same script.

Button Methods

The button methods in the form [zDialogs]"LWJS2_Inject" trigger the actual "injection" and execution of the JavaScripts. These are simple one line methods that specify the plug-in area as well as the name of the script in the [JavaScripts] table.

Conclusion

With 4D Live Window 1.1 the developer has the ability to execute any JavaScript function loaded into a 4DLW plug-in area. Additionally JavaScript can be injected into the page, giving the developer complete control over its contents without needing to modify the source.

This Technical Note described the technique of using 4D Live Window to perform JavaScript injection on both a local HTML page as well as a "live" page from the Web.

A Note about 4D Web 2.0 Pack

The products in 4D Web 2.0 Pack are a departure from most other 4D products. As 4D Web 2.0 Pack is a subscription-based product it is expected that incremental releases will be made. Thus please note that this Technical Note is based on 4D Live Window 1.1.2. As new features are implemented this Technical Note may become obsolete (faster than most other 4D products). For the latest information on 4D Live Window consult the latest documentation and also check the 4D Web 2.0 Pack Wiki:

http://daxipedia.4d.com

For the latest news about 4D Web 2.0 Pack please see:

http://www.4d.com/products/4dweb20pack.html