Creating Add-ins for Microsoft® Expression Web 4

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

Jim Cheshire

que

Creating Microsoft® Expression Web 4 Add-ins:

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

Jim Cheshire

que®

CREATING MICROSOFT EXPRESSION WEB 4 ADD-INS:

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

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

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.jimcobooks.com or www.jimcosoftware.com. You can also email him at jcheshire@jimcobooks.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

1

EXPRESSION WEB 4 ADD-IN BASICS

Add-ins in Expression Web

No software application can meet all needs for all people. No matter how many features Microsoft packs into Expression Web, there will always be designers who find that something's missing. Fortunately, Expression Web is extensible so that users of the application can add functionality.

Previous versions of Expression Web were extensible only using a technology called Component Object Model, or COM. Using Visual Studio, Microsoft's development studio, a developer can create a COM add-in that adds functionality to Expression Web. (If you've used any of my add-ins that I distribute from Jimco Software, you've used a COM add-in.) There is a significant barrier to entry for COM add-in development, and most web designers don't possess the skill set necessary to develop them.

note

You can view a tutorial on how to create a COM add-in at http://jimcobooks.com/articles/061130/Default.aspx.

Expression Web 4 still allows for COM add-ins, but it adds an exciting new add-in architecture that allows web designers to use existing skills in HTML, XML, and JavaScript to develop add-ins. HTML is used to create user-interfaces, and JavaScript is used to add functionality to Expression Web 4 add-ins.

For more information on adding functionality to an add-in using JavaScript, see "Adding Functionality with JavaScript," in the online Chapter 2 on page 59.

In this book, I cover everything you need to know to build add-ins using the new JavaScript and HTML extensibility model. I also cover the basics of how you can call into a COM add-in from a JavaScript add-in.

Expression Web 4 JavaScript Add-ins

Using the JavaScript add-in model in Expression Web 4, you can create add-ins with three different main components: panels, dialog boxes, and commands.

Panels are available from the Panels menu. The interface for the panel is developed using HTML code, and functionality is provided using JavaScript. Panels open as floating panels, but users of the add-in can drag and drop the panel to dock it if desired.

Dialog boxes are displayed as a modal dialog of a specific size. The dialog box interface is developed using HTML, and functionality is provided using JavaScript. JavaScript can be used to control the functionality of the dialog box itself as well as add functionality for the add-in.

Commands add a menu command or a toolbar button that runs JavaScript code when the menu item or toolbar button is clicked. Commands are also used to launch dialog boxes.

You may already be familiar with using JavaScript to add functionality to a web page running inside a browser. When you use JavaScript to develop Expression Web add-ins, you are not interacting with a web page in the browser. Instead, you are using JavaScript to interact with files (web pages and other files) inside the Expression Web interface. You can also use JavaScript to interact with Expression Web itself.

I'll explain how to do all of that in the next few chapters, but before we get into the details of implementing an add-in, it's important to understand some of the basics that make up an Expression Web add-in.

note

Microsoft intends for JavaScript add-ins to be used with disk-based sites only. If you use a JavaScript add-in with a server-based site accessed using HTTP or FTP, you may encounter unexpected results.

I'll provide guidance on what you can expect with server-based sites in Chapter 4, "Expression Web 4 JavaScript API Reference."

tip

Microsoft's documentation says that there are three types of add-ins: panel, dialog box, and command add-ins. However, since it's possible for a single add-in to have any combination of those three, I believe it's more accurate to call these components of an add-in and not refer to them as types of add-ins.

The Makeup of Expression Web Add-ins

Expression Web add-ins are made up of a collection of files. HTML files are used to present a user interface for panels and dialog-boxes. JavaScript code is often included directly in the HTML file, but can also be included as a separate script file with a `.js` file extension. Add-ins can also include any number of collateral files such as image files, CSS files, and so on.

tip

Since add-ins can be created right from within Expression Web, the easiest way to create an add-in is to create a new disk-based site and then create your add-in files within that site.

