Pro Office 2007 Development with VSTO

Copyright © 2009 by Ty Anderson

ISBN-13 (pbk): 978-1-4302-1072-6

ISBN-13 (electronic): 978-1-4302-1071-9

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `http://www.apress.com/info/bulksales`.

The source code for this book is available to readers at `http://www.apress.com`.

CHAPTER 1

Introduction to VSTO 2008

Organizations of all types run their operations with the help of Microsoft Office. Since the initial release of Word, what has become the suite of applications known as Microsoft Office is no longer a tool only for the individual authoring documents. Today, the 2007 Microsoft Office system (Office 2007) is a business platform that allows teams of individuals to automate their complex business processes.

Enabling Office as a development platform for business is the reason Microsoft created Visual Studio Tools for the Microsoft Office System (VSTO). VSTO provides you with modern, .NET capabilities to build complicated, enterprise-class solutions using managed code with Office as the underlying platform. In this chapter, you will learn

- How to identify projects that fit VSTO
- The difference between application-level and document-level add-ins
- How to create application-level and document-level add-ins
- The development features VSTO provides
- The purpose of the `ThisAddin`, `ThisDocument`, and `ThisWorkbook` classes
- The purpose of deployment and application manifests

An Overview of VSTO 2008

The official product name is Visual Studio Tools for the Microsoft Office System 2008. Microsoft certainly does not have the best imagination when naming its products, but its names rarely leave you wondering what it is that the product does. This couldn't be truer with VSTO, as it does exactly what the name implies. VSTO extends Visual Studio's toolset to Office and allows you to build Office-based solutions using managed code (Visual Basic .NET or C# only).

VSTO relies on the .NET Framework, which means all the power and features available with .NET are available to you when programming with Office. For example, when creating an Excel worksheet solution, you can add WinForm controls to the worksheet. Also, if you want to create a custom action pane in an Office application, the form you will use is a .NET WinForm user control, which has all the features you expect from WinForm controls. These tools are a serious improvement over the control set available using Visual Basic for Applications (VBA)—the traditional Office scripting or macro language.

VBA: NOT DEAD YET!

VBA has had a long and good life and will continue its life for the foreseeable future as the macro language for Office. However, it will eventually fade into the sunset, because Microsoft does not have a road map for future VBA language enhancements. It has not added any features to the core VBA language or the Office Visual Basic Editor since Office XP. In fact, the initial VSTO team included many of the existing VBA team members. Despite VBA's stunted growth, Office 2007 contains hundreds of new objects and methods related to Office's new features, such as the RibbonX application programming interface (API). You can build complete solutions with VBA, but you don't have access to the features provided by VSTO.

The first version of VSTO allowed for the creation of custom Word and Excel document solutions in Visual Basic .NET (VB .NET) and C#. This initial version of VSTO implemented a simple model that relied on custom document properties. These properties told VSTO the location of the .NET assembly and the name of the startup class to call within it. This model separated the code from the document by calling the linked .NET assembly and took full advantage of the .NET code-access security model. This release established a .NET beachhead within the Office development community, as it allowed developers to build *document-level* add-ins with managed code. Prior to VSTO, document-level add-ins were only possible using VBA.

The second version of VSTO included two releases: VSTO 2005 and VSTO 2005 Second Edition (VSTO 2005 SE). These two versions combined to extend the VSTO feature set to include application-level add-ins for Excel, Word, Outlook, PowerPoint, InfoPath, and Visio. Beyond including support for these new types of add-ins, Microsoft made other improvements to the existing document-level add-ins. These two versions of VSTO together provided support for Office 2003 and Office 2007.

As the third major release of VSTO, VSTO 2008 rolls up all the features from the previous VSTO versions and includes many new features for Office 2007 development. The preexisting Office 2003 features still remain, making VSTO 2008 the most powerful set of development tools for Office.

Existing Features Prior to VSTO 2008

I want to start the discussion of VSTO's features by talking about the feature set that already existed prior to VSTO 2008. I'll then explain the new tools and capabilities included with VSTO 2008. It's important to understand the existing feature set, as it is the foundation for everything in VSTO 2008. The combination of the old and new features has caused VSTO 2008 (the third major release of VSTO) to leave its infancy and join the ranks of mature development tools.

Application-Level Add-Ins

An application-level add-in is a class-library project that produces an assembly associated with a supported Office application. This assembly runs as an add-in within the host application's domain. Using this type of project, you have access to all of the .NET namespaces as well as the target Office application's object model. In fact, the project provides a class named `ThisApplication`, which gives you easy access to the Office application object.

The project type is a simplified version of the Shared Add-in that was once the recommended framework for building Office add-ins with managed code. With this version, however, you don't need to worry about creating a Component Object Model (COM)-based wrapper—or shim—for final deployment with Office, as VSTO does this for you.

■ **Note** A shim is complex and requires a C++ wrapper class. I recommend reading "Implementing IDTExtensibility2 in an Automation Add-in" by Andrew Whitechapel (`http://blogs.officezealot.com/whitechapel/archive/2005/06/09/4756.aspx`).

Document-Level Add-Ins

Document-level add-ins are the original VSTO add-in type. Like application-level add-ins, a document-level add-in produces an assembly. The difference, however, is this assembly attaches to an Office document. Currently, only Word documents, Word templates, Excel worksheets, and Excel templates are supported. Whichever type you choose, the document stores a link to the assembly, and the assembly remains external to the document. This deployment strategy is different from VBA macros, which are embedded inside the document. Document-level add-ins give you access to all .NET namespaces and the target document's objects. Instead of `ThisApplication`, the class name is either `ThisWorkbook` or `ThisDocument`.

With document-level add-ins, you have the ability to write code that reads and writes data related to the business objects represented by the document. You don't need to worry about finding the value in cell A2 in Excel or, worse, the customer name typically found on page 2, paragraph 2 in Word. Instead, you can reference data using WinForm data-binding objects to access the data directly without messing with the Office objects.

Custom Task Panes and Action Panes

Custom Task Pane forms and Action Pane forms are the task-oriented vertical windows that typically display next to a document (see Figure 1-1). Task panes are associated with an application-level add-in, while action panes are associated with document-level add-ins. Besides this one contextual difference, they are basically the same. These panes allow you to build context-sensitive user interfaces for your supported task or scenario using Windows Forms and WinForm controls. For example, you can build a custom pane that connects to your company's sales data, lists current contacts, allows the user to select a value, and inserts the data into the document.

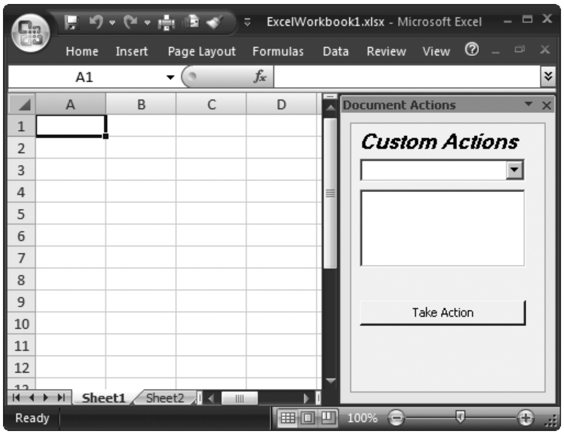


Figure 1-1. *A custom VSTO action pane filled with WinForm controls*

Outlook Form Regions

Outlook is one of the more popular Office development targets, because it acts as the operation hub for millions of Office users every day. VSTO allows you to create custom form regions to extend the various Outlook forms with additional layout regions. These regions display as part of the targeted Outlook form (see Figure 1-2). Form regions provide the abilities to extend the default page area and replace standard forms.

