Office Solutions Architecture Overview

Visual Studio 2005

Visual Studio Tools for Office enables you to create two types of solutions: document-level customizations and application-level add-ins. They have the following configurations:

- Document-level customizations consist of a managed code assembly that is attached to a Microsoft Office Word 2003 document or Microsoft Office Excel 2003 workbook.
- Application-level add-ins consist of a managed code assembly that runs as an add-in in a Microsoft Office application.

The types of solutions you can create depend on which combination of Microsoft Visual Studio 2005 Tools for the Microsoft Office System (VSTO 2005) or Microsoft Visual Studio 2005 Tools for the 2007 Microsoft Office System (VSTO 2005 SE) you have installed. For more information, see Features Available by Product Combination.

Document-Level Customizations

In their most basic form, customizations that are created using VSTO 2005 consist of two files: a Word document or Excel workbook, and an assembly that Visual Studio compiles as a .dll file. This section describes the typical architecture of these solutions and how they function from the perspective of the developer and the end user.

Typical Architecture

In a document-level solution, the assembly is linked to the document but stored separately. A document or workbook with a linked managed code assembly is said to have managed code extensions. For more information, see Assemblies in Office Projects Overview.

The Word document or Excel workbook has an invisible control embedded in it called the Runtime Storage Control. The Runtime Storage Control stores cached data and an application manifest that contains the location of the assembly. For more information, see Runtime Storage Control Overview.

Steps Involving the Developer and Designer at Design Time

These are the steps involved in the design-time experience from the point of view of the developer and (optionally) designer:

1. The developer creates a document-level project in Visual Studio. The project includes the document and the assembly that runs behind the document. The document might already exist (perhaps created by a designer), or a new document can be created along with the project. For more information, see How to: Create Visual Studio Tools for Office Projects.

2. The designer (either the developer who creates the project or someone else) creates the final look and feel of the document for the end user.

Steps Involving the End User at Run Time

These are the steps involved in the run-time experience from the point of view of the end user:

1. The end user opens a customized document or workbook.
2. The document or workbook loads the compiled assembly.
3. The assembly responds to events in the document or workbook.

Developer and End User Perspective Compared

Because the developer works primarily in Visual Studio, and the end user works in Microsoft Office 2003, there are two ways of understanding Word and Excel solutions that use managed code extensions.

Developer's Perspective	End User's Perspective
Using Visual Studio, the developer writes code that is accessible to Word and Excel. Although it might seem that the developer is creating an executable file that runs Word or Excel, the process actually works the other way around. The document is associated with an assembly and contains a pointer to that assembly. When the document opens, Word or Excel locates the assembly and runs the code in response to all handled events.	Those who use the solution simply open the document or workbook (or create a new document from a template) just as they would open any other Office file. The assembly provides customizations in the document or workbook such as automatically populating it with current data, or showing a dialog box to request information. The code performs these actions without the user knowing that anything is different from any other Office document.

How Managed Code Works with Office Documents

Word and Excel contain a loading mechanism called the Visual Studio Tools for Office loader. When Word documents and Excel workbooks that contain managed code extensions are opened, the Visual Studio Tools for Office loader in Word and Excel starts the common language runtime (CLR) and loads the assembly. The assembly can capture events that are raised in the document or workbook.

The CLR enables the use of managed code that is written in a language supported by the Microsoft .NET Framework. In your solution, you can do the following:

- Respond to events that are raised in the document, workbook, or the Office application itself (for instance, when a user clicks a menu item).
- Write code against the object model to automate the Office application's features.

The managed code communicates with the Office COM components through the primary interop assembly (PIA) in your Word or Excel project. For more information about primary interop assemblies, see Office Primary Interop Assemblies and Office Solutions Development Overview.

