

"Office development using managed code has hit new strides with Visual Studio 2008, and personally, I can't wait to take advantage of the answers I find in this book to build great applications."

—From the Foreword by **Ken Getz**,
senior consultant, MCW Technologies



Visual Studio Tools for Office 2007

VSTO for Excel, Word, and Outlook



Eric Carter
Eric Lippert

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The .NET logo is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries and is used under license from Microsoft.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Carter, Eric.

Visual Studio tools for Office 2007 : VSTO for Excel, Word, and Outlook / Eric Carter, Eric Lippert. — 2nd ed.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-321-53321-0 (pbk. : alk. paper)

1. Microsoft Visual BASIC. 2. BASIC (Computer program language) 3. Microsoft Visual studio. 4. Microsoft Office. I. Lippert, Eric. II. Title.

QA76.73.B3C3452 2009

005.13'3—dc22

2009000638

Copyright © 2009 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671-3447

ISBN-13: 978-0-321-53321-0

ISBN-10: 0-321-53321-6

Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.
First printing, February 2009



Foreword

FACE THE CHALLENGE of composing a foreword to this particular book with some amount of trepidation. Let's face it: The names on the cover of this book inspire some amount of awe. It is humbling to know that one's words will introduce what one believes is to be the seminal work on a given topic, and believe me, I'm relatively sure that this one will meet those lofty goals. When approached with the invitation to grace the front matter of the book, my first response was to wonder what I could possibly add; couldn't they find some luminary at Microsoft to preface the book? It seems, however, that an outside voice adds some credence to the proceedings, so, dear reader, I speak meekly in the presence of greatness.

First, a little about me (it's the last chance I'm going to get in this short piece): I've been lurking about, programming Office in its various guises, for upward of 10 years. I've written a lot about the wonders, and gotchas, of Office development, and survived the glory years surrounding Office 2000, when it looked like Office might finally make a successful integrated development platform. Around 2001, it became clear that no matter how hard I and like-minded folks wanted Office to become a respected development standard, it just wasn't going to make it with VBA as the programming language.

With the release of Visual Studio Tools for Office 2003, it finally looked like we had made some progress. No longer relegated to the 1990s, Office developers could embrace .NET and all its goodness, taking advantage of managed code, code-access security, xcopy deployment, and all the rest

that .NET supplied. I loved this product, but it never really reached critical mass with the developer community. Most likely, the fact that you could only use COM-based controls on documents, and the fact that the product supplied no design-time experience at all, made it a slow starter.

Around that time, I remember very clearly sitting down at some Microsoft event and meeting Eric Carter. I didn't really know who he was at the time (and he certainly didn't know anything about me), but he seemed nice enough, and we chatted for several hours about Office development in general and about VSTO in specific. Only later did I learn that he was high up in the development side of the product. (I spent hours worrying that I had said something really stupid while we were chatting. Hope not.) We began a long correspondence, in which I've more often than not made it clear that I've got a lot to learn about how .NET and Office interact. I've spent many hours learning from Eric's blog, and Eric Lippert's blog is just as meaty. If you are spending time doing Office development, make sure you drop by both:

<http://blogs.msdn.com/ericlippert/>

http://blogs.msdn.com/eric_carter/

I spent some measurable hours perusing the draft copy of this book and in each chapter attempted to find some trick, some little nugget, that I had figured out on my own that didn't appear in the book. I figured that if I was going to review the book, I should add something. The result: I was simply unable to find anything missing. Oh, I'm sure you'll find some little tidbit that you've figured out that won't appear here, but in my quick pass, I wasn't able to. I thought for sure I would catch them on something. Alas, I failed. And, I suppose, that's a good thing, right? Every time I thought I had them in a missing trick, there it was, right there in print. What that means is that you'll have the best possible reference book at your fingertips. Of course, you need to get your expectations set correctly; it's simply not possible, even in a 60-page chapter, to describe the entirety of the Excel or Word object model. But E&E have done an excellent job of pointing out the bits that make the biggest impact on .NET development.



If you're reading this foreword before purchasing the book, just do it. Buy the thing. If you've already bought it, why are you reading this? Get to the heart of the matter—skip ahead, and get going. You can always read this stuff later. There's a considerable hill ahead of you, and it's worth the climb. Office development using managed code has hit new strides with the release of Visual Studio 2008, and personally, I can't wait to take advantage of the answers I find in this book to build great applications.

—Ken Getz, senior consultant for MCW Technologies



Preface

IN 2002 THE first release of Visual Studio .NET and the .NET Framework was nearing completion. A few of us at Microsoft realized that Office programming was going to miss the .NET wave unless we did something about it.

What had come before was Visual Basic for Applications (VBA), a simple development environment integrated into all the Office applications. Each Office application had a rich object model that was accessed via a technology known as COM. Millions of developers identified themselves as “Office developers” and used VBA and the Office COM object models to do everything from automating repetitive tasks to creating complete business solutions that leveraged the rich features and user interface of Office. These developers realized that their users were spending their days in Office. By building solutions that ran inside Office, they not only made their users happy, but also were able to create solutions that did more and cost less by reusing functionality already available in the Office applications.

Unfortunately, because of some limitations of VBA, Office programming was starting to get a bad rap. Solutions developed in VBA by small workgroups or individuals would gain momentum, and a professional developer would have to take them over and start supporting them. To a professional developer, the VBA environment felt simple and limited, and of course, it enforced a single language: Visual Basic. VBA embedded code in every customized document, which made it hard to fix bugs and update solutions because a bug would get replicated in documents across the

enterprise. Security weaknesses in the VBA model led to a rash of worms and macro viruses that made enterprises turn VBA off.

Visual Studio .NET and the .NET Framework provided a way to address all these problems. A huge opportunity existed not only to combine the richness of the new .NET Framework and developer tools with the powerful platform that Office has always provided for developers, but also to solve the problems that were plaguing VBA. The result of this realization was Visual Studio Tools for Office (VSTO).

The first version of VSTO was simple, but it accomplished the key goal of letting professional developers use the full power of Visual Studio .NET and the .NET Framework to put code behind Excel 2003 and Word 2003 documents and templates. It let professional developers develop Office solutions in Visual Basic and C#. It solved the problem of embedded code by linking a document to a .NET assembly instead of embedding it in the document. It also introduced a new security model that used .NET code-access security to prevent worms and macro viruses.

The second version of VSTO, known as VSTO 2005, was even more ambitious. It brought with it functionality never available to the Office developer before, such as data binding and data/view separation, design-time views of Excel and Word documents inside Visual Studio, rich support for Windows Forms controls in the document, the ability to create custom Office task panes, server-side programming support against Office—and that's just scratching the surface. Although the primary target of VSTO is the professional developer, that does not mean that building an Office solution with VSTO is rocket science. VSTO makes it possible to create very rich applications with just a few lines of code.

The third version of VSTO, which this book focuses on, shipped as a core feature of Visual Studio 2008. It is sometimes said that it takes Microsoft three versions to get something right, and we truly feel that this version of VSTO has the most amazing support for Office programming that Microsoft has ever built. In VSTO, you can now build add-ins for all the major Office applications; you can build application-level custom task panes; you can customize the new Office Ribbon; you can modify Outlook's UI using Forms Regions, and you can easily deploy everything you



build using ClickOnce. The Office 2007 applications themselves are more extensible and provide many new programmability features.

If you've been reluctant to use VSTO because of the issues in previous versions—such as the difficulty of deployment, the nonsupport of VSTO in the Visual Studio Professional SKU, and the limited support for add-ins—we're happy to tell you that these issues have been fixed in the third version of VSTO.

This book tries to put in one place all the information you need to succeed using VSTO to program against Word 2007, Excel 2007, and Outlook 2007. It introduces the Office 2007 object models and covers the most commonly used objects in those object models. In addition, this book helps you avoid some pitfalls that result from the COM origins of the Office object models. This book also provides necessary background for developers using VSTO to customize Visio 2007, Publisher 2007, PowerPoint 2007, and InfoPath 2007. Although it doesn't specifically focus on these applications, it teaches how to use the VSTO add-in model, how to create custom task panes and ribbons, and how to code against Office object models using C#.

This book also provides an insider view of all the rich features of VSTO. We participated in the design and implementation of many of these features; therefore, we can speak from the unique perspective of living and breathing VSTO for the past six years. Programming Office using VSTO is powerful and fun. We hope you enjoy using VSTO as much as we enjoyed writing about it and creating it.

—Eric Carter
—Eric Lippert
January 2009

■ ■ 16 ■ ■

Working with Outlook Form Regions

Introduction to Form Regions

In Outlook 2007, developers have the ability to extend the Outlook UI by creating a special kind of Outlook extension called an Outlook form region. *Form regions* are used primarily to customize Inspector windows, which we introduced in Chapter 10, “Working with Outlook Events.” *Inspector windows* are the Outlook windows that appear when you double-click an Outlook item—a mail item in your inbox or a task in a task list, for example. With form regions you can do things like add pages to the Inspector window, replace all the existing pages in an Inspector window with your own page, or dock some custom UI onto an existing page. You can also use a certain type of Outlook form region (an Adjoining form region) to customize the reading pane in Outlook Explorer windows.

Creating a New Form Region

To begin our exploration of Outlook form regions, let’s create a simple one by using Visual Studio 2008. Start by creating a new Outlook add-in project by choosing File > New > Project. In the New Project dialog box that appears, create a new Outlook 2007 add-in, as shown in Figure 16-1.

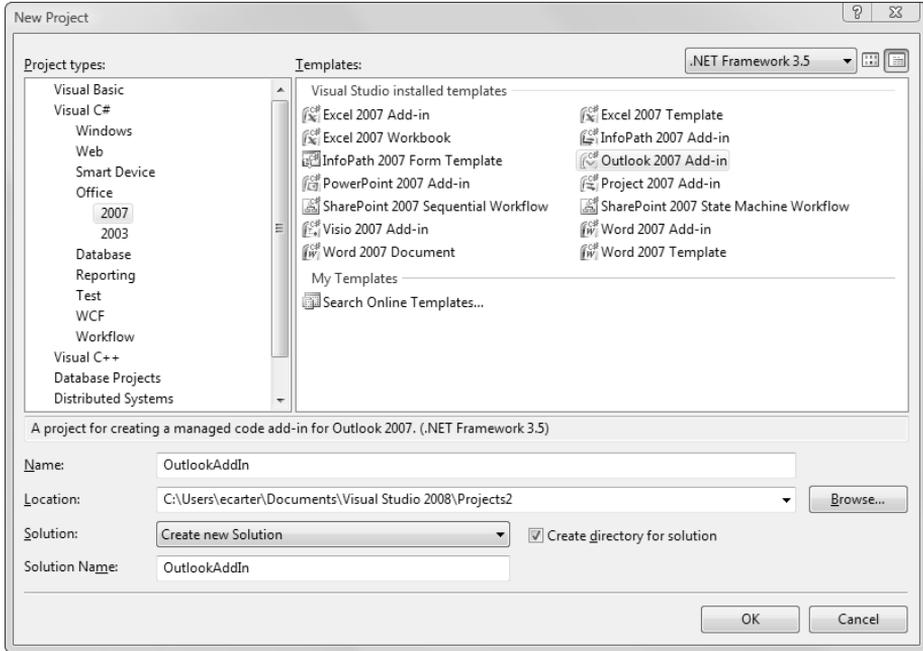


Figure 16-1: Creating a new Outlook 2007 add-in.

Now, in your new add-in project, choose **Project > Add New Item**. Click the **Office** category to filter to show just the Office-specific items. In the list of Office items, click **Outlook Form Region**, as shown in Figure 16-2. Name the form region—just use the default name **FormRegion1**. Then click the **Add** button.

A wizard appears, as shown in Figure 16-3. The first step in the wizard is to decide whether you want to create an Outlook form region or import a form region that was previously designed in Outlook with Outlook’s built-in form designer. For this introduction, click **Design a New Form Region**. This option lets you use Windows Forms and keeps our editing experience within Visual Studio. Later in the chapter we show you how to use the Outlook built-in form designer, as well as discuss when you might want to use Outlook’s form designer instead of Windows Forms.

After you decide whether to design a new form region with Windows Forms or to import an existing Outlook form region designed in Outlook, click the **Next** button to move to the second page of the wizard, shown in Figure 16-4, which allows you to pick the type of form region you want to create.

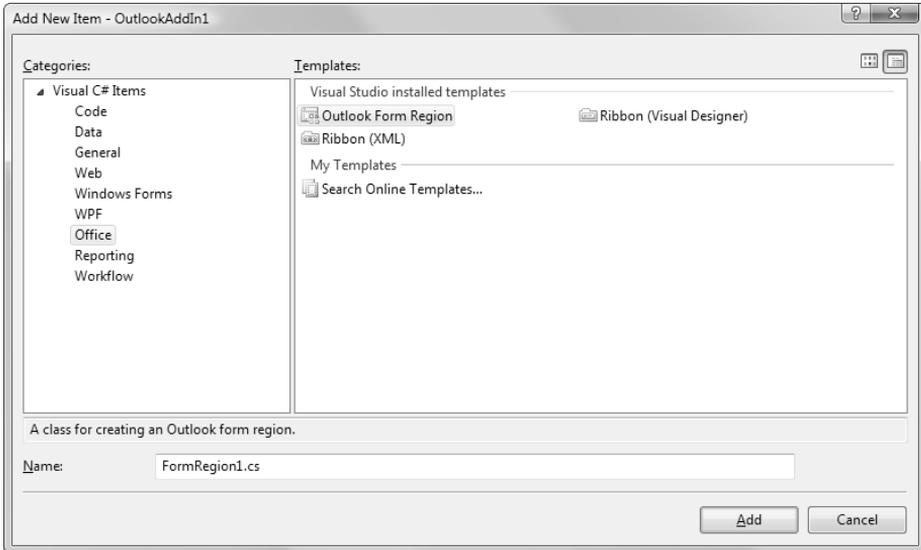


Figure 16-2: Adding an Outlook form region to an Outlook 2007 add-in project.

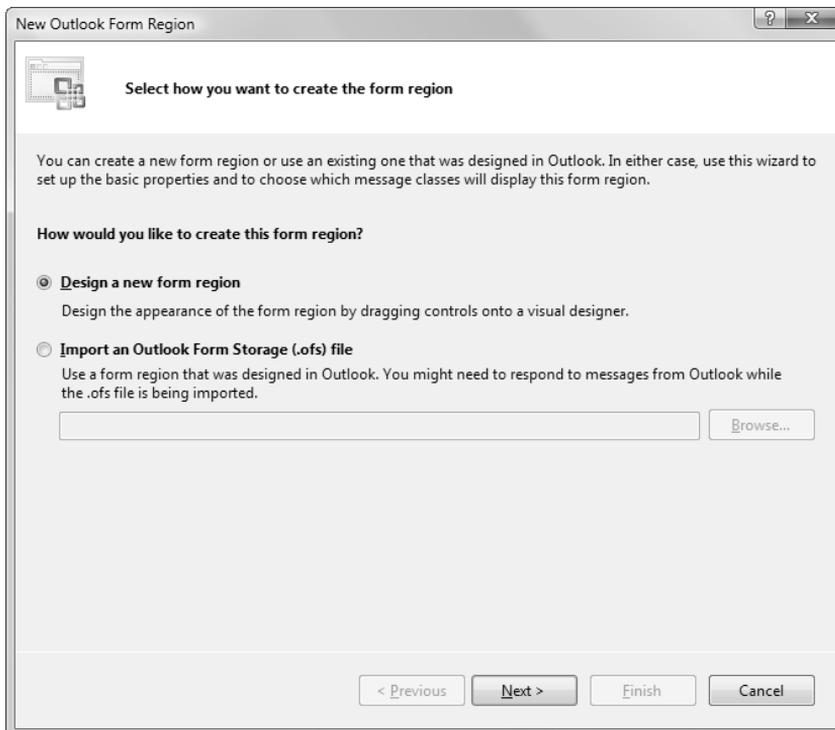


Figure 16-3: Selecting the form technology to use to create the form region.

To understand the types of form regions that are available in Figure 16-4, we must take a step back and discuss Inspector windows in some additional detail. Form regions are used primarily in Outlook Inspector windows. An Outlook Inspector window can have multiple pages associated with it, and Ribbon buttons are used to switch between the pages associated with a particular Inspector window. Consider the Inspector window that appears when you double-click an Outlook task, as shown in Figure 16-5.

Figure 16-5 has two Ribbon buttons in the Show group: Task and Details. In Figure 16-5 the Task button is selected and the Task page is displayed. The Task page is the default page for the Task Inspector window and is displayed first whenever a task is opened. If you click the Details button, the view changes to the Details page, as shown in Figure 16-6.

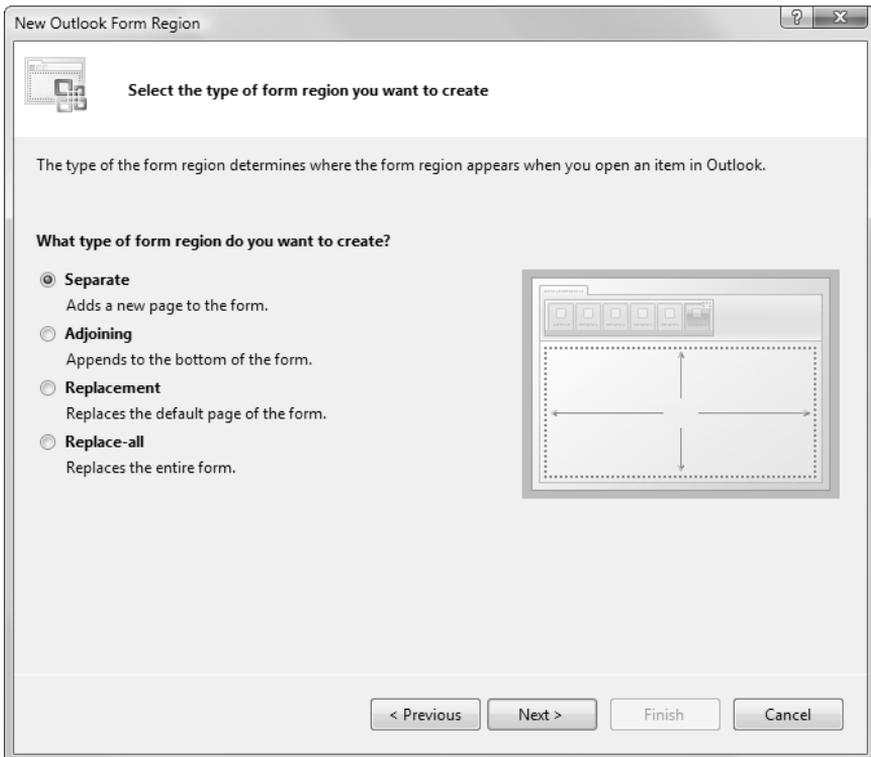


Figure 16-4: Selecting the type of form region to create: Separate.

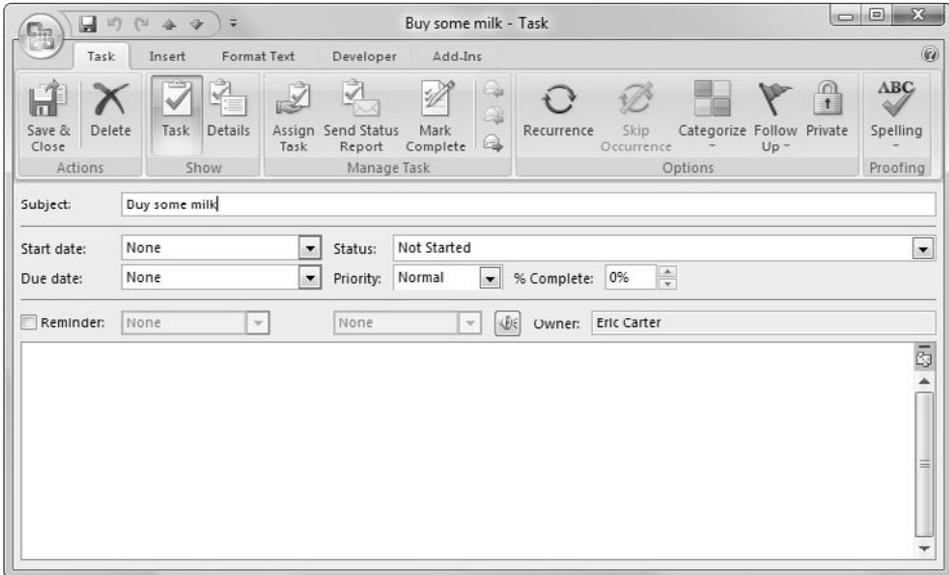


Figure 16-5: A task Inspector window with the Task page selected.

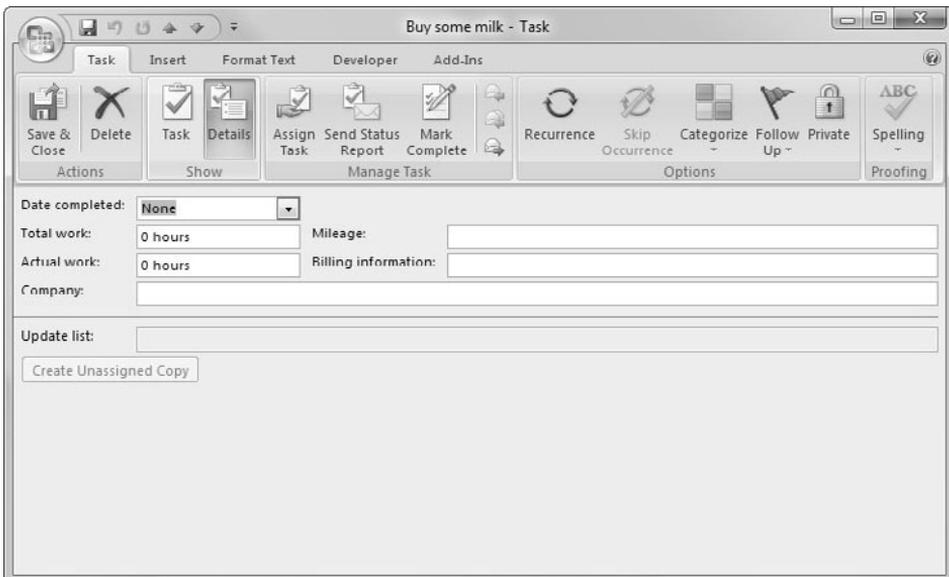


Figure 16-6: A Task Inspector window with the Details page selected.

With this background, you're ready to go back to Figure 16-4 and make sense of the options. A Separate form region adds a new page (and a new Ribbon button to activate that page) to an Inspector window. So you could add a new page to the Task Inspector window to show something like sub-tasks that must be completed to finish the main task. In Figure 16-4 the wizard also displays a nice graphic to help you remember what a Separate form region is. In this case the graphic emphasizes that you get a new Ribbon button to display the new page, and you have complete control of the new page that is shown.

Figure 16-7 shows what the wizard displays when you select Replacement instead of Separate as the type of form region. A Replacement form region allows you to replace the default page of the Inspector window. So in the task example, you could replace the Task page (the default page for a Task Inspector window), but the Details page would still be available.

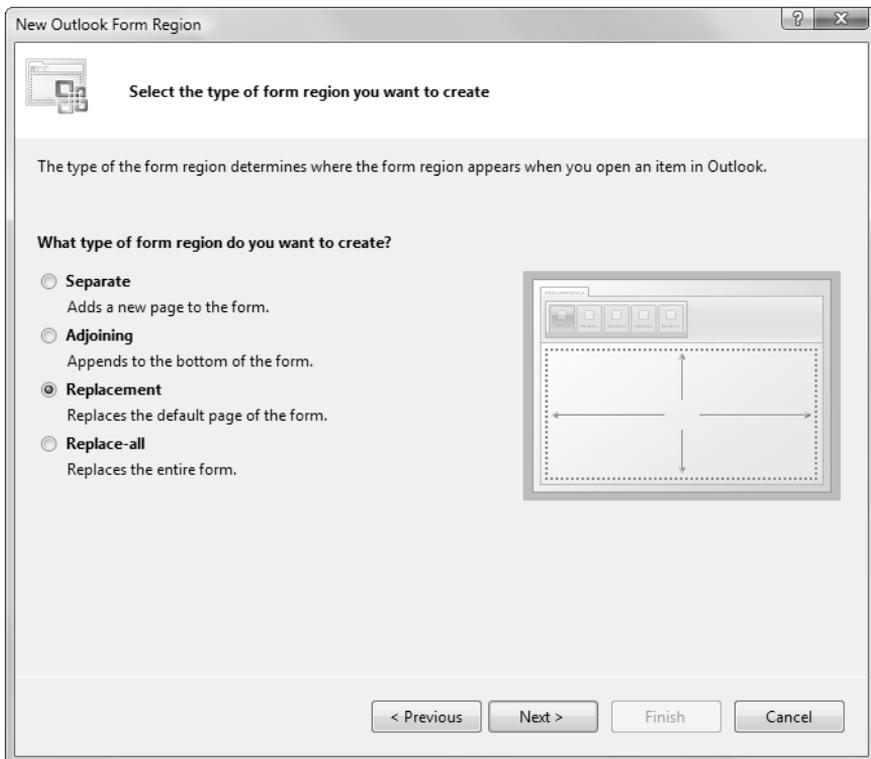


Figure 16-7: Selecting the type of form region to create: Replacement.

Figure 16-8 shows what the wizard displays when you select Replace-All as the type of form region. A Replace-All form region allows you to replace all available pages and make your page available only in the Inspector window. So in the task example, you could replace both the Task page and the Details page; your page would be the only page displayed in the Inspector window.

When you think about Replacement and Replace-All form region types, you realize that replacing the default pages for an Outlook item type is a pretty powerful capability—actually too powerful, in a way, because you could change the default page for an Outlook item type, such as a task, and implement a new default page that prevents the user from editing key data associated with that task. You may forget to provide a way to set the priority of a task in your Replacement or Replace-All form region, for example. Indeed, the creators of Outlook didn't want to give you quite that much power, enough to possibly break key functionality of Outlook.

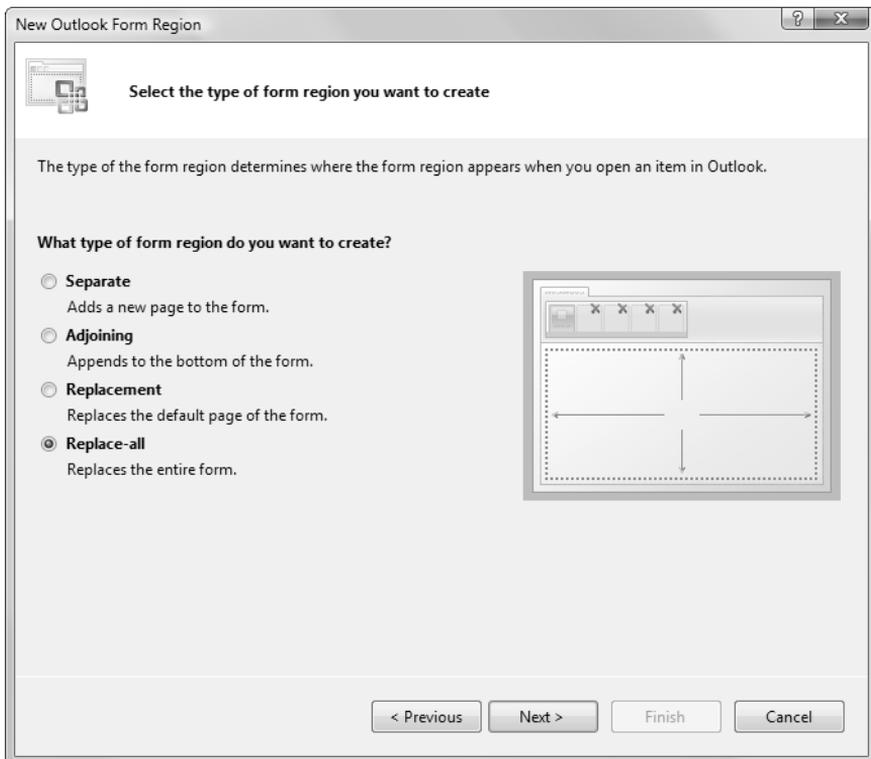


Figure 16-8: Selecting the type of form region to create: Replace-All.