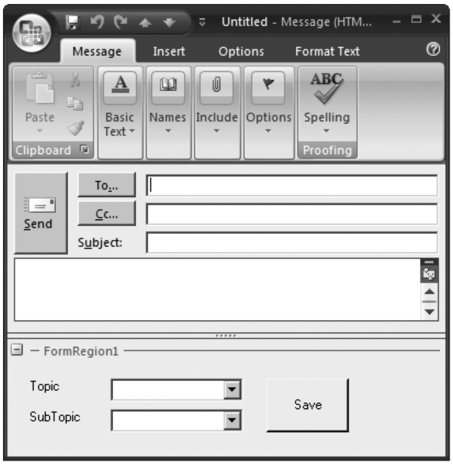


Figure 1-2. *A custom form region displaying within an Outlook mail item*

Customizing the Default Page Area

You can extend the default page area of any Outlook item's standard form, including e-mail, appointments, and tasks. Previously, the default page for most Outlook forms was not open to developer customization. Now, you have the ability to extend Outlook forms to capture and display those missing fields your users have been clamoring to see. For those who require a plethora of customizations, VSTO supports adding up to 30 extra pages to any of the standard Outlook forms.

Replacing Standard Forms

You now have the ability to replace Outlook forms with your customized version. The ability to perform this trick was previously possible but definitely not supported by Microsoft.

OUTLOOK FORM REPLACEMENT PRIOR TO VSTO 2008

Replacing standard Outlook forms with your own custom version is possible without VSTO 2008, but it is unsupported by Microsoft. In fact, if you were to attempt to replace the standard Contact form, for example, you would be required to implement all the features required by the form without any help from VSTO. The secret is to capture the event that opens an Outlook form (`NewInspector`), cancel the opening of the form, and instead, open your custom version. Although possible, this strategy requires a lot of work to reimplement the features your users would expect a Contact form to include. To see how to implement the old, unsupported method for replacing Outlook forms, read an article I wrote on the topic: "Outlook the Way You Want It—Build Custom Outlook GUIs with WinForms & VSTO" (`http://www.devx.com/MicrosoftISV/Article/29261`).

Ribbon Customizations

Office 2007 includes a new toolbar user interface for Word, Excel, PowerPoint, and Outlook (only partially implemented in Outlook) known as the Office Fluent User Interface, or Ribbon (see Figure 1-3). VSTO allows you to customize the Ribbon by adding your own tabs, groups, and buttons. This is much like creating new command bars in previous versions of Office. VSTO 2008 provides a Ribbon Designer, which greatly improves the developer experience for creating Ribbon customizations. I'll explain the designer later in this chapter.

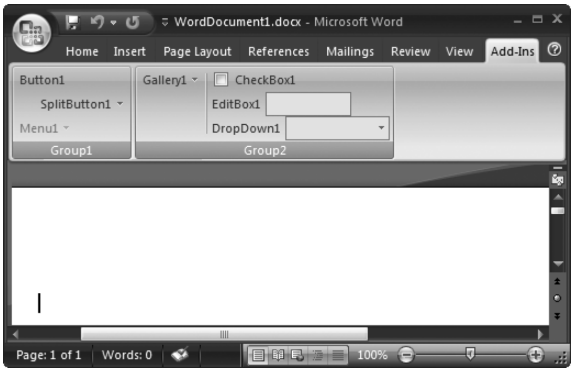


Figure 1-3. *A customized Ribbon displayed in Word*

Smart Tags

A smart tag is a class library attached to a Word or Excel document. Smart tags scan for defined strings of text that have actions attached to them. VSTO allows you to maintain a list of actionable values and recognize them when they appear in either a Word or Excel document. Anytime a desired value is recognized, VSTO displays a menu of actions that you have defined for that type of string. Because you can define the various types of strings the VSTO add-in should identify, you can display different actions for each value type.

VSTO smart tags affect only the targeted document. This means the smart tag only reacts to identified values in its associated document. It doesn't react to any other document, as it isn't in scope. The ability to attach a smart tag to a single document (*single-document scoping*) is a key distinction from Office smart tags that operate at the application level and respond to identified values across all open documents.

You can implement smart tags to provide easy access to contextual actions within the document window. For example, a proposal document could scan for known client names and provide a menu that allows a user to view additional client data, such as contact details and financial details.

Host Controls

Host controls extend Office objects for interaction with managed code. They are based on native, COM-based Office controls and behave like their native Office counterparts. However, host controls add properties and methods that allow you to respond to that instance of the control.

Bookmark

The Bookmark control, supported by Word, identifies a placeholder inside a Word document. You can bind this control to a data source. This control automatically reflects any changes to the underlying data source.

XMLNode

The XMLNode control, supported by Word, only exists if an Extensible Markup Language (XML) schema is attached and mapped to the document. It also supports data binding to a data source. You can manipulate this object directly with code.

XMLNodes

The XMLNodes control, supported by Word, only exists if an XML schema is attached and mapped to the document. This control contains a collection of mapped `XMLNode` objects. It does not support data binding.

NamedRange

The NamedRange control, supported by Excel, contains a `Range` object and supports data binding. A range is a collection of Excel cells. The NamedRange control provides you with the ability to access the range directly and code against its events. This is not possible with the Office-based `Range` object, where you need to respond to every change event to determine if the desired range triggered the change.

ListObject

The ListObject control, supported by Excel, contains an Excel list and supports data binding. It exposes properties and methods that allow you to reference the list and code directly against it.

Chart

The Chart control, supported by Excel, contains a `Chart` object, supports data binding, and exposes properties and methods that allow you to reference the list and code directly against it.

XMLMappedRange

The XMLMappedRange control, supported by Excel, contains a range with an XML schema mapped against it. This range supports data binding and direct manipulation through code via its exposed properties and methods.

Visual Document Designers

VSTO provides full-featured designers for Word and Excel (see Figure 1-4). These designers open Word and Excel documents within Visual Studio and treat the documents as a design surface similar to a Windows form. You can drag and drop controls, edit properties, and switch between Code and Design views to your heart's delight. This feature greatly enhances the Office development experience, as you can write code behind the document as you build its corresponding interface. In addition, the full Office menus integrate within Visual Studio, so you can edit the underlying document as you normally would in Word or Excel.

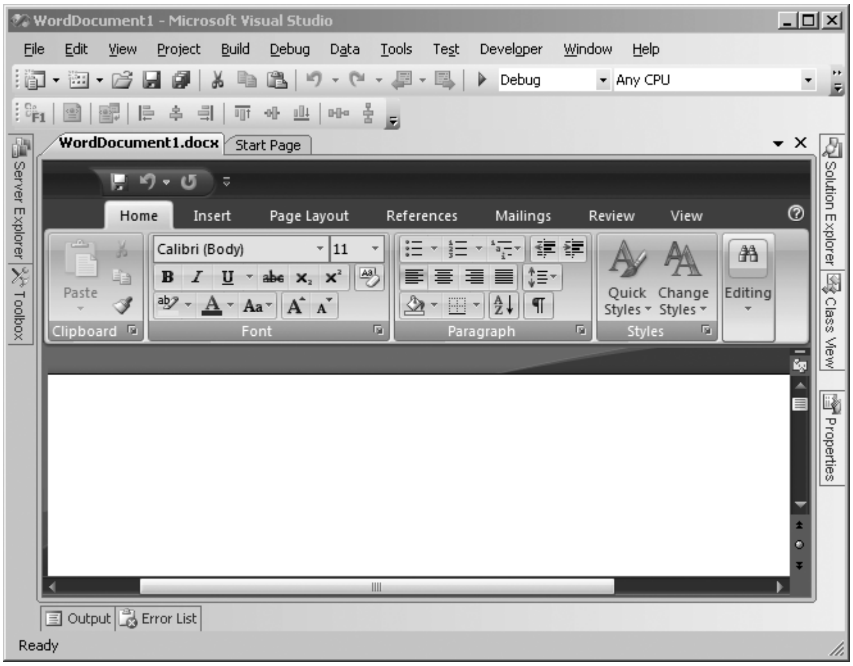


Figure 1-4. *A Word-based VSTO solution displaying the Word visual designer*

Data Programming

VSTO allows you to access and manipulate data within Office documents directly, without relying on the Office object models. VSTO employs the model-view-controller (MVC) pattern to separate the data from the Office document user interfaces. (See `http://en.wikipedia.org/wiki/Model-view-controller` for more information.)

