

Creating Add-ins

for Microsoft®

Expression Web 4

Using Existing HTML and
JavaScript Skills to Build Add-ins
for Microsoft Expression Web

CREATING MICROSOFT EXPRESSION WEB 4 ADD-INS: USING EXISTING HTML AND JAVASCRIPT SKILLS TO BUILD ADD-INS FOR MICROSOFT EXPRESSION WEB

Copyright © 2011 by Que Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-7897-4102-8

ISBN-10: 0-7897-4102-4

First Printing: February 2011

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Que Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Microsoft is a registered trademark of Microsoft Corporation.

Expression is a registered trademark of Microsoft Corporation.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Associate Publisher

Greg Wiegand

Acquisitions Editor

Loretta Yates

Development Editor

Todd Brakke

Managing Editor

Sandra Schroeder

Project Editor

Seth Kerney

Copy Editor

Geneil Breeze

Indexer

Cheryl Lenser

Proofreader

Linda Seifert

Technical Editor

Kathleen Anderson

Publishing Coordinator

Cindy Teeters

Book Designer

Anne Jones

Compositor

Bronkella Publishing, Inc.

CONTENTS

1 Expression Web 4 Add-in Basics 1

- Add-ins in Expression Web 1
- Expression Web 4 JavaScript Add-ins 2
 - The Makeup of Expression Web Add-ins 2
- XML Basics 3
- General Manifest Elements and Attributes 4
 - src (optional) 4
 - legacy (optional) 5
 - developer (optional) 5
 - navigationalallowed (optional) 5
 - <name> (required) 6
 - <description> (optional) 6
 - <author> (optional) 6
 - <version> (optional) 7
 - <homepage> (optional) 7
 - <minversion> (optional) 7
 - <guid> (optional) 7
 - <load> (optional) 8
- Commands and Dialog Boxes 8
 - id (required) 8
 - filetype (optional) 8
 - onclick (optional) 9
- Menus and Toolbars 10
 - <menuitem> (optional) 10
 - <toolbaritem> (optional) 12
- Panels 13
 - <panel> (optional) 14
- Menu and Command Bar Reference 16
 - Menus 17
 - Toolbars 36

2 Creating and Manipulating an Add-in User Interface 47

- Planning an Add-in 47
- Creating the Manifest with the Add-in Builder 48
 - Creating the Manifest 48
 - Editing the Manifest 53
- Creating the User Interfaces 53
 - Creating a Custom Page Size for

- Panels 54
- Creating the Panel's Interface 55
- Creating the Options Dialog Interface 58
- Adding Functionality with JavaScript 59
 - JavaScript for panel.htm 59
 - JavaScript for options.htm 65
 - The Set Page Title Dialog 66
- Accessing Managed Classes from JavaScript 68
 - Creating a Managed Class 69
 - Editing the Add-in Manifest to Load the Managed Class 72
 - Calling the Managed Class 73
- Summary 73

3 Packaging, Testing, and Debugging Add-ins 75

- Creating an Add-in Installation Package 75
- Testing and Debugging Add-ins 76
 - Testing Add-ins 76
 - Debugging Add-ins Using Expression Web 77
 - Debugging Add-ins Using Visual Studio 81
- Summary 85

4 Expression Web 4 JavaScript API Reference 87

- Conventions Used in this Reference 87
- xweb.application Object 88
 - xweb.application.version Property 88
 - xweb.application.chooseFile Method 89
 - xweb.application.endDialog Method 90
 - xweb.application.handleEvent Method 91
 - xweb.application.newDocument Method 92
 - xweb.application.openDocument Method 94
 - xweb.application.refreshFileListing Method 94

