4D Live Window JavaScript Part 1 – Offscreen Areas

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

Technical Note 07-31

Abstract

Version 1.1 of the 4D Live Window (4DLW) plug-in, which is part of 4D Web 2.0 Pack, adds features to support the execution of JavaScript. For technical reasons the plug-in does not support offscreen areas. However, with the addition of the ability to treat JavaScript functions as "code libraries", the ability to use a 4DLW plug-in area as an offscreen area is desirable. This Technical Note presents an implementation that allows this.

A demo 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. Perhaps more importantly, this feature allows the 4D developer to treat JavaScript functions as code libraries. This is the ability that will be explored in this Technical Note.

Of course, in most 4D plug-ins this kind of interaction might occur off-screen. Unfortunately, due to technical limitations of the underlying Web browsers, there is currently no way to create an "offscreen" plug-in area with 4DLW. Instead this Technical Note uses a 4D Window that contains a 4DLW plug-in area but is hidden off-screen. With this solution the JavaScript loaded in the hidden window can be treated as if it were a code library.

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_JavaScriptReturn***. Here is the documentation for the function:

Web_JavaScriptReturn (Area; Function; Result; {Parameter1; Parameter2; Parameter3; Parameter4; Parameter5}) -> error code

Parameter	Type	Description
Area	Longint	4D Live Window area
Function	Text	Name of an existing JavaScript function
Result	Text	Result of JavaScript function
Parameter1	Text	Optional Parameter 1
Parameter2	Text	Optional Parameter 2
Parameter3	Text	Optional Parameter 3
Parameter4	Text	Optional Parameter 4
Parameter5	Text	Optional Parameter 5
Function result	Longint	Error code (0 = No error)

The command Web_JavaScriptReturn permits executing an existing JavaScript function with up to 5 parameters, returning the result.

The parameter Function must name an existing JavaScript function, otherwise the error -15004 is returned. The JavaScript function can be "injected" using Web_JavaScriptExecute before using this command. Parameters for this function can be passed (up to 5 parameters). The routine always expects and returns parameters as text, even if the JavaScript function uses other variable types.

Implementation

At its core the design implemented in the example database is simply a dialog "hidden" by moving it offscreen. The dialog contains form that has a 4DLW plug-in area in it. In this way the 4DLW plug-in area can be used without the user knowing it is there. This dialog is referred to as the JavaScript Utility Window throughout this Technical Note.

Naming Convention

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

*LWJS_FN_*	These methods represent individual JavaScript function calls.
*LWJS_MISC_*	Miscellaneous methods.
*LWJS_UTILWIN_*	Methods that pertain to the "offscreen area"
*UTIL_*	Utility methods, completely generic.
*zLWJS_K_*	Custom "constants".

Interprocess variables follow this convention as well.

JavaScript Utility Window

The JavaScript Utility Window form is called "LWJS_UtilWin" and is attached to the [LWJS_Dialogs] table.

The window type of the JavaScript Utility Window is a "Palette Window" so that it minimally interferes with focus.

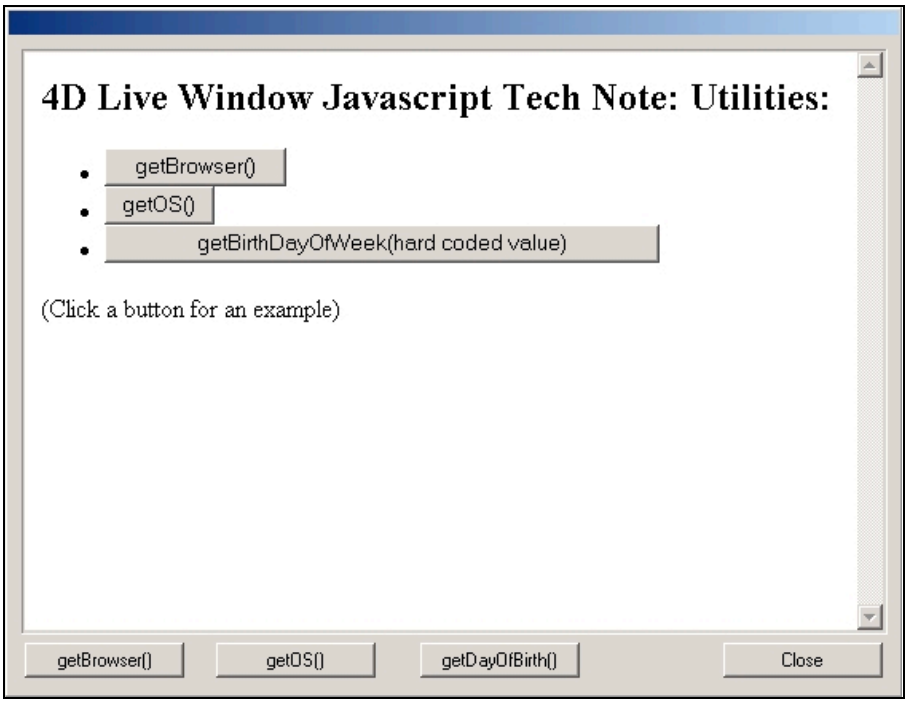
The window is opened offscreen by calculating the screen size and then adding on some padding. For example:

```
` I want to open the utils window offscreen...
$left:=Screen width+100
$top:=0

$right:=$left+300
$bottom:=$top+500

$winref:=Open window($left;$right;$top;$bottom;Palette window )
```