VSTO stores the data as an XML data island. Since the data is XML-based, you can create typed datasets to access the underlying data schema by name. This strategy removes a major source of pain with traditional Office development methods that require navigating the user interface to access data. With VSTO, data access is direct and simple due to VSTO's data-caching features and the `ServerDocument` object.

Data Caching

The data contained in the XML data island enables offline data caching. Data retrieved externally to the document fills the data island. After the document closes, the data island remains filled with the data, enabling offline usage. Think of the classic airplane example, where the data cache is offline, and the account executive is downloading sales spreadsheets and reviewing them on the plane trip to meet the client.

ServerDocument

The XML data island also exposes methods for manipulating supported Office document data directly. The `ServerDocument` object provides the APIs for reading and writing data with server-side code. This doesn't require the code to execute in Windows Server; instead this refers to the location of the execution code—either the server or the client, which could be one and the same.

The data-programming features are available with document-level add-ins, meaning only Word and Excel are supported.

What's New in VSTO 2008

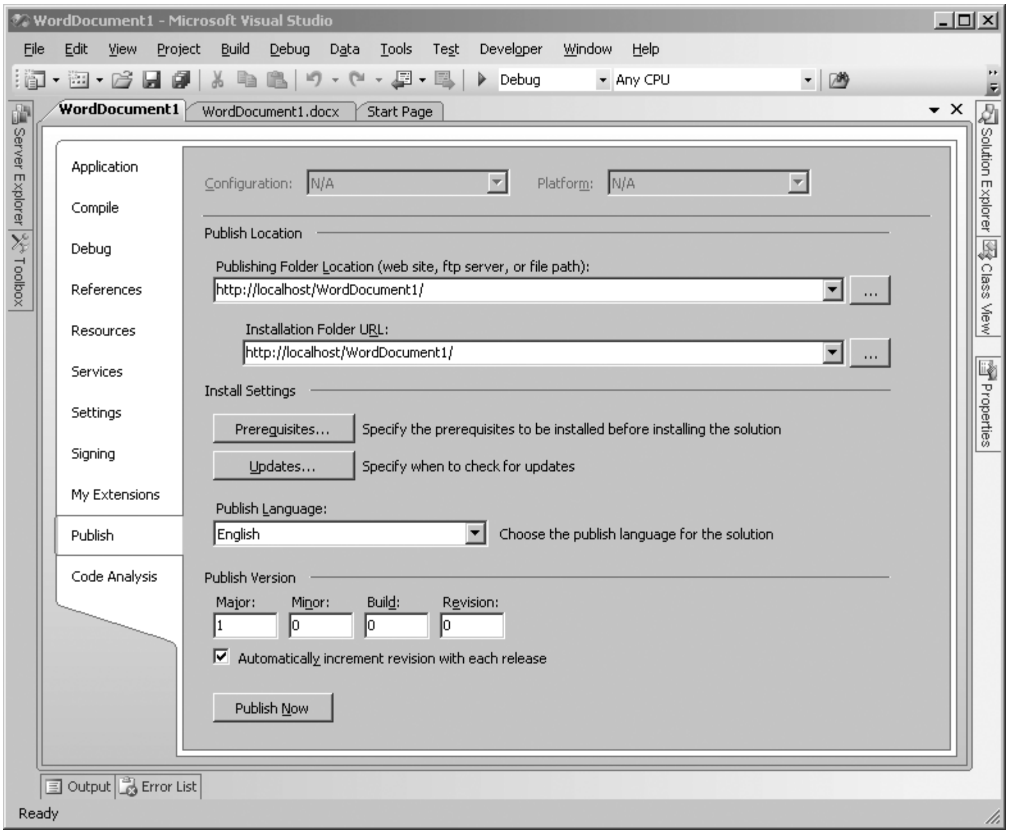
VSTO 2008 targets solutions developed with Office 2007. The new features included in this version largely correspond with the new features in Office 2007, such as the new Office XML file formats, the Ribbon user interface, Outlook form regions, and Word content controls. However, this release does more than support Office's new features; it also adds support for .NET technologies like ClickOnce deployment. Each tool helps you build powerful, Office-based solutions in less time and with less effort.

ClickOnce Deployment

Finally, ClickOnce deployment exists for Office development. ClickOnce is a self-updating deployment technology that allows for deployment of applications with minimal user interaction. The idea is to make installing and updating applications as easy as clicking a link . . . once. Once deployed, the add-in checks the server each time it initializes to determine if an updated version exists; if it finds one, the add-in self-updates.

This deployment model has not been possible with Office solutions due to the COM architecture of Office and its reliance upon the Windows registry. Office applications read the registry to load any associated add-ins on application startup. ClickOnce does not support writing keys to the registry, making ClickOnce an impossibility.

VSTO 2008 changes this dire situation by supporting ClickOnce installations for application-level add-ins. VSTO provides a Publish Wizard (see Figure 1-5) to guide you through the creation of a deployment package (called a ClickOnce manifest). This is a powerful and much needed tool for Office development.



Configuration: N/A Platform: N/A

Publish Location

Publishing Folder Location (web site, ftp server, or file path): http://localhost/WordDocument1/

Installation Folder URL: http://localhost/WordDocument1/

Install Settings

Prerequisites... Specify the prerequisites to be installed before installing the solution

Updates... Specify when to check for updates

Publish Language: English Choose the publish language for the solution

Publish Version

Major: 1 Minor: 0 Build: 0 Revision: 0

☑ Automatically increment revision with each release

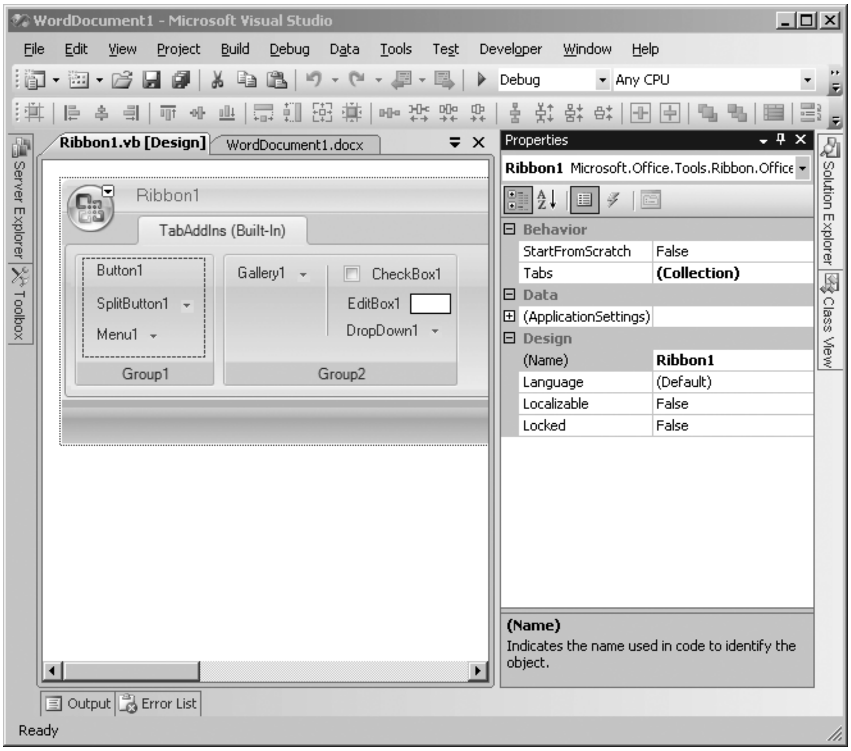
Figure 1-5. *The ClickOnce Publish Wizard in action*

Unfortunately, document-level add-ins are not supported and continue to use the VSTO 2005 deployment model.