- xweb.application.
 - setActiveDocument Method 95
- xweb.application.
 - setPanelVisibility Method 96
- xweb.application.showModalDialog Method 97
- xweb.application.settings Object 98
 - xweb.application.settings.read Method 99
 - xweb.application.settings.write Method 99
- xweb.developer Object 100
 - xweb.developer.write Method 100
 - xweb.developer.writeLine Method 101
- xweb.document Object 101
 - xweb.document.anchors Property 103
 - xweb.document.applets Property 103
 - xweb.document.embeds Property 104
 - xweb.document.filename Property 104
 - xweb.document.forms Property 105
 - xweb.document.frames Property 105
 - xweb.document.images Property 106
 - xweb.document.isXHTML Property 106
 - xweb.document.links Property 106
 - xweb.document.location Property 107
 - xweb.document.name Property 108
 - xweb.document.pathFromSiteRoot Property 108
 - xweb.document.scripts Property 109
 - xweb.document.selection Property 109
 - xweb.document.
 - appendScriptReference Method 110
 - appendStyleReference Method 111
 - close Method 112
 - getElementById Method 112
 - getElementsByAttributeName Method 113
 - getElementsByTagName Method 114
 - xweb.document.
 - getScriptElementByCode Method 114
 - getScriptElementByFile Method 115
 - getStyleElementByCode Method 116
 - getStyleElementByFile Method 116
 - insertBeforeHtml Method 117
 - save Method 118
 - saveAs Method 118
 - synchronizeViews Method 119
- xweb.file Object 119
 - xweb.file.copy Method 120
 - xweb.file.createFile Method 121
 - xweb.file.createFolder Method 121
 - xweb.file.deleteFile Method 122
 - xweb.file.exists Method 123
 - xweb.file.getAttributes Method 123
 - xweb.file.getCreationDate Method 124
 - xweb.file.getModificationDate Method 124
 - xweb.file.getSize Method 125
 - xweb.file.listFiles Method 125
 - xweb.file.read Method 126
 - xweb.file.setAttributes Method 127
 - xweb.file.write Method 128
- htmlElement Object 128
 - htmlElement.childNodes Property 128
 - htmlElement.className Property 129
 - htmlElement.id Property 129
 - htmlElement.innerHTML Property 130
 - htmlElement.innerText Property 131
 - htmlElement.nextSibling Property 132
 - htmlElement.outerHtml Property 133
 - htmlElement.parentNode Property 134
 - htmlElement.previousSibling Property 134
 - htmlElement.tagName Property 135

- htmlElement.getAttribute
Method 135
- htmlElement.removeAttribute
Method 136
- htmlElement.setAttribute
Method 136
- xweb.document.selection Object 137
 - selection.end Property 137
 - selection.start Property 137
 - selection.text Property 138
 - selection.append Method 138
 - selection.insert Method 139
 - selection.set Method 139
 - selection.remove Method 140
 - selection.replace Method 140
 - selection.wrap Method 141

ABOUT THE AUTHOR

Jim Cheshire is the owner of Jimco Software and Books and is the author of several design books and books on the Amazon Kindle and Barnes and Noble Nook. In his real job, Jim works as a senior escalation engineer at Microsoft on the ASP.NET, IIS, and Expression Web teams. He has worked on the FrontPage, Visual Basic, ASP, IIS, and ASP.NET teams at Microsoft for more than 12 years.

You can reach Jim by visiting one of his websites: www.jimcobook.com or www.jimcosoftware.com. You can also email him at jcheshire@jimcobook.com.

DEDICATION

This book is dedicated to my lovely wife, Becky, and my two children. I love you all very much.

ACKNOWLEDGMENTS

I owe a debt of gratitude to my editors at Que Publishing. Loretta, it's been such a pleasure to work with you over the years. Todd, thanks for your consistent work to improve what comes out of my mind. To Kathleen and Ian, thank you for your hard work in ensuring that this book is technically accurate and easy to understand. To Seth, thanks for your commitment to a quality book. Although the cover of this book bears my name only, the book would have not been possible without the commitment of all of you, and I am sincerely thankful for all your hard work.

Thanks to Anna Ullrich, Paul Bartholomew, Justin Harrison, Steve Guttman, Erik Saltwell, Marc Kapke, Mike Calvo, and Erik Mikkelson at Microsoft, all of whom were of great help in answering questions that arose during the writing of this book. I also owe John Dixon at Microsoft a special thank you for always being available for questions about add-in development. Without John's assistance, the last part of this book simply wouldn't have been possible.

—Jim

WE WANT TO HEAR FROM YOU!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an associate publisher for Que Publishing, I welcome your comments. You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that I cannot help you with technical problems related to the topic of this book. We do have a User Services group, however, where I will forward specific technical questions related to the book.