To jump ahead a little, select Replacement or Replace-All as the form region type and then skip two steps ahead in the wizard by clicking the Next button twice. You see the wizard page shown in Figure 16-9, where you determine which Outlook message classes you want this form region to be associated with. When you select Replacement or Replace-All, notice that all the standard message classes (Appointment, Contact, Task, and so on) are grayed out in this dialog box. Outlook won't let you replace the default page or replace all the pages for standard message classes because you may break key features of Outlook. To use Replacement and Replace-All form region types, you must define a custom message class. A custom message class can reuse all the existing functionality of a built-in message class such as Appointment, Contact, or Task and acts as a specialized version of those built-in Outlook item objects. We discuss working with custom message

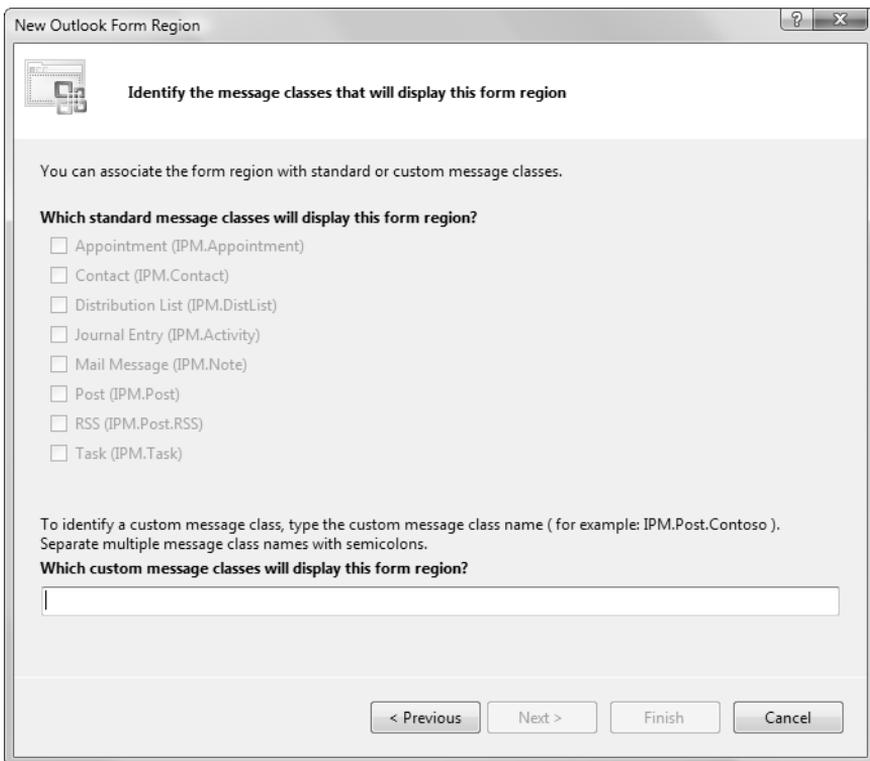


Figure 16-9: Replacement and Replace-All form regions can be associated only with custom message classes.

classes in more detail later in this chapter, in the section “Form Region Types and Custom Message Classes,” because you must understand that concept to use Replacement and Replace-All form region types.

Moving back to the page in the wizard where you pick the form region type, consider the final form region type: Adjoining, shown in Figure 16-10. An Adjoining form region is appended to the bottom of the default page for an Inspector. Multiple adjoining form regions can be associated with the same message class, so potentially you can have several Adjoining form regions displayed in one Inspector window’s default page. Adjoining form regions have headers that allow them to be collapsed and expanded to make more room in the default page when needed.

Another interesting application of an Adjoining form region is in an Explorer window. Specifically, an Adjoining form region can be used in the reading pane that is displayed in an Explorer window. In much the

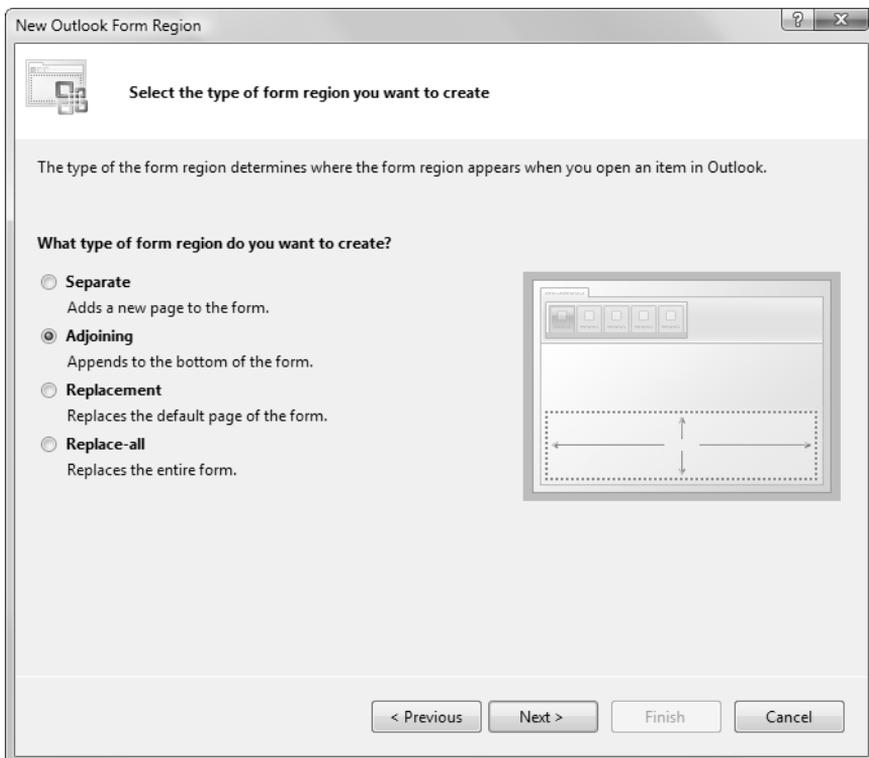


Figure 16-10: Selecting the type of form region to create: Adjoining.

same way that they are used in the default page of an Inspector window, multiple Adjoining form regions can be associated with an Outlook message class and can be displayed in the reading pane. Form regions displayed in the reading pane can also be collapsed to their headers. Replacement and Replace-All form regions can be used in the reading pane as well, although in this case they replace what is shown in the reading page and can be used only for custom message classes.

Now that you're familiar with all the form region types, select Adjoining as the form region type and click the Next button to move to the next page of the wizard, shown in Figure 16-11. In this dialog box, you set the name for the form region that will be displayed in the UI, so pick a friendly name. Title and Description are grayed out because you're creating an Adjoining form region; those options are enabled only for Replacement and Replace-All form region types.

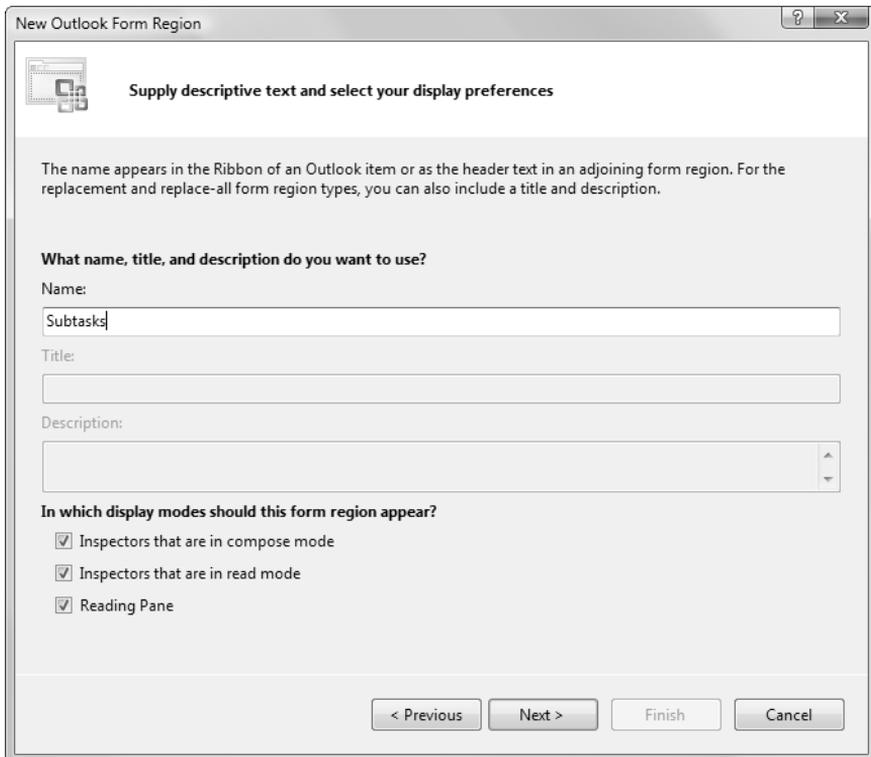


Figure 16-11: Setting descriptive text and display preferences.

This page of the wizard also has three check boxes that specify when the form region is displayed. The first check box sets whether the form region is displayed for an Inspector window that is in compose mode. An Inspector window is in compose mode when you create a new instance of the Outlook item associated with it—when you create a new task, for example. The second check box sets whether the form region is displayed for an Inspector window that is in read mode. An Inspector window is in read mode when you open an existing item—a mail message, for example. Finally, the third check box sets whether to display the form region in reading-pane view.

For this example, keep all the boxes checked and click the Next button to pick which Outlook message classes to associate the form region with, as shown in Figure 16-12. For this example, select Task. Note that you can associate the same form region with multiple built-in Outlook message

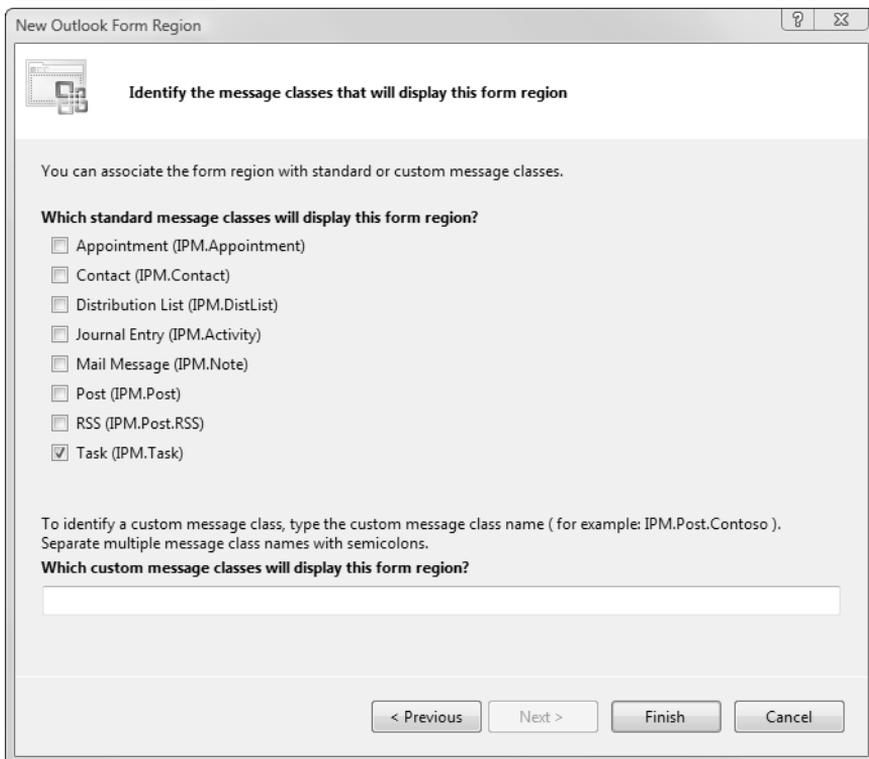


Figure 16-12: Picking which message classes will display a form region.

classes. You could have a form region that displays for both Tasks and Mail messages, for example. You can also associate a form region with custom message classes, which we discuss later in this chapter. As we describe earlier in this section, Replacement and Replace-All form region types can be associated only with custom message classes.

Associate the form region with the built-in Task type, and click the Finish button to exit the wizard. Visual Studio creates a new project item called `FormRegion1.cs`, as shown in Figure 16-13. It displays a visual designer in which you can drag and drop Windows Forms controls from the toolbox to construct the form region. This visual designer is much like the one you use to design user controls and task panes.

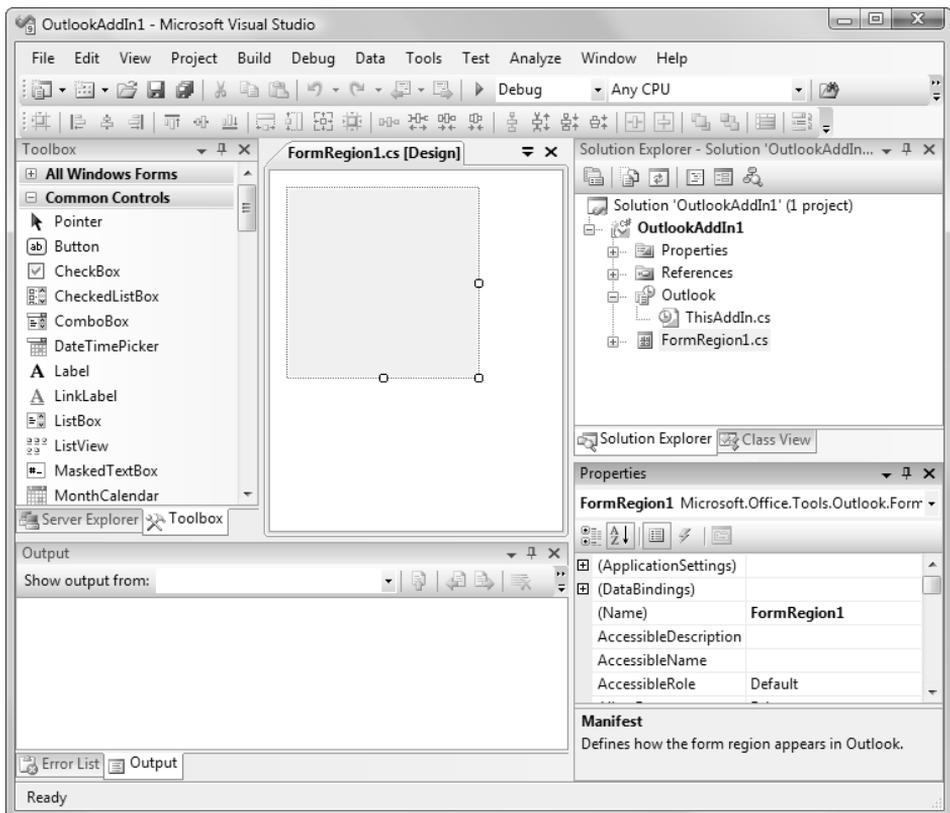


Figure 16-13: The newly created form region project item in visual design view.

Customizing a Form Region

Your goal is to add a form region in which subtasks can be associated with a task. First, drag and drop a list box control and a button to create a new task and delete an existing task. Because the user can resize the form region, use the Anchor property of the controls to anchor the list box to the top, left, bottom, and right, and anchor the buttons to the bottom and left. Figure 16-14 shows the final form region.

Before you go any further, run the add-in project and see what happens. Press F5 to build and run Outlook with the add-in project loaded. If you click a task in a task list and show reading view (by choosing View > Reading Pane > Bottom), you see that the adjoining form region is displayed docked at the bottom of reading-pane view for a task, as shown in Figure 16-15. If you double-click a task, the Adjoining form region is docked at the bottom of the default page for the Inspector window, as shown in Figure 16-16. After you've run your project, if you want to remove the form region and add-in from Outlook, choose Build > Clean.

Let's examine the adjoining form region a little more. First, notice that the Name you specified in Figure 16-11 is displayed as the caption above the Adjoining form region. To the left of the form region caption is a -/+ button that expands and collapses the form region. In Figure 16-17 you see what an Adjoining form region looks like when it is collapsed. Remember that several Adjoining form regions could be displayed in one Inspector window or reading pane; the ability to expand and collapse them is important, because it allows the end user to manage screen real estate.

Also, notice that when you resize the reading pane or the Inspector window, the form region has a default height. When the user adjusts the

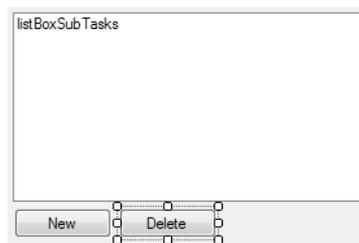


Figure 16-14: A simple form region.

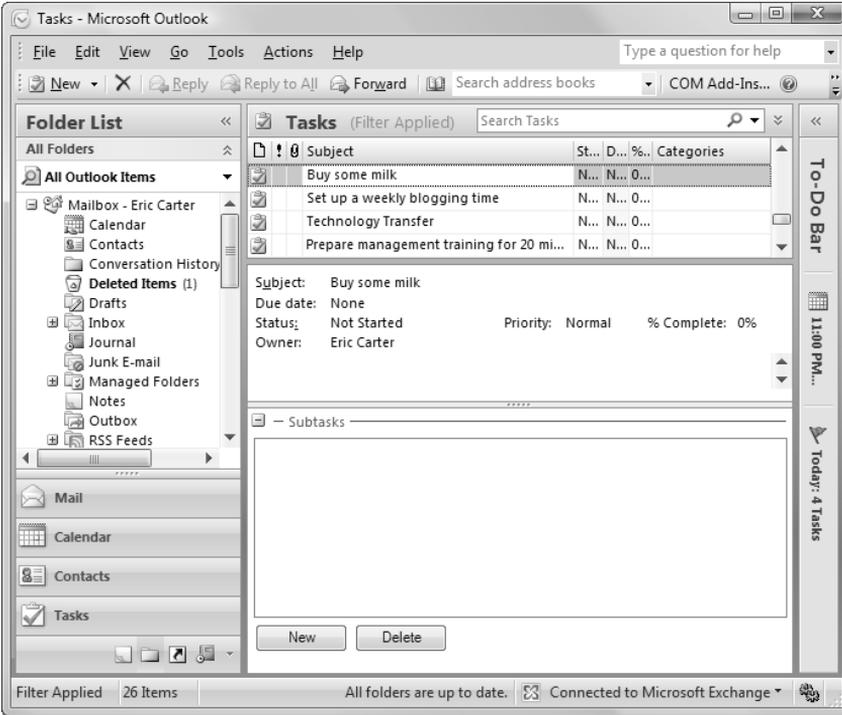


Figure 16-15: An Adjoining form region in the reading pane.

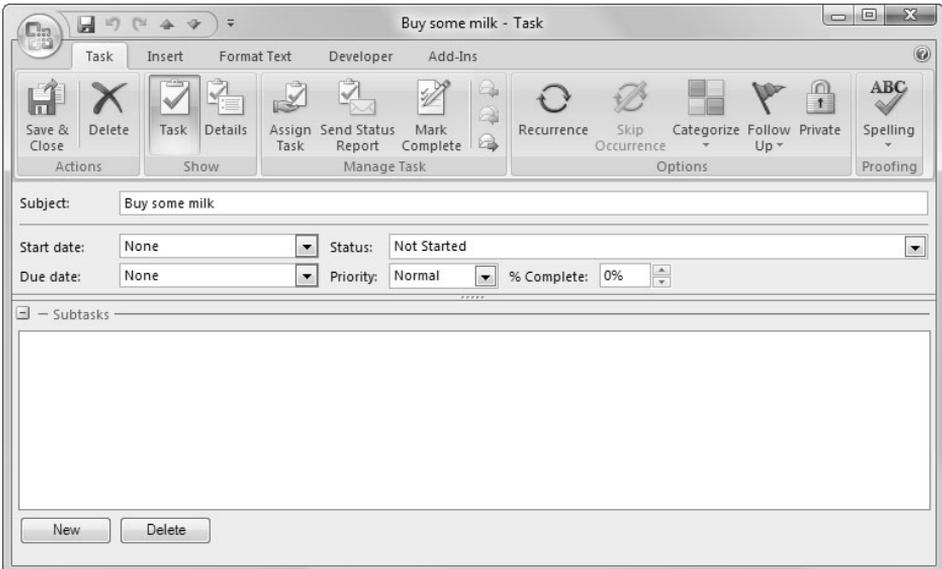


Figure 16-16: An Adjoining form region in the default page of an Inspector window.

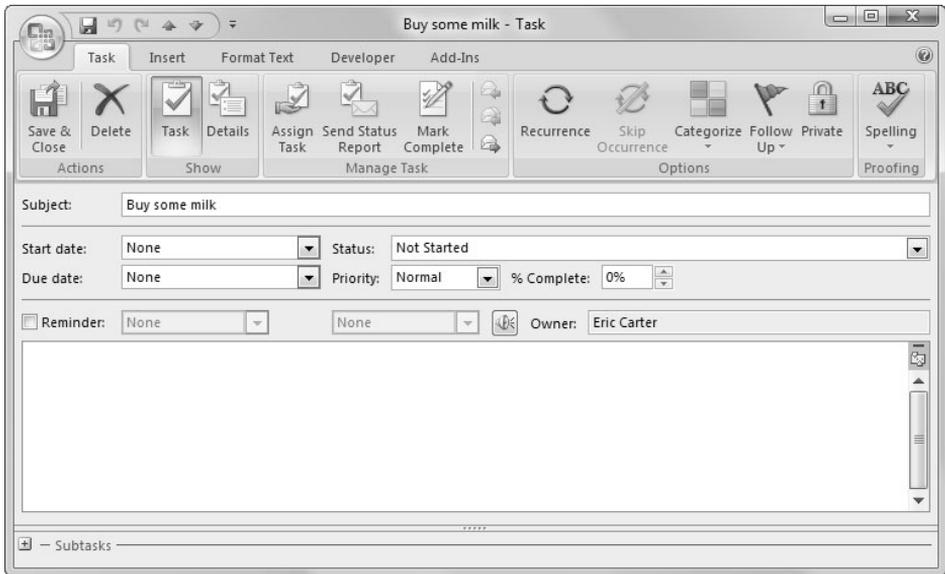


Figure 16-17: A collapsed Adjoining form region.

size of the form region, Outlook remembers the height and uses that height the next time the reading view is displayed. If you size the window small enough that the default height of the form region can't be displayed, a vertical scroll bar appears, as shown in Figure 16-18. This minimum height represents the height you set when you designed the form region. To have a smaller or larger minimum height, simply adjust the height of the visual design surface for the form region inside Visual Studio.

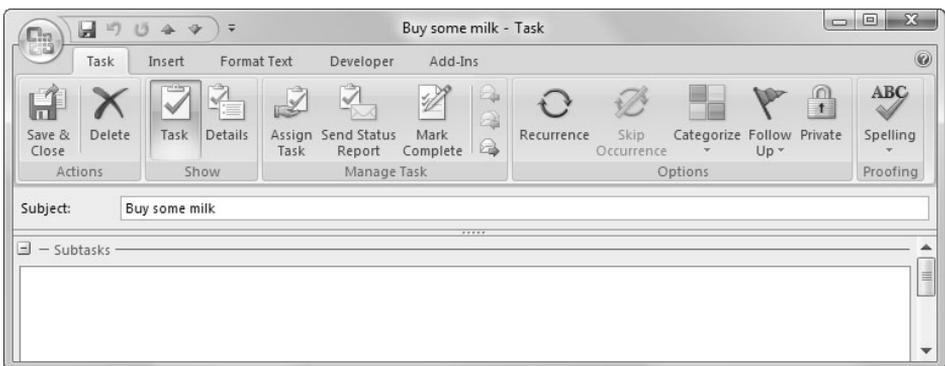


Figure 16-18: The effect of default height on the form region's vertical scroll bar.

Now exit Outlook and go back to the add-in project to put some code behind the form region. Right-click `FormRegion1.cs` in the Solution Explorer, and choose View Code from the context menu. The default code for a form region is shown in Listing 16-1. There are three event handlers of interest in our class `FormRegion1`. The first is actually in a nested class called `FormRegion1Factory`. This nested class provides a method called `FormRegion1Factory_FormRegionInitializing` where you can write code to decide whether to show the form region for a given Outlook item. The `FormRegionInitializing` event handler is passed a parameter `e` of type `FormRegionInitializingEventArgs` that can be used to get the Outlook item that the form region is about to be shown for (`e.OutlookItem`) and to cancel the showing of the form region if necessary by setting `e.Cancel` to `true`. Don't hold a reference to the Outlook item (`e.OutlookItem`) that is about to be shown; it is provided for use only during the event handler.

The form region class itself (`FormRegion1`) has a `FormRegionShowing` event handler that is invoked before the form region is displayed (but too late to prevent the display of the form region altogether; that is what `FormRegionInitializing` is for). In the `FormRegionShowing` event handler, you can write code to initialize your form region. In this event handler, you can use the property `this.OutlookItem` to access the Outlook item associated with the form region.

When the form region is closed, the `FormRegionClosed` event handler is invoked. This event handler is a good place to save any changes made to the Outlook item by your form region and to do any final cleanup.