When the Document is Opened

The following steps occur when an Office document with managed code extensions is opened:

1. The Visual Studio Tools for Office loader in Word or Excel checks the custom document properties to see if there are managed code extensions associated with the document. For more information, see Custom Document Properties Overview.
2. If there are managed code extensions, the Visual Studio Tools for Office loader starts the Visual Studio Tools for Office runtime.
3. The runtime creates an application domain and sets policy for the application domain not to trust the My Computer Zone.
4. The Microsoft .NET Framework validates the evidence presented by the assembly against the policy. If it fails, an error is raised. If it passes, the process continues.
5. The runtime checks for assembly updates, using the application manifest and deployment manifest. If any updates are necessary, they are performed now. For more information, see Application and Deployment Manifests in Office Solutions.
6. The runtime loads the assembly into the application domain.
7. The runtime runs the assembly.

Visual Studio Tools for Office Loader and Application Domains

The Visual Studio Tools for Office loader in Word and Excel is also used by other managed code solution types, including managed smart documents and smart tags. The loader creates a separate application domain for each document. Having a separate application domain for each document increases security, and also ensures that the when the document is closed, all the code is shut down. For more information about application domains, see Application Domains Overview.

The managed code solution types are designed to work in a single application domain around a single document. They are not designed for cross-document communication.

Supported Document Formats for Customizations

The following formats are supported in document-level solutions:

Microsoft Office Word 2003

- Word Document (.doc)
- Word Template (.dot)
- WordprocessingML (.xml)

Note

VSTO 2005 does not include a project template for creating a solution based on a WordprocessingML file. However, VSTO 2005 does support attaching managed code extensions to existing WordprocessingML files. For more information, see How to: Attach Managed Code Extensions to Documents.

Microsoft Office Excel 2003

- Excel Workbook (.xls)
- Excel Template (.xlt)

If you base your solution on a Word or Excel document that was originally created in an earlier version of Word or Excel, VSTO 2005 will update the format to that of a Word 2003 or Excel 2003 document.

You should design managed code extensions only for documents in the supported formats. Otherwise, certain events might not be raised when the document opens in the application. For example, the Open event is not raised when you use managed code extensions with workbooks saved in the Excel XML Spreadsheet format or in the Web Page (.htm; .html) format.

Compatibility with Word 2007 and Excel 2007

If a user opens a Word 2003 or Excel 2003 customization on a computer that has Microsoft Office Word 2007 or Microsoft Office Excel 2007 installed, the customization will work in most cases. However, Excel 2003 customizations that have ListObject controls might not work as expected.

To make sure that existing Excel 2003 customizations continue to work on computers that have Excel 2007 installed, you must install the VSTO 2005 SE version of the Visual Studio Tools for Office runtime on the end user computer. This version enables Excel customizations that have **ListObject** controls to work as expected when the workbook is opened in Excel 2007. For more information about the Visual Studio Tools for Office runtime, see How to: Prepare End User Computers to Run Office Solutions and How to: Install the Visual Studio Tools for Office Runtime.

Note

Word 2003 and Excel 2003 customizations can be opened in Word 2007 and Excel 2007 only if the document format is not changed. If the user converts the document to one of the Microsoft Office Open XML Formats, the customization might not work as expected. For more information about the Microsoft Office Open XML Formats, see "Introducing the Office (2007) Open XML File Formats" (http://go.microsoft.com/fwlink/?LinkId=72040).

Application-Level Add-ins

Add-ins that are created by using Visual Studio Tools for Office consist of an assembly that is loaded into a Microsoft Office application as an add-in. Add-ins that are created by using Visual Studio Tools for Office have access to the Microsoft .NET Framework as well as the application's object model. When you build an add-in project, Visual Studio compiles the assembly into a .dll file and creates a separate application manifest file. The application manifest points to the assembly, or to the deployment manifest if the solution uses one. For more information, see Application and Deployment Manifests in Office Solutions.

How Managed Code Works with Add-ins

Visual Studio Tools for Office provides a loader for add-ins that are created by using Visual Studio Tools for Office. This loader is named AddinLoader.dll. When a user starts the Microsoft Office application, this loader starts the common language runtime (CLR) and the Visual Studio Tools for Office runtime, and then loads the add-in assembly. The assembly can capture events that are raised in the application.

Note

To run add-ins that are created by using VSTO 2005 SE, end user computers must have the VSTO 2005 SE version of the Visual Studio Tools for Office runtime on the end user computer. For more information, see How to: Prepare End User Computers to Run Office Solutions and How to: Install the Visual Studio Tools for Office Runtime.