When you write, please be sure to include this book's title and author as well as your name, email address, and phone number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@quepublishing.com

Mail: Greg Wiegand
Associate Publisher
Que Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

PACKAGING, TESTING, AND DEBUGGING ADD-INS

Creating an Add-in Installation Package

If you're going to write add-ins, you're going to have to deal with software bugs. Even the simplest add-in is likely to contain bugs, so testing and debugging your add-ins is a must. However, before you can test and debug an add-in, you need to install the add-in.

If you used the Add-in Builder to create your add-in manifest, installation of the add-in is done automatically for you. However, you'll still want to create an installation package so that others can install your add-in.

➔ *For more information on using the Add-in Builder, see "Creating the Manifest with the Add-in Builder," p. 48.*

Creating an add-in installation package is easy to do. Simply zip the folder that contains your add-in files and then change the file extension to `.xadd`. (You'll need to ensure that Windows is configured to show file extensions.) You can create a zip file by right-clicking on your add-in's folder and selecting Send To, Compressed (zipped) Folder from



note

A great place to share your add-in is the Microsoft Expression Gallery at <http://gallery.expression.microsoft.com/>.



tip

You cannot double-click on an `.xadd` file to install an add-in. To install an add-in, you need to use the Manage Add-ins dialog in Expression Web.

the menu. Once the folder has been zipped, change the file extension of the Zip file to `.xadd`. Your add-in is now ready for installation.

Testing and Debugging Add-ins

You've spent many hours developing a really cool add-in, and you're eager to upload it to the Expression Gallery so that other Expression Web users can use it. However, before you do, you should spend some time testing and debugging your add-in so that users don't experience problems. Even if your add-in is completely free, if users experience problems, they will complain and make sure to let other users know about their trouble. Testing and debugging are critical to the success of any add-in.

Testing Add-ins

The easiest way to test an add-in is to use it yourself during your normal usage of Expression Web. However, that's not the best way. It's likely that other people who may use your add-in (assuming you decide to share it with others) use Expression Web differently than you do. Subtle changes in the way that an add-in is used can expose serious bugs that you'll want to deal with before you share your add-in with others.

The specific scenarios you use when testing your add-in will differ depending on the complexity of your add-in. For example, testing the Page Info add-in was fairly simple because the act of opening and closing pages causes all the JavaScript functions used by the add-in to execute. However, when you're dealing with a more complex add-in, you may have JavaScript functions that execute only when certain conditions are met. In these situations, it's usually best to build a test page that's designed to call each of your functions. If an error is encountered, you can then debug the error more easily.

I follow some general rules when testing all of my add-ins:

- Test the add-in with disk-based sites, FTP sites, and HTTP sites.
- Test the add-in with pages at numerous folder levels within my site.
- Test the add-in with no sites open in Expression Web.
- If the add-in requires user input, conduct tests with no input and with bogus input.
- Find people willing to test the add-in for me and provide feedback.

I find (and fix) plenty of problems when going through the first four of these rules, but I'm always amazed at how many problems I miss that are uncovered by other people using my add-ins. Regardless of how much effort you put into predicting how users will use your add-in, you will still encounter a surprising number of unexpected scenarios once your add-in gets into the hands of users. When you provide your add-in to testers, make sure that you give them some guidelines for reporting bugs. For example, let them know that you need detailed, step-by-step instructions for reproducing



tip

Keep in mind that how any particular function works is often impacted by external factors. Create as many test scenarios as you can when testing your functions. This section gives you some pointers on how to do that.

any problems they encounter. Provide an example of a bug submission so that they'll know what you expect. If you don't, you often won't get enough detail to reproduce a particular problem.

As an example, the following steps are not sufficient to reproduce a problem:

1. Opened my site.
2. Started editing my page.
3. Got the error message.

However, the following steps would likely allow you to reproduce a problem and get it corrected:

1. Opened a site at <http://www.mysite.com>.
2. Opened `aboutme.htm` from the root folder.
3. Switched to Design View.
4. Added the text "Click Here for Info".
5. Selected the text and clicked the Insert Hyperlink button in Expression Web.
6. Added a hyperlink to `moreinfo.htm` in the root folder of the site.
7. Clicked outside that hyperlink and got the error.

If you provide your testers with examples of what you expect, you'll get better information from them.

It's also important that you understand a tester's expectation compared with what was experienced when encountering a problem. Not all problems that testers encounter are actually bugs. When a tester encounters a problem, your add-in may be operating exactly the way you designed it, but your design may not be what the tester expects. For that reason, I like to have testers include steps that were performed, what was expected, and what actually happened.