Listing 16-1: The Default Code in a New Windows Forms-Based Form Region

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Office = Microsoft.Office.Core;
using Outlook = Microsoft.Office.Interop.Outlook;

namespace OutlookAddIn1
{
    partial class FormRegion1
    {
        #region Form Region Factory
        [Microsoft.Office.Tools.Outlook.
```

```
    FormRegionMessageClass(Microsoft.Office.Tools.Outlook.  
        FormRegionMessageClassAttribute.Task)]  
[Microsoft.Office.Tools.Outlook.  
    FormRegionName("OutlookAddIn1.FormRegion1")]  
public partial class FormRegion1Factory  
{  
    // Occurs before the form region is initialized.  
    // To prevent the form region from appearing, set e.Cancel  
    // to true. Use e.OutlookItem to get a reference to the  
    // current Outlook item.  
    private void FormRegion1Factory_FormRegionInitializing(  
        object sender, Microsoft.Office.Tools.Outlook.  
        FormRegionInitializingEventArgs e)  
    {  
    }  
}  
#endregion  
  
// Occurs before the form region is displayed.  
// Use this.OutlookItem to get a reference to the current  
// Outlook item. Use this.OutlookFormRegion to get a reference  
// to the form region.  
private void FormRegion1_FormRegionShowing(object sender,  
    System.EventArgs e)  
{  
}  
  
// Occurs when the form region is closed.  
// Use this.OutlookItem to get a reference to the current  
// Outlook item. Use this.OutlookFormRegion to get a reference  
// to the form region.  
private void FormRegion1_FormRegionClosed(object sender,  
    System.EventArgs e)  
{  
}  
}  
}
```

Listing 16-2 shows a simple implementation for the subtasks form region. You don't need to write any code in `FormRegionInitializing` because you always want to display your form region. In `FormRegionShowing`, write some code to get a custom `UserProperty` object from the Outlook item with which the form region is associated. The custom `UserProperty` we will associate with the Outlook item will have the identifier "SubTasks" You'll use this custom `UserProperty` to store the subtasks that are edited by the

form region. If the `UserProperty` isn't associated with the Outlook item yet, create the `UserProperty` for the Outlook item in `FormRegionInitializing`. The "SubTasks" user property contains a string value that contains subtasks delimited by a new line. You parse any subtasks that are in the string and populate the list box for the form region with the subtasks.

In `FormRegionClosed`, you do the reverse: Grab all the entries out of the list box and concatenate them into a string in which subtasks are separated by new lines. If the subtasks have been changed, set the "SubTasks" `UserProperty`'s value to the new string and save the associated Outlook item.

Finally, a simple implementation for the Add button just adds the current time as a new subtask; a complete implementation would include a dialog box with an edit box in which the user could type a subtask description. The Delete button deletes the selected list item.

Listing 16-2: Form Region Code for a Simple Subtasks Form Region Based on Windows Forms

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Office = Microsoft.Office.Core;
using Outlook = Microsoft.Office.Interop.Outlook;

namespace OutlookAddIn1
{
    partial class FormRegion1
    {
        Outlook.TaskItem task;
        Outlook.UserProperty subtasks;

        #region Form Region Factory
        [Microsoft.Office.Tools.Outlook.
            FormRegionMessageClass(Microsoft.Office.Tools.Outlook.
                FormRegionMessageClassAttribute.Task)]
        [Microsoft.Office.Tools.Outlook.
            FormRegionName("OutlookAddIn1.FormRegion1")]
        public partial class FormRegion1Factory
        {
            // Occurs before the form region is initialized.
            // To prevent the form region from appearing, set e.Cancel
            // to true. Use e.OutlookItem to get a reference to the
            // current Outlook item.
```

```
private void FormRegion1Factory_FormRegionInitializing(
    object sender, Microsoft.Office.Tools.Outlook.
    FormRegionInitializingEventArgs e)
{
}
}
#endregion

// Occurs before the form region is displayed.
// Use this.OutlookItem to get a reference to the current
// Outlook item. Use this.OutlookFormRegion to get a reference
// to the form region.
private void FormRegion1_FormRegionShowing(object sender,
    System.EventArgs e)
{
    task = this.OutlookItem as Outlook.TaskItem;
    if (task != null)
    {
        // Check for custom property SubTasks
        subTasks = task.UserProperties.Find("SubTasks", true);
        if (subTasks == null)
        {
            subTasks = task.UserProperties.Add("SubTasks",
                Outlook.OlUserPropertyType.olText, false,
                Outlook.OlUserPropertyType.olText);
        }
    }
}

// Convert string
string subTasksString = subTasks.Value.ToString();
if (!String.IsNullOrEmpty(subTasksString))
{
    string[] delimiters = new string[1];
    delimiters[0] = System.Environment.NewLine;
    string[] tasks = subTasksString.Split(delimiters,
        StringSplitOptions.RemoveEmptyEntries);
    for (int i = 0; i < tasks.Length; i++)
    {
        listBoxSubTasks.Items.Add(tasks[i]);
    }
}
}

// Occurs when the form region is closed.
// Use this.OutlookItem to get a reference to the current
// Outlook item. Use this.OutlookFormRegion to get a reference
// to the form region.
private void FormRegion1_FormRegionClosed(object sender,
    System.EventArgs e)
```

```

{
    if (subTasks == null || task == null)
        return;

    string oldTasks = subTasks.Value.ToString();
    StringBuilder builder = new StringBuilder();

    foreach (object o in listBoxSubTasks.Items)
    {
        string t = o as string;
        if (!String.IsNullOrEmpty(t))
        {
            builder.AppendLine(t);
        }
    }

    string newTasks = builder.ToString();

    if (!String.IsNullOrEmpty(newTasks) &&
        !String.IsNullOrEmpty(oldTasks))
    {
        if (newTasks.CompareTo(oldTasks) == 0)
            return; // no changes
    }

    subTasks.Value = newTasks;
    task.Save();
}

private void buttonNew_Click(object sender, EventArgs e)
{
    // Just add current time as a subtask for simplicity
    listBoxSubTasks.Items.Add(
        System.DateTime.Now.ToShortTimeString());
}

private void buttonDelete_Click(object sender, EventArgs e)
{
    if (listBoxSubTasks.SelectedItem != null)
    {
        listBoxSubTasks.Items.RemoveAt(
            listBoxSubTasks.SelectedIndex);
    }
}
}
}

```

When you run the form region, it displays as before, but now the Add and Delete buttons work, and you can add subtasks (set to the current time) to the current task.

Form Region Types and Custom Message Classes

Table 16-1 summarizes the behaviors and capabilities of the four types of form regions we introduced in the preceding sections. Figure 16-19 shows what the subtasks form region you created in the introduction looks like when it is changed to a Separate form region type. To make this change in Visual Studio, simply click the form region surface in the visual designer and change the `FormRegionType` property in the property grid. (This property can be a bit hard to find initially; it is a child property of the expandable `Manifest` property in the property grid.) Now when you open the Task Inspector, an additional Ribbon button appears in the Show group with the name of the separate form region—in this example, Subtasks. Subtasks is not the default page (Task is the default page), but when you click the Subtasks button, the form region page is displayed.

TABLE 16-1: Behavioral Capabilities of the Four Form Region Types

	Separate	Adjoining	Replacement	Replace-All
Inspector window behavior	Adds a new page	Appends to the bottom of the default page	Replaces the default page	Replaces all pages
Reading pane behavior	N/A	Appends to the bottom of the reading pane	Replaces the reading pane	Replaces the reading pane
Can customize standard built-in message classes	Yes	Yes	No—custom message classes only	No—custom message classes only

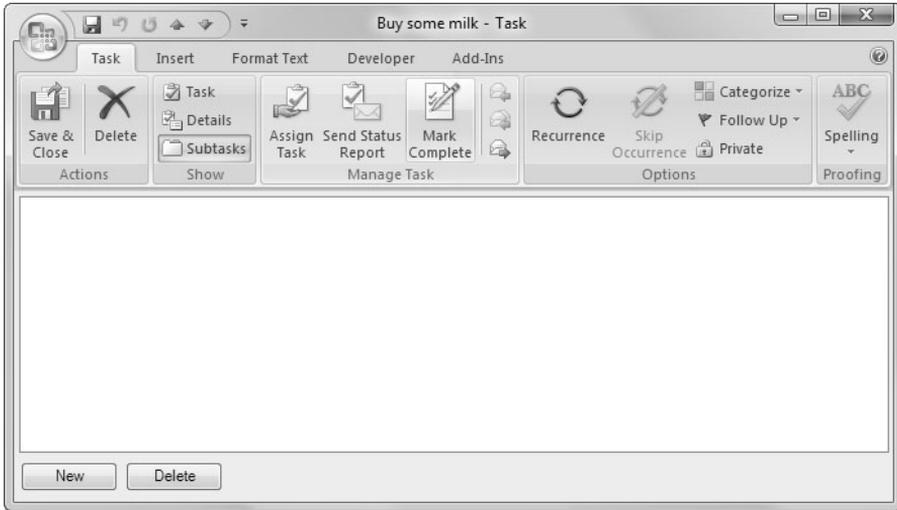


Figure 16-19: A Separate form region version of the Subtasks form region.

Built-In and Custom Message Classes

To convert the example form region to a Replacement or Replace-All form region type, you need to learn a little bit more about built-in and custom message classes. The type of all Outlook items is identified by a string value called a message class. Table 16-2 lists the message classes associated with some of the major Outlook item types.

You can define your own custom message class by defining your own message class string. The message class string must begin with a built-in message class string from Table 16-2 to ensure that you inherit the behavior associated with a built-in message class; Outlook does not support having a “baseless” message class that doesn’t inherit behavior from a built-in Outlook type. Then you append your own unique identifier to the message class string to create a unique message class string. If you want to create a custom message class based on Outlook’s built-in contact type that extends the contact with some information about the Facebook user ID associated with that contact, for example, your message class string might be "IPM.Contact.FacebookAware". The important thing is that your custom message class string start with a built-in message class identifier ("IPM.Contact", for example) and have some additional identifier that

TABLE 16-2: Built-In Outlook Message Classes

Outlook Item Type	Message Class String
Appointment	IPM.Appointment
Contact	IPM.Contact
Distribution List	IPM.DistList
Journal Entry	IPM.Activity
Mail Message	IPM.Note
Post	IPM.Post
RSS Post	IPM.Post.RSS
Sharing Invitation	IPM.Sharing
Task	IPM.Task

won't be picked by another add-in developer. So you might make it more unique by embedding your company name, as in "IPM.Contact.FacebookAwareAddisonWesley".

You can use these unique string custom message classes to create Outlook items with the `Items.Add` method on an Outlook Folder object. You can modify the code of the add-in you created in the introduction to edit the `ThisAddIn_Startup` method so that it creates an Outlook item with a custom message class based on `Task`, to be called "IPM.Task.MySubTaskAwareTask". Listing 16-3 shows the new `ThisAddIn.cs` code file.

Listing 16-3: An Outlook Add-In That Creates a New Outlook Item with a Custom Message Class Based on Task

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml.Linq;
using Outlook = Microsoft.Office.Interop.Outlook;
using Office = Microsoft.Office.Core;
```

```
namespace OutlookAddIn1
{
    public partial class ThisAddIn
    {
        private void ThisAddIn_Startup(object sender,
            System.EventArgs e)
        {
            Outlook.Folder taskList =
                Application.Session.GetDefaultFolder(
                    Outlook.OlDefaultFolders.olFolderTasks)
                as Outlook.Folder;

            Outlook.TaskItem taskItem = taskList.Items.Add(
                "IPM.Task.MySubTaskAwareTask") as Outlook.TaskItem;

            taskItem.Subject = "IPM.Task.MySubTaskAwareTask Created On "+
                System.DateTime.Now.ToLongDateString();

            taskItem.Save();
        }

        #region VSTO generated code
        private void InternalStartup()
        {
            this.Startup += new System.EventHandler(ThisAddIn_Startup);
        }
        #endregion
    }
}
```

Now that the Add-in creates a new task with a custom message class on startup, you can modify the form region to be a Replacement form region type. To do this, double-click `FormRegion1.cs` in the Solution Explorer to activate the form region designer. In the Properties window, pick `FormRegion1` in the list of controls. Expand the Manifest section of the Properties window, and set the `FormRegionType` to Replacement.

Now you need to change the `FormRegion1Factory` so that it associates the form region with the custom message class "IPM.Task.MySubTaskAwareTask" rather than with the built-in message class for a task, "IPM.Task". To do this, you need to edit an attribute of the `FormRegion1Factory` class. Looking at the `FormRegion1Factory` class, you see two custom attributes: `FormRegionMessageClass` and `FormRegionName`. `FormRegionMessageClass` tells the factory what message class to show the form

region for. Because you associated the form region with a task when you created it in the form region wizard, the `FormRegionMessageClass` attribute is set to display for the string specified by the constant `Microsoft.Office.Tools.Outlook.FormRegionMessageClassAttribute.Task`. This string is a constant string that is set to "IPM.Task". The `FormRegionName` attribute is set to the fully qualified name of the form region class—in this case, "OutlookAddIn1.FormRegion1". Both custom attributes are shown here:

```
#region Form Region Factory
[Microsoft.Office.Tools.Outlook.FormRegionMessageClass(
    Microsoft.Office.Tools.Outlook.
    FormRegionMessageClassAttribute.Task)]
[Microsoft.Office.Tools.Outlook.FormRegionName(
    "OutlookAddIn1.FormRegion1")]
public partial class FormRegion1Factory
{
```

Change the `FormRegionMessageClass` attribute to take the custom message class string "IPM.Task.MySubTaskAwareTask", as follows:

```
#region Form Region Factory
[Microsoft.Office.Tools.Outlook.FormRegionMessageClass(
    "IPM.Task.MySubTaskAwareTask")]
[Microsoft.Office.Tools.Outlook.FormRegionName(
    "OutlookAddIn1.FormRegion1")]
public partial class FormRegion1Factory
{
```

Now when you run the add-in, a new task with custom message class "IPM.Task.MySubTaskAwareTask" is created in the Startup handler for the add-in. When the new task with the custom message class is opened, an Inspector window with the default page replaced is displayed, as shown in Figure 16-20. Note in the Show group on the Ribbon, the default is now the Subtasks form region. The original default page, Task, is no longer visible in the pages that can be shown for the task.

Finally, you can go back to the add-in and use the Properties window to set the `FormRegionType` to Replace-All. When the add-in is run and a task with the custom message class is opened, the Inspector window has all the pages removed except for your form region, as shown in Figure 16-21.

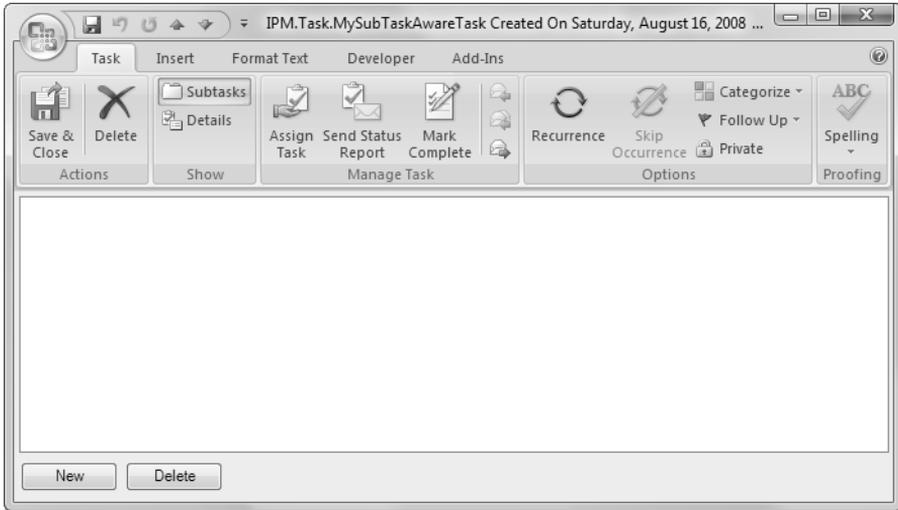


Figure 16-20: A Replacement form region version of the Subtasks form region.

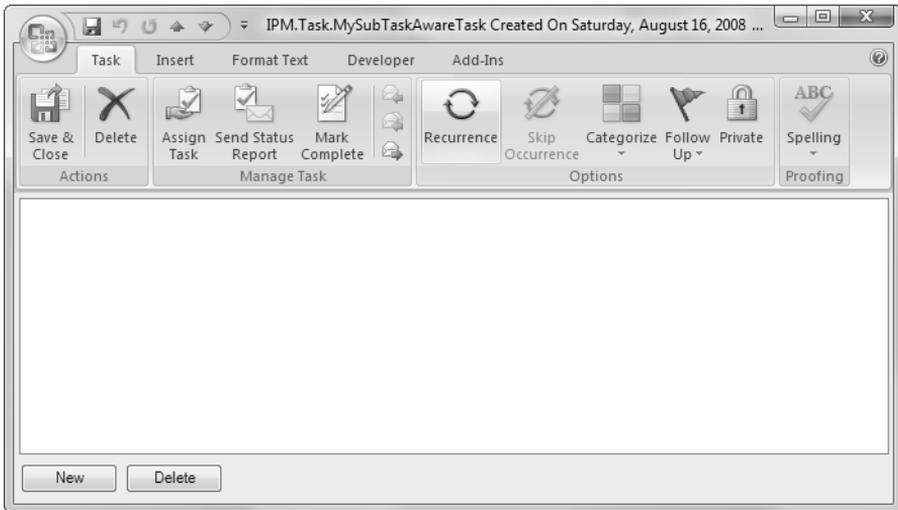


Figure 16-21: A Replace-All form region version of the Subtasks form region.

NOTE

In Figure 16-21, the Show group on the Ribbon is no longer displayed. Because only one page is available to be displayed (your form region), there is no longer a reason to display the Show group, as no other pages can be selected.

Creating an Outlook Forms-Based Form Region

As you saw in the introduction, there are two ways to create form regions in VSTO. The first is to use the Windows Forms designers inside Visual Studio. The second is to use the Outlook Forms Designer inside Outlook. A form region designed with the Outlook Forms Designer is integrated with your .NET code through COM interop. The form region and form controls are COM objects that Visual Studio generates wrappers to communicate with.

If you use Windows Forms, you can use a forms engine that is .NET-based, which may be more familiar to you. The design-time experience for using Windows Forms is much more integrated with Visual Studio. If you use the Outlook Forms Designer, you need to design the form in Outlook and then import it into Visual Studio. If you decide that you want to change the layout or the controls on the form, you have to delete the form region in Visual Studio, go back to Outlook and redesign the form region, and then reimport it into Visual Studio via the wizard.

Although using Windows Forms is more convenient, Outlook Forms have a lot of features that Windows Forms do not. Outlook Forms, for example, have automatic data binding support to bind to properties of the Outlook items with which they are associated. Also, some Outlook Forms controls are more Outlook-like than any of the Windows Forms controls you have available to you. These controls allow you to replicate functionality available in built-in Outlook Inspector windows. Outlook Forms provide controls such as these: the Outlook Body control, which allows you to edit the item body of an Outlook item with the same editor provided for built-in Outlook editors; a Business Card control, which displays the same business-card view that is built into Outlook; and a Category control, which provides a UI for visualizing the categories with which an Outlook item is associated. So in many cases you may pick an Outlook Forms-based form region because it provides more Outlook-aware controls for you to use.

The first step in creating a form region by using Outlook Forms is launching Outlook. Next, choose Tools > Forms > Design a Form to bring up the Design Form dialog box, shown in Figure 16-22. Pick the type of built-in Outlook item type that you want to start from—for this example, Task—and then click the Open button.

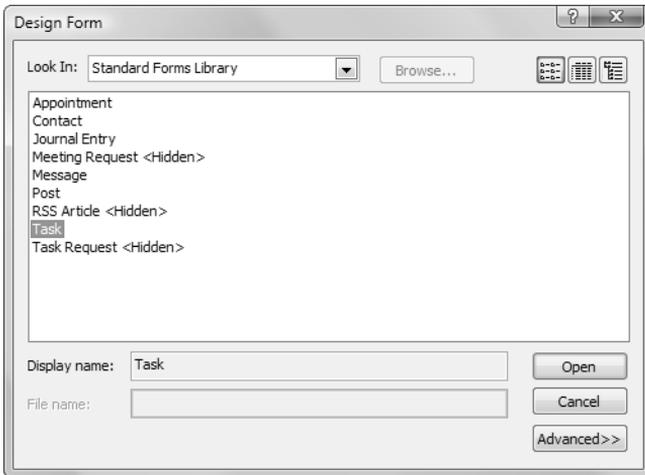


Figure 16-22: The Design Form dialog box in the Outlook Forms Designer.

Next, drop down the Design button and then the Form Region button, and choose New Form Region, as shown in Figure 16-23. (You can also use the Open Form Region command under the Form Region button if you already have a form region in an .OFS file.) A new page titled (Form Region) will appear; you can design your form region in that page.

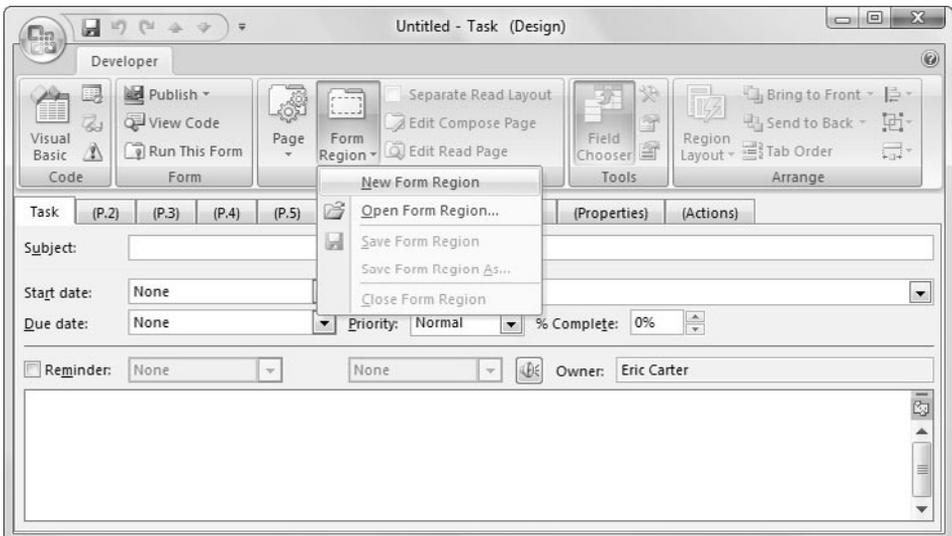


Figure 16-23: Creating a new form region in the Outlook Forms Designer.

The Tools group in the Ribbon allows you to bring up the design-time tools you will need. The Field Chooser tool lets you drag and drop fields into your form region from the Outlook item that the form region will display. These fields are automatically data bound—an advantage over Windows Forms, which require you to write additional code to bind your controls to the Outlook item associated with your form region. Also in the Tools group, the Controls Toolbox button brings up the toolbox, which displays a set of standard controls. The Advanced Properties button displays the properties window, which you can use to set properties for the selected control in the Forms Designer.

The initial set of tools in the Controls toolbox doesn't have any of the cool controls we mentioned earlier, so let's get them added to the toolbox. Right-click a blank area of the Controls toolbox, and choose Custom Controls from the context menu. The Additional Controls dialog box appears, as shown in Figure 16-24. Check all the controls in the list that start with *Microsoft Office Outlook*; then click the OK button.

Figure 16-25 shows the final design environment with all the tools visible and all the additional Outlook controls added to the Controls toolbox.

Now drag some Outlook controls to the design surface. You'll create the same form region you created in the introduction but use Outlook Forms this time. Find the `OlkListBox` control by hovering over the controls in the

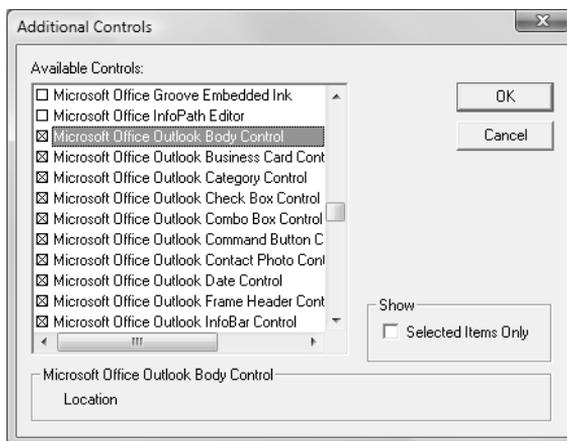


Figure 16-24: Add all the controls that start with *Microsoft Office Outlook* to the Controls toolbox.



Figure 16-25: The Outlook Forms Designer with the toolbox, Properties window, and Field Chooser tool.

Controls toolbox and finding the control that shows `OlkListBox` in its tooltip. Drag and drop a `OlkListBox`, and size it to fill most of the design surface while leaving a strip at the bottom for buttons. Right-click the list-box control you added to the form, and choose Properties from the context menu to bring up the Properties dialog box, shown in Figure 16-26. Click the Layout tab, and drop down the Horizontal combo box to pick Grow/Shrink with Form. This setting allows the list box to size to fill the Inspector window.

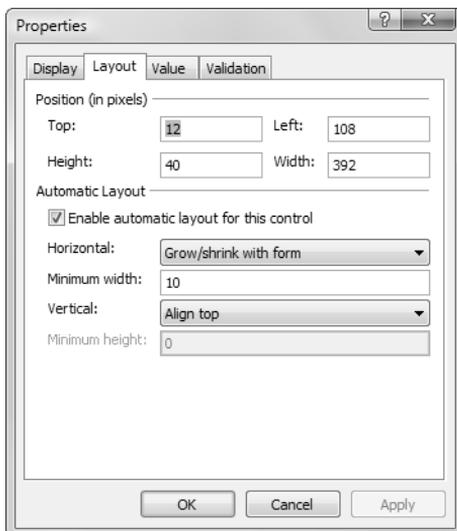


Figure 16-26: The Properties dialog box with the Layout tab.

While the Properties dialog box is open, take some time to explore the rest of it. The Properties dialog box has a Display tab that lets you set the caption of the control, visibility, font, and color. The Layout tab lets you set size and position, as well as several useful autosizing and alignment settings. The Value tab lets you set up an automatic binding to an Outlook item property. Finally, the Validation tab lets you set up some validation rules for the control.

Next, drag two additional Outlook controls onto the design surface. Drag and drop two `OlkCommandButton` controls at the bottom of the Outlook Form region. The `OlkCommandButton` will display with a look and feel more consistent with the Outlook UI than with a `CommandButton`. Right-click each of the `OlkCommandButton` controls, and choose Properties from the context menu to display the Properties window. In the Display tab, set the caption of one button to Add and the other to Delete. Also, in the Layout tab, set the Vertical drop-down menu to Align Bottom for each of the two buttons to ensure that the buttons stay at the bottom of the form when the form is resized. The final form region should look like Figure 16-27.

With a form region designed, you need to export the form region to an Outlook form region file with a `.OFS` extension, which then can be imported into Visual Studio. To save the form region as an `.OFS` file, drop down the Design button; then drop down the Form Region button and choose Save Form Region As to bring up the Save dialog box. Save it as `MyFormRegion` for this example, as shown in Figure 16-28.

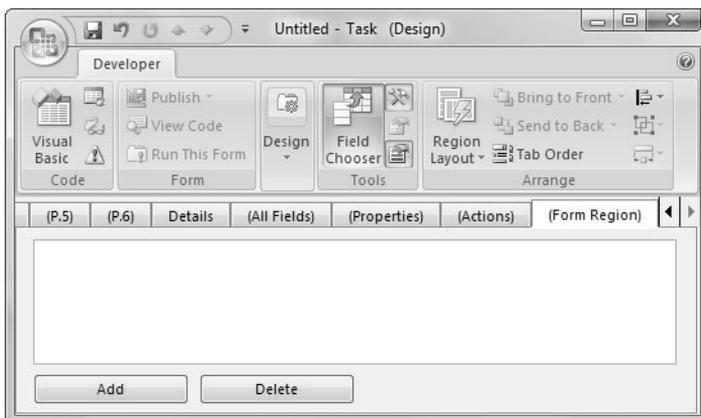


Figure 16-27: A form region designed in the Outlook Form Designer.

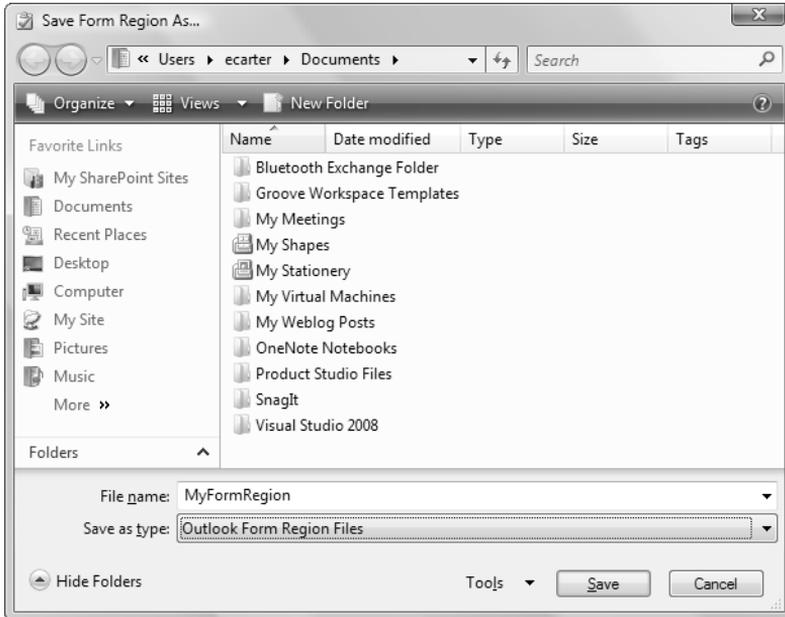


Figure 16-28: Saving the form region to an Outlook form region file (.OFS file).

TIP

At this point, we recommend that you exit Outlook before moving back to Visual Studio. The process of importing a .OFS file from Visual Studio involves Visual Studio starting up Outlook and talking to it to process the OFS file, and we've found that this process works best if you don't already have Outlook already open.

Start Visual Studio, and either create a new Outlook add-in project or open an existing Outlook add-in project. Choose Project > Add New Item. Click the Office category to show just the Office-specific items. In the list of Office items, click Outlook Form Region (refer to Figure 16-2). Name the form region—for this exercise, FormRegion2. Then click the Add button.

In the first page of the New Outlook Form Region wizard that appears, pick Import an Outlook Form Storage (.ofs) File, as shown in Figure 16-29. Click the Browse button, and locate the .OFS file that you saved earlier.

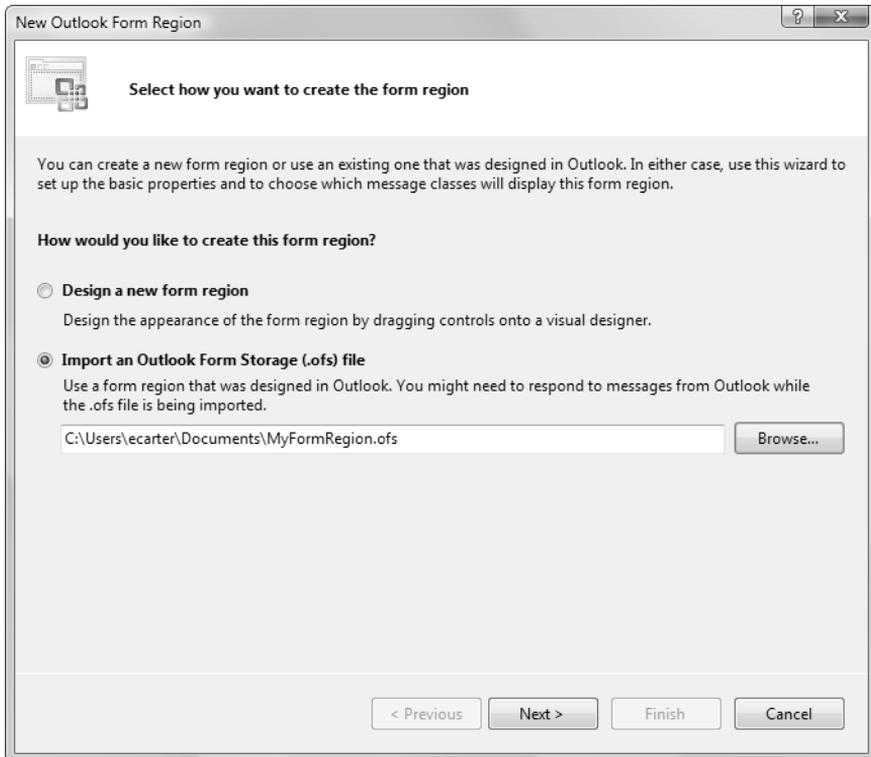


Figure 16-29: Importing an .OFS file in the New Outlook Form Region wizard.

With a .OFS file selected, click the Next button to move to the page where you set the type of the form region. For this example, select Separate as the Form Region type. Then click the Next button. On the next page, set the name you want to use for the form region—for this exercise, Subtasks—and make sure that the check boxes for Inspectors that are in compose mode and Inspectors that are in read mode are checked. Click the Next button to move to the final page. On this page, make sure that only the check box next to Task is checked; then click the Finish button. Visual Studio creates a new project item for the form region.

No visual designer is displayed within Visual Studio—just generated code. As we mention earlier in this chapter, if you want to change the form region, you have to delete the form region code item from your Visual Studio project, go back to Outlook and reopen the .ofs file, modify your form

region, save it as a .ofs file, exit Outlook, and then re-create the form region in Visual Studio.

TIP

If you are changing only the layout of the form region, you can edit the .ofs file without regenerating the form region in Visual Studio; just copy the modified .ofs file over the old .ofs file in your project folder.

The code looks very similar to the code for a Windows Forms-based form region. As before, you have a form region class with a nested form region factory class. The form region factory class has a `FormRegionInitializing` event handler where you can write code to determine whether to show the form region. The event handler is passed a parameter `e` of type `FormRegionInitializingEventArgs` that can be used to get the Outlook item that the form region is about to be shown for (`e.OutlookItem`) and to cancel the showing of the form region if necessary by setting `e.Cancel` to `true`.

The form region class has a `FormRegionShowing` event handler that is invoked before the form region is displayed (but too late to prevent the display of the form region altogether). In this event handler, you can write code to initialize the form region and use `this.OutlookItem` to access the Outlook item associated with the form region.

When the form region is closed, the `FormRegionClosed` event handler is invoked. This event handler is a good place to save any changes made to the Outlook item by your form region and do any final cleanup.

There are also some major differences between a Windows Forms-based form region and an Outlook Forms-based form region. Because there is no design view, no property grid like the one in Windows Forms allows you to interact with key settings—especially the Manifest settings that are editable in a Windows Forms-based form region. To compensate for this deficiency in Outlook Forms-based form regions, VSTO adds a second method called `InitializeManifest` to the nested form region factory code, as shown in Listing 16-4. In this method, you can modify the code to change the Form region type or any of the other settings that you initially set in the Form Region wizard.

Listing 16-4: The Default Code in a New Outlook Forms-Based Form Region

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Resources;
using System.Text;
using Office = Microsoft.Office.Core;
using Outlook = Microsoft.Office.Interop.Outlook;

namespace OutlookAddIn2
{
    public partial class FormRegion2
    {
        #region Form Region Factory

        [Microsoft.Office.Tools.Outlook.
            FormRegionMessageClassAttribute.Task]
        [Microsoft.Office.Tools.Outlook.
            FormRegionName("OutlookAddIn2.FormRegion2")]
        public partial class FormRegion2Factory
        {
            private void InitializeManifest()
            {
                ResourceManager resources =
                    new ResourceManager(typeof(FormRegion2));
                this.Manifest.FormRegionType =
                    Microsoft.Office.Tools.Outlook.FormRegionType.Separate;
                this.Manifest.Title = resources.GetString("Title");
                this.Manifest.FormRegionName =
                    resources.GetString("FormRegionName");
                this.Manifest.Description =
                    resources.GetString("Description");
                this.Manifest.ShowInspectorCompose = true;
                this.Manifest.ShowInspectorRead = true;
                this.Manifest.ShowReadingPane = false;
            }

            // Occurs before the form region is initialized.
            // To prevent the form region from appearing, set e.Cancel to
            // true. Use e.OutlookItem to get a reference to the current
            // Outlook item.
            private void FormRegion2Factory_FormRegionInitializing(
                object sender, Microsoft.Office.Tools.Outlook.
                FormRegionInitializingEventArgs e)
            {
            }
        }
    }
}
```

```
#endregion

// Occurs before the form region is displayed.
// Use this.OutlookItem to get a reference to the current
// Outlook item. Use this.OutlookFormRegion to get a reference
// to the form region.
private void FormRegion2_FormRegionShowing(object sender,
    System.EventArgs e)
{
}

// Occurs when the form region is closed. Use this.OutlookItem
// to get a reference to the current Outlook item. Use
// this.OutlookFormRegion to get a reference to the form/
// region.
private void FormRegion2_FormRegionClosed(object sender,
    System.EventArgs e)
{
}
}
}
```