Ribbon Designer

Customizing the Ribbon is now supported with a design-time control known as the Ribbon Designer (see Figure 1-6). This designer allows you to create custom Ribbon tabs in the same manner as designing Windows Forms. With this tool, you can quickly create new tabs, extend Office's built-in tabs, add control groups, and draw controls within them. The Ribbon controls function similarly to Windows Forms controls with familiar objects, properties, and events.

The Ribbon user interface is XML-based, so peculiarities exist. For example, you can't double-click the Ribbon control to attach an event. Instead, you assign the name of the method you want to execute as an attribute of the control.



Properties

Ribbon1 Microsoft.Office.Tools.Ribbon.Office

Behavior	
StartFromScratch	False
Tabs	**(Collection)**
Data	
(ApplicationSettings)	
Design	
(Name)	**Ribbon1**
Language	(Default)
Localizable	False
Locked	False

(Name)
Indicates the name used in code to identify the object.

Figure 1-6. *The Ribbon Designer*

Ribbon XML

Although the Ribbon Designer is a much-needed and easy-to-use tool, it does not support all Ribbon customization features. If you need to go beyond the features provided in the Ribbon Designer, you will need to write Ribbon XML manually. Fortunately, VSTO 2008 provides a Ribbon (XML) project template to support these needs.

Use the Ribbon (XML) template if you want to

- Include standard Ribbon groups in your custom tab or another standard Ribbon tab
- Add standard controls to your custom group
- Add buttons to the Quick Access Toolbar (the button in the upper left-hand corner of Ribbon-supported Office applications, as shown in Figure 1-7)

- Override default event handlers for standard controls with your custom code
- Share your Ribbon (XML) customizations among multiple add-ins

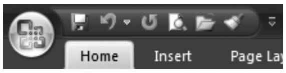
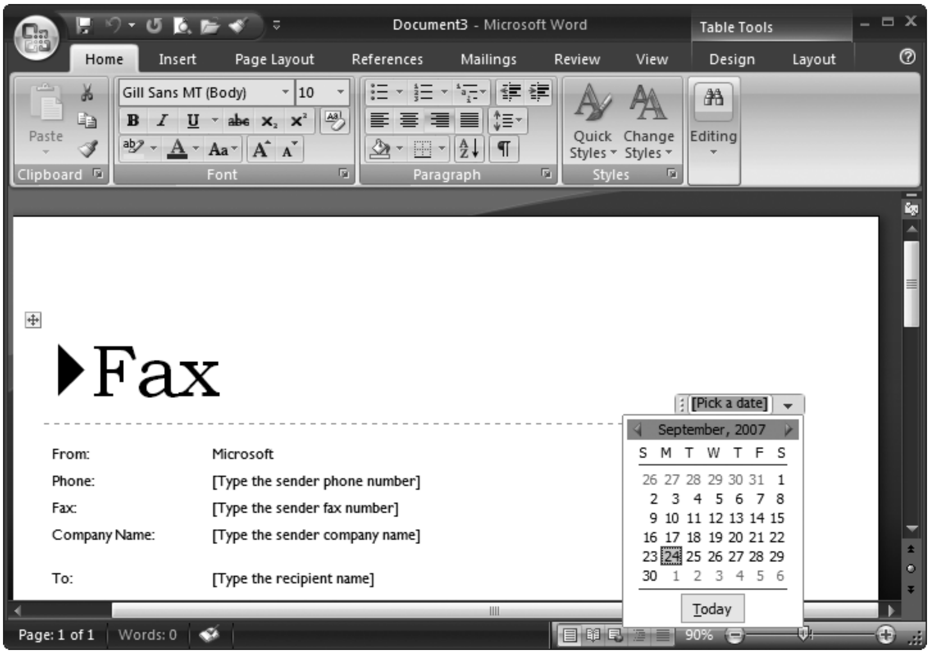


Figure 1-7. *A customized Quick Access Toolbar*

Word Content Controls

Word 2007 includes a new set of controls known as *content controls*. These controls add structure to Word documents by defining a data region and its data type. These controls ease the effort required to edit the typed data within a Word document. In addition, they allow for better support of data types, as you can limit what type of data you can input within the control.

VSTO provides the developer experience expected within Windows Forms. You draw content controls onto the document surface (see Figure 1-8). You can write code to respond to the control's events, set its properties, and so on.



From: Microsoft
Phone: [Type the sender phone number]
Fax: [Type the sender fax number]
Company Name: [Type the sender company name]

To: [Type the recipient name]

Figure 1-8. *A Word content control containing a DateTime data type*

Outlook Form Region Wizard

VSTO automates the creation of custom Outlook form regions with the Form Region Wizard. This wizard eliminates the need for you to manually create the XML required for custom form regions. Once added to your Outlook add-in project, this wizard takes you through a five-step process for creating the shell for your custom region. Utilizing the wizard, you can

- Create new form regions or import the Outlook Form Storage (`.ofs`) file created within Outlook's form designer
- Specify the form region's location
- Specify display preferences
- Associate the region to Outlook items, including mail, tasks, and the calendar

After the wizard creates the form region, you code against it as if it were a Windows form by drawing controls, setting properties, attaching event handlers, and more.

SharePoint Workflow

Given the popularity of SharePoint 2007, the VSTO team came through for you by adding support for building custom SharePoint workflows. These project templates allow you to build custom processes that control the life cycle of documents and lists. Support for both sequential (a series of actions) and state-machine (a set of states, transitions, and actions) workflows exists.

Custom workflows are perfect for business processes whose complexity goes beyond the standard SharePoint workflows. A good example is the process of responding to government-related Requests for Proposals (RFPs). This business process could require input from multiple individuals and external partners. With VSTO, you can create a complex, state-machine workflow to manage the various stages of building the proposal.

The VSTO Programming Model

The beauty of using Office to build solutions is that it is familiar to your users. They use it every day and even if they don't necessarily love it, they are accustomed to it and know how to accomplish their work with it. This familiarity is one of the more compelling reasons to build VSTO solutions.

Another compelling reason to create VSTO solutions is that you build them using a tool known and loved by .NET developers, Visual Studio 2008. VSTO not only helps take care of your users, but it also provides the tools you need in a familiar environment. Your VSTO learning curve is low, because you're not required to learn a significant number of new programming skills.

Enabling .NET Development

Each Office application is a COM-based program and is completely unaware of the .NET Framework. COM was Microsoft's specification for providing object interoperability prior to

.NET. All of Office's interfaces are based on the COM specification, which means COM remains a key component of Office development.

Traditionally, you would build Office solutions with Office's embedded macro language, VBA, or you would use classic Visual Basic to automate Office applications from your custom application. Also, you would need to pick from a plethora of Office add-in technologies such as COM add-ins (managed or unmanaged), smart tags, Excel file add-ins (`.xla` files), Word file add-ins (`.dot` files), and many more.

Each of these pre-VSTO add-in technologies were great, because they provided flexibility in how you extended Office. The problem, however, was that Office lacked a single development model. Each technology provided its own interface and development model. Building Office solutions in pre-VSTO days required a large amount of patience and a desire to learn a multitude of interfaces.

VSTO 2008 still supports the previous methods for building Office add-ins. It doesn't replace them or consolidate them into a single, overarching programming model. Instead, it provides the Office extensibility architecture for Office 2007 and beyond. If you want to extend Office with VSTO, you only need to know how to build an add-in. You no longer need to determine whether or not you should customize a menu using an Excel add-in instead of managed-code, COM add-in (also known by its Visual Studio template name, Shared Add-in). Using VSTO, you only need to know which project template to select and how to work within the VSTO framework. In the remainder of this chapter, I'll discuss the VSTO programming model along with the key objects and classes you will use when building VSTO-based Office solutions.

Model Overview

VSTO supports both Office 2003 and Office 2007, with the latter receiving more support and features. To build a VSTO solution, begin by opening Visual Studio and selecting the project template appropriate for your development needs. VSTO provides templates for building both application-level and document-level solutions to varying degrees. For example, document-level add-in templates are only available for Word and Excel. In addition, Access, Publisher, and Groove lack application-level add-in support. Table 1-1 lists the application support for each project type.