Once you are able to reproduce a problem, you can then debug it and fix it. Let's review some of the tools available for debugging add-ins.

**tip**

No amount of testing can identify all software bugs. Even simple add-ins are likely to be released with some undiscovered bugs.

Debugging Add-ins Using Expression Web

Debugging is the process of identifying and fixing problems in software. You can debug your add-ins from within Expression Web using a couple of different tools: the Extensibility Tester and debug consoles.

The Extensibility Tester

The Extensibility Tester is an add-in developed by Microsoft for testing the JavaScript API. Using the Extensibility Tester, you can test JavaScript code easily outside your add-in project. For example, suppose your add-in needs to get the path of the file currently open in Expression Web, but you aren't sure whether you want to use `xweb.document.filename` or `xweb.document.location`.

Using the Extensibility Tester, you can evaluate each of these to determine which property is appropriate for your particular situation.

You can download the Extensibility Tester from <http://gallery.expression.microsoft.com/en-us/ExtensibilityTester>. After you download and install it, select Panels, Extensibility Tester to open it. The Extensibility Tester is divided into four windows:

- **Input**—Expressions to evaluate are added here. To evaluate an expression, click the Evaluate button.
- **Output**—If no error occurs when evaluating the expression, the result is displayed here.
- **Error**—If an error occurs when evaluating the expression, the error is displayed here.
- **Events**—Displays events in real-time.

➡ *For more information on events, see Chapter 4, “Expression Web 4 JavaScript API Reference.”*

In Figure 3.1 the expression `xweb.document.filename` is being evaluated. Because a file is not currently open in Expression Web, “Object required” is displayed in the Error window. An “Object required” error means that the JavaScript expression refers to an object that doesn’t exist. In this case, the `xweb.document` object doesn’t exist because a file isn’t open in Expression Web.