The CLR enables the use of managed code that is written in a language supported by the Microsoft .NET Framework. In your solution, you can do the following:

- Respond to events that are raised in the application itself (for instance, when a user clicks a menu item).
- Write code against the object model to automate the application.

After the assembly has been loaded, the add-in has access to the application's objects, such as documents or mail. The managed code communicates with the application's COM components through the primary interop assembly in your add-in project. For more information about primary interop assemblies, see Office Primary Interop Assemblies and Office Solutions Development Overview.

How an Add-in Is Loaded

The following steps occur when a user starts the application:

1. The application checks a set of registry keys for entries that identify add-ins created using Visual Studio Tools for Office. These registry entries provide information about the add-in and point to the application manifest for the add-in. For more information about these registry entries, see Deploying Application-Level Add-ins.
2. If the application finds registry entries for an add-in created using Visual Studio Tools for Office, the application starts AddinLoader.dll, which is the loader for add-ins created using Visual Studio Tools for Office.

One of the registry entries specifies the location of the application manifest. The application manifest points to the add-in assemblies, or to the deployment manifest, if the solution uses one.

3. The loader starts the Visual Studio Tools for Office runtime, and sends the contents of the application manifest to the Visual Studio Tools for Office runtime.
4. The runtime creates an application domain and sets policy on the application domain not to trust the My Computer Zone.
5. The Microsoft .NET Framework validates the evidence presented by the assembly against the policy of the application domain. If it fails, an error is raised. If it passes, the process continues.
6. The runtime checks for assembly updates, using the application manifest and deployment manifest. If any updates are necessary, they are performed now. For more information, see Application and Deployment Manifests in Office Solutions.
7. The runtime loads the assembly into the application domain.
8. The runtime runs the assembly.

Compatibility with Different Versions of Microsoft Office

VSTO 2005 provides a project template for Outlook 2003 add-ins. Add-ins that you create by using this project template can be loaded only by Outlook 2003.

VSTO 2005 SE provides separate project templates for Microsoft Office 2003 add-ins and 2007 Microsoft Office add-ins. However, add-ins that you create by using these project templates can be loaded by later versions of Microsoft Office, and in some cases, earlier versions.

The following table describes the scenarios for using add-ins created by using VSTO 2005 SE with other versions of Microsoft Office.

	Microsoft Office 2003 project templates	**2007 Microsoft Office project templates**
Does the add-in work with later versions of Microsoft Office?	Yes.	Forward compatibility is part of the design.
Does the add-in work with earlier versions of Microsoft Office?	No.	These add-ins can be used with Microsoft Office 2003, but not with earlier versions. Also, the add-in must only use features that are available in the 2003 release.

Shutdown Behavior of Outlook 2003 Add-ins

The shutdown process for Outlook 2003 add-ins created using Visual Studio Tools for Office is different from the shutdown process of add-ins that implement the **IDTExtensibility2** interface.

If you are migrating an existing Outlook 2003 add-in that implements the **IDTExtensibility2** interface to Visual Studio Tools for Office, you should be aware of these differences.

If an add-in that implements the **IDTExtensibility2** interface has a reference to one or more Outlook objects, then Outlook 2003 does not call the `OnDisconnection` method of the add-in. If the add-in has object references that are cleaned up only in the `OnDisconnection` method, then Outlook 2003 never calls the `OnDisconnection` method. As a result, the add-in is never unloaded, and Outlook 2003 never shuts down.

Note

The behavior described in this section does not apply to Outlook 2007 add-ins. Outlook 2007 always calls the `OnDisconnection` method in an add-in, even if the add-in still has references to Outlook objects.

Outlook 2003 add-ins created using Visual Studio Tools for Office are unloaded in a way that avoids this potential problem. The Visual Studio Tools for Office runtime raises the **Microsoft.Office.Tools.Outlook.Application.Shutdown** event of the add-in and unloads the application domain of the add-in if the add-in does not have references to any **Microsoft.Office.Interop.Outlook.Explorer** or **Microsoft.Office.Interop.Outlook.Inspector** objects when