Add-ins also contain a special XML file called a *manifest*. The add-in manifest (named `addin.xml`) describes the add-in to Expression Web. It's used to tell Expression Web the components included with the add-in, the files included with the add-in, and other information necessary for the proper functioning of the add-in. The manifest also includes information such as the add-in's name, the name of the developer, the version number, and so on.

Listing 1.1 shows a simple add-in manifest for an add-in that displays a panel.

Listing 1.1 A Simple Manifest File

```
<addin>
  <name>Simple Panel</name>
  <version>1.0</version>
  <description>A simple panel add-in for Expression Web.</description>
  <author>Jim Cheshire</author>
  <homepage>http://www.jimcosoftware.com</homepage>
  <panel src="default.html" id="panel1"
        title="Simple Panel" filetype="HTML-DOM"
        activate="enableControls" deactivate="disableControls" />
</addin>
```

Add-in files can be saved into any folder, but the manifest must be inside the root folder for the add-in. Once you're ready to use an add-in, you simply Zip the folder where the files are located and then rename the Zip file so that it has a `.xadd` file extension. You can then install the add-in using the Add-ins dialog available by selecting Tools, Add-ins in Expression Web.

I'll cover all of this in explicit detail in the next few chapters, but what I want you to take away from this is that no specialized tools or complicated skill sets are required to create and deploy Expression Web 4 add-ins. You can create them right inside Expression Web, and you can create an installable package by simply renaming a Zip file. It really couldn't be simpler.

XML Basics

Before I go into the details of the add-in manifest, you'll need to know some basics about XML. XML syntax is actually used for the XHTML code with which you are probably at least somewhat familiar.

At the top level of an XML file is the *root element*. There is exactly one root element; no fewer, no more. Beneath the root element are *child elements*. There can be any number of child elements in an XML file, and each child element can also have its own child elements. The element directly above a child element is known as the *parent element*.

In the manifest in Listing 1.1, the `<addin>` element is the root element. All other elements are child elements.

All elements may contain one or more *attributes*. An attribute is defined within the XML element itself and consists of the attribute name followed by the attribute value in quotes. The following line of XML shows an attribute called `developer` with a value of `yes`.

```
<addin developer="yes">
```

caution

XML is case-sensitive, so pay attention to case when writing XML.

Attribute names are case-sensitive, and attribute values must be enclosed in quotes.

All child elements in an XML file may contain attributes or text content. Text content is textual information surrounded by a particular element. In the XML code that follows, `Simple Panel` is the text content for the `<name>` element.

```
<name>Simple Panel</name>
```

The < and & characters are not legal characters for XML content. If you want to use these characters for attribute or text content, you need to use what's called an *entity reference*. The entity reference for < is `<` and the entity reference for & is `&`. You might notice that both entity references contain the & character. The only time the & character is valid in an XML file is when it is used with an entity reference.

The following XML snippet defines a name of Jack & Jill Add-ins.

```
<name>Jack &amp; Jill Add-ins</name>
```

Armed with this basic information about XML, you are now ready to learn the details about the add-in manifest. This chapter covers all the elements and attributes that you can use in an add-in manifest. Later chapters go into much more detail about how to use specific elements and attributes.

tip

Some XML files use camel case for attribute names. However, in an add-in manifest, all attribute names are lowercase.

note

An XML element can contain both attributes and text content.

note

If you'd like more information about XML, read *Special Edition Using XML* from Que Publishing.

General Manifest Elements and Attributes

As mentioned previously, the `<addin>` element is the root element of the add-in manifest. Four optional attributes are available for the `<addin>` element.

tip

Each element and attribute documented in the following sections includes a graphical representation of where that particular element or attribute fits within the manifest's hierarchy.

`src` (optional)

```
<addin src='script.js'>
```

The `src` attribute is used to specify JavaScript source files that contain functions that the manifest references. This attribute is typically used for commands that don't have a user interface. Scripts that are used with panels and dialog boxes are referenced within the HTML files used with those components.