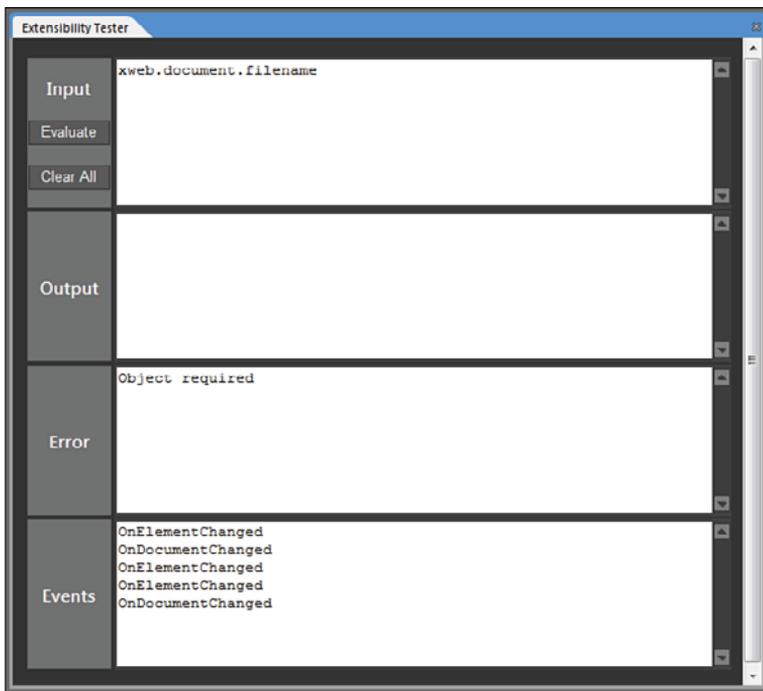


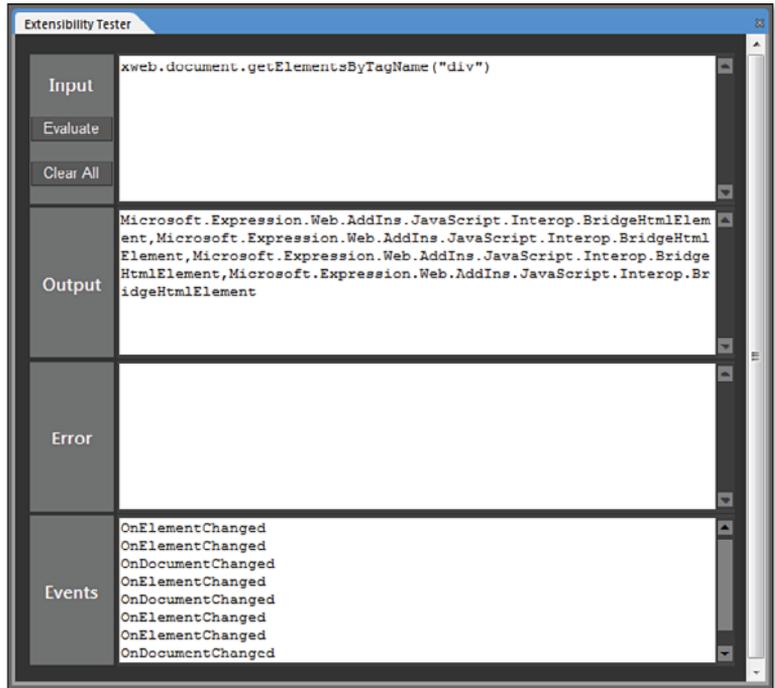
Figure 3.1
The Extensibility Tester add-in makes it possible to test JavaScript expressions easily.

In some cases, the output you get from the Extensibility Tester may not be sufficient to resolve a problem. For example, in Figure 3.2, the following code is being evaluated:

```
xweb.document.getElementsByTagName("div")
```

Figure 3.2

The output from the Extensibility Tester isn't always as helpful as you might want it to be.



The Output window displays the value `Microsoft.Expression.Web.AddIns.JavaScript.Interop.BridgeHtmlElement` four times. (The `BridgeHtmlElement` object is an internal object that Expression Web uses to identify an HTML element.) If I want to find out additional information about a particular `div`, I need to change my input to refer to a particular element. The `getElementsByTagName` method returns a zero-based array of elements, so I can use the following code to get a reference to the first `div` on the page:

```
xweb.document.getElementsByTagName("div")[0]
```

Figure 3.3 shows the output obtained when evaluating the `innerHTML` property of the first `div` on the page.



tip

The Extensibility Tester is a great way to experiment with the JavaScript API and learn more about how to develop add-ins.

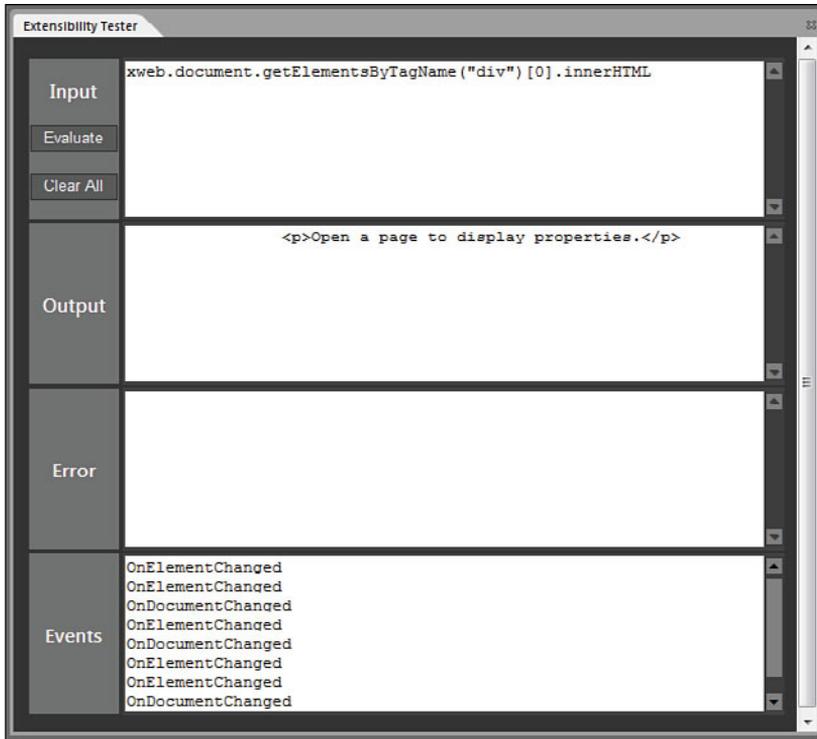


Figure 3.3
By using a more specific input string, I am able to get useful information about a particular div.