Table 1-1. *Application Support by Project Type*

Add-In Types	Supported Office 2007 Applications	Supported Office 2003 Applications
Application-level add-ins	Outlook Excel Word InfoPath PowerPoint Project Visio	Outlook Excel Word PowerPoint Project Visio
Document-level add-ins	Word Excel	Word Excel
SharePoint workflows	SharePoint Server 2007	Not supported

Once you create a new project and Visual Studio loads it into the editor, building with VSTO is similar to building a Windows Forms application. However, having Office in the mix does add to the complexity of your solution, and the inner workings of how you move from a VSTO project template to an executing assembly working with Office applications or documents require further explanation.

First, let me explain the main development paradigms of VSTO.

Office Objects Are .NET Classes

VSTO provides custom classes for each supported Office application and document. These classes (`ThisAddin`, `ThisDocument`, and `ThisWorkbook`) are fully typed .NET classes that incorporate and extend the Office object model. They provide the interfaces for the VSTO managed code to call Office's COM-based objects.

Managed Controls Hosted Within Documents

Managed controls (also called *host controls*) are .NET controls that connect to their actual Office counterpart. They are not the same as the object they represent; instead, they provide a reference to the actual object and provide the ability to extend its class, as you would expect to do with a .NET class.

Separation of Business Logic and View

VSTO provides .NET controls for objects such as the Excel NamedRange control and the Word Bookmark control. These controls provide direct access to their related objects in the actual document. You no longer have to navigate the document and parse values until you find the one you want. Simply drop these controls onto the document (or workbook) and manipulate them by name, just as you would expect to do with a WinForm application.

Now that you understand the main developer benefits of VSTO and understand that, more or less, building a VSTO add-in is much like building a Windows Forms application, it is time to learn how to build each type of add-in. In this chapter, I'm concerned only with application-level and document-level add-ins. I'll cover SharePoint workflow projects in Chapter 15.

Understanding Application Add-Ins

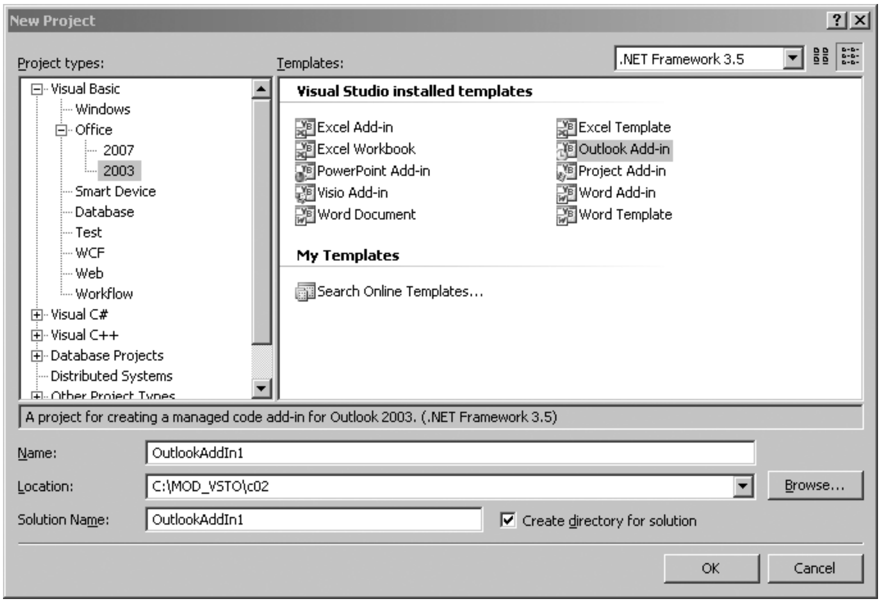
The best method to learn how VSTO works is to build with it. In this section and the next, you will learn the basic developer workflow for building VSTO solutions. As you build each add-in, I will explain each of the main components and the role they play within the overall solution.

As mentioned earlier, a VSTO application-level add-in is an assembly executing against a VSTO-supported Office application such as Outlook, Excel, or PowerPoint. Open Visual Studio, and let's walk through how to build an add-in that targets Outlook.

Creating an Outlook Application Add-In

Since Outlook is one of the most popular target applications for VSTO add-ins, let's use it to walk through building our first sample add-in. Complete the following steps:

1. Open Visual Studio.
2. Open the New Project dialog box (File ➤ New ➤ Project) and select Visual Basic ➤ Office ➤ 2007 in the Project Types section (see Figure 1-9).


New Project

Project types: Visual Basic, Windows, Office, 2007, 2003, Smart Device, Database, Test, WCF, Web, Workflow, Visual C#, Visual C++, Database Projects, Distributed Systems, Other Project Types

Templates: .NET Framework 3.5

Visual Studio installed templates: Excel Add-in, Excel Workbook, PowerPoint Add-in, Visio Add-in, Word Document, Excel Template, Outlook Add-in, Project Add-in, Word Add-in, Word Template

My Templates: Search Online Templates...

A project for creating a managed code add-in for Outlook 2003. (.NET Framework 3.5)

Name: OutlookAddIn1

Location: C:\MOD_VSTO\c02

Solution Name: OutlookAddIn1

☑ Create directory for solution

Figure 1-9. *Selecting a VSTO project template in Visual Studio*

Since VSTO supports both Office 2003 and Office 2007, two nodes exist to make it easy for you to select a project type that supports your targeted version of Office. The available project types follow the data presented Table 1-1.

3. Select the Outlook Add-in template from the Templates section and click OK. Make things easy on yourself, and accept the default values for the Name, Location, and Solution Name, as this is a sample application.

With this blank add-in ready to go, let's review the main components that make it function.

Main Add-In Components

After Visual Studio creates the project, take a look at the Solution Explorer window to review the objects that make up the add-in project (see Figure 1-10). Application add-ins initially consist of a single class and references to the host application, which in this case is Outlook. The class implements the `IDTExtensibility2` interface required for Office add-ins.

■ **Note** VSTO dramatically simplifies the implementation of the `IDTExtensibility2` interface. In short, with VSTO, you only need to implement two of the five interface methods. Behind the scenes, VSTO implements the other three methods. You can learn more by reading “VSTO support for Outlook” by Eric Carter (`http://blogs.msdn.com/eric_carter/archive/2005/06/06/423986.aspx`).

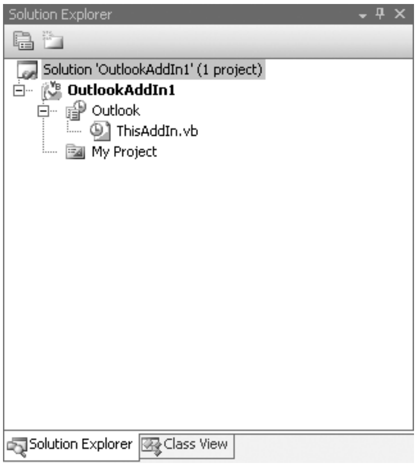


Figure 1-10. *The contents of the Outlook add-in displayed in Solution Explorer*

The ThisAddin Class Every VSTO application add-in contains a class named `ThisAddin`. This class acts as the entry point to the add-in and is the starting point for implementing your code. Listing 1-1 contains the full code stub for the newly created `ThisAddin`.