Another obvious difference is that the form region class created when you import a .OFS file does not derive from `System.Windows.Forms.UserControls`. It derives from a class in VSTO called `Microsoft.Office.Tools.Outlook.ImportedFormRegion`.

As you write your code in the form region class, you will find that VSTO has created member variables for all the controls you used in the .OFS file. For this example, you created a `OlkListBox` with a default name of `OlkListBox1`, an `OlkCommandButton` with a name of `OlkCommandButton1`, and an `OlkCommandButton` with a name of `OlkCommandButton2`. The import of the .OFS file converts the names used in the Outlook Form designer to camel case, so you have three controls named `olkListBox1`, `olkCommandButton1`, and `olkCommandButton2`.

These controls are of types that come from the `Microsoft.Office.Interop.Outlook` namespace. This namespace has types for many of the built-in Outlook controls. Some of the controls in the toolbox generate types that come from the `Microsoft.Vbe.Interop.Forms` namespace. Table 16-3 shows the names of the controls in Outlook's Controls toolbox and the .NET types associated with these controls.

TABLE 16-3: Mapping Between Outlook Controls and .NET Types

Name	Type
Microsoft Forms 2.0 CheckBox	Microsoft.Office.Interop.Outlook.OlkCheckBox
Microsoft Forms 2.0 ComboBox	Microsoft.Office.Interop.Outlook.OlkComboBox
Microsoft Forms 2.0 CommandButton	Microsoft.Office.Interop.Outlook.OlkCommandButton
Microsoft Forms 2.0 Frame	Microsoft.Vbe.Interop.Forms.UserForm
Microsoft Forms 2.0 Image	Microsoft.Vbe.Interop.Forms.Image
Microsoft Forms 2.0 Label	Microsoft.Office.Interop.Outlook.OlkLabel
Microsoft Forms 2.0 ListBox	Microsoft.Office.Interop.Outlook.OlkListBox
Microsoft Forms 2.0 MultiPage	Microsoft.Vbe.Interop.Forms.MultiPage
Microsoft Forms 2.0 OptionButton	Microsoft.Office.Interop.Outlook.OlkOptionButton
Microsoft Forms 2.0 ScrollBar	Microsoft.Vbe.Interop.Forms.ScrollBar
Microsoft Forms 2.0 SpinButton	Microsoft.Vbe.Interop.Forms.SpinButton
Microsoft Forms 2.0 TabStrip	Microsoft.Vbe.Interop.Forms.TabStrip
Microsoft Forms 2.0 TextBox	Microsoft.Office.Interop.Outlook.OlkTextBox
Microsoft Forms 2.0 ToggleButton	Microsoft.Vbe.Interop.Forms.ToggleButton

Continues

TABLE 16-3: Mapping Between Outlook Controls and .NET Types (Continued)

Name	Type
Microsoft Office Outlook Business Card Control	Microsoft.Office.Interop.Outlook.OlkBusinessCardControl
Microsoft Office Outlook Category Control	Microsoft.Office.Interop.Outlook.OlkCategory
Microsoft Office Outlook Check Box Control	Microsoft.Office.Interop.Outlook.OlkCheckBox
Microsoft Office Outlook Combo Box Control	Microsoft.Office.Interop.Outlook.OlkComboBox
Microsoft Office Outlook Command Button Control	Microsoft.Office.Interop.Outlook.OlkCommandButton
Microsoft Office Outlook Contact Photo Control	Microsoft.Office.Interop.Outlook.OlkContactPhoto
Microsoft Office Outlook Date Control	Microsoft.Office.Interop.Outlook.OlkDateControl
Microsoft Office Outlook Frame Header Control	Microsoft.Office.Interop.Outlook.OlkFrameHeader
Microsoft Office Outlook InfoBar Control	Microsoft.Office.Interop.Outlook.OlkInfoBar
Microsoft Office Outlook Label Control	Microsoft.Office.Interop.Outlook.OlkLabel

TABLE 16-3: Mapping Between Outlook Controls and .NET Types (Continued)

Name	Type
Microsoft Office Outlook List Box Control	Microsoft.Office.Interop.Outlook.OlkListBox
Microsoft Office Outlook Option Button Control	Microsoft.Office.Interop.Outlook.OlkOptionButton
Microsoft Office Outlook Page Control	Microsoft.Office.Interop.Outlook.OlkPageControl
Microsoft Office Outlook Recipient Control	Microsoft.Office.Interop.Outlook._DRecipientControl
Microsoft Office Outlook Sender Photo Control	Microsoft.Office.Interop.Outlook.OlkSenderPhoto
Microsoft Office Outlook Text Box Control	Microsoft.Office.Interop.Outlook.OlkTextBox
Microsoft Office Outlook Time Control	Microsoft.Office.Interop.Outlook.OlkTimeControl
Microsoft Office Outlook Time Zone Control	Microsoft.Office.Interop.Outlook.OlkTimeZoneControl
Microsoft Office Outlook View Control	Microsoft.Office.Interop.OutlookViewCtl.ViewCtl
Microsoft Office Outlook Body Control	Microsoft.Office.Interop.Outlook._DDocSiteControl

As you write code against Outlook controls, you will discover that you sometimes need to cast the primary Outlook control types listed in Table 16-3 to either of two different types: `Microsoft.Office.Interop.Outlook.OlkControl` and `Microsoft.Vbe.Interop.Forms.Control`. When you cast to an `OlkControl`, you can set properties to configure Outlook-specific layout and binding options like those that are settable by the Properties dialog box in the Outlook Forms Designer. When you cast to a `Control`, you can set basic positioning properties that are common to all controls. Remember that before writing code to cast to a `Microsoft.Vbe.Interop.Forms.Control`, you must ensure that your project has a reference to the Microsoft Forms 2.0 Object Library.

Listing 16-5 is similar to Listing 16-2. The only difference is that it uses Outlook Forms controls, so some of the code for adding and removing items to the `OlkListBox` is different.

Listing 16-5: Form Region Code for a Simple Subtasks Form Region Based on Outlook Forms

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Resources;
using System.Text;
using Office = Microsoft.Office.Core;
using Outlook = Microsoft.Office.Interop.Outlook;

namespace OutlookAddIn2
{
    public partial class FormRegion2
    {
        Outlook.TaskItem task;
        Outlook.UserProperty subTasks;

        #region Form Region Factory

        [Microsoft.Office.Tools.Outlook.
            FormRegionMessageClass(Microsoft.Office.Tools.Outlook.
                FormRegionMessageClassAttribute.Task)]
        [Microsoft.Office.Tools.Outlook.FormRegionName(
            "OutlookAddIn2.FormRegion2")]
        public partial class FormRegion2Factory
        {
            private void InitializeManifest()
            {
                ResourceManager resources =
                    new ResourceManager(typeof(FormRegion2));
            }
        }
    }
}
```

```
this.Manifest.FormRegionType = Microsoft.Office.
    Tools.Outlook.FormRegionType.Separate;
this.Manifest.Title = resources.GetString("Title");
this.Manifest.FormRegionName = resources.
    GetString("FormRegionName");
this.Manifest.Description = resources.
    GetString("Description");
this.Manifest.ShowInspectorCompose = true;
this.Manifest.ShowInspectorRead = true;
this.Manifest.ShowReadingPane = false;

}

// Occurs before the form region is initialized.
// To prevent the form region from appearing, set e.Cancel to
// true. Use e.OutlookItem to get a reference to the current
// Outlook item.
private void FormRegion2Factory_FormRegionInitializing(
    object sender, Microsoft.Office.Tools.Outlook.
    FormRegionInitializingEventArgs e)
{
}
}

#endregion

// Occurs before the form region is displayed.
// Use this.OutlookItem to get a reference to the current
// Outlook item. Use this.OutlookFormRegion to get a reference
// to the form region.
private void FormRegion2_FormRegionShowing(object sender,
    System.EventArgs e)
{
    this.olkCommandButton1.Click += new
        Microsoft.Office.Interop.Outlook.
        OlkCommandButtonEvents_ClickEventHandler(
            olkCommandButton1_Click);
    this.olkCommandButton2.Click += new
        Microsoft.Office.Interop.Outlook.
        OlkCommandButtonEvents_ClickEventHandler(
            olkCommandButton2_Click);

    task = this.OutlookItem as Outlook.TaskItem;
    if (task != null)
    {
        // Check for custom property SubTasks
        subTasks = task.UserProperties.Find("SubTasks", true);
        if (subTasks == null)
        {
```

```
        subTasks = task.UserProperties.Add("SubTasks",
            Outlook.OlUserPropertyType.olText, false,
            Outlook.OlUserPropertyType.olText);
    }
}

// Convert string
string subTasksString = subTasks.Value.ToString();
if (!String.IsNullOrEmpty(subTasksString))
{
    string[] delimiters = new string[1];
    delimiters[0] = System.Environment.NewLine;
    string[] tasks = subTasksString.Split(delimiters,
        StringSplitOptions.RemoveEmptyEntries);
    for (int i = 0; i < tasks.Length; i++)
    {
        olkListBox1.AddItem(tasks[i], i);
    }
}
}

// Occurs when the form region is closed. Use this.OutlookItem
// to get a reference to the current Outlook item. Use
// this.OutlookFormRegion to get a reference to the form
// region.
private void FormRegion2_FormRegionClosed(object sender,
    System.EventArgs e)
{
    if (subTasks == null || task == null)
        return;

    string oldTasks = subTasks.Value.ToString();
    StringBuilder builder = new StringBuilder();

    for (int i = 0; i < olkListBox1.ListCount; i++)
    {
        string t = olkListBox1.GetItem(i);
        if (!String.IsNullOrEmpty(t))
        {
            builder.AppendLine(t);
        }
    }

    string newTasks = builder.ToString();

    if (!String.IsNullOrEmpty(newTasks) &&
        !String.IsNullOrEmpty(oldTasks))
    {
```

```
        if (newTasks.CompareTo(oldTasks) == 0)
            return; // no changes
    }

    subTasks.Value = newTasks;
    task.Save();
}

// New Button
void olkCommandButton1_Click()
{
    // Just add current time as a subtask for simplicity
    this.olkListBox1.AddItem(
        System.DateTime.Now.ToShortTimeString(),
        this.olkListBox1.ListCount);
}

// Delete button
void olkCommandButton2_Click()
{
    if (this.olkListBox1.ListIndex != -1)
    {
        olkListBox1.RemoveItem(olkListBox1.ListIndex);
    }
}
}
```

Outlook Form Region Programmability

In this section, we examine in more detail the Outlook form region classes that VSTO creates. As with many other VSTO project items, it uses partial classes to display user code (code edited by the developer of the add-in) and associate it with generated code (code generated by Visual Studio) to build the final Outlook form region class.

The VSTO Form Region Class

In the example from the first section of this chapter, right-clicking `FormRegion1.cs` and choosing `View Code` from the context menu shows the user code:

```
partial class FormRegion1
```

You can also see the generated code by expanding `FormRegion1.cs` and double-clicking the file `FormRegion1.Designer.cs`, which is a child of

FormRegion1.cs. Here, you see this line of code, showing that a VSTO form region class derives from `Microsoft.Office.Tools.Outlook.FormRegionControl`:

```
partial class FormRegion1 :  
    Microsoft.Office.Tools.Outlook.FormRegionControl
```

Then, if you look at the definition of `FormRegionControl`, you see that it derives from `System.Windows.Forms.UserControl`. A form region class is primarily a Windows Forms `UserControl` with extensions like the implementation of the `IFormRegion` interface, which is used by VSTO to start and shut down a form region:

```
public class FormRegionControl : UserControl, IFormRegion
```

When you import an .OFS file, the form region class derives from `Microsoft.Office.Tools.Outlook.ImportedFormRegion`.

The Form Region Factory

The form region factory is mostly an internal implementation detail of how VSTO supports Outlook form regions. You can do some advanced things with custom form region factories, such as having your form region classes in a separate assembly from the add-in or having a single factory to handle multiple form region classes. But outside these advanced scenarios, the form region factory does peek through in one significant way: It exposes the `FormRegionInitializing` method, which can be handled in your code to prevent a form region from being displayed for a particular Outlook item based on criteria you set. As you might expect, the Factory object creates an instance of your form region class every time an Outlook item requires a form region to be displayed. If you have an `Adjacent` form region that displays in the reading pane for a list of mail items, for example, each time the selection changes to a different mail item, a new instance of your form region class is created for the current mail item. The factory object is invoked first, and if the `FormRegionInitializing` method doesn't cancel the creation of the form region by the implementation of the method setting the `Cancel` property of the `e` parameter to `true`, a new instance of the form region class is created.

When you import an .OFS file, the form region factory also has a method called `InitializeManifest` in which you can write code to modify settings for the form region, such as the form region type. With Windows Forms-based form regions, you typically modify these form region settings in the Properties window, and no `InitializeManifest` method is in the form region factory.

Another key element of the form region factory class is the `FormRegionMessageClass` attribute, which sets the message classes—both built-in and custom—the form region will be displayed for. Listing 16-6 shows the attributes of a form region factory class associated with three built-in message classes (`Appointment`, `Contact`, and `Note`) and one custom message class that derives from `Task` (`IPM.Task.Foo`). VSTO provides constant strings in the `Microsoft.Office.Tools.Outlook.FormRegionMessageClassAttribute` namespace for each of the built-in Outlook message classes. If you want, you can interchange the constant string. The string `"IPM.Task"`, for example, is equivalent to the `Microsoft.Office.Tools.Outlook.FormRegionMessageClassAttribute.Task` constant.

Listing 16-6: A Form Region Factory Class Associated with Three Built-In Message Classes and One Custom Message Class via `FormRegionMessageClass` Attributes