Note that the *process* displaying the dialog is **not** hidden (e.g. with HIDE PROCESS). Hiding the process would disable the 4DLW plug-in area.

The HTML page loaded into the 4DLW plug-in area contains three simple JavaScript functions. This file is located in the "Extras" folder of the sample database and is called "4DLW_JS_Utilities.html".

Here is a screenshot of the window:



The buttons embedded in the HTML page are for testing the JavaScript code in the context of the browser. The form buttons test the code from within the context of 4D. Both are present simple for testing and not necessary for the functionality.

Custom Event Model

Since the offscreen area is running in its own process this process must be called (using CALL PROCESS) in order to respond to the application's needs. In order to support multiple behaviors and provide the ability to interact with the JavaScript Utility Window a "custom event" model is used. This consists of an interprocess variable used to store the event, and an event handler used to process the event.

Once an event has been generated the JavaScript Utility Window handles the event and, if necessary, provides a result. The calling method waits for the result as appropriate.

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 no mentioned here.

The JavaScript "Functions"

The example database implements 3 JavaScript functions as 4D Project Methods. In this way the JavaScript code can be treated as any other 4D code. These are obviously very simple examples, but the idea presented here can be extended to include thousands of existing JavaScript functions.

LWJS_FN_GetBrowser

This method uses the "navigator.userAgent" JavaScript object to get information about the Web browser being used in the 4DLW plug-in area. A string is returned that indicates the browser in use. There are no parameters to this method.

LWJS_FN_GetDayOfBirth

This method calculates the day (of the week) of birth based on a birth date. Uses Zeller's algorithm and was adapted from an example at:

http://www.javafile.com/calc/calc3.php

This method accepts 1 parameter, a 4D Date, representing the birthday.

Note that ***Web_JavaScriptReturn*** only accepts string values as parameters, so the input parameter must be converted into strings.

LWJS_FN_GetOS

This method uses the "navigator.appVersion" JavaScript object to get information about the Operating System in use. A string is returned that

identifies whether a Mac, Windows, or unknown OS is in use. There are no parameters to this method.

Other Methods

These methods are of note.

LWJS_UTILWIN_OnOutsideCall

This Project Method contains that actual code to make the JavaScript calls (in addition to the code that manages the window itself). This is the event handler of the custom event model.

LWJS_UTILWIN_Hide
LWJS_UTILWIN_Show

These two project methods are used to "hide" and "show" the JavaScript Utility Window. These methods notify the Utility Window process, which manages the window by moving it on or off the visible screen.

LWJS_UTILWIN_Refresh

This method is useful for development. It tells the JavaScript Utility Window process to refresh the HTML page being displayed in the 4DLW plug-in area. In this way changes to the JavaScript will be taken into account.

LWJS_UTILWIN_Restart

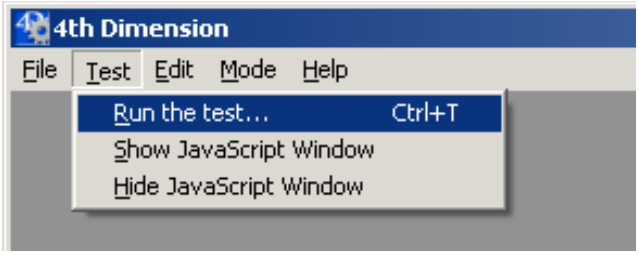
This method is also useful for development. It shuts down the JavaScript Utility Window process and re-launches it. Use this to pick up changes to the Utility Window form.

LWJS_UTILWIN_Stop

This method shuts down the JavaScript Utility Window process. Additionally it waits for the process to end, so that the database can quit gracefully.

Using the Example Database

The example database starts up in Custom Menus mode. There is a menu item that will run though a summary of the features as well as show the JavaScript Utility window, as indicated below:



Additionally you can show and hide the window with the respective menu items.

Conclusion

With 4D Live Window 1.1 the developer has the ability to execute any JavaScript function loaded into a 4DLW plug-in area. This Technical Note described a technique that can be used in order to implement an offscreen plug-in area for the 4DLW plug-in. By combining these two designs the 4D developer has the ability to treat any JavaScript function as a 4D method.

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