Listing 1-1. *A Stubbed-Out ThisAddin Class*

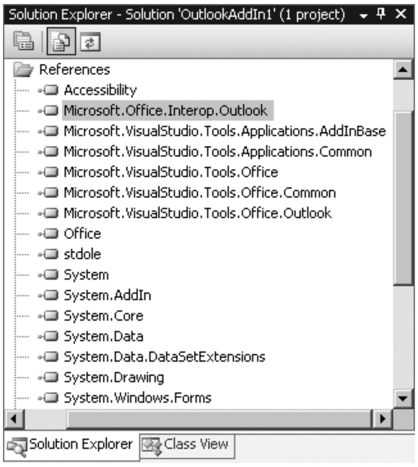
```
Public Class ThisAddIn

    Private Sub ThisAddIn_Startup(ByVal sender As Object, ByVal e As _
      System.EventArgs) Handles Me.Startup

    End Sub
    Private Sub ThisAddIn_Shutdown(ByVal sender As Object, ByVal e As _
      System.EventArgs) Handles Me.Shutdown

    End Sub

End Class
```

The class contains two methods: `Startup` and `Shutdown`. These are appropriate for initializing and cleaning up your add-in, respectively. The `Startup` event executes when the host application loads the add-in into memory. This means the host application has completed its initialization process and all of its objects are available for you to access and manipulate with your code. The `Shutdown` method executes just prior to the host application unloading it from memory. Use the `Shutdown` method to perform any required cleanup.

Both methods contain the same arguments. For application add-ins, the `sender` argument passes a reference to the add-in, providing you with access to the classes' properties and methods.

Perhaps the most important property of `ThisAddin` is the `Application` property. This property is a reference to the current instance of the host application, which is important to Office development. The `Application` object is always the parent object within Office applications; all other objects are linked as child objects. Thus, if you have access to the `Application` object, you have access to every other object in the application.

Office Primary Interop Assemblies Within Solution Explorer, click the Show All Files button to view the project's References folder (see Figure 1-11). VSTO automatically includes a reference to the host application Outlook (`Microsoft.Office.Interop.Outlook`). This is a reference to Outlook's Primary Interop Assembly (PIA), which is just one of the Office PIAs. Interop is short for interoperability and in the context of VSTO, refers to the managed assemblies that wrap the Office COM-based interfaces for use within .NET development. A full listing of all Office PIAs is available on the Microsoft Developer Network (MSDN) site at `http://msdn2.microsoft.com/en-us/library/15s06t57(VS.90).aspx`.



```
References
  Accessibility
  Microsoft.Office.Interop.Outlook
  Microsoft.VisualStudio.Tools.Applications.AddInBase
  Microsoft.VisualStudio.Tools.Applications.Common
  Microsoft.VisualStudio.Tools.Office
  Microsoft.VisualStudio.Tools.Office.Common
  Microsoft.VisualStudio.Tools.Office.Outlook
  Office
  stdole
  System
  System.AddIn
  System.Core
  System.Data
  System.Data.DataSetExtensions
  System.Drawing
  System.Windows.Forms
```

Figure 1-11. *The project's References folder for a VSTO Outlook add-in*

The Office PIAs contain the Office type definitions for each Office application and are what make coding Office with managed code a possibility. In order for managed code to communicate with COM-based code, it needs to understand the types of objects it is calling. The PIAs provide these descriptions and let the managed code know what object types and data types it should use to call the referenced COM object.

A SET OF PIAS FOR EVERY VERSION OF OFFICE

Microsoft publishes a version of PIAs for each new version of Office starting with Office 2003. It is important to make sure you use the Microsoft version of the PIAs and not allow Visual Studio to generate one by creating a reference to an Office application. Although this will work, Microsoft doesn't support it, and it will most likely cause problems that you could easily avoid by using the correct PIAs for your targeted version of Office.

Manifest Files VSTO utilizes two types of XML-based manifest files for deploying and updating add-ins. The first manifest is the application manifest, which contains the information that VSTO needs to locate the assemblies included within the add-in. The application manifest stores such information as the add-in's name, version, public key token, and language. It also stores information about assembly dependencies and prerequisites, as well as the name of the class that serves as the application entry point. Basically, if an add-in needs a file to execute properly, this manifest will include the needed information. Listing 1-2 contains a sample application manifest.

Listing 1-2. *A Sample VSTO Application Manifest*

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1
 assembly.adaptive.xsd" manifestVersion="1.0" xmlns:asmv3=
"urn:schemas-microsoft-com:asm.v3" xmlns:dsig=
"http://www.w3.org/2000/09/xmldsig#" xmlns="urn:schemas-microsoft-com:asm.v2"
 xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns:asmv2=
"urn:schemas-microsoft-com:asm.v2" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xmlns:co.v1="urn:schemas-microsoft-com:clickonce.v1">
<asmv1:assemblyIdentity name="OutlookAddIn1.dll" version="1.0.0.0"
  publicKeyToken="b213866c46f1a387" language="en" processorArchitecture=
"msil" type="win32" />
<description xmlns="urn:schemas-microsoft-com:asm.v1">
OutlookAddIn1 - Outlook add-in created with Visual Studio Tools
for Office</description>
<application />
<entryPoint>
<co.v1:customHostSpecified />
```

```
</entryPoint>
<trustInfo>
<security>
<applicationRequestMinimum>
<PermissionSet Unrestricted="true" ID="Custom" SameSite="site" />
<defaultAssemblyRequest permissionSetReference="Custom" />
</applicationRequestMinimum>
<requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
<requestedExecutionLevel level="asInvoker" />
</requestedPrivileges>
</security>
</trustInfo>
<dependency>
<dependentAssembly dependencyType="preRequisite" allowDelayedBinding="true">
<assemblyIdentity name="Microsoft.Office.Interop.Outlook" version="12.0.0.0"
 publicKeyToken="71E9BCE111E9429C" language="neutral" />
</dependentAssembly>
</dependency>
<vstav1:addIn xmlns:vstav1="urn:schemas-microsoft-com:vsta.v1">
<vstav1:entryPoints>
<vstav1:entryPoint class="OutlookAddIn1.ThisAddIn">
<assemblyIdentity name="OutlookAddIn1" version="1.0.0.0" language="neutral"
processorArchitecture="msil" />
</vstav1:entryPoint>
</vstav1:entryPoints>
<vstav1:update enabled="true">
<vstav1:expiration maximumAge="7" unit="days" />
</vstav1:update>
<vstav1:application>
<vstov3:customization xmlns:vstov3="urn:schemas-microsoft-com:vsto.v3">
<vstov3:appAddIn application="Outlook" loadBehavior="3" keyName="OutlookAddIn1">
<vstov3:friendlyName>OutlookAddIn1</vstov3:friendlyName>
<vstov3:description>OutlookAddIn1 - Outlook add-in created with
Visual Studio Tools for Office</vstov3:description>
</vstov3:appAddIn>
</vstov3:customization>
</vstav1:application>
</vstav1:addIn>
```

The second manifest is the deployment manifest, which serves to identify the current version of the application that should be deployed to the user's machine. The information included in this manifest largely matches that of the application manifest, as it contains the name, version, language, and so on, but in much less detail. It does not list all of the add-in's assemblies and other required files. This file only serves to identify the current version and point to the application manifest associated with that version. Listing 1-3 contains a sample of a deployment manifest.

Listing 1-3. *A Sample VSTO Deployment Manifest*

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1
  assembly.adaptive.xsd" manifestVersion="1.0" xmlns:asmv3=
"urn:schemas-microsoft-com:asm.v3" xmlns:dsig=
http://www.w3.org/2000/09/xmldsig#" xmlns:co.v1="urn:schemas-microsoft-
com:clickonce.v1" xmlns="urn:schemas-microsoft-com:asm.v2" xmlns:asmv1=
"urn:schemas-microsoft-com:asm.v1" xmlns:asmv2=
"urn:schemas-microsoft-com:asm.v2" xmlns:xrml=
"urn:mpeg:mpeg21:2003:01-REL-R-NS" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance">
<assemblyIdentity name="OutlookAddIn1.vsto" version="1.0.0.0"
publicKeyToken="b213866c46f1a387" language="en"
processorArchitecture="msil" xmlns="urn:schemas-microsoft-com:asm.v1" />
<description asmv2:publisher="Microsoft" asmv2:product="OutlookAddIn1"
xmlns="urn:schemas-microsoft-com:asm.v1" />
<deployment install="false" mapFileExtensions="true" />
<dependency>
<dependentAssembly dependencyType="install" codebase=
"OutlookAddIn1_1_0_0_0\OutlookAddIn1.dll.manifest" size="11993">
<assemblyIdentity name="OutlookAddIn1.dll" version="1.0.0.0"
publicKeyToken="b213866c46f1a387"
language="en" processorArchitecture="msil" type="win32" />
<hash>
<dsig:Transforms>
<dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
</dsig:Transforms>
<dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<dsig:DigestValue>9oVorZqzS8vRJFnPSCNnjHIiXo4=</dsig:DigestValue>
</hash>
</dependentAssembly>
</dependency>
```

CLICKONCE AND VSTO

Both the application and the deployment manifests are based on the ClickOnce manifest schemas. However, they do not support the full ClickOnce schemas and ignore any nonsupported element.