```
[Microsoft.Office.Tools.Outlook.FormRegionMessageClass(  
    Microsoft.Office.Tools.Outlook.  
    FormRegionMessageClassAttribute.Appointment)]  
[Microsoft.Office.Tools.Outlook.FormRegionMessageClass(  
    Microsoft.Office.Tools.Outlook.  
    FormRegionMessageClassAttribute.Contact)]  
[Microsoft.Office.Tools.Outlook.FormRegionMessageClass(  
    Microsoft.Office.Tools.Outlook.  
    FormRegionMessageClassAttribute.Note)]  
[Microsoft.Office.Tools.Outlook.FormRegionMessageClass(  
    "IPM.Task.Foo")]  
[Microsoft.Office.Tools.Outlook.FormRegionName(  
    "OutlookAddIn2.FormRegion3")]  
public partial class FormRegion3Factory
```

The Manifest Object

Most of the properties that control how a VSTO form region is displayed by Outlook are found in the `FormRegionManifest` object returned by the `Manifest` property of a VSTO form region. Behind the scenes, setting properties on the `FormRegionManifest` object manipulates an XML manifest

that describes the form region. This manifest is provided to Outlook when the add-in loads. You can modify the properties of the `FormRegionManifest` object via the property grid for Windows Forms-based form regions by clicking the form region surface in the designer and then using the Properties window to set properties associated with the Manifest property, as shown in Figure 16-30.

To set properties of the manifest object for a Outlook Forms-based form region, write code in the `InitializeManifest` method of the form region fac-

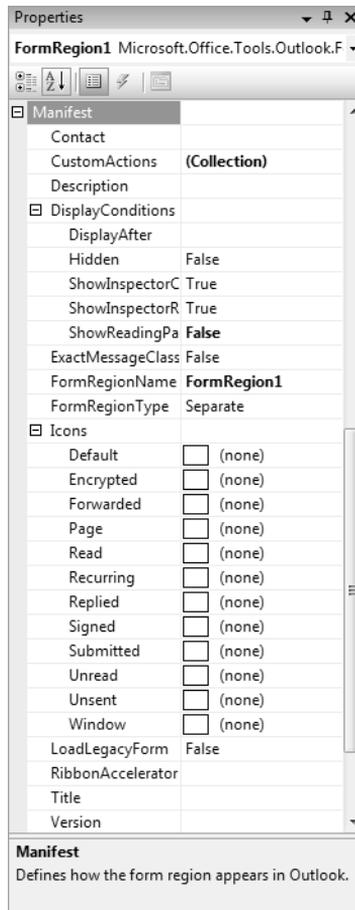


Figure 16-30: Setting properties on the manifest object for a Windows Forms-based form region.

tory. If you try to set properties on the manifest object outside these two mechanisms (the Properties window for Windows Forms and the InitializeManifest method for Outlook Forms), chances are that Outlook will already have asked for the manifest object's settings, and your code will generate an `InvalidOperationException`. You can use the `Locked` property of the manifest object to check whether Outlook has already retrieved the settings from the manifest object. If `Locked` is set to `true`, any code you write against the manifest object will have no effect.

Table 16-4 describes the various properties of the manifest object. The table refers several times to the Choose Form dialog box, which you invoke by dropping down the New button in the Explorer window and choosing Choose Form, as shown in Figure 16-31. The dialog box shown in Figure 16-32 appears, allowing you to pick Replacement or Replace-All form regions (which are associated with custom message classes). This way, an end user can create an Outlook item with a custom message class that you defined and associated with a form region. Table 16-5 describes the icons that can be used by a form region.

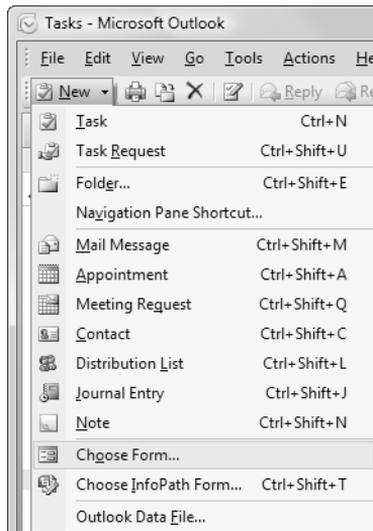


Figure 16-31: Creating a new Outlook Item by using the Choose Form button.

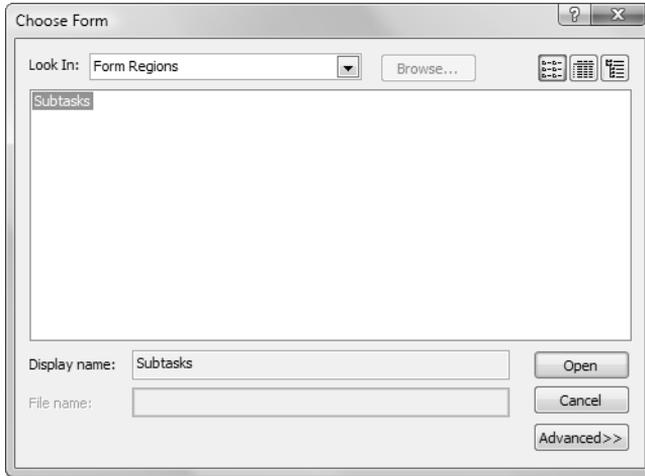


Figure 16-32: The Choose Form dialog box.

TABLE 16-4: Properties of the Manifest Object

Name	Type	What It Does
Contact	string	Gets and sets the name used in the Choose Form dialog box for Replacement and Replace-All form regions.
CustomActions	FormRegion-CustomAction-Collection	A collection with custom actions associated with the form region. The custom actions appear in a Custom Actions group in the Ribbon of the Inspector window showing the form region.
Description	string	Gets and sets the description used in the Choose Form dialog box for Replacement and Replace-All form regions.
DisplayAfter	string	Gets or sets the name of the form region to display before this form region.
ExactMessage-Class	bool	If set to true, this property prevents a custom message class derived from the message class for the form region from displaying the form region.

TABLE 16-4: Properties of the Manifest Object (*Continued*)

Name	Type	What It Does
Form-RegionName	string	Gets and sets the name used for the Ribbon button associated with this form region or the header associated with an Adjoining form region.
FormRegion-Type	FormRegion-Type	Gets and sets an enum specifying the form region type: Adjoining, Replacement, Replace-All, or Separate.
Hidden	bool	If set to true, this form region won't be displayed in the Choose Form dialog box.
Icons	FormRegion-ManifestIcons	Sets the icons used by the form region (see Table 16-5).
Locked	bool	Returns true if Outlook has already queried the manifest object for its settings. Any code you write against the manifest object after Locked is set to true has no effect.
Ribbon-Accelerator	string	Gets and sets the keyboard shortcuts for Separate, Replacement, and Replace-All form regions.
ShowInspector-Compose	bool	Gets and sets whether the form region is shown when an Inspector window is in compose mode.
ShowInspector-Read	bool	Gets and sets whether the form region is shown when an Inspector window is in read mode.
ShowReading-Pane	bool	Gets and sets whether a form region is shown for the reading pane.
Title	string	Gets and sets the name that appears in the Actions menu and the Choose Form dialog box for Replacement and Replace-All form regions.

TABLE 16-5: Icons on the Manifest Object

Name	What It Does	Applies to Form Region Types
Default	16 × 16-pixel icon that is used by default.	Replacement and Replace-All
Encrypted	16 × 16-pixel icon for encrypted items.	Replacement and Replace-All
Forwarded	16 × 16-pixel icon for forwarded items.	Replacement and Replace-All
Page	Icon used in the Ribbon of an Inspector window for the button that activates the form region. Use a PNG file for this icon.	Separate, Replacement, and Replace-All
Read	16 × 16-pixel icon for read items.	Replacement and Replace-All
Recurring	16 × 16-pixel icon for recurring items.	Replacement and Replace-All
Replied	16 × 16-pixel icon for replied-to items.	Replacement and Replace-All
Signed	16 × 16-pixel icon for digitally signed items.	Replacement and Replace-All
Submitted	16 × 16-pixel icon for items in the Outbox that are submitted for sending.	Replacement and Replace-All
Unread	16 × 16-pixel icon for unread items.	Replacement and Replace-All
Unsent	16 × 16-pixel icon for items in the Drafts folder that are not yet sent.	Replacement and Replace-All
Window	Appears in the notification area and in the Alt+Tab window for Inspector windows displaying the form region. Use a 32 × 32-pixel icon.	Replacement and Replace-All

Other Key Properties and Methods

Several other properties and methods associated with a Outlook form region class created with VSTO are worth pointing out. We've already talked about the `OutlookItem` property, which returns as an object the Outlook Item associated with the Outlook form region. You can cast the object returned by the `OutlookItem` property to the type of Outlook item you expect, based on what built-in or custom message classes your form region is associated with.

The `OutlookFormRegion` property returns the underlying Microsoft `.Office.Interop.Outlook.FormRegion` object, which represents your form region in the Outlook object model. Table 16-6 shows some of the key properties and methods on this object.

Globals Support

Whenever you create a form region in a VSTO project, it is added to the `Globals` object for the project. You can access the currently active form

TABLE 16-6: Key Properties and Methods on the `FormRegion` Object Returned by the `OutlookFormRegion` Property

Name	Type	What It Does
<code>Detail</code>	<code>string</code>	Gets and sets the name displayed in the header after the display name of an Adjoining form region.
<code>Inspector</code>	<code>Inspector</code>	Returns the Inspector window object associated with the form region.
<code>IsExpanded</code>	<code>bool</code>	Read-only property that returns <code>true</code> when an Adjoining form region is expanded.
<code>Language</code>	<code>int</code>	Returns the locale ID (LCID) for the current language used by Outlook.
<code>Reflow()</code>		Method that forces Outlook to size an Adjoining form region so that all controls are visible.
<code>Select()</code>		Makes the form region the active form region and forces it to be visible.

regions in three ways: You can see all the active form regions for a particular Inspector window, for a particular Explorer window, or for all open windows. You can access only form regions provided by your add-in; you can't access form regions provided by other add-ins by using the Globals object.

Listing 16-7 shows a subroutine that uses all three ways of accessing active form regions to get the count of active form regions associated with the active Explorer window, the count of active form regions associated with the active Inspector window, and the total count of all active form regions.

Listing 16-7: Three Methods of Accessing Active Form Regions: All Active, for an Explorer Window, and for an Inspector Window

```
private void ShowActiveFormRegions()
{
    Outlook.Explorer explorer =
        Globals.ThisAddIn.Application.ActiveExplorer();
    Outlook.Inspector inspector =
        Globals.ThisAddIn.Application.ActiveInspector();

    System.Windows.Forms.MessageBox.Show(
        String.Format("{0} total form regions",
            Globals.FormRegions.Count.ToString()));

    if (explorer != null)
        System.Windows.Forms.MessageBox.Show(
            String.Format("{0} for regions for the active Explorer",
                Globals.FormRegions[explorer].Count.ToString()));