Debug Consoles

If your add-in is in developer mode, you can use the debug console to aid in debugging your add-in. To access the debug console for your add-in, select it from the Panels menu in Expression Web. The menu item for your add-in's debug console will be named using the name of your add-in's folder. Therefore, if your add-in is installed into the `C:\Users\jim\AppData\Roaming\Microsoft\Expression\Web 4\Addins\MyAddin` folder, the debug console can be opened by selecting Panels, MyAddin (console).

The debug console displays some error information automatically. For example, if your add-in's manifest contains errors, the debug console may display helpful information about the error. However, you can also use the `xweb.developer` object to write directly to the debug console.



tip

As discussed in Chapter 1, "Expression Web 4 Add-in Basics," to specify that an add-in run in developer mode, set the `developer` attribute in the manifest's `addin` element to `yes`.



tip

If an error is encountered in your manifest (and when some other errors occur), Expression Web creates a file called `errors.txt` in the root of your site that contains the error message.

The `xweb.developer` object has two methods that can be used to write content to the debug console:

- `xweb.developer.write(string)`—Displays the value passed as a string in the debug console. The string value can either be an explicit string or an expression that evaluates to a string.
- `xweb.developer.writeLine(string)`—Equivalent to the `write` method except that `writeLine` includes a carriage return at the end of the string so that each string appears on a new line.

Consider the following code snippet:

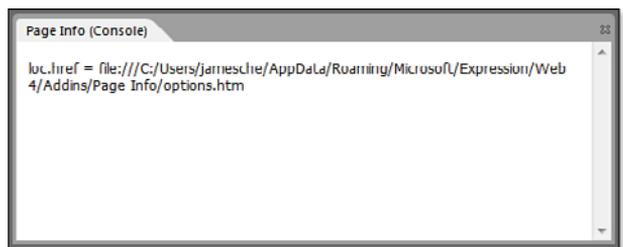
```
var loc = xweb.document.location;
var linkDiv = xweb.document.getElementById("link");
linkDiv.innerHTML = "<a href=\"" + loc.href + "\">Click Here</a>";
```

Suppose that when this code runs, the hyperlink created within the `div` points to a URL that's not what I expect. By adding a call to `xweb.developer.write`, you can easily determine the value of `loc.href` before you use it in your code. Here's the same code with an additional line that writes the value of `loc.href` to the debug console.

```
var loc = xweb.document.location;
xweb.developer.writeLine("loc.href = " + loc.href);
var linkDiv = xweb.document.getElementById("link");
linkDiv.innerHTML = "<a href=\"" + loc.href + "\">Click Here</a>";
```

Figure 3.4 shows the contents of the debug console when this code is run.

Figure 3.4
The debug console displaying the value of `loc.href`.



Debugging Add-ins Using Visual Studio

The debug console and Extensibility Tester are nice tools for troubleshooting add-ins, but if you really want a full-featured debugger, Visual Studio is your best choice.

To debug your add-in in Visual Studio, follow these steps:

1. Open Internet Explorer and select Tools, Internet Options.
2. Click the Advanced tab.
3. Uncheck Disable Script Debugging (Internet Explorer).



tip

To clear the contents of the debug console, close and reopen Expression Web.

4. Uncheck **Disable Script Debugging (Other)**. Your Internet Options dialog should look like the one shown in Figure 3.5.
5. Click **OK**.
6. Ensure that all panels and dialogs for your add-in are closed and close Expression Web.
7. Launch Visual Studio and open the file that contains the script you want to debug.
8. Right-click on the line where you want a breakpoint and select **Breakpoint, Insert Breakpoint**.
9. Launch Expression Web, but do not open your add-in's panel or dialog box.
10. Switch back to Visual Studio and select **Debug, Attach to Process**.
11. Locate **ExpressionWeb.exe** in the Available Processes window and make sure that **Script** appears in the Type column as shown in Figure 3.6. If it does, proceed to step 15.
12. If **Script** does not appear in the Type column, click the **Select** button.
13. Select the **Debug These Code Types** radio button and check the **Script** check box.
14. Click **OK**.
15. Select **ExpressionWeb.exe** from the Available Processes list and click the **Attach** button.
16. Switch back to Expression Web and access your add-in.

At this point, Visual Studio should break into the process when your breakpoint is hit and you'll have access to all of the debugging tools available in Visual Studio.