To understand how the two manifest files work together at runtime, see the "VSTO Runtime Overview" section later in this chapter.

Windows Registry Entries Office utilizes the Windows registry to discover if any add-ins should be loaded when an Office applications loads. In order for an Office application to know of an add-in, a special key must exist that tells the host how to locate the add-in. When the host application loads, it checks the registry for any settings under this node to determine the add-in's entry point. The data entered here depends on the targeted version of Office, but the registry node for each version is `HKEY_CURRENT_USER\Software\Microsoft\Office\APPLICATION NAME\Addins\ADDIN ID` node.

For Office 2007, the registry stores the location of the add-in's deployment manifest. The name of the key is `Manifest`, and it stores the location of the add-in's deployment manifest.

For Office 2003, both the `Manifest Name` and `Manifest Location` values must exist in the registry. In addition, the manifest in this case is the application manifest, not the deployment manifest.

AS WITH ALL THINGS IN LIFE, EXCEPTIONS EXIST

I have no idea why Visio's add-in registry key does not share the same location as its Office brethren. The registry key for Visio add-ins is stored in the HKEY_CURRENT_USER\Software\Microsoft\Visio\APPLICATION NAME\Addins*ADDIN ID* node.

Also, an Office 2003 add-in can become a machine-level add-in if you change the first node to HKEY_LOCAL_MACHINE. For Office 2007, this strategy causes Office to ignore the add-in. That said, strategies exist for creating a machine-level add-in installation. If enabling your add-in for use by all users of a single machine is important to your Office 2007 development efforts, I suggest that you read "Deploying your VSTO Add-In to All Users" by Misha Shneerson (`http://blogs.msdn.com/mshneer/archive/2007/09/04/deploying-your-vsto-add-in-to-all-users-part-i.aspx`).

Understanding Document Add-Ins

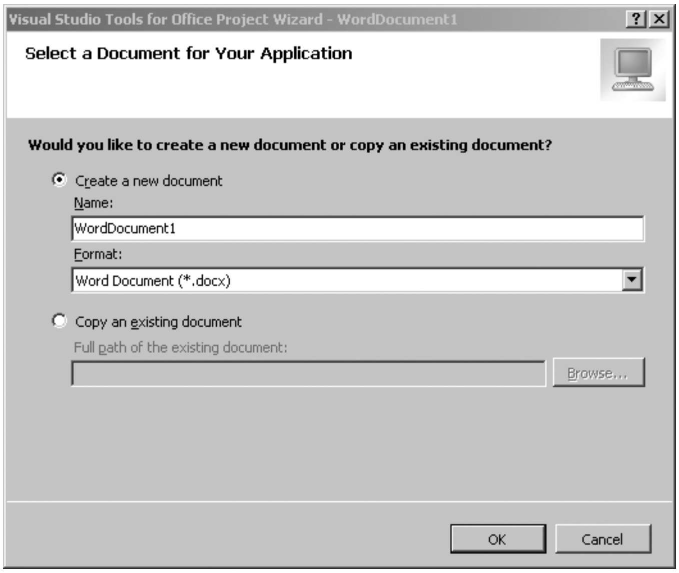
Document add-ins are managed assemblies attached to an Office document (Excel or Word only). The code is not embedded as part of the document but instead resides in a separate location such as a URL, a network share, or another location available to the end user. This scenario allows for a development experience that resembles Windows Forms development. Here, the form canvas is the Word or Excel document. The managed code resides *behind* the document and provides for true separation between your code and the form. When a user opens the document, the Office application locates and loads the assembly, enabling your code to respond to the document's events and manipulate its objects. Just like application add-ins, document level add-ins utilize the Office PIAs to communicate with the COM-based types found within the Office object model.

For the sake of this discussion, I will walk through a Word document add-in. The structure is largely the same for a Word and Excel object. Where they diverge, I will point out their differences, but suffice it to say, the developer model for each is similar. If you understand one, you understand the other.

Creating a Word Document Add-In

For this sample, you will create a Word document VSTO project in Visual Studio. Complete the following steps to build the add-in:

1. Open Visual Studio and open the New Project dialog box.
2. Select the Word Document project template and click the OK button.
3. The Office Project Wizard displays to guide you through creating the add-in project's shell. The first step requires you to either create a new Word document or specify an existing one (see Figure 1-12).



Visual Studio Tools for Office Project Wizard - WordDocument1

Select a Document for Your Application

Would you like to create a new document or copy an existing document?

[✓ Create a new document]

Name: WordDocument1

Format: Word Document (*.docx)

Copy an existing document

Full path of the existing document:

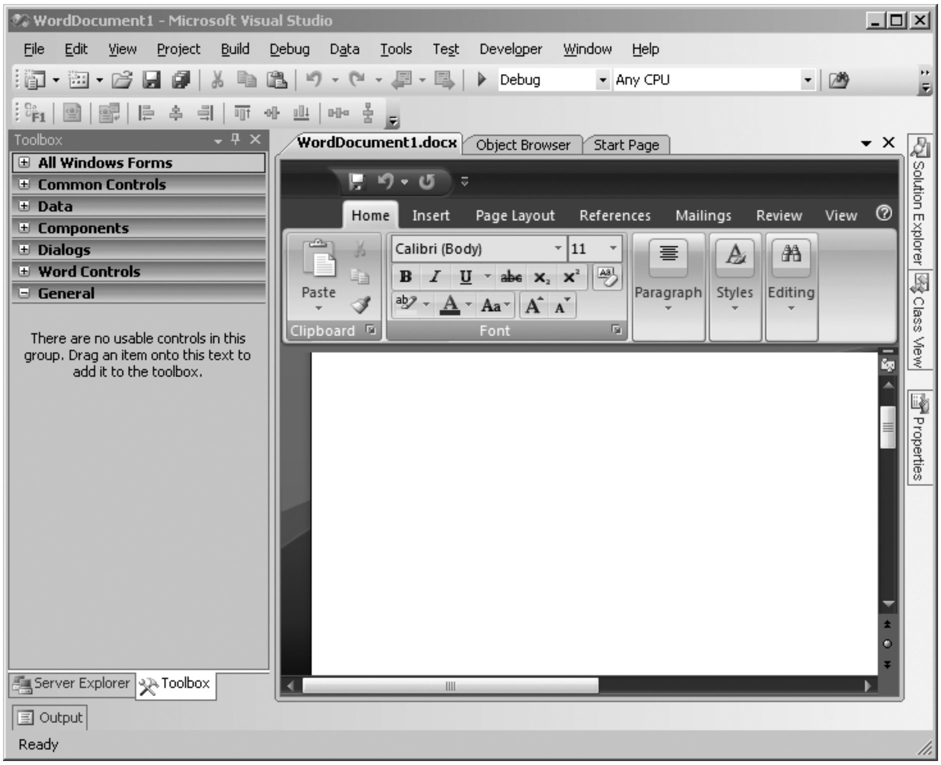
Figure 1-12. *Selecting a target document for the add-in*

If you had an existing document in use by your organization already, you could copy it to the add-in project by selecting the second option and specifying its path. The original would remain where it is, and you would code against a copy.