    if (inspector != null)
        System.Windows.Forms.MessageBox.Show(
            String.Format("{0} for regions for the active Inspector",
                Globals.FormRegions[inspector].Count.ToString()));
}
```

Conclusion

With the new form region feature in Outlook 2007, developers have a powerful new way to customize the Outlook UI. The four types of form regions—Separate, Adjacent, Replacement, and Replace-All—provide a wide variety of UI options. In this chapter, you saw how form regions can be created to customize Inspector windows as well as the reading pane.

You also learned about custom message classes, as they are required for using the Replacement and Replace-All form region types.

VSTO supports two form technologies to create form regions with: Windows Forms and Outlook Forms. You saw how Outlook Forms can be created and imported into Visual Studio and how the controls are accessed from managed code. You also dived deeper into the programming model to discover additional ways to customize form regions.

Also in this chapter you saw a simple way in which form regions integrate with the Ribbon: They automatically add a button to switch to the form region page, for example.

In the next chapter, you see how VSTO supports creating a wider range of Ribbon customizations.



Index

Symbols

- + (tooltips), 119
- % (percent), for Alt key, 177
- : (colon), as Range operator, 220
- [] (index operator)
 - accessing items in collections, 323
 - iterating over Categories collection, 509–510
- ^ (caret), for Ctrl key, 177
- + (plus), for Shift key, 177

A

- A1-style reference format, 220–221, 224
- ActionEventArgs, Smart Tags, 860–862
- Actions property, SmartTag class, 856
- Actions, Smart Tags
 - adding action events, 860–862
 - creating, 856
 - varying number of, 866–869
- Actions task pane
 - adding custom user control to, 684–686
 - architecture of, 680–681
 - arranging, 681–684
 - attaching/detaching, 695–697
 - contextually changing, 686–689
 - detecting orientation of, 689–691
 - properties and methods to avoid, 697
 - scrolling, 691
 - showing/hiding, 691–694
 - working with, 680
 - WPF controls in, 697–699
- Activate method
 - Excel windows, 198
 - Outlook, 433
 - Word Application object, 22
 - Word windows, 338
 - worksheets, 204
- Activation events
 - aggregated chart sheets and chart hosts, 615
 - aggregated Workbook object, 613
 - aggregated Worksheet object, 614
 - Excel, 120–128
 - Outlook, 432–434
 - Word documents, 279–282
- Active/selected object properties
 - Excel, 171–172
 - Outlook, 476
 - Word, 319–320
- ActiveChart property, 182
- ActiveDocument property, Word, 17
- ActiveSheet property, 182
- ActiveWindow property, Word, 352
- ActiveX controls
 - custom task panes hosting, 704
 - Document Actions task pane hosting, 680–681
 - moving to Windows Forms controls, 627–628
 - VBA and, 649
 - Word and Excel hosting, 644–645
 - working with shapes, 215–216
- Adapters
 - data programming, 897–899
 - defined, 892

- Add-ins
 - automation. *See* Automation add-ins
 - as basic patterns of Office solutions, 51–52
 - COM. *See* COM add-ins
 - hosted code and, 52–54
 - Office 2003, 72
 - Outlook 2003 vs. Outlook 2007, 77–78
 - removing, 77
 - types of, 69
 - VSTO. *See* VSTO add-ins
- Add-Ins dialog box, 92, 97
- Add method
 - action pane controls, 681
 - bookmarks, 393
 - columns, 545
 - custom DocumentProperty, 194–195
 - custom task panes, 702, 710
 - folders, 520
 - items, 541–543, 548
 - names, 203
 - tables, 394
 - user control, 704–705
 - Word application object, 65
 - Word documents, 345
 - workbooks, 179–180
 - worksheets or charts to workbooks, 191–192
- Add New Project dialog, 107–108
- Add or Remove Programs
 - Excel documents, 991
 - VSTO add-ins, 976–977
- AddControl method
 - controls added at runtime, 658–659
 - group box control added to Word document, 659–660
 - OLEObject (Excel) and OLEControl (Word) returned by, 663–664
 - Windows Forms controls, 663–664
- AddCustomization method, Server-Document object, 944–946
- AddIn object
 - methods, 619
 - Outlook, 429–432
- AddressEntry, 514–515
- Addresses, Range object, 223–226
- AddressLists collection, NameSpace object, 515–517
- AddressLists property, NameSpace object, 515–517
- AddStore method, Outlook, 509
- AddStoreEx method, Outlook, 509
- Adjoining type, form region
 - customizing, 735–737
 - manifest object properties, 743
 - overview of, 731–732
- Administrative privileges, for registering/unregistering COM interop, 101–102
- ADO.NET
 - data binding, 913–914
 - datasets, 894
 - disconnected programming strategy, 894–895
 - reference material for, 881
- After parameter, worksheets, 191
- Aggregation
 - class hookup and cookies and, 591–594
 - of objects, 578–580
 - obtaining aggregated objects, 580–581
 - overview of, 576–578
 - Windows Forms controls and, 581–582
- Alerts
 - Excel, 165–166
 - Word, 313–314
- Aliases, namespaces and, 64
- Alignment tools, Windows Forms editor, 641
- AllowEdit Ranges collection, 208, 211
- Alt modifier key, 176
- Alt+Ctrl+Shift+S (Document Actions task pane), 674
- AND operator, 539
- Application object, Excel, 163–178
 - activation/deactivation events, 122–123
 - active/selected object properties, 171–172
 - automation executable creating, 55–57
 - built-in functions, 173
 - calculate events, 135
 - cell ranges, 174
 - change events, 139
 - classes, 45, 47
 - close events, 148
 - collections returned by properties, 172
 - delegates, 44, 47
 - dialog boxes and alerts, 165–166
 - double-click and right-click events, 129
 - editing and, 168–169
 - e-mailing workbooks, 175

- EnableEvents property, 167–168
 - events and, 115
 - events raised by, 29–30
 - file and printer settings, 170–171
 - hyperlink events, 142
 - interfaces, 44, 46
 - keyboard commands sent to Excel, 176–178
 - look and feel properties, 169–170
 - mouse pointer, 166
 - new functions in Excel 2007, 173
 - new workbook and worksheet events, 116–120
 - open events, 150
 - overview of, 163
 - print events, 149
 - Quit method, 176
 - Range property, 20
 - save events, 150
 - screen updating behavior, 164–165
 - selection change events, 145
 - spell checking, 174–175
 - status bar messages, 166–167
 - Undo method, 176
 - Union and Intersection methods, 226
 - window resizing events, 146–147
 - Windows property, 195
 - workbook calculation settings, 172–173
- Application object, Outlook, 475–486
- active/selected object properties, 476
 - application-level events, 427–428
 - copying files into Outlook folders, 485–486
 - ItemSend events, 74–76
 - list of events, 426–427
 - overview of, 475
 - properties returning collections, 477–479
 - Quit method, 486
 - search methods, 479–485
- Application object, Word, 311–332
- accessing items in collections, 323–342
 - activation/deactivation events, 279–282
 - active/selected object properties, 319–320
 - adding methods to, 65
 - close events, 276–278
 - CommandBar events, 303–306
 - declaring instances of, 64–65
 - dialogs and alerts, 313–314
 - document navigation, 324–326
 - event interfaces, 269
 - file dialogs, 328–329
 - File Save format options, 328
 - grammar/spell checking, 330–331
 - mail merge events, 290–294
 - mouse events, 283–285
 - mouse pointer appearance, 314–315
 - new and open events, 273–275
 - objects associated with, 263
 - Options property, 326–327
 - overview of, 311
 - print events, 279–282
 - properties, methods, and events, 14–15
 - properties that return collections, 320–323
 - Quit method, 67, 332
 - reasons for multiple event interfaces, 267–269
 - save events, 278–279
 - screen updating behavior, 312–313
 - selection events, 285–287
 - startup and shutdown events, 271
 - status bar or windows caption messages, 315–316
 - Templates property, 341
 - user information, 330
 - user interface properties, 316–319
 - Visible property, 67
 - what it does, 9
 - window sizing events, 287
 - Word object model and, 262
 - XML events, 287
- Application property, AddIn object, 619–620
- Application windows, in custom task panes, 710
- Apply method
- PropertyPage interface, 411
 - View object, 529
- Appointment class, message classes, 730
- Architecture
- actions task pane, 680–681
 - custom task panes, 704–705
 - Windows Forms hosting, 643
- Areas of cells, Range object, 227–228
- Arguments
- aggregated content control objects, 612–613
 - aggregated Document object, 602–603

- Arrange method
 - Excel windows, 197–198
 - Word windows, 339–341
 - as operator, for item type, 551–552
 - ASP.NET, 931–934
 - .aspx Web form, 934–936
 - Attachment events, Outlook, 466–467
 - AttachmentAdd event, Outlook, 466
 - AttachmentContextMenuDisplay event, Outlook, 442
 - AttachmentRead event, Outlook, 466
 - AttachmentRemove event, Outlook, 466
 - Authentication, in Word, 366
 - AutoCorrect dialog box
 - displaying Smart Tags, 96, 247–248
 - managing Smart Tags in Excel, 854
 - managing Smart Tags in Word, 853–854
 - Smart Tags page of, 874
 - Automation add-ins
 - creating for user-defined functions, 99–101
 - debugging, 107–108
 - deploying, 107–108
 - loading, 101–105
 - overview of, 70, 92
 - reflection to work around locale issue, 237–238
 - types of add-ins, 70
 - Automation executables
 - basic patterns of Office solutions, 51–52
 - console application for automating already-running application, 57–59
 - console application for automating Word, 59–69
 - Excel programming with, 90
 - locale issue and, 235–236
 - Outlook programming with, 405–406
 - overview of, 54–57
 - Word programming with, 243
 - AutoSave, Word documents, 278
 - AutoShape, Excel shapes, 215
- B**
- Back-end data source, 891
 - Basic function, Word, 335
 - BDC (Business Data Catalog), 4
 - parameter, worksheets, 191
 - BeforeAttachmentAdd event, Outlook, 466
 - BeforeAttachmentSave event, Outlook, 467
 - BeforeAttachmentWriteToTempFile event, Outlook, 467
 - BeforeAutoSave event, Outlook, 459–461
 - BeforeCaptionShow event, Smart Tags, 861
 - BeforeDelete event, Outlook, 449
 - BeforeFolderMove event, Outlook, 439
 - BeforeFolderSwitch events, Outlook, 437
 - BeforeItemCopy event, Outlook, 453
 - BeforeItemCut event, Outlook, 453
 - BeforeItemMove event, Outlook, 449
 - BeforeItemPaste event, Outlook, 453
 - BeforeMaximize events, Outlook, 434–435
 - BeforeMinimize events, Outlook, 435
 - BeforeMove events, Outlook, 435
 - BeforeSzie events, Outlook, 435
 - BeforeViewSwitch events, Outlook, 436
 - Binding manager, 914
 - Binding sources
 - defined, 892
 - as proxies, 900
 - Blurry quality, in Windows Forms controls, 653–654
 - Bookmarks
 - aggregated Bookmark object, 605–607
 - data-bound Word documents and, 889
 - working with, 392–394
 - Box control, Ribbon
 - child controls and, 809
 - overview of, 781–782
 - Boxing, C#, 12–13
 - Breaks, inserting nonprinting, 385–387
 - BreakSideBySide method, Word windows, 340
 - Browser property
 - document navigation, 324–326
 - range navigation, 380–382
 - wrapper, 35
 - Building Block Gallery content control, 397
 - Built-in functions, Excel Application object, 173
 - Built-in images, Ribbon, 847
 - Built-in properties, items, 561–565
 - Built-In Tab, adding commands to, 845–846
 - BuiltinDocumentProperties property
 - Excel, 183
 - Word, 357–358

Business applications, Office and, 3–4
Business Data Catalog (BDC), 4

Button control, Ribbon
overview of, 782–784
properties, 785
reordering buttons, 807

Button Group control, Ribbon
adding to Excel workbook project,
816–817
child controls and, 809
overview of, 784

ButtonClick events, Gallery control,
830–831

Buttons
adding to Explorer window, 495–496
adding to Inspector window, 505–506

C

C#
boxing, 12–13
code behind and, 92–93, 245
converting parameterized properties to
methods, 12
creating Excel workbooks, 80
debugging library project, 107–108
exception handling support, 312
syntax of, 13
interoperability with VSTO, 582–583
.NET Framework and, 4–5
not supporting parameterized properties,
18–19
try, catch, and finally blocks, 165
verbosity of Word in, 25
writing Smart Tags, 96, 248

C++ classes, exposed as COM objects,
14

CA (Certificate Authority)
obtaining publisher certificate from,
1000
overview of, 999

Cache Viewer, client-side ServerDocu-
ment utility, 939–941

Cached data, 52

Cached data object
CachedDataHostItem collection, 950
CachedDataHostItem object, 950–951
CachedDataItem object, 951–952
overview of, 948–949

CachedData property, ServerDocument
object, 948–949

CachedDataHostItem, ServerDocument
object, 948, 950

CachedDataHostItemCollection, Server-
Document object, 948, 950

Caching data, in XML data island
accessing XML data island, 931
adding/removing members, 910–911
custom data types, 909–910
ICachedType interface, 911–913
overview of, 908–909

Calculate events, Excel, 135–139

Calculation settings, workbooks, 172–173

Callback methods, connecting to Excel
Application object, 31–32

CapsLock property, Word, 15–16

Caption property, Word windows,
315–316

Case sensitivity, Smart Tags, 857

catch blocks, C#, 165

catch blocks, C#, 167

Categories property, Namespace object,
509–510

CDO (Collaboration Data Objects), 405

CDs

deploying Excel documents, 995

deploying to, 1009–1010

deploying VSTO add-ins, 981

Cell method, Table object, 65–66, 395

Cells

formatting, 234

naming, 186–188

populating tables, 395

Range object and, 219–222, 228

selecting/activating, 174

Windows Forms controls insertion
behavior, 640–641

worksheet properties, 220

Certificate Authority (CA)

obtaining publisher certificate from,
1000

overview of, 999

Certificates, ClickOnce

obtaining publisher certificate,
999–1000

overview of, 995–996

security checks and, 1006

signing deployment manifest with pub-
lisher certificate, 1000–1002

test certificate (self-cert), 996–998

trusting publisher certificates, 1002

- Change events
 - CommandBar.ComboBox, 156
 - Excel, 139–141
 - Outlook, 448–452
- Characters collection, Range objects
 - returned by, 355–356
- Characters, inserting nonprinting, 385–387
- Chart collections, 189–192
- Charts
 - activation/deactivation events, 122–123
 - adding to workbooks, 191–192
 - aggregated form, 615
 - calculate events, 136
 - double-click and right-click events, 129–130
 - selection change events, 145
 - Sheets collection and, 109–110
 - working with ChartObject, 216–217
- Charts property, 182
- Check Box control, Ribbon, 784–786
- CheckSpelling method
 - Excel, 174–175
 - Word, 27–28
- Child controls, Ribbon
 - adding to controls, 805–807
 - Collection Editor method for adding, 810
 - cutting, copying, pasting, 808–809
- Choose Form dialog box, form regions, 770
- Clean Solution, removing add-ins, 77
- Clear method
 - CachedData object, 949
 - clearing ranges, 231
 - detaching actions pane, 695
- ClearData method, CachedData object, 949
- ClearSearch method, Explorer object, 500–501
- Click events
 - CommandBarButton, 33–34
 - Gallery control, 830
- ClickOnce
 - CD or USB key, deploying to, 1009–1010
 - certificates, 995–998
 - deploying Excel documents, 982
 - deploying VSTO add-ins, 958–959
 - IIS, deploying to non-IIS websites, 1009
 - inclusion list for trust decisions, 1004–1005
 - Install Settings, 961–963
 - installing add-ins, 966–974
 - installing Excel documents, 986–991
 - license to code, 998–999
 - Mage tool for editing manifests, 1010–1014
 - Office Trusted Locations, 1004
 - overview of, 955
 - post install phase, for add-ins, 974–978
 - post install phase, for Excel documents, 991–992
 - post publishing phase, 965–966
 - prerequisites for VSTO solutions, 956–958
 - Publish Location settings, 959–960
 - Publish Version settings, 963
 - publisher certificates, 999–1002
 - publishing add-ins, 963–965
 - publishing Excel documents, 982–986
 - registering vs. full install, 969–970
 - security, 995
 - security checks, 1005–1009
 - SharePoint, deploying to, 1010
 - trust prompting, 1002–1004
 - updating add-ins, 978–980
 - updating Excel documents, 992–994
 - variations on add-in deployment, 980–982
 - variations on Excel document deployment, 992–994
 - Windows Installer, custom setups, 1010
- Client-side ServerDocument utility, 939–941
- Close events
 - Excel, 148
 - form regions, 448
 - Outlook, 435–436, 456–459
 - Word, 276–278
- Close method
 - Item object, 560–561
 - Outlook, 457
 - ServerDocument object, 943–944
 - Word documents, 347–348, 350–352
 - workbooks, 181, 185
- Code behind
 - as basic pattern of Office solutions, 51–52
 - custom property pages and, 411

- for Document object's New event, 275
 - documents, 78–85
 - Excel programming with, 92–93
 - Excel workbook Ribbon project, 818
 - form regions, 738
 - hosted code and, 52–54
 - Ribbon Gallery control, 828
 - Windows Forms controls, 641–643
 - Word programming with, 245
- Code, hosted. *See* Hosted code
- Collaboration Data Objects (CDO), 405
- Collapse method, Range object, 382
- Collection Editor, Ribbon designer, 810–811
- Collections, 11–13. *See also* by individual type
- Color Categories dialog box, items, 566
- Columns
- controls creating when dragging, 887
 - Count property, 229
 - working with, 229–230
 - worksheet properties, 221
- Columns collection, Table object, 545
- Columns property, Table object, 545
- COM
- adding COM references, 42–44
 - Document Actions task pane and, 678
 - interfaces for unmanaged code, 39–40
 - Word components, 243
 - Word's COM type library, 26
- COM add-ins
- Excel, 90–91
 - locale issue and, 235–236
 - Outlook, 406–407
 - reflection to work around locale issue, 237–238
 - types of add-ins, 69–70
 - Word, 244
- COM Add-Ins dialog box, 90–92, 244, 406–407
- COM interop
- C# and Visual Basic objects masquerading as COM objects, 89
 - customizing Word and, 243
 - Office/.NET communication, 39
 - Outlook and, 405
 - overview of, 25–26
 - registering/unregistering for, 99, 101–102
- COM objects, C++ classes exposed as, 14
- Combo Box control, Ribbon
- Collection Editor and, 810
 - overview of, 786–787
- ComboBox change event, CommandBar object, 156
- ComboBox content control, Word, 397
- , (comma), for union operator, 220
- CommandBar object
- adding buttons/menus to Explorer window, 495–496
 - adding buttons/menus to Inspector window, 505–506
 - events, 156–158
 - helper method, 443
 - hierarchy of, 153–154
 - Ribbon compared with, 35
 - uses of, 154–156
 - Word events and, 303–306
- CommandBarButton
- Click events and, 33–34, 156, 443
 - Ribbon compared with, 35
- CompareSideBySideWith method
- Excel windows, 197
 - Word windows, 340
- Complex data binding
- Excel and, 905–908, 915
 - overview of, 900
- Concurrency manager, 914
- Console applications
- automation of Excel via console application, 57–59
 - automation of Word via console application, 59–69
 - handling new workbook and worksheet events, 117–119
- Constructors, ServerDocument object, 942–943
- Contact class, message classes, 730
- Content controls, Word, 396–403
- aggregated form, 609–613
 - events, 297–303
 - properties and methods, 402–403
 - Windows Forms controls compared with, 652
 - working with, 396–403
 - working with programmatically, 400–401
- Content property, Range object, 370–371
- ContentControl object, 402

- ContentControls property, Document and Range objects, 400
- Context
 - Document Actions task pane, 686–689
 - hosted code, 53
- Context menus
 - customizing with CommandBar, 303
 - Outlook, 442–447
- ContextMenuClose event, Outlook, 443
- Contextual tabs, Fluent UI, 778
- Control state, in Windows Forms controls, 650–653
- Controls
 - ActiveX. *See* ActiveX controls
 - added at runtime are not saved, 664–665
 - content. *See* Content controls, Word
 - data-bindable, 892, 900–901
 - data-bound controls for spreadsheets, 886–889
 - dynamic. *See* Dynamic controls
 - host. *See* Host controls
 - hosted code running in response to, 54
 - Office controls compared with Windows Forms controls, 581–582
 - user, 410–413, 684–686
 - Windows Forms controls. *See* Windows Forms controls
- Controls collection
 - adding controls at runtime, 658–659, 661–663
 - aggregated Document object, 604
 - controls typed as objects not controls, 666
 - deleting controls at runtime, 664
 - enumerating/searching, 585–586
 - Excel methods for adding Windows Forms controls, 662–663
 - overview of, 585
 - why controls added at runtime are not saved, 664–665
 - Windows Forms controls, 661–663, 681
 - Word and Excel host controls added dynamically, 586–590
- Controls group, Ribbon, 297
- Controls, Ribbon
 - Box control, 781–782
 - Button control, 782–784
 - Button Group control, 784
 - Check Box control, 784–786
 - Combo Box control, 786–787
 - cutting, copying, pasting controls, 808–809
 - Drop Down control, 787–789
 - Edit Box control, 789–790
 - Gallery control, 790–792
 - Group control, 792–793
 - Label control, 793
 - Menu control, 793–795
 - Office Button control, 799
 - properties, 782
 - properties that cannot be changed at runtime, 839
 - reordering controls, 807–808
 - Ribbon control, 795
 - RibbonControl object, 781
 - selecting controls, 806
 - Separator control, 795–796
 - Split Button control, 796–797
 - Tab control, 798
 - Toggle Button control, 798–799
- Controls toolbox
 - adding tools to, 751
 - Windows Forms controls not in, 636–638
- ConvertToTable method, Range object, 66
- Cookies, hooking up aggregated objects to classes, 591–594
- Copy event, Outlook, 453–455
- Copy method
 - for items, 558
 - for ranges, 231
 - for sheets, 192
 - for worksheets, 204
- Copying Ribbon controls, 808–809
- CopyTo method, Folder object, 534
- Count property
 - cells, 228
 - collections, 11
 - columns, 229
 - worksheets, 191
- Create New Folder dialog box, 524
- CreateAndConnect helper method, 666–668
- CreateItem method, Outlook Application object, 543–544, 548
- Creating add-in project, 825–836
- Creating automation add-in, 99–101
- Creating custom property pages, 410–413
- Creating custom tabs, 802, 826

- Creating form regions
 - design-time tools for, 751
 - options for, 724–725
 - Outlook Forms Designer for, 749
 - overview of, 723
- Creating items, 548–551
- Creating objects, 8, 55–57
- Creating ranges, 226–227
- Creating windows, 186–188, 338
- Creating Word documents, 345
- Creating workbooks, 80, 82, 179–180
- Ctrl modifier key, 176
- Ctrl+F5, for running projects, 255
- CurrentFolder property, Explorer object, 489–490
- CurrentRegion property, Excel Range object, 230
- CurrentUser property, Namespace object, 514–515
- CurrentView property
 - Folder object, 529
 - Outlook, 490–491
- Cursor property
 - mouse pointer, 166
 - System object, 314
- Custom action events, Outlook, 467–470
- Custom class, for document-level Smart Tags, 869–873
- Custom message classes, 730
- Custom properties
 - items, 561–565
 - Outlook, 410–419
 - worksheets, 206–208
- Custom task panes
 - adding custom user control to, 708–710
 - application types, 708
 - application windows and, 710
 - architecture of, 704–705
 - CustomTaskPane object, 706–708
 - CustomTaskPanes collection, 705–706
 - Excel-specific issues, 717–719
 - Outlook-specific issues, 711–714
 - overview of, 701–703
 - Word-specific issues, 714–717
 - working with, 704
 - WPF controls in, 719–721
- Custom user control, 708–710
- Custom View Organizer dialog box, 529
- CustomDocumentProperties property, Workbook object, 184

- Customizing Excel, 89
- Customizing Word, 243
- CustomPropertyChange event, Outlook, 455–456
- CustomTaskPane object, 706–708
- CustomTaskPanes collection
 - Add method, 702
 - AddIn object, 620
 - Excel example, 705–706
 - overview of, 705–706
 - working with, 704
- Cut event, Outlook, 453–455
- Cutting, copying, pasting, Ribbon controls, 808–809

D

- DASL queries, 540
- Data binding
 - ADO.NET, 913–914
 - bindable controls, 892, 900–901
 - bindable host items and host controls, 914–915
 - complex and simple, 905–908
 - components, 891–892
 - creating data-bound Excel documents, 901–905, 929–930
 - creating data-bound spreadsheets, 882
 - creating data-bound Word documents, 889–891
 - overview of, 881
 - for spreadsheets, 886–889
 - VSTO add-ins and, 921–928
 - for Word documents, 908
- Data programming, 881
 - adapters, 897–899
 - ADO.NET, 913–914
 - binding sources as proxies, 900
 - cached members, adding/removing from data island, 910–911
 - caching custom data types, 909–910
 - caching data in XML data island, 908–909
 - complex and simple data binding, 905–908
 - components needed for data binding, 891–892
 - creating data-bound Excel document, 901–905
 - creating data-bound spreadsheets, 882

- Data programming, *continued*
 - creating data-bound Word documents, 889–891
 - data-bindable controls, 900–901
 - data-bindable host items and host controls, 914–915
 - data binding and dynamic controls from add-ins, 921–928
 - data binding in Word documents, 908
 - data-bound controls for spreadsheets, 886–889
 - data source, defining, 882–886
 - data source security, 892–894
 - datasets, 892–894
 - disconnected strategy, 894–895
 - ICachedType interface in cached data control, 911–913
 - ListObject host control events, 918–921
 - ListObject host control exceptions, 921
 - ListObject host control properties and methods, 915–918
 - overview of, 881–882
 - typed/untyped datasets, 895–897
- Data, separating from view
 - benefits of, 576
 - MVC pattern and, 575
 - VSTO programming and, 574–576
- Data Source Configuration Wizard, 882–886
- Data sources
 - back-end, 891
 - binding sources as proxies, 900
 - data-bound spreadsheets and, 882–886
 - data-bound Word documents and, 890
 - security of, 892–894
- Datasets
 - choosing database objects, 886
 - defined, 892
 - disconnected strategy and, 894–895
 - overview of, 894
 - tree view, 887
 - typed/untyped, 895
- DataTable object, binding to ListObject control to, 83–85
- Date Picker content control, 397
- Dates
 - converting to DateTime format, 239–241
 - DateTime object for, 236–237
- DateTime object
 - converting Excel dates to, 239–241
 - overview of, 236–237
- DateTimePicker control
 - on Document Actions task pane, 632
 - on Document surface, 630–631
 - in Word documents, 651–653
- Deactivate method, Outlook, 433
- Deactivation events
 - Excel, 120–128
 - Word, 279–282
- Deactivation events, Outlook, 432–434
- Debugging
 - Excel user-defined functions, 107–108
 - setting up debugger, 67
- DefaultFilePath property, Excel Application object, 170
- DefaultSaveFormat property, Excel Application object, 170
- Define Names dialog box, 203
- Delegates
 - aggregated Document object, 602
 - Excel Application object, 44, 47
- Delete events, Outlook items, 448–452
- Delete methods
 - controls at runtime, 664
 - folders, 522
 - items, 558–560
 - names, 203
 - ranges, 232
 - worksheets, 204
- Deleted Items folder, 558
- Deploying automation add-ins, Excel, 107–108
- Deploying Excel documents
 - installing documents, 986–991
 - overview of, 982
 - post install, 991–992
 - publishing to Web server, 982–986
 - updating documents, 992–994
 - variations on document deployment, 992–994
- Deploying to CD or USB key, 981, 995, 1009–1010
- Deploying to non-IIS websites, 1009
- Deploying to SharePoint server, 995, 1010
- Deploying VSTO add-ins
 - Install Settings, 961–963
 - installing add-ins, 966–974
 - overview of, 958–982
 - post install, 974–978
 - post publishing, 965–966
 - Publish Location settings, 959–960

- Publish Version settings, 963
- publishing add-ins, 963–965
- updating add-ins, 978–980
- variations on add-in deployment, 980–982
- Deployment manifests
 - editing, 1010–1014
 - security checks and, 1005–1006
- Description property, Folder object, 519
- Design Form dialog box, Forms Designer, 750
- Designer view, Visual Studio 2008, 80–81
- Details pages, Inspector windows, 726–727
- Developers
 - benefits of .NET Framework to, 4–5
 - code behind and, 78
 - Office programming and, 4
- Dialog boxes
 - controlling in Excel, 165–166
 - Windows Forms controls as, 631–634
- Dialog object, Word, 332–337
 - dialogs and alerts, 313–314
 - File dialogs, 328–329
 - getting/setting fields, 334–337
 - overview of, 332
 - showing dialog box and executing actions, 332–333
 - showing dialog box and preventing execution of actions, 333–334
 - tabbed dialog boxes, 333
- Directories, for installed add-ins, 977
- Disconnected strategy, ADO.NET, 894–895
- Discovery, of hosted code, 53
- Display method
 - dialog boxes, 333–334
 - folders, 534
 - items, 560
- Display options, form regions, 733
- DLLs
 - add-ins and, 52
 - Smart Tags, 94–95, 247
 - XLL, 98–99
- DockPositionChanged event, Custom-TaskPane object, 706
- Document Actions task pane, 673
 - architecture of, 680–681
 - attaching/detaching, 695–697
 - contextually changing, 686–689
 - custom user control added to, 684–686
 - detecting orientation of, 689–691
 - methods/properties to avoid, 697
 - overview of, 673–680
 - scrolling, 691
 - showing/hiding, 691–694
 - Windows Forms controls added to, 681–684
 - Windows Forms controls in, 630–631
 - WPF controls in, 697–699
- Document-level Smart Tags
 - action events, 860–862
 - actions, varying number of, 866–869
 - custom class, 869–873
 - issues with Smart Tag properties, 873–887
 - overview of, 855–860
 - regular expressions, 864–866
 - terms, varying number of, 863–864
- Document object, Word, 343–369
 - activation/deactivation events, 279–282
 - adding controls at runtime, 658
 - adding host controls dynamically, 586–590
 - adding Windows Forms controls to document surface, 634–635
 - aggregated form, 599–605
 - changing template attached to a document, 353
 - close events, 276–278
 - closing and saving documents, 350–352
 - code behind controls on documents or spreadsheets, 641–643
 - content control events, 298–299
 - event interfaces, 269
 - grammar/spell checking, 362–363
 - layout of Windows Forms controls on document surface, 641
 - mouse events, 283–285
 - navigating with Word Application object, 324–326
 - new and open events, 273–275
 - objects associated with, 264
 - overview of, 348–349
 - password protection, 367–369
 - preserving dirty state, 349–350
 - print events, 282–283
 - printing documents, 364–365
 - properties, 357–362
 - properties that return collections, 353–357

- Document object, Word, *continued*
 - Range object and, 65
 - read-only protection, 365–367
 - reasons for multiple event interfaces in Word, 267–269
 - removing dynamic controls, 590
 - save events, 278–279
 - saving/reloading dynamic controls, 590–591
 - selection events, 285–287
 - sizing events, 287
 - sync events, 288
 - Undo/Redo methods, 369
 - windows associated documents, 352
 - Word object model and, 262
 - WPF controls in documents, 669–671
 - XML events, 287–288
- Document properties, Workbook object, 183–184
- DocumentProperties collections
 - accessing DocumentProperty object in, 193–194
 - adding custom DocumentProperty to, 194–195
 - iterating over, 192–193
 - overview of, 192
 - working with in Word, 358–362
- DocumentProperty object, 192–195
 - accessing in DocumentProperty collections, 193–194
 - adding custom DocumentProperty to DocumentProperty collection, 194–195
 - overview of, 192
 - working with in Word, 358–362
- Documents collection, Word
 - accessing documents in, 344–345
 - closing all open documents, 347–348
 - creating new documents, 345
 - iterating over open documents, 343
 - opening existing documents, 345–347
 - overview of, 11, 343
 - saving all open documents, 348
- Documents, Excel
 - adding Windows Forms controls to document surface, 634–635
 - code behind controls on documents or spreadsheets, 641–643
 - customizing, 594–596
 - deploying. *See* Deploying Excel documents
 - getting host item object for, 622
 - layout of Windows Forms controls on document surface, 641
 - server-generated, 97
 - smart documents, 93
 - Windows Forms controls and WPF controls on document surface, 629–631
- Documents, Word
 - activation/deactivation events, 279–282
 - adding controls at runtime, 658
 - adding host controls dynamically, 586–590
 - adding Windows Forms controls to, 634–635
 - changing template attached to, 353
 - close events, 276–278
 - closing and saving, 350–352
 - code behind controls on, 641–643
 - content control events, 298–299
 - creating data-bound document, 889–891
 - data binding in, 908
 - event interfaces, 269
 - grammar/spell checking, 362–363
 - layout of Windows Forms controls on, 641
 - mouse events, 283–285
 - navigating, 324–326
 - new and open events, 273–275
 - password protection, 367–369
 - preserving dirty state, 349–350
 - print events, 282–283
 - printing, 364–365
 - read-only protection, 365–367
 - reasons for multiple event interfaces, 267–269
 - removing dynamic controls, 590
 - save events, 278–279
 - saving/reloading dynamic controls, 590–591
 - selection events, 285–287
 - ServerDocument methods, 944–947
 - sizing events, 287
 - sync events, 288
 - Undo/Redo methods, 369
 - windows associated with, 352
 - WPF controls, 669–671
 - XML events, 287–288

- Double-clicking
 - adding controls to document surface, 636
 - Excel events, 128–135
 - Drag and drop
 - adding controls to document surface, 635–636
 - adding List box control, 735
 - controls from Visual Studio toolbox to Ribbon design surface, 802–803
 - rules for dragging in Ribbon, 804
 - Drawing method, adding controls to document surface, 635–636
 - Drop Down control, Ribbon
 - Collection Editor and, 810
 - overview of, 787–789