Source file paths are relative to the add-in's top-level folder, and you can add multiple source files by separating them with a comma.

`legacy` (optional)

```
<addin legacy='yes'>
```

The `legacy` attribute is set to `yes` when an add-in needs to access the legacy object model for Expression Web. The legacy object model provides access to the object model used by add-in developers in previous versions of Expression Web, and it has some additional functionality compared to the JavaScript API.

Use of the legacy object model is outside the scope of this book.

`developer` (optional)

```
<addin developer='yes'>
```

When set to `yes`, the `developer` attribute aids in testing and troubleshooting your add-in during development. By default, the context menu that appears when you right-click on a page is disabled for add-in panels and dialog boxes. By including the `developer` attribute with a value of `yes`, you can remove this restriction, allowing you to easily refresh the user interface of your panel or dialog box, view the source of your interface, and so on.

There are other benefits to setting the `developer` attribute to `yes` when you are developing your add-in. I'll cover how you can use developer mode to debug your add-in in Chapter 4.

note

You should not set the `developer` attribute to `yes` unless you are in the process of developing your add-in because doing so enables debugging tools that are not meant for end-users.

`navigationallowed` (optional)

```
<addin navigationallowed='yes'>
```

By default, Expression Web does not allow you to navigate away from the current page from within a panel or a dialog box. For example, if your dialog box contains a hyperlink, clicking the hyperlink does nothing by default.

In some situations, it may be necessary to allow a user of your add-in to navigate to other pages within your add-in's user interface. For example, if your add-in's interface is in the form of a wizard, you may want to implement each step of the wizard using a separate page. In such a scenario, setting the `navigationallowed` attribute to `yes` allows a user to navigate through the pages of your wizard.

<name> (required)

```
<addin>
   <name='My Add-in'/>
```

The <name> element specifies the name of your add-in. Once your add-in is installed into Expression Web, the value specified by the <name> attribute is displayed in the Manage Add-ins dialog as shown in Figure 1.1.

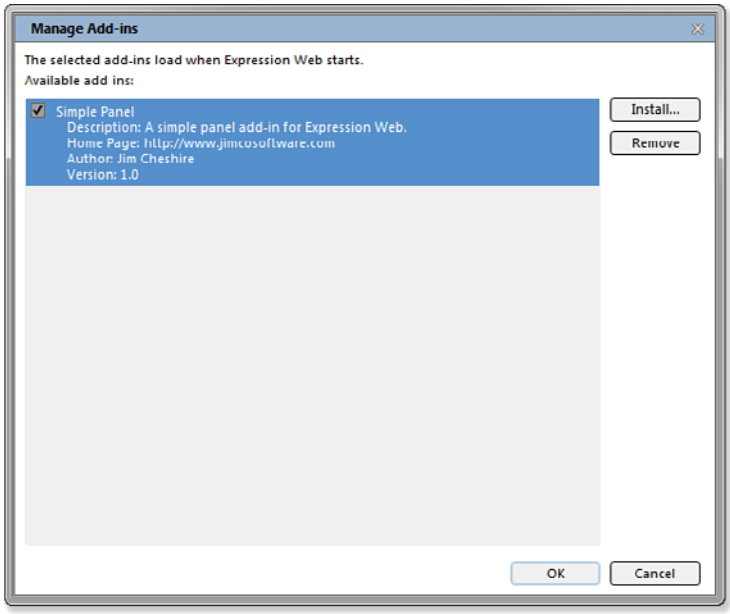

Figure 1.1
The name shown in the Manage Add-ins dialog is controlled by the <name> element. This add-in uses the manifest shown in Listing 1.1.

<description> (optional)

```
<addin>
   <description='Simple Add-in'/>
```

The <description> element is used to provide a description of your add-in that appears in the Manage Add-ins dialog as shown previously in Figure 1.1

The <description> element is optional, but it's a good idea to include it so that users of your add-in can more easily identify it inside the Manage Add-ins dialog.

<author> (optional)

```
<addin>
   <author='Jim Cheshire'/>
```