Using this method, you can set a breakpoint and debug from that point. However, you can also use Visual Studio to debug unexpected errors that may occur while testing your add-in.

Once you have script debugging enabled in Internet Explorer, when a script error is encountered, you are asked whether you want to debug the file containing the script as shown in Figure 3.7. If you click **Yes**, the Visual Studio Just-In-Time Debugger dialog displays and you can choose a debugger as shown in Figure 3.8. Choose **New Instance of Visual Studio 2010** and click **Yes** to debug your add-in. (Your version of Visual Studio may differ.)

 **note**

Debugging add-ins in Expression Web requires a full version of Visual Studio. You cannot debug Expression Web add-ins with any of the Express editions.

 **note**

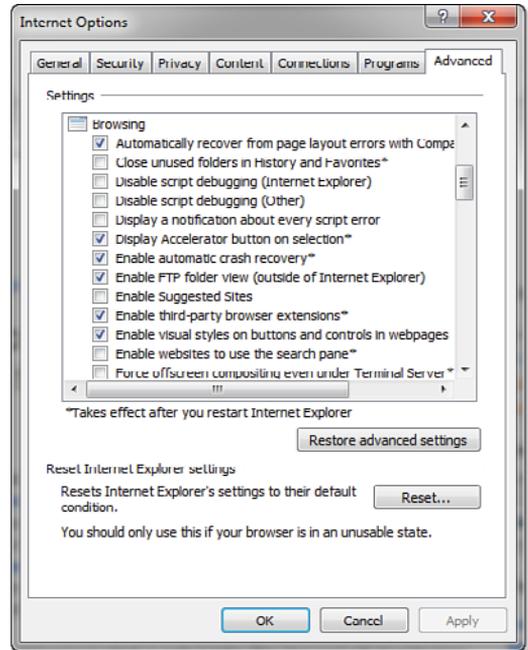
Expression Web add-ins actually run inside an Internet Explorer control. That's why you need to ensure that script debugging is enabled in Internet Explorer before you can debug your add-ins in Visual Studio.

 **note**

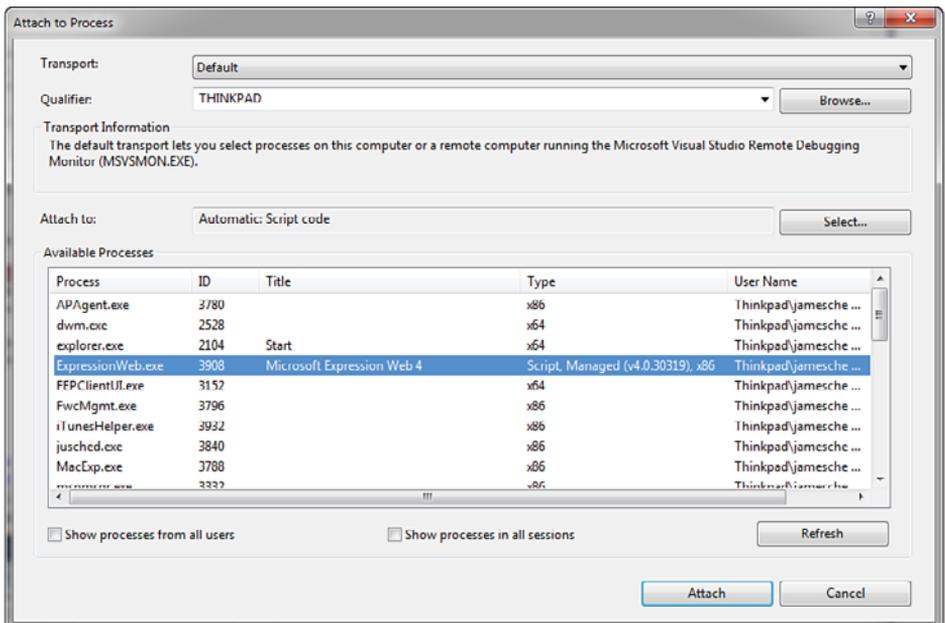
Thanks to John Dixon on the Expression Web product team for documenting details on how to do this on his blog. You can read John's blog post at <http://blogs.msdn.com/b/jdixon/archive/2010/09/02/debugging-javascript-add-ins-within-expression-web-4.aspx>.