 - Drop-down menus, Smart Tags and, 851
 - Drop regions, Ribbon controls, 805
 - DropDown List content control, 397
 - Duplicate property, Range object, 379
 - Dynamic controls, 584–585
 - adding, 586–590
 - from application-level add-in, 921–928
 - removing, 590
 - saving/reloading, 590–591
 - Dynamic worksheets, 624–625
 - Dynamism, Ribbon, 838–840
- E**
- E-mail
 - emailing workbooks, 175
 - methods, 175
 - Outlook events, 461–465
 - E-postage events, Word, 288
 - Edit Box control, Ribbon, 789–790
 - Editing Directly in Cell option, Excel, 168–169
 - Editing/editors
 - Collection Editor, 810–811
 - editing in Excel, 168–169
 - editing range values, 231
 - Windows Forms editor, 641
 - Word editor, 501–505, 829
 - ElementHost control
 - adding at runtime, 669
 - adding programmatically at runtime, 638
 - adding WPF controls to custom task pane, 719–721
 - overview of, 628–629
 - using WPF controls in Actions task pane, 697–699
 - Embed Smart Tags option, Word Options dialog box, 874
 - EnableEvents property, Excel Application object, 167–168
 - End property, Word ranges, 373
 - End User License Agreement (EULA), 966–968
 - EndOf method, Range object, 375–376
 - EntireColumn property, worksheets, 221
 - EntireRow property, worksheets, 221
 - Entry points, hosted code, 53
 - EntryID property, NameSpace object, 510–514
 - Enumerations
 - Categories collection, 509–510
 - DocumentProperties collections, 192–193
 - documents in Word, 343
 - items in Outlook, 534–537
 - Names collection in Excel, 202
 - overview of, 13
 - properties returning, 16–17
 - shapes, 215–216
 - stores in Outlook, 508–509
 - subfolders, in Folders collection, 520
 - windows in Word, 196, 338–339
 - workbooks, 178
 - worksheets, 190
 - EULA (End User License Agreement), 966–968
 - Event-driven programming, 583–584
 - Event handlers
 - adding to workbooks and worksheets, 119–120
 - custom handler for server data, 936–939
 - form regions, 738–739
 - generating with Events view, 642
 - Outlook Forms Designer, 756
 - Toggle Button control, 821
 - Visual Studio generating in Word, 270–271
 - Events
 - hosted code running in response to, 53–54
 - method names colliding with event names, 38–39
 - Office object model and, 28–33

Events, *continued*

- Ribbon designer, 803–805
 - VSTO events that are not in Excel, 159–162, 307–310
- Events, Excel**
- activation/deactivation events, 120–128
 - add-in install/uninstall events, 147
 - aggregated NamedRange, XmlMappedRange, and ListObject, 616
 - Application object, 29–30
 - calculate events, 135–139
 - change events, 139–141
 - before close events, 148
 - CustomTaskPane object and, 706
 - double-click and right-click events, 128–135
 - hyperlink events, 141–144
 - list of less used, 158–159
 - new workbook and new worksheet events, 116–120
 - open events, 150–153
 - overview of, 115
 - before print events, 148–149
 - before save events, 149–150
 - selection change events, 145
 - Smart Tags action events, 860–862
 - toolbar and menu events, 153–158
 - VSTO events that are not in Excel, 159–162
 - window resizing events, 146–147
 - XML import/export events, 147
- Events, Outlook**
- activation/deactivation events, 432–434
 - Application level, 427–428
 - attachment events, 466–467
 - close events, 435–436
 - context menu events, 442–447
 - copy/paste/cut/delete events, 453–455
 - custom action events, 467–470
 - e-mail events, 461–465
 - event ordering, 430
 - folder change events, 439–442
 - form region events, 447–448
 - item addition/deletion/change/move events, 448–452
 - item events, 448
 - item load, unload, and BeforeAutosave events, 459–461
 - ItemSend events, 74–76
 - list of less used events, 471–473

- new window events, 434
 - open/read/write/close events, 456–459
 - ordering, 430
 - overview of, 425–427
 - property change events, 455–456
 - startup and quit events, 429–432
 - view and selection change events, 436–439
 - window events, 434–435
- Events, Word, 267–310**
- activation/deactivation events, 279–282
 - aggregated Bookmark object, 606
 - aggregated content control objects, 611
 - aggregated XMLNode and XMLNodes controls, 607
 - close events, 276–278
 - CommandBar events, 303–306
 - content control events, 297–303
 - e-postage events, 288
 - mail merge events, 288–297
 - mouse events, 283–285
 - new and open events, 273–275
 - new events on aggregated Document object, 600–601
 - overview of, 267
 - print events, 282–283
 - reasons for multiple Application and Document event interfaces, 267–269
 - save events, 278–279
 - selection events, 285–287
 - Smart Tags action events, 860–862
 - startup and shutdown events, 271–273
 - sync events, 288
 - Visual Studio generation of event handlers, 270–271
 - VSTO events that are not in Word, 307–310
 - window sizing events, 287
 - XML events, 287–288
- Excel**
- ActionsPane changing based on Selection events, 687–688
 - attaching/detaching actions pane solutions, 696
 - blurry controls in, 653–654
 - complex data binding and, 915
 - console application for automating, 57–59

- custom task panes and, 702, 708, 717–719
 - custom user control added to task pane, 685–686
 - CustomTaskPanes collection, 705–706
 - Document Actions task pane. *See* Document Actions task pane
 - events. *See* Events, Excel
 - hosting ActiveX controls, 644–645
 - Insert Function button (formulas), 103–105
 - list of key objects, 10
 - methods for adding Windows Forms controls, 662–663
 - multiple Ribbons in, 837–838
 - object model, 89, 108–112
 - object model hierarchy, 8
 - objects, 8–9
 - OLEObject properties added to Windows Forms control, 654–656
 - optional parameters, 20
 - OrientationChanged events for Action-Pane, 690–691
 - overloads for helper method, 661–662
 - Ribbon in Excel 2007, 777
 - Ribbon tab IDs and, 814
 - showing/hiding Actions Pane, 692–693
 - VSTO features enabled for Excel 2007, 6
 - Windows Forms compared with Office controls, 581–582
 - Windows Forms controls inserted into, 640–641
- Excel Controls, Visual Studio, 82
- Excel programming
- additional user-defined functions, 106–107
 - automation add-ins, 92
 - automation executables, 90
 - code behind, 92–93
 - creating automation add-in, 99–101
 - customizing Excel, 89
 - debugging user-defined functions, 107–108
 - deploying automation add-ins, 107–108
 - loading automation add-ins, 101–105
 - new objects in Excel 2007, 89–90
 - object model and, 108–112
 - research services, 97–98
 - server-generated documents, 97
 - smart documents, 93
 - Smart Tags, 94–97
 - user-defined functions, 98–99
 - VSTO add-ins, 90–92
 - XLA add-ins, 97
 - XML Expansion Packs, 93–94
- Excel Smart Tags
- adding, 859–860
 - configuring, 853–854
 - creating custom, 869–873
 - overview of, 851–853
 - regular expressions used with, 864–866
 - varying number of actions, 867–869
- Exceptions, properties throwing, 18
- Exchange Client Extensions, 405
- Exists method, bookmarks, 393
- Expand method, Range object, 374–375
- Expanded event, form regions, 448
- Explorer collection, 486–488
- Explorer object, 488–501
- activation/deactivation events, 432–434
 - buttons/menus added to Explorer window, 495–496
 - close events, 436
 - copy/paste/cut/delete events, 453–455
 - Explorer window, 494
 - list of events, 426–427
 - Navigation Pane of Explorer window, 496–499
 - new window events, 434
 - overview of, 488–489
 - as primary Outlook window, 428
 - selected folder/view/items, 489–494
 - setting/clearing active search for Explorer window, 500–501
 - Web views associated with folders, 499–500
 - window events, 434–435
- Explorer window
- buttons/menus added to, 495–496
 - events, 434–435
 - Navigation Pane of, 496–499
 - setting/clearing active search for, 500–501
 - working with, 494
- Export events, XML, 147
- Extensions, to Word/Excel object models
- aggregation and Windows Forms controls, 581–582
 - aggregation, inheritance, and implementation, 576–578

Extensions, to Word/Excel object models,
continued

- C# interoperability and, 582–583
- event model and, 583–584
- hooking up aggregates, 578–580
- obtaining aggregated objects, 580–581
- overview of, 576
- Tag field, 583

F

F5 shortcut, for running Outlook, 735

Fields, Word dialog boxes, 334–337

File converters, 328

File dialogs, 328–329

File Save format options, Word Application object, 328

File settings, Excel Application object,
170–171

FileDialog property, 18–19

Filename, workbooks, 186

FilePrint method, Word, 335

Files, copying into Outlook folders,
485–486

Filter strings, queries and, 538–541

finally blocks

- C#, 165
- displaying messages in Excel status bar,
167

Financial Symbol Smart Tag, 853

Find and Replace dialog, 232–234

Find command, Excel, 631

Find method

- finding text in ranges, 232–234
- Items collection, 537–541
- parameters of, 233

Find property, Range object, 389–391

FindNext method

- finding text in ranges, 232
- Items collection, 538–541

Fluent user interface, 777–778

Folder change events, Outlook, 439–442

Folder object, 519–534

- copying/moving folders, 534
- displaying folders in Explorer view,
534

GetTable method, 545

identifiers for folders, 519–520

item access, 523–529

MAPIFolder object and, 440, 519

overview of, 519

StorageItem objects, 531–534

stores, 520–522

subfolder access, 520–523

view settings, 529–531

Folder Switch events, Outlook, 437

FolderAdd event, Outlook, 439

FolderContextMenuDisplay event, Outlook,
442

FolderPath property, Folder object, 519

FolderRemove event, Outlook, 439

Folders. *See also* Folder object

copying files into, 485–486

copying/moving, 534

CurrentFolder property, 489–490

displaying in Explorer view, 534

EntryID property, 510–514

GetDefaultFolder method, 510

identifiers, 519–520

root folders of Outlook stores, 507–508

Select Folder Dialog box, 518

subfolder access, 520–523

switch events, 437

Web views associated with, 499–500

Folders collection

- iterating over subfolders in, 520
- overview of, 440

Folders property, Folder object, 520

Font object, 388

Font property, Range object, 235, 387–389

foreach iteration

- Categories collection, 509–510
- DocumentProperties collections,
192–193
- documents, 343
- GetEnumerator method and, 178

items, 534–537

Names collection, 202

object model, 13

properties returning, 16–17

shapes, 215–216

sheets, 190

stores, 508–509

subfolders in Folders collection, 520

windows, 196, 338

Form region events, Outlook, 447–448

Form region factory, 766–767

Form regions, 723–775

Adjoining type, 731–732

built-in and custom message classes,
744–748

- Choose Form dialog box, 770
- code behind, 738
- creating with Forms Designer, 749
- creating with Visual Studio 2008, 723
- customizing, 735–743
- display options, 733
- event handlers, 738
- expanding/collapsing, 735–737
- form region factory, 766–767
- FormRegion class, 765–766
- Globals support for, 773–774
- Inspector windows and, 726–727
- key properties and methods, 773
- manifest object, 767–770
- manifest object icons, 772
- manifest object properties, 770–771
- mapping Outlook controls and .NET types, 759–761
- message classes associated with, 733–734
- methods for accessing, 774
- naming, 732
- options for creating, 724–725, 749
- overview of, 723
- programmability of, 765
- Replacement and Replace-All types, 728–731
- running add-in project, 735
- selecting type of form region to create, 724–726
- Separate type, 728–731
- subtasks, 739–743
- types of, 743–744

Format Cells dialog box, 234

Format errors, Excel objects, 238

Formats

- File Save format options, 328
- Range object, 234, 387–389

FormRegion class

- event handlers, 738–739
- Outlook Forms Designer and, 756
- overview of, 765–766

Forward event, Outlook, 463

Forward method, Outlook, 568

FullName property, Word Document object, 349

Functions

- new in Excel 2007, 173
- user-defined. *See* User-defined functions, Excel

G

Gallery control

- Click and ButtonClick events, 830
- code behind, 828
- Collection Editor and, 810–811
- drag and drop to Group control, 826
- editing Buttons collection associated with, 827
- Items collection associated with, 828–829
- overview of, 779, 790–792
- Properties window, 826–827

GenerateItemsMessage helper function, Outlook, 453

get_Address method, for range addresses, 223–226

GetCustomizationVersion method, ServerDocument object, 947

GetDefaultFolder method, NameSpace object, 510–511

get_End method, for ranges, 230

GetEnumerator method, 178, 196, 338. *See also* Enumerations

get_FileDialog method, 328

get_Information method, for ranges, 373–374

GetInspector method, Item object, 560–561

get_Item method

- accessing documents, 344–345
- accessing items, 323
- accessing sheets, 190–191
- accessing windows, 196, 339
- accessing workbooks, 179
- working with cells, 228

GetNextRow() method, Table object, 545–546

get_Offset method, ranges, 226–227

GetPageInfo method, PropertyPage interface, 411

GetProperty method, PropertyAccessor object, 565

get_Range method, worksheets, 219–222

GetSpellingSuggestions method, 330–331

GetStorage method, Folder object, 532

GetTable method, Folder object, 545

get_Value method, ranges, 231

GetVstoObject method, 623, 666

GIF files, 817

- Globals class
 - ActionsPane control and, 676
 - add-in projects, 620
 - VSTO programming and, 598–599
- Globals support, form regions, 773–774
- GoTo method, Range object, 380–382
- Grammar/spell checking
 - Excel Application object, 174–175
 - Word, 362–363
 - Word Application object, 330–331
- Group box control
 - adding to Word document, 659
 - example of a control not in Word/Excel control toolbox, 637
- Group content control, 397
- Group control, Ribbon
 - adding to Excel workbook project, 813
 - drag and drop Gallery control to, 826
 - drag and drop Menu control to, 832
 - modifying, 813–816
 - overview of, 792–793
 - Properties window for, 805
 - reordering, 807–808
- Groups
 - read-only protection and, 365–366
 - Ribbon, 778–779
- GUID, 101
- H**
- HasAxis property, aggregated charts, 615
- Height property, Excel windows, 199
- Helper classes, aggregated Document object, 604
- Helper functions
 - for examining collections, 321–323
 - for modifying Word interface, 318–319
- Helper methods
 - for adding controls at runtime, 659
 - CommandBar object, 443
 - ServerDocument object, 944–947
- Hiding/showing task panes, 691–694
- Hierarchy, object
 - Excel, 109–112
 - Outlook, 419–420
 - Word, 262
- Host controls. *See also* Dynamic controls
 - aggregated Bookmark, 605–607
 - aggregated XMLNode and XMLNodes, 607–609
 - aggregating Word/Excel objects, 578–580
 - data binding, 914–915
 - data related events, 918–921
 - data related exceptions, 921
 - data related properties and methods, 915–918
 - removing dynamic controls, 590
 - saving/reloading dynamic controls, 590–591
 - Tag field, 583
 - Windows Forms compared with, 573
- Host items
 - add-ins, 619
 - aggregated content controls, 610–611
 - aggregated Word/Excel objects, 578–580
 - aggregated workbooks, 613–614
 - aggregated worksheets, 614–615
 - data binding, 914–915
 - extensions to Word/Excel object models, 577–578
 - getting host item control for ListObject object, 623
 - getting host item object for documents, worksheets, or workbooks, 622
 - mapping cookies to unmanaged host items, 593
 - Tag field, 583
 - Windows Forms compared with, 573
- Hosted code. *See also* Add-ins; Code behind
 - overview of, 52–53
 - running after start-up, 53–54
 - unloading, 54
- Hosting architecture, Windows Forms controls. *See* Windows Forms control host
- Hyperlink events, Excel, 141–144
- I**
- IBindableComponent, 914
- ICachedType interface, 911–913
- Icons, manifest object, 772
- IDispatch interface, 780
- IDs, data source security, 893
- ignore.xml file, Smart Tags, 857
- IIS (Internet Information Server)
 - deploying to non-IIS websites, 1009
 - publishing Excel documents to Web server, 983–986
- Images, built-in images for Ribbon, 847

- implementation, VSTO programming, 576–578
- Import events, XML, 147
- Inclusion list
 - for ClickOnce trust decisions, 1004–1005
 - security checks and, 1007
- Index operator ([])
 - accessing items in collections, 323
 - iterating over Categories collection, 509–510
- Index property, Worksheet object, 204
- InfoPath
 - custom task panes and, 702, 708
 - VSTO features enabled for InfoPath 2007, 6
- Information Rights Management (IRM), 366
- inheritance, VSTO programming, 576–578
- Insert Function button, Excel formulas, 103–105
- Insert tab, bookmark, 392
- InsertBreak method, Word, 386
- Inspector object, 501–506
 - activation/deactivation events, 432–434
 - buttons/menus added to Explorer window, 505–506
 - close events, 436
 - custom task panes and, 711–714
 - detail views of Outlook item types, 422
 - events, 426–427
 - Inspector window, 501
 - item associated with, 501
 - new window events, 434
 - overview of, 501
 - as primary Outlook window, 428
 - properties and methods, 502
 - window events, 434–435
 - Word editor, 501–505, 829
- Inspector windows
 - buttons/menus added to, 505–506
 - customizing. *See* Form regions
 - form regions, 726–727
 - Ribbon associated with, 826
 - Task and Details pages, 726–727
 - what they are, 723
 - working with, 501
- Inspectors collection, 486–488
- Install Settings, ClickOnce, 961–963
- Installation Folder URL, ClickOnce, 959
- Installing Excel documents
 - overview of, 986–991
 - post install, 991–992
- Installing PIAs, 40–41
- Installing VSTO add-ins
 - overview of, 966–974
 - post install, 974–978
- IntelliSense, 324
- Interfaces
 - Excel Application object, 44, 46
 - Fluent user interface, 777
 - properties for controlling look and feel of, 169–170
 - reasons for multiple event interfaces in Word, 267–269
 - Word properties controlling user interface, 316–319
- InternalStartup method
 - Excel worksheets, 78
 - VSTO, 74
 - Word, 271
- Internet Information Server (IIS)
 - deploying to non-IIS websites, 1009
 - publishing Excel documents to Web server, 983–986
- Interoperability, C#, 582–583
- Intersection operator, cells, 220
- Invalid type errors, Excel objects, 238
- IRibbonExtensibility, 780
- IRM (Information Rights Management), 366
- is operator, for item type, 552
- IsCacheEnabled method, ServerDocument object, 947
- IsCustomized method, ServerDocument object, 947
- Item method, Names collection, 202
- ItemAdd event, 449
- ItemChange event, 449
- ItemContextMenuDisplay event, 443
- ItemProperties collection, 561–565
- ItemProperties property, 561–565
- ItemRemove event, 448–449
- Items
 - accessing items in a folder, 523–529
 - adding items to collections, 541–544
 - addition/deletion/change/move events, 448–452
 - built-in and custom properties, 561–565
 - comparing `01ItemType` types, 550

Items, *continued*

- copy/paste/cut/delete events, 453–455
- copying/moving, 558
- creating, 548–551
- creating item types, 550–551
- deleting, 558–560
- displaying in Inspector view, 560–561
- identifying item type, 551–556
- Inspector window associated with, 501
- list of, 426–427
- list of properties, 557
- load, unload, and BeforeAutoSave events, 459–461
- mail properties and methods, 566–569
- ObjectClass compared with item types, 553
- open/read/write/close events, 456–459
- overview of, 448
- properties and methods, 547
- property change event, 455–456
- PropertyAccessor, 565
- saving, 566
- showing Color Categories dialog box for, 566
- types of, 420–421
- Items collection, 534–547
 - adding items to collections, 541–544
 - associated with Gallery control, 828–829
 - finding items, 537–541
 - iterating over Outlook items, 534–537
 - overview of, 534–547
 - Table object and, 544–547
- Items property, Folder object, 524
- ItemSend events, 74–76, 462
- Iteration
 - Categories collection, 509–510
 - DocumentProperties collections, 192–193
 - documents in Word, 343
 - items in Outlook, 534–537
 - Names collection in Excel, 202
 - Office, 13
 - properties returning, 16–17
 - shapes, 215–216
 - stores in Outlook, 508–509
 - subfolders, in Folders collection, 520
 - windows in Word, 196, 338–339

- workbooks, 178
- worksheets, 190

J

- Jet query string, 537–541

K

- Key tips, Ribbon controls, 783
- Keyboard commands, sending to Excel, 176–178

L

- Label control, Ribbon
 - adding to Excel workbook project, 816
 - overview of, 793
- Late-bound properties, getting/setting, 334–337
- Layout, Windows Forms controls on document/worksheet surface, 641
- Left property, Windows object, 199
- License to code, ClickOnce, 998–999
- List box control, 735
- ListObject host control
 - data related events, 918–921
 - data related exceptions, 921
 - data related properties and methods, 915–918
- ListObject object
 - binding to DataTable, 83–85
 - getting host item control for, 623
 - host control, 615–619
 - properties for working with tables, 217–218
 - tables and list and, 82
- ListObjects collection, 217–218
- Load behavior, VSTO add-ins, 976
- Load event, Outlook, 459–461
- Loading automation add-ins, Excel, 101–105
- Locale issue, 235–239
 - automation executables and COM add-ins and, 235–236
 - reflection to work around, 237–238
 - switching thread to English and back, 237
 - VSTO and, 238–239
- Locales, Excel, 105, 163
- Logical operators, queries and, 539
- Look and feel properties, Excel Application object, 169–170

M

Macros

- macro recording feature, 4
- security of, 649

Mage tool

- editing deployment manifest with, 1010
- setting friendly name and description, 1013–1014
- setting product name, publisher name, and support URL, 1011–1013
- signing deployment manifest with publisher certificate, 1000–1001

Mail merge

- options for customizing, 289
- Word events, 288–297

Mail Merge Wizard, 289–294

Mail properties, Outlook items, 566–569

Mailing tab, Mail Merge Wizard, 289

MailItem, 543, 566–569

Manifest object

- icons, 772
- overview of, 767–770
- properties, 743, 770–771

MAPIFolder object, 519

MAPILogonComplete event, 429–432

Menu control, Ribbon

- adding child controls to, 805–807
- drag and drop to Group control, 832–833
- ItemsLoading event, 833–836
- overview of, 793–795

Menus

- adding to Explorer window, 495–496
- adding to Inspector window, 505–506
- Excel events, 153–158
- Ribbon replacing, 777

Message classes

- associating with form regions, 733–734
- built-in, 745
- custom, 730, 744–748

Messages

- displaying in Excel status bar, 166–167
- displaying in Word status bar, 315–316

Methods. *See also* by individual types

- common to items, 549
- content control, 402–403
- Document Actions task pane, 697
- e-mail, 175
- form regions, 773
- Inspector object, 502

- method names colliding with event names, 38–39

object base class, 21

Open and Save dialog boxes, 328

with optional parameters, 23–24

with parameters and a return type, 22–23

with parameters and no return type, 22

properties compared with, 21

Shapes collection, 216

SmartTag class, 856

Store object, 521–522

Word add-in object, 60

Microsoft Certificate Server, 1000

Microsoft Office Fluent user interface, 777

Microsoft.Office.Core namespace, 112

Microsoft.Office.Tools.Excel namespace, 855

Microsoft.Office.Tools.Word namespace, 855

Modal dialog box, Windows Forms controls as, 631–634

Model-View-Controller (MVC), 575

Modeless dialog box, Windows Forms controls as, 631–634

Modifier keys, 176

Mouse

- pointer appearance in Excel, 166
- pointer appearance in Word, 314–315
- Word events, 283–285

Move events, Outlook, 448–452

Move method

- Item object, 558
- MailItem and PostItem and, 543
- Range object, 377–379
- sheets, 192
- Word windows, 338
- worksheets, 204

MoveStart method, Range object, 379

MoveTo method, Folder object, 534

MoveUntil method, Range object, 378–379

MoveUntilStart method, Range object, 379

MSI installer

- deploying Excel documents via, 995
- deploying VSTO add-ins via, 981

MVC (Model-View-Controller), 575

“My button stopped working” issue, 33–38

N

- Name object, 203
 - Name property
 - Folder object, 519
 - Word Document object, 348
 - Workbook object, 181
 - NamedRange object, 615–619
 - Names
 - accessing in Names collection, 202–203
 - form regions, 732
 - worksheet object, 206
 - Names collection, 201–203
 - accessing name in, 202–203
 - iterating over, 202
 - Name object and, 203
 - overview of, 202
 - worksheet names, 206
 - Names property, workbooks, 206
 - Namespace object, 506–518
 - adding/removing stores, 509
 - AddressLists property, 515–517
 - checking if Outlook is offline, 510
 - CurrentUser property, 514–515
 - EntryID property, 510–514
 - getting standard folders, 510–511
 - iterating over open stores, 508–509
 - Master Category list, 509–510
 - overview of, 506–507
 - root folders of stores, 507–508
 - Select Folder Dialog box, 518
 - Select Names Dialog box, 518
 - Namespaces
 - aliases and, 64
 - Excel, 855
 - Word, 855
 - Navigation
 - Word documents, 324–326
 - Word ranges, 380–382
 - Navigation Pane, of Explorer window, 496–499
 - .NET Framework, 4–7
 - code behind model for Word, 245
 - developers and, 4–5
 - limitations of Office programming with, 6–7
 - mapping Outlook controls to .NET types, 759–761
 - prerequisites for ClickOnce, 957
 - programming support for Office applications, 5
 - security policy, 955
 - Smart Documents and, 678–680
 - New events
 - aggregated Document object, 602
 - Inspector object, 35–36
 - Outlook windows, 434
 - Word, 273–275
 - Word windows, 338
 - New functions in Excel 2007, 173
 - new keyword
 - creating automation executable, 55–57
 - creating objects and, 8
 - New Project dialog box, Visual Studio 2008, 61
 - New Window method, Word windows, 338
 - NewMail event, Outlook, 461
 - NewMailEx event, Outlook, 462
 - NewWindow method, workbooks, 188
 - Next method, Range object, 378
 - NOT operator, queries and, 539
 - NumberFormat property, in cell range formatting, 234
- O**
- OBAs (Office Business Applications), 4
 - object base class, 21
 - Object model, 581–582
 - C# interoperability and, 582–583
 - collections, 11–13
 - enumerations, 13
 - events, 28–33
 - Excel object model, 8, 89, 108–112
 - list of key objects, 9–10
 - method names colliding with event names, 38–39
 - methods, 21–24
 - “my button stopped working” issue, 33–38
 - new in Office 2007, 8
 - object model hierarchy, 7–8
 - optional properties in Word, 24–28
 - Outlook object model, 419–423
 - parameterized properties, 18–21
 - properties, 15–18, 21
 - Word object model, 261–262
 - Objects, aggregated
 - hooking up, 578–580
 - obtaining, 580–581

- Objects, Excel
 - Application object. *See* Application object, Excel
 - collections, 189–192
 - DateTime for dates, 236–237, 239–241
 - document properties, 192–195
 - extensions to Excel object model, 576–577
 - format and invalid type errors, 238
 - locale issue and, 235–239
 - Names collection and Names object, 201–203
 - new objects in Excel 2007, 89–90
 - object model, 108–112
 - Range object. *See* Range object, Excel
 - Windows collection, 195–199
 - Windows object, 199–201
 - Workbook object. *See* Workbook object
 - Workbooks collection. *See* Workbooks collection
 - Worksheet object. *See* Worksheet object
- Objects, Outlook
 - Application object. *See* Application object, Outlook
 - Explorer and Inspector collections, 486–488
 - Explorer object. *See* Explorer object
 - Folder object. *See* Folder object
 - Inspector object. *See* Inspector object
 - items. *See* Items
 - Items collection. *See* Items collection
 - Namespace object. *See* Namespace object
- Objects, Word
 - Application object. *See* Application object, Word
 - bookmarks, 392–394
 - content controls, 396–403
 - Dialog object. *See* Dialog object, Word
 - Document object. *See* Document object, Word
 - extensions to Word object model, 576–577
 - overview of, 262
 - Range object. *See* Range object, Word
 - Shape object, 266
 - tables, 394–396
 - templates, 341–343
 - VSTO events not in Word, 307–310
 - windows. *See* Windows, Word
- Office 2003
 - add-ins, 72
 - Smart Documents, 678–680
- Office 2007
 - custom task panes in, 701
 - prerequisites for ClickOnce, 956
 - Ribbon model for defining buttons, 35
- Office 2007 installer, 40–41
- Office Business Applications (OBAs), 4
- Office Button control
 - adding child controls to, 805–807
 - features of Fluent UI, 777–778
 - modifying menu, 840–845
 - overview of, 799
- Office Interop API extensions, 7
- Office programming
 - business applications, 3–4
 - collections, 11–13
 - enumerations, 13
 - events, 28–33
 - method names colliding with event names, 38–39
 - methods, 21–24
 - “my button stopped working” issue, 33–35
 - .NET Framework and, 4–7
 - object model, 7–11
 - optional properties in Word, 24–28
 - parameterized properties, 18–21
 - PIAs and, 39–48
 - professional developers and, 4
 - properties, 15–18, 21
 - reasons for using, 3
 - reasons for using VSTO 3.0, 5–6
- Office Ribbon. *See* Ribbon
- Office solutions
 - add-ins. *See* Add-ins
 - automation executables. *See* Automation executables
 - code behind. *See* Code behind
 - patterns of, 51–52
- Office Trusted Locations, 988–989, 1004
- Office.Microsoft.Office.Core, 63
- Offline property, Namespace object, 510
- .OFS file extension, 753–755
- OLEControl
 - AddControl method returning, 663–664
 - properties merged with Windows Forms control, 654–657

- OLEObjects
 - AddControl method returning, 663–664
 - properties merged with Windows Forms control, 654–657
 - working with, 214–215
 - ObjectClass, 553
 - OnShutdown method
 - AddIn object, 619
 - aggregated Document object, 602
 - aggregated Worksheet object, 614
 - OnStartup method
 - AddIn object, 619
 - aggregated Document object, 602
 - aggregated Worksheet object, 614
 - OnUpdate event, CommandBar object, 156
 - Open dialog box, Word, 328
 - Open events
 - aggregated Document object, 602
 - Excel, 150–153
 - Outlook, 456–459
 - Word, 273–275
 - Open method
 - opening existing documents, 345–347
 - opening workbooks, 180–181
 - Visible property, 352
 - Open XML formats, 52
 - Optional parameters
 - Add method, 345
 - for addresses, 224–225
 - Item method, 202
 - methods with, 23–24
 - Open method, 346–347
 - PrintOut method, 364
 - Protect method, 209–210
 - SaveAs method, 351–352
 - in Word, 24–28
 - for worksheets, 191
 - Options dialog box
 - checking installed add-ins, 974
 - configuring Smart Tags in Outlook, 408
 - custom property page added to, 416
 - Excel options, 168–169
 - Smart Tags options, 874
 - Word options, 326–327
 - OR operator, queries and, 539
 - OrientationChanged events, ActionsPane controls, 689–691
 - Outlook
 - add-ins, 406–407, 723–724
 - automation executables, 405–406
 - creating Outlook 2007 add-in, 70–77
 - custom property pages, 410–419
 - custom task panes and, 702, 708, 711–714
 - events. *See* Events, Outlook
 - form regions. *See* Form regions
 - item types, 420–421
 - list of key objects, 10
 - mapping Outlook controls to .NET types, 759–761
 - New Inspector events, 35–36
 - object model, 419–423
 - objects, 8
 - Ribbon in Outlook 2007, 777
 - Smart Tags, 408–410
 - tab IDs, Ribbon for, 814
 - VSTO add-ins, 77–78
 - VSTO features enabled for Outlook 2007, 6
 - ways to customize, 405
 - Outlook Forms Designer
 - accessing/selecting control properties, 752–753
 - adding tools to Controls toolbox, 751
 - compared with Windows Forms, 749
 - creating form regions with, 749
 - Design Form dialog box, 750
 - design-time tools, 751
 - dragging controls to design surface, 751–753
 - event handlers, 756
 - exporting form region to Visual Studio, 753–755
 - form region class, 756
 - Windows Forms-based form region compared with, 756–758, 762–765
- P**
- Page Layout, inserting characters and breaks, 385–387
 - PageChange events, Outlook, 435
 - PageSetup property, printers, 364
 - Paragraphs
 - InsertParagraph method, 386
 - Paragraph object, 11
 - Parameterized indexes, C#, 582–583, 604
 - Parameterized properties
 - Office, 18–21
 - parameterized indexes compared with, 582–583, 604

- Parameters and a return type, methods with, 22–23
- Parameters and no return type, methods with, 22
- partial classes
 - code behind and, 78
 - Globals class and, 599
 - in VSTO, 73–74
- Pascal casing code, in Visual Studio, 32
- Password protection, Word documents, 367–369
- Passwords, data source security, 893
- Paste event, Outlook, 453–455
- Pasting Ribbon controls, 808–809
- Path information, workbooks, 186
- Persistent tagging, generated by Smart Tags, 855
- Persona menu, Smart Tag support in Outlook, 409
- PIAs (Primary Interop Assemblies)
 - adding references to, 62–63
 - browsing, 44–48
 - installing, 40–41
 - list of common, 42
 - as .NET assembly, 14
 - overview of, 39–40
 - Pascal casing and, 32
 - prerequisites for ClickOnce, 957
 - referencing, 41–44
 - wiki documenting, 59
 - Word and Excel assemblies, 576–577
- Picture content control, 397
- Pictures, working with shapes, 215–216
- Plain text content control, 397
- PNG files, 817
- Pop-up menus, Smart Tags and, 94, 247, 851
- Position property, Ribbon, 846–847
- Positioning, Windows object, 199
- PostItem, 543, 566–569
- PowerPoint
 - custom task panes and, 702, 708
 - VSTO features enabled for PowerPoint 2007, 6
- Pragmatic ADO.NET* (Wildermuth), 881
- Primary Interop Assemblies. *See* PIAs (primary interop assemblies)
- Principle of least privilege, data sources, 893