4. For our purposes, we want to use a new document, so keep the selection with "Create a new document" and click OK. Visual Studio will create the project.

Once created, Visual Studio will display the `WordDocument.docx` file within a Visual Studio designer (see Figure 1-13). This designer is one of the killer features of developing with VSTO, as it enables Office development from the comfy confines of Visual Studio. You can draw

WinForm controls, data controls, common controls, and more. Anything available to you in a Windows Forms application is available to you as you build document add-ins.



There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Figure 1-13. *The VSTO document designer for a Word document*

Main Add-In Components

Just like the application add-ins, a VSTO document add-in contains a single stubbed-out class that acts as the entry point for the add-in and serves as the starting point of your customization efforts. Unlike the application add-ins, document add-ins contain either a Word or Excel document (these could be either document or template types) that acts as the target, or *host document.*

The Host Document (WordDocument1.docx) The host document is the target document that the add-in attaches to during runtime. It has the `ThisDocument` class attached as a child node in Solution Explorer (see Figure 1-14). You can perform all normal Word document functions to build out the document structure as needed. This includes applying formatting, inserting tables, and attaching the XML structure. Since the Ribbon is available within the designer, all Word menus are available.

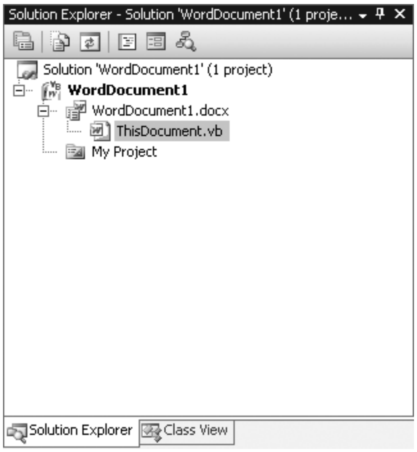

Solution Explorer - Solution 'WordDocument1' (1 proje...
Solution 'WordDocument1' (1 project)
WordDocument1
WordDocument1.docx
ThisDocument.vb
My Project
Solution Explorer
Class View

Figure 1-14. *The WordDocument1.docx node with ThisDocument.vb attached*

The ThisDocument Class The `ThisDocument` class functions almost exactly like the application add-ins' `ThisAddin` class (see Listing 1-4). It contains two methods for startup and shutdown. This class functions exactly like `ThisAddin`, except it executes against a document. Therefore, events in this class occur after the host application has already loaded and fired its `Startup` and `Shutdown` events. Another key difference is the object type passed in the `sender` argument. For document add-ins, `sender` is a reference to the host application (Word or Excel).

Listing 1-4. *A Stubbed-Out ThisDocument Class*

```
Public Class ThisDocument

    Private Sub ThisDocument_Startup(ByVal sender As Object, ByVal e As _
      System.EventArgs) Handles Me.Startup

    End Sub

    Private Sub ThisDocument_Shutdown(ByVal sender As Object, ByVal e As _
      System.EventArgs) Handles Me.Shutdown

    End Sub

End Class
```

These events are the ideal location for code that initializes the object needed by your add-in—for example, menu customizations, data objects, user interfaces, and project-level references to the host application object.

EXCEL'S MAIN DOCUMENT ADD-IN OBJECTS

Following Excel's nomenclature, the main class's name is `ThisWorkbook`, which represents the entire Excel file and all its contents. In addition, the `Worksheet` class represents a single tab included in `ThisWorkbook`. Each of these classes has `Startup` and `Shutdown` events that work as you would expect.

Upon starting up the add-in, the `ThisWorkbook.Startup` method executes. Then, each individual worksheet within the Excel workbook executes its `Worksheet.Startup` method. The sequence follows the order of worksheets in the physical file—for example, Sheet1, Sheet2, Sheet3, and so on. The shutdown sequence follows the same order as the `Startup` method.

Custom Document Properties and Manifest Files Instead of registry entries, document add-ins rely on custom document properties named `_AssemblyName` and `_AssemblyLocation`. The `_AssemblyName` stores a Globally Unique Identifier (GUID) string that refers to the entry point of the VSTO loader. The `_AssemblyLocation` stores the location of the deployment manifest. There isn't anything too special about these properties except for the data they contain. Otherwise, they are just typical document properties that you could create and edit as you would any custom document property. The good news is VSTO handles the creation and editing for you as part of the solution publishing process.

Document add-ins also utilize deployment and application manifests. These function in the same manner as application add-ins with the deployment manifest containing version information and the location of the application manifest. The application manifest contains all the information required to install and run the add-in.

VSTO Runtime Overview

The VSTO runtime performs two functions. First, it provides the set of classes used to extend the host application and host documents with your custom code. The classes included in the runtime are managed assemblies. These assemblies handle all communication between your code and the COM-based Office applications. Assemblies for error handling, caching data offline, and hosting controls in Office documents, as well as those providing VSTO's internal functions, are all part of the runtime. This is true for both application-level and document-level add-ins. (For full details on every VSTO assembly, check out MSDN at `http://msdn2.microsoft.com/en-us/library/bb608603(VS.90).aspx`.)

Second, the runtime includes a runtime loader that loads the correct version of the VSTO runtime for the executing Office application. For Office 2007–based projects, VSTO utilizes version 3.0 of the VSTO runtime (`VSTOEE.dll`). Projects targeting Office 2003, however, load the VSTO 2005 SE runtime. That said, solutions originally built for Office 2003 will load in Office 2007, as the VSTO 2005 SE runtime is supported by Office 2007 applications.

In addition to the runtime loader, the VSTO runtime has a solution loader (`VSTOLoader.dll`) whose job it is to load the solution assembly. The solution loader performs numerous tasks as outlined.

Implementing the IDTExtensibility2 Interface

The `IDTExtensibility2` is the COM add-in interface introduced with Office 2000. This interface is still part of the foundation of Office add-ins no matter which technology (Visual Basic 6, .NET, VSTO, Delphi, et al.) you use to develop them. The fact is `IDTExtensibility2` (`http://msdn2.microsoft.com/en-us/library/extensibility.idtextensibility2(VS.80).aspx`) must be implemented for an Office add-in, and the solution loader handles this for you.

Implementing the IManagedAddin Interface

`IManagedAddin` is a new interface with Office 2007 applications that you must implement for any assembly that loads a managed add-in with Office 2007. The `IManagedAddin` (`http://msdn2.microsoft.com/en-us/library/aa942112(VS.90).aspx`) interface exists specifically to work with the VSTO runtime, but since it is an interface, you could theoretically create your own add-in loader.

Creating a Separate Application Domain for the Add-In

One of the main criticisms of Office add-ins built with older technologies (i.e., COM add-ins) was due to the fact that Office add-ins all shared the same application domain. This meant that if one add-in misbehaved and corrupted in memory space, then all other loaded add-ins were taken out as well. VSTO corrects this flaw by loading each VSTO add-in into a separate application domain. This ensures that your solidly architected, well-behaved add-in will remain unaffected by the lesser-built add-ins.

Checking Security Levels

VSTO applications run under a strict model of full trust by the Global Assembly Cache (GAC) in order to execute. The solution loader checks the GAC to determine if the add-in has the appropriate level of permissions to run from its location.

Summary

VSTO is a tool for professional developers building enterprise-level applications with Office 2007. An ideal example of when you would use VSTO is with a business process, where the data within an Office application or document represents an actual business object, such as an invoice, a proposal, or a work order.

VSTO implements two models for building Office add-ins: application-level add-ins and document-level add-ins. Although these two models are similar, their intended use is different. Application-level add-ins attach to the targeted host Office application object and can access all documents and objects. The main class is `ThisApplication`. Document-level add-ins attach to a single document and work with objects and data within the document. The main class is either `ThisDocument` or `ThisWorkbook` (for Word or Excel, respectively).

VSTO provides several features that make building applications with Office more enjoyable. These features bridge the divide between Office and Visual Studio and enable you to work with Office in a manner similar to building Windows Forms–based applications.