Figure 3.5

You need to uncheck both Disable Script Debugging check boxes in Internet Explorer before you can debug your add-in.

**Figure 3.6**

Make sure that Visual Studio is configured to debug scripts.



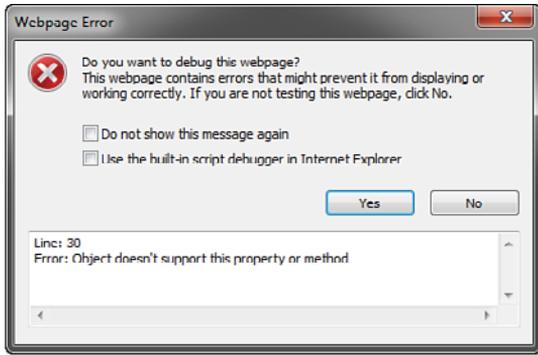


Figure 3.7
Internet Explorer displays a script error dialog when script debugging is enabled.

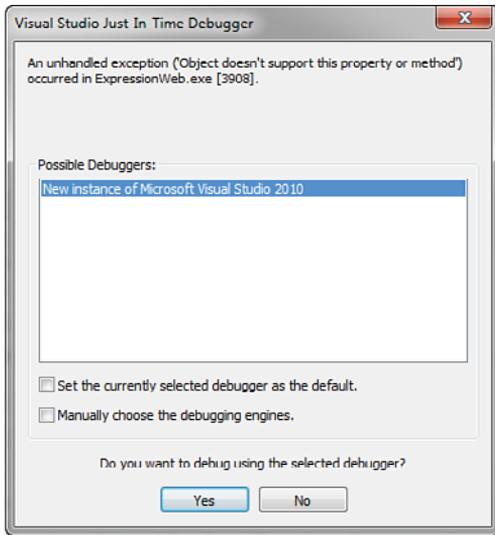


Figure 3.8
The Visual Studio JIT debugger can debug scripts that generate errors.

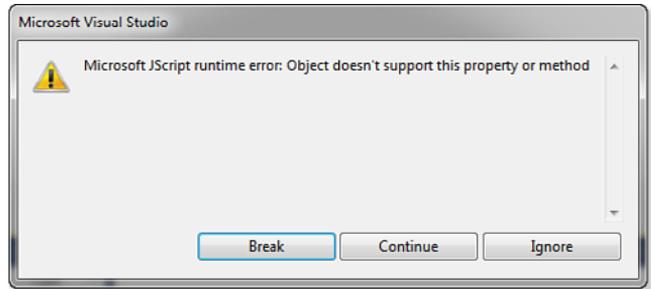
When Visual Studio launches, it automatically attaches the debugger to Expression Web, and the script error is displayed as shown in Figure 3.9. Click Break and Visual Studio breaks at the point where the error occurred.

note

You can read details on all of Visual Studio's debugging tools at <http://msdn.microsoft.com/en-us/library/sc65sadd.aspx>.

Figure 3.9

The script error is displayed within Visual Studio. Clicking Break allows you to debug it.



Perpetual Script Errors

Once you enable script debugging in Internet Explorer, an error message is displayed each time a script error is encountered. If you're developing a panel for your add-in and you have an error in your script that runs when the panel is activated or deactivated, it's possible that a script error will be displayed each time you open a page or close a page. If an error occurs when you are closing a page and you click No to the prompt asking whether you want to debug the error, Expression Web will not close the page.

To correct such a situation, you need to first correct the script error that is causing your problem. Once you do that and you save the file containing the script, right-click on your panel and select Refresh. Your panel then runs the corrected script and you should be able to open and close pages without an error message.

You should also know that when script debugging is enabled, you may see errors occur on many sites as you browse the Internet. For that reason, it's a good idea to disable script debugging again after you've finished debugging your Expression Web add-in.

Summary

In this chapter, you familiarized yourself with the tools available for troubleshooting and debugging your add-ins. You now have all the skills necessary for developing and debugging add-ins in Expression Web. However, we've only touched on the capabilities of the JavaScript API.

In the next chapter, you'll find a complete reference on the Expression Web 4 JavaScript API. The API reference is designed to be a valuable tool for looking up information on the API while you're developing add-ins. However, it's also helpful to read the reference so that you can familiarize yourself with all the capabilities of the JavaScript API.