- Print events
 - Excel, 148–149
 - Word documents, 282–283
- Printer settings, Excel, 170–171
- Printing Word documents, 364–365
- Printing workbooks, 188
- PrintOut method
 - Word documents, 364–365
 - workbooks, 188–189
- Privileges, data source security, 893
- Programmability
 - form regions, 765
 - Ribbon, 780
- Programmatic content controls, Word, 400–401
- Programming in Excel. *See* Excel programming
- Programming in Word. *See* Word programming
- Programs and Features, Windows Vista, 991
- Project 2007, 6
- Properties, Excel
 - active/selected object properties, 171–172, 181–182
 - collections returned by, 172
 - CustomTaskPane object, 707
 - look and feel of user interface, 169–170
 - Name object, 203
 - optional, 19–20
 - windows, 200–201
 - workbook collections returned by, 182
 - workbooks, 82
 - for worksheet/document protection, 213–214
 - worksheets, 206–208
- Properties, Office
 - methods compared with, 21
 - object model and, 21
 - overview of, 15–18
 - parameterized, 18–21
 - SmartTag class, 856
- Properties, Outlook
 - active/selected object properties, 476
 - collections returned by, 477–479
 - common to all items, 549
 - Explorer window, 494
 - form regions, 773
 - list of item properties, 557
 - manifest object, 743, 770–771
 - Store object, 521–522

- Properties, Ribbon
 - Button control, 785
 - controls, 782
 - Properties window
 - for adding event handlers, 119–120, 270
 - Ribbon designer, 803–805
 - Properties, Word
 - active/selected object properties, 319–320
 - collections returned by, 320–323, 353–357
 - content control, 402–403
 - font object, 388
 - Inspector object, 502
 - late-bound, 334–337
 - objects returned by, 17
 - Open and Save dialog boxes, 328
 - read-only, 16
 - throwing exceptions, 18
 - Word add-in object, 60
 - Word Document object, 357–362
 - Word user interface, 316–319
 - PropertyAccessor object, Outlook items, 565
 - PropertyAccessor property, Outlook items, 565
 - PropertyChange event, Outlook, 455–456
 - PropertyPage interface, in Outlook, 411
 - Protect Document task pane, 365
 - Protect method
 - optional parameters, 209–210
 - workbooks, 188–189
 - worksheets, 208–209
 - Proxy objects, binding sources as, 900
 - Publish Folder Location text box, ClickOnce, 959
 - Publish Location settings, ClickOnce, 959–960
 - Publish Now button, ClickOnce, 963–965
 - Publish Properties page
 - Install settings, 961–963
 - Publish Location settings, 959–960
 - Publish Now button, 963–965
 - Publish Version settings, 963
 - Publish Version settings, ClickOnce, 963
 - Publisher certificates, ClickOnce, 999–1002
 - obtaining, 999
 - signing deployment manifest with, 1000–1002
 - trusting, 1002
 - Publishing add-ins
 - overview of, 963–965
 - post publishing, 965–966
 - Publishing Excel documents, to Web server, 982–986
- Q**
- Query Web method, 257–260
 - Quick Access toolbar, Fluent UI, 777–778
 - Quit event
 - Outlook, 429–432
 - Word, 272
 - Quit method
 - Excel Application object, 176
 - for exiting Word, 332
 - Outlook Application object, 486
 - Word Application object, 67
- R**
- RIC1-style references, in addressing, 223
 - Radio buttons, 650
 - Range method, 370–371
 - Range object, Excel, 219–235
 - addresses, 223–226
 - aggregated version, 615–619
 - areas, 227–228
 - cells, 228
 - copying/clearing/deleting ranges, 231–232
 - creating ranges, 226–227
 - editing range values, 231
 - finding text in ranges, 232–234
 - formatting range cells, 234
 - Locked property, for worksheet protection, 212–213
 - object hierarchy and, 112–113
 - overview of, 219
 - for particular cell or range of cells, 219–222
 - regions, 230
 - right-click and double-click events and, 128
 - rows and columns, 229–230
 - selecting a range, 230
 - Range object, Word, 369–391
 - changing ranges, 374–377
 - collapsing ranges, 382
 - ConvertToTable method, 66
 - Document object, 65
 - Find/Replace properties, 389–391

- formatting, 387–389
- getting a range, 370–373
- getting text from ranges, 382–384
- grammar/spell checking, 362–363
- identifying ranges, 373–374
- inserting nonprinting characters/
breaks, 385–387
- moving ranges, 377–379
- navigating ranges, 380–382
- object model and, 262
- objects associated with, 265
- overview of, 369–370
- properties that return collections,
353–356
- setting text in ranges, 384–385
- stories and, 380
- Range operator (:), 220
- Range property, Excel Application object,
20
- Ranges, cell
 - formatting, 234
 - naming, 186–188
 - noncontiguous (areas), 227
 - Range object for, 219–222
 - selecting/activating, 174
- RCW (Runtime Callable Wrapper), 35
- Read event, Outlook, 456–459
- Read-only properties, Word, 16
- Read-only protection, Word documents,
365–367
- RecentFiles property, Excel Application
object, 170
- Recipient objects, 514–515
- Recipients collection, 568
- Recognizers
 - Excel, 94–95
 - Smart Tags and, 854
 - Word, 247
- Redo methods, Word documents, 369
- References
 - adding to Word 2007 PIA, 62
 - addresses, 223–224
 - PIAs, 41–44
 - to ServerDocument object, 934
- RefersTo properties, Name object, 203
- Reflection
 - setting properties on Dialog object
with, 336
 - work around for locale issue,
237–238
- Regex class, 864
- Regions, Range object, 230
- Registering ClickOnce, 969–970
- Registering COM interop, 101–102
- Registering Research services, 255–256
- Registration Web method, 255–256
- Registry
 - discovering installed add-ins, 245
 - inclusion list for trust decisions,
1004–1005
 - settings for installed add-ins, 975
- Registry, discovering installed add-ins,
407
- Regular expressions, Smart Tags and,
864–866
- Remove method
 - Controls collection, 664
 - Folders collection, 521–522
- RemoveAt method, Controls collection,
664
- RemoveCustomization method
 - aggregated Document object, 605
 - aggregated Workbook object, 613
 - detaching actions pane and, 695
 - ServerDocument object, 947
- Replace-All type, form region
 - built-in and custom message classes,
744–748
 - manifest object properties, 743
 - overview of, 728–731
- Replace property, Word Range object,
389–391
- Replacement type, form region
 - built-in and custom message classes,
744–748
 - manifest object properties, 743
 - overview of, 728–731
- Reply event, Outlook, 462
- Reply method, Outlook, 569
- ReplyAll event, Outlook, 462
- Research options, in Research task pane,
255
- Research services, Excel, 97–98
- Research services, Word, 249–261
 - getting started with, 250–254
 - overview of, 249–250
 - Query Web method, 257–260
 - registering, 255–256
 - resources for, 261
 - using, 256–261

- Research task pane
 - research options in, 255
 - writing research services in Excel, 97–98
 - writing research services in Word, 249–250
 - ResetSideBySideWith method, Word windows, 340
 - ResizeBegin events, Windows Forms controls, 643
 - ResizeEnd events, Windows Forms controls, 643
 - Restrict method, Items collection, 540
 - Ribbon, 777–850
 - adding commands to Built-In Tab, 845–846
 - advanced topics, 836–837
 - Box control, 781–782
 - built-in images for Ribbon, 847
 - Button control, 782–784
 - Button Group control, 784
 - for buttons and menus, 80–82
 - Check Box control, 784–786
 - Combo Box control, 786–787
 - CommandBar compared with, 303
 - Controls group in Developer tab of, 297
 - coordinating with task panes, 848
 - creating in Excel workbook project. *See* Workbooks, Ribbon in
 - creating in Outlook add-in project, 825–836
 - defining buttons, 35
 - delaying loading of add-in with, 848–850
 - Drop Down control, 787–789
 - dynamism and, 838–840
 - Edit Box control, 789–790
 - in Excel workbook project, 812–825
 - exporting to XML, 848
 - Gallery control, 790–792
 - Group control, 792–793
 - hosted code and, 54
 - Label control, 793
 - limitations of, 780–781
 - Menu control, 793–795
 - modifying Office Button menu, 840–845
 - multiple Ribbons in Word and Excel, 837–838
 - Office Button control, 799
 - overview of, 777–779
 - Position property, 846–847
 - programmability of, 780
 - replacing entire Ribbon, 838
 - Ribbon control, 795
 - RibbonControl object, 781
 - Separator control, 795–796
 - Split Button control, 796–797
 - Tab control, 798
 - Toggle Button control, 798–799
 - Tools group, 751
 - Windows Forms components and, 848
 - Ribbon control, 795
 - Ribbon designer, Visual Studio
 - adding child controls to Office Button, Split Button, and Menu controls, 805–807
 - Collection Editor, 810–811
 - cutting, copying, pasting controls, 808–809
 - overview of, 800–801
 - Properties window, tasks, and events, 803–805
 - reordering controls, 807–808
 - selecting controls, 806
 - toolbox, 802–803
 - as WYSIWYG designer, 809
 - RibbonControl object, 781
 - Rich text content control, 397
 - Rich text control, Word documents, 297
 - Right-click events, Excel, 128–135
 - Root folders, Outlook stores, 507–508
 - Rows
 - Count property, 229
 - populating tables, 395
 - working with, 229–230
 - worksheet properties, 221
 - Runtime Callable Wrapper (RCW), 35
- S**
- Save As dialog, Word documents, 278
 - Save As method
 - Word documents, 351
 - Workbook object, 185
 - Save dialog box, Word, 328
 - Save events
 - Excel, 149–150
 - Word, 278–279
 - Save method
 - Outlook items, 566
 - ServerDocument object, 943–944

- Word documents, 348, 350
- Workbook object, 185
- SaveCopyAs method, Workbook object, 185
- Saved property
 - preserving dirty state of documents, 349–350
 - Workbook object, 184–187
- Screen tips, Ribbon controls, 783
- ScreenRefresh method, Word Application object, 313
- ScreenUpdating property
 - Excel Application object, 164–165
 - Word Application object, 312–313
- Search methods
 - Controls collection, 585–586
 - Explorer object, 500–501
 - Outlook, 479–485
- Security
 - data sources, 892–894
 - macro, 649
 - Outlook, 406
 - Windows Forms controls, 648–649
 - worksheet objects, 208–214
- Security checks, ClickOnce, 1005–1009
- Security, ClickOnce
 - certificates, 995–998
 - checks, 1005–1009
 - inclusion list for trust decisions, 1004–1005
 - license to code, 998–999
 - Office Trusted Locations, 1004
 - overview of, 995
 - publisher certificates, 999–1002
 - trust prompting, 1002–1004
- Security policy, .NET Framework, 955
- Select Folder Dialog box, NameSpace object, 518
- Select method, range, 230
- Select Names Dialog box, NameSpace object, 518
- SelectContentControlsByTag, Word Document object, 400
- Selection change events
 - Excel, 145
 - Outlook, 436–439
- Selection collections, Outlook, 492
- Selection events, Word, 285–287
- Selection property
 - Explorer object, 492
 - Word Range object, 372
- self-cert
 - license to code, 998–999
 - types of certificates, 996–998
- Send event, Outlook, 462
- Send method, Outlook, 568, 569
- SendKeys method, keyboard commands, 176–178
- Sentences collection, Word Range object, 355–356
- Separate type, form region
 - manifest object properties, 743–744
 - overview of, 728–731
- Separator control, Ribbon, 795–796
- Server data, 929–953. *See also* ServerDocument object
 - accessing XML data island, 931
 - ASP.NET and, 931–934
 - client-side ServerDocument utility, 939–941
 - custom handler for, 936–939
 - data-bound VSTO documents, 929–930
 - populating documents with, 929–930
 - setting up server, 934–936
 - XML file formats, 930–931
- Server document pattern, 52
- Server-generated documents
 - Excel programming and, 97
 - Word programming and, 248–249
- ServerDocument class
 - Excel documents on server, 97
 - Word documents on server, 248–249
- ServerDocument object
 - AddCustomization method, 944–946
 - cached data object model, 948–952
 - client-side ServerDocument utility, 939–941
 - constructors, 942–943
 - GetCustomizationVersion method, 947
 - IsCustomized method and IsCacheEnabled method, 947
 - overview of, 941–942
 - reference to, 934
 - RemoveCustomization method, 947
 - saving and closing documents, 943–944
 - static helper methods, 944
- Service, mapping cookies to unmanaged host items, 592–593
- SetColumns method, Items collection, 535–537

- SetProperty method, PropertyAccessor object, 565
- SetRange method, Range object, 379
- set_Style method
 - Word font settings, 387–389
 - Word tables, 395
- set_Value method, Excel Range object, 231
- Shape object
 - methods, 216
 - object hierarchy and, 112, 114
 - objects associated with, 266
 - Word, 262
 - working with, 215–216
- Shapes collection
 - methods, 216
 - working with, 215–216
- Shared add-ins. *See* COM add-ins
- SharePoint server, deploying to, 995, 1010
- Sheets
 - accessing in collections, 190–191
 - activating/deactivating, 122–123
 - copying/moving, 192
 - double-click and right-click events, 129–130
 - iterating over open, 190
- Sheets collection
 - accessing sheets in collections, 190–191
 - copying/moving sheets, 192
 - Iterating over open sheets, 190
 - Workbook object, 109–110
 - working with, 189–192
- Sheets property, 182
- Shift-clicking, for selecting items in a folder, 492
- Shift modifier key, 176
- Shift+Tab, for iterating through Ribbon controls, 807
- ShortcutContextMenuStripDisplay event, Outlook, 442
- Show All Files button, Solution Explorer, 591
- Show method, dialog boxes, 332–333
- ShowCategoriesDialog method, Outlook items, 566–569
- Showing/hiding task panes, 691–694
- Shutdown events
 - Outlook, 429–432
 - Word, 271–273
- Shutdown methods, Excel worksheets, 78
- Shutdown sequences, VSTO programming, 596–598
- Simple data binding, 900, 905–908
- Smart documents
 - Excel programming and, 93
 - Office 2003, 678–680
 - Word programming and, 246–247
- Smart Tags, 851–880
 - action events, 860–862
 - actions, varying number of, 866–869
 - configuring in Word and Excel, 853–854
 - creating application-level, 874–879
 - creating document-level, 855–860
 - custom class, 869–873
 - Excel programming and, 94–97
 - issues with Smart Tag properties, 873–887
 - Outlook programming and, 408–410
 - overview of, 851–853
 - persistent tagging, 855
 - regular expressions, 864–866
 - terms, varying number of, 863–864
 - VSTOSmartTags collection, 604
 - Word programming and, 247–248
- SmartTag class, 855–856
- Solution Explorer
 - adding user controls, 684
 - creating Excel workbooks, 82
 - deploying automation add-ins, 107–108
 - Outlook add-in project in, 71
 - Show All Files button, 591
 - View Code, 82–83, 411, 738
- Sort method, Items collection, 535–537
- Spell checking
 - Excel Application object, 174–175
 - Word, 362–363
 - Word Application object, 330–331
- Split Button control, Ribbon
 - adding child controls to, 805–807
 - overview of, 796–797
- Spreadsheets
 - .aspx Web form for downloading, 934–936
 - code behind on, 641–643
 - custom handler for server data, 936–939
 - inserting functions, 103–105
 - populating with server data, 929, 933
 - Windows Forms controls on, 629–631, 634–635
 - WPF controls on, 629–631

- Spreadsheets, data-bound, 882–889
 - controls, 886–889
 - data source, 882–886
 - overview of, 882
- Start property, Word ranges, 373
- Start-up events, code behind and, 78
- StartOf method, Range object, 377
- Startup
 - custom task panes and, 714–717
 - VSTO programming, 596–598
- Startup events
 - Outlook, 429–432
 - Word, 271–273
- Startup methods, Excel worksheets, 78
- Static helper methods, ServerDocument object, 944–947
- Status bars
 - displaying messages in Excel status bar, 166–167
 - displaying messages in Word status bar, 315–316
- StatusBar property, Excel Application object, 166–167
- StorageItem object, Folder object, 531–534
- Store objects, 520, 521
- Store property, Folder object, 520
- StoreContextMenuDisplay event, Outlook, 442
- StoreID property, Namespace object, 510–514
- Stores
 - adding/removing, 509
 - folder object, 520–522
 - iterating over open, 508–509
 - root folders of, 507–508
 - StoreID property, 510–514
- Stores collections, 508–509
- Stores property, 508–509
- Story concept
 - in Word, 370
 - Word ranges and, 380
- StoryRanges collection, Word Range object, 371, 373
- Styles
 - A1-style reference format, 220–221, 224
 - applying to ranges, 235
 - RIC1-style references, in addressing, 223
 - set_Style method, 387–389, 395

- Subfolder access, Folder object, 520–523
- Subtasks, form regions, 739–743
- Super tips, Ribbon controls, 783
- Sync events, Word, 288
- System object, Cursor property, 314
- System.Data.DataSet, 896
- System.Runtime.InteropServices
 - .Marshal.BindToMoniker, 57
- System.Runtime.InteropServices
 - .Marshal.GetActiveObject, 57

T

- Tab control
 - Properties window and, 803
 - Ribbon, 798
- Tab IDs, Ribbon, 814
- Tab key
 - automating code generation, 119
 - for iterating through Ribbon controls, 807
- TabAddIns tab, 812
- Tabbed dialog boxes, 333
- Table adapter manager, 892, 899
- Table object, Outlook, 544–547
- Table object, Word, 65–66
- Tables collection, Word, 65, 394
- Tables, Excel, 217–218
- Tables, Word
 - format of, 66
 - generating, 65–66
 - working with, 394–396
- Tabs, Ribbon
 - adding commands to Built-In Tab, 845–846
 - creating custom, 826
 - overview of, 778–779
 - when to create custom tab, 802
- Tag field, VSTO programming, 583
- Tags, persistent Smart Tags, 855
- Task class, message classes, 730
- Task pages, Inspector windows, 726–727
- Task panes
 - custom application-level, 676
 - Document Actions task pane. *See* Document Actions Task Pane
 - overview of, 674–675
 - Ribbon coordinated with, 848
 - Styles task pane, 674
- Tasks, Ribbon designer, 803–805

- Templates collection, Word, 341
 - Templates property, Word Application object, 341
 - Templates, Word
 - accessing from a collection, 324
 - changing template attached to a document, 353
 - overview of, 341–343
 - VSTO vs. VBA, 245
 - working with, 341–343
 - Terms collection, Smart Tags, 863–864
 - Terms property, SmartTag class, 856
 - Terms, Smart Tags
 - setting terms to be recognized, 856
 - varying number of, 863–864
 - Test certificate (self-cert)
 - license to code, 998–999
 - types of certificates, 996–998
 - Text, finding in ranges, 232–234
 - Text property, Range object
 - getting text from ranges, 382–384
 - setting text in ranges, 384–385
 - ThisApplication property
 - aggregated Document object, 605
 - aggregated Workbook object, 613
 - Tips, Ribbon controls, 783
 - TLBIMP, 40, 45
 - Toggle Button control, Ribbon
 - adding to Excel workbook project, 816–819
 - event handlers and, 821
 - naming, 818
 - overview of, 798–799
 - Toolbar events, Excel, 153–158
 - Toolbars, Ribbon replacing, 777
 - Toolbox, Ribbon controls in, 802–803
 - Tools group, Ribbon, 751
 - Tooltips, (+=), 119
 - Top property, Windows object, 199
 - Trust, ClickOnce
 - inclusion list for trust decisions, 1004–1005
 - Office Trusted Locations, 1004
 - prompts, 1002–1004
 - publisher certificates and, 1002
 - security checks and, 1005–1009
 - try blocks, C#, 165, 167
 - Type, optional parameters for worksheets, 191
 - Typed datasets, 895–897
- U**
- UIs (user interfaces)
 - Excel Application object, 44, 46
 - Fluent user interface, 777
 - properties for controlling look and feel of, 169–170
 - reasons for multiple event interfaces in Word, 267–269
 - Word properties controlling, 316–319
 - UNC (Universal Naming Convention), 960
 - Undo method
 - Excel Application object, 176
 - Word documents, 369
 - Uniform resource locators (URLs), 959–960
 - Union operator (,), for combining multiple cells, 220
 - Universal Naming Convention (UNC), 960
 - Unload event, Outlook, 459–461
 - Unmanaged code
 - COM interfaces for, 39
 - Office applications written in, 14
 - Untyped datasets, 895–897
 - Updating Excel documents, 992–994
 - Updating VSTO add-ins, 978–980
 - URLs (uniform resource locators), 959–960
 - USB key
 - deploying Excel documents via, 995
 - deploying to, 1009–1010
 - deploying VSTO add-ins via, 981
 - UsedRange property, Excel, 230
 - User controls
 - custom control added to Document Actions task pane, 684–686
 - custom property pages and, 410–413
 - User-defined functions, Excel
 - additional, 106–107
 - creating automation add-in for, 99–101
 - debugging, 107–108
 - loading automation add-ins, 101–105
 - overview of, 98–99
 - User Forms, Windows Forms controls replacing, 628
 - User information, Word Application object, 330
 - User interfaces. *See* UIs (user interfaces)

- UserControl object
 - ActionPane control and, 680–681
 - Add method, 704–705
 - adding to Custom task panes, 708–710
 - custom task panes and, 702
 - VSTOContainerControl derived from, 644
- V**
- VARIANT, COM type, 26
- VBA (Visual Basic for Applications)
 - ActiveX controls and, 649
 - code behind pattern in, 52
 - event-driven programming, 583–584
 - Excel writing custom functions in, 98–99
 - Office programming performed with, 4
- VBADE, 63
- View Code, Solution Explorer, 82–83, 411, 738
- View events, Outlook, 436–439
- View object, Outlook
 - Apply method, 529
 - CurrentView property, 490–491
 - View Type property, 530
- View, separating from data. *See* Data, separating from view
- View Type property, Outlook View object, 530
- ViewContextMenuDisplay event, Outlook, 443
- Views collection, Outlook, 529–531
- Views property, Folder object, 529–531
- ViewSwitch events, Outlook, 437
- Visible property
 - names, 203
 - Open method, 352
 - Word Application object, 67
 - worksheets, 204–205
- VisibleChanged event, CustomTaskPane object, 706
- Visio 2007, 6
- Visual Basic
 - code behind, 92–93, 245
 - optional parameters in Word, 25
 - Smart Tags, 96, 248
- Visual Basic for Applications. *See* VBA (Visual Basic for Applications)
- Visual Studio
 - creating console application with, 60
 - creating form regions with, 723
 - Designer view, 80–81
 - Excel Controls, 82
 - generating event handlers in Word, 270
 - Pascal casing code in, 32
 - Ribbon designer, 800–801
 - toolbox for Ribbon designer, 802–803
 - Windows Forms editor, 641
- Visual Studio 2008 Tools for Office 2007.
 - See* VSTO 3.0
- Visual Studio Professional, 5
- Visual Studio Solution Explorer, 42
- VSTO 3.0
 - prerequisites for ClickOnce, 958
 - reasons for using, 5–6
- VSTO add-ins
 - building for Outlook, 406–407
 - creating form region and, 723–724
 - creating Outlook add-In, 70–77
 - data binding from, 921–928
 - deploying. *See* Deploying VSTO add-ins
 - document controls in application-level add-ins, 666–668
 - document-level features in application-level add-ins, 621–624
 - Excel programming with, 90–92
 - install/uninstall events, 147
 - load behavior, 976
 - Outlook 2003 vs. Outlook 2007, 77–78
 - overview of, 69
 - programming model for, 619–621
 - Ribbon for delaying loading, 848–850
 - Ribbon in add-in project, 825–836
 - Smart Tags at add-in level, 874–879
 - Windows Forms control in application-level task pane, 702
 - for Word, 244–245
- VSTO programming model
 - add dynamic controls, 586–590
 - for add-ins, 619–621
 - aggregated objects, 578–581
 - aggregation and Windows Forms controls, 581–582
 - aggregation, inheritance, and implementation, 576–578
 - Bookmark object, 605–607
 - C# interoperability, 582–583
 - chart sheet host items and chart host controls and, 615
 - class hookup and cookies, 591–594

- VSTO programming model, *continued*
 - content control classes, 609–613
 - Controls collection, 585
 - document-level features used in application-level add-ins, 621–624
 - Document object, 599–605
 - dynamic controls, 584–586
 - dynamic worksheets, 624–625
 - enumerating/searching Controls collection, 585–586
 - event-driven programming, 583–584
 - Globals class in Excel, 598–599
 - inspecting generated code, 594–596
 - MVC design pattern, 575
 - overview of, 573–574
 - Range objects and, 615–619
 - removing dynamic controls, 590
 - saving/reloading dynamic controls, 590–591
 - separating data and view, 574–576
 - startup and shutdown sequences, 596–598
 - Tag field, 583
 - Workbook object and, 613–614
 - Worksheet object and, 614–615
 - XMLNode and XMLNodes controls and, 607–609
 - VSTO.Action. *See* Actions, Smart Tags
 - VSTOContainerControl, 644
 - VstoSmartTags collection, 604, 864, 874
- W**
- WCF (Windows Communication Foundation), 5
 - Web server, publishing Excel documents to, 982–986
 - Web services
 - Excel research service, 97–98
 - Word research service, 249–256
 - Web views, associated with folders, 499–500
 - What You See Is What You Get (WYSIWYG) designer, 809
 - WholeStory method, Range object, 380
 - Width property, Windows object, 199
 - Wikis
 - listing of WordWiki implementation, 67–69
 - overview of, 59
 - text format for, 60
 - Window events, Outlook, 434–435
 - Windows collection, Excel, 195–199
 - accessing windows in, 196–197
 - arranging Excel windows, 197–199
 - iterating over open windows, 196
 - overview of, 195–196
 - Windows collection, Word, 196–197
 - Windows Communication Foundation (WCF), 5
 - Windows, Excel, 199–201
 - accessing in Windows collection, 196–197
 - activating/deactivating, 123
 - arranging windows, 197–199
 - displaying settings associated with, 199–201
 - overview of, 199
 - positioning, 199
 - window resizing events, 146–147
 - Windows Forms
 - Forms Designer compared with, 749
 - Ribbon and, 848
 - VSTO programming compared with, 573
 - Windows Forms control host
 - hosting ActiveX controls, 644–645
 - limitations of, 649–650
 - overview of, 643
 - security of, 648–649
 - why VSTO controls are derived from Windows Forms, 645–648
 - Windows Forms controls
 - ActiveX controls moved to, 627–628
 - AddControl method, 663–664
 - added to Document Actions task pane, 681–684
 - adding to documents, 634–635
 - aggregation and, 581–582
 - Application-level add-in for creating control in a document, 666–668
 - blurry quality of, 653–654
 - code behind, 641–643
 - Content controls compared with, 652
 - control state not saved in documents, 650–653
 - controls added at runtime are not saved, 664–665
 - Controls collection, 661–663, 666
 - deleting at runtime, 664
 - displayed in custom task panes, 703

- on document surface, 629–631
 - excluded from controls toolbox, 636–638
 - hosting ActiveXcontrol, 644–645
 - hosting architecture, 643
 - hosting model limitations, 649–650
 - hosting model security, 648–649
 - insertion behavior in Excel, 640–641
 - insertion behavior in Word, 638–639
 - layout on Document/Worksheet surface, 641
 - methods for adding, 635–636
 - modal or modeless forms, 631–634
 - OLEObject/OLEControl properties merged with, 654–657
 - overview of, 627
 - running at runtime, 658–660
 - uses of, 628
 - why VSTO controls are derived from Windows Forms, 645–648
- Windows Forms editor, Visual Studio, 641
- Windows Forms Programming in C#* (Sells), 881
- Windows Installer
- custom setups for deployment solutions, 956, 1010
 - prerequisites for ClickOnce, 956
 - setup reboot and, 970
- Windows object, Word, 9
- Windows Presentation Foundation. *See* WPF (Windows Presentation Foundation)
- Windows property, Excel
- Application object, 195
 - returning open windows, 172
 - what it does, 182
- Windows Security Center, 406
- Windows Vista, 407
- Windows Vista, Programs and Features, 991
- Windows, Word, 338–341
- accessing in Windows collection, 339
 - activation/deactivation events, 280
 - ActiveWindow property, 352
 - arranging, 339–341
 - Caption property, 315–316
 - creating new, 338
 - iterating over open windows, 338–339
 - overview of, 338
 - selection change events, 285
 - sizing events, 287
 - working with windows associated with a document, 352
- Windows XP, 407
- Windows XP, Add or Remove Programs, 976–977, 991
- Word
- attaching/detaching actions pane solutions, 696
 - blurry controls in, 653–654
 - COM type library, 26
 - console application for automating, 59–69
 - custom task panes and, 702, 708, 714–717
 - data binding in Word documents, 908
 - Document Actions task pane. *See* Document Actions task pane
 - events. *See* Events, Word
 - hosting ActiveX controls, 644–645
 - list of key objects, 9
 - multiple Ribbons in, 837–838
 - new objects, 9
 - object model, 261–262
 - object model hierarchy, 7–8
 - objects, 8
 - OLEObject properties added to Windows Forms control, 654–657
 - optional parameters, 20–21, 24–28
 - overloads for helper method, 661
 - Ribbon in Word 2007, 777
 - Ribbon tab IDs for, 814
 - showing/hiding Actions Pane, 693–694
 - Smart Tags, 409
 - VSTO features enabled for Word 2007, 6
 - Windows Forms compared with Office controls, 581–582
 - Windows Forms controls inserted into, 638–639
- Word Basic, 335
- Word collections
- accessing items in, 323–342
 - accessing windows in, 339
 - properties returning important, 320–323
 - Range objects returned by, 355–356
- Word editor, 501–505
- Word objects. *See* Objects, Word

- Word programming, 243–266
 - automation executables, 243
 - code behind, 245
 - options for customizing Word, 243
 - research services. *See* Research services, Word
 - server-generated documents, 248–249
 - smart documents and XML Expansion Packs, 246–247
 - Smart Tags, 247–248
 - VSTO add-ins, 244–245
 - Word object model, 261–262
- Word Smart Tags
 - at add-in level, 874–879
 - adding, 857–859
 - configuring, 853–854
 - creating custom, 869–873
 - overview of, 851–853
 - regular expressions used with, 864
- WordArt, 215–216
- WordEditor property, Inspector object, 829
- Workbook events
 - activation/deactivation, 121–123
 - new event, 116–120
- Workbook object, 181–189
 - accessing document properties, 183–184
 - aggregated form, 613–614
 - collections returned by properties, 182
 - creating/activating windows, 186–188
 - events, 115
 - naming cell ranges, 186–188
 - overview of, 181
 - printing workbooks, 188
 - protecting workbooks, 188–189
 - saved property, 184–185
 - Sheets collection, 109–110
- Workbooks
 - accessing in workbooks collection, 179
 - activating/deactivating, 120–121
 - active/selected object properties, 181–182
 - calculate events, 135–136
 - calculation settings, 172–173
 - change events, 139
 - close events, 148
 - closing all open, 181
 - creating new, 179–180
 - creating with C#, 80
 - customizing, 79
 - deploying. *See* Deploying Excel documents
 - double-click and right-click events, 129
 - emailing, 175
 - event handlers for, 119–120
 - host item object for, 622
 - hyperlink events, 142
 - iterating over open, 178
 - names, 206
 - open events, 150
 - opening existing, 180–181
 - print events, 149
 - properties for saving, 187
 - property returning open, 172
 - Ribbon in, 812
 - save events, 150
 - saving, 184–186
 - selecting workbook to associate with code behind, 81
 - selection change events, 145
 - ServerDocument methods, 944–947
 - smart documents and, 93
 - window resizing events, 146–147
 - XML Expansion Packs and, 93–94
- Workbooks collection, 178–181
 - accessing workbook in, 179
 - closing all open workbooks, 181
 - creating new workbook in, 179–180
 - iterating over open workbooks in, 178
 - opening existing workbook in, 180–181
 - overview of, 178
- Workbooks property, Excel Application object, 172
- Worksheet collections
 - accessing sheets, 190–191
 - adding worksheets to workbooks, 191–192
 - copying sheets, 192
 - iterating over open sheets, 190
 - moving sheets, 192
 - overview of, 189
- Worksheet events
 - activation/deactivation, 122–123
 - new event, 116–120
- Worksheet object, 204–218
 - adding controls at runtime, 658
 - aggregated form, 614–615

- custom properties, 206–208
- events and, 115
- managing worksheets, 204–206
- names, 206
- protecting, 208–214
- working with ChartObjects, 216–217
- working with OLEObjects, 214–215
- working with Shapes, 215–216
- working with tables, 217–218

WorksheetFunction property, 173

Worksheets

- activating/deactivating, 122–123
- adding host controls dynamically, 586–590
- adding to collections, 191–192
- calculate events, 136
- Cells property, 220
- change events, 140
- Columns property, 221
- customizing, 594–596
- double-click and right-click events, 129–130
- dynamic, 624–625
- event handlers for, 119–120
- get_Range method, 219–222
- hooking up aggregated, 592–593
- host item object for, 622
- hyperlink events, 142
- layout of Windows Forms controls on, 641
- managing, 204–206
- removing dynamic controls, 590
- Rows property, 221
- saving/reloading dynamic controls, 590–591
- selection change events, 145
- Sheets collection and, 109–110
- Startup and Shutdown methods, 78

Worksheets property, 182

WPF controls

- in custom task panes, 719–721
- in Document Actions Task Pane, 697–699
- on document surface, 629–631

- in documents, 669–671
- via ElementHost control, 628–629

WPF (Windows Presentation Foundation), 5

Write event, Outlook, 456–459

WYSIWYG (What You See Is What You Get) designer, 809

X

XLA add-ins, 97

XLA files, 147

XLL, 98–99

XML

- cached data manifest, 948
- Document Actions task pane and, 678–680
- Excel events, 147
- pattern, 52
- Ribbon exported to, 848
- Ribbon programmability and, 780
- selection change events, 285
- Word events, 287–288

XML data island. *See also* Caching data, in XML data island

- accessing, 931
- .aspx Web form for server data from, 934–936
- custom handler for server data from, 936–939

XML Expansion Packs

- Excel programming and, 93–94
- Word programming and, 246–247

XML file formats, server data and, 930–931

XMLMappedRange object, 615–619

XMLNode control, 607–609

XMLNodes controls, 607–609

Z

Zones, trusted

- overview of, 1002–1003
- security checks and, 1006

Zoom levels, control hosting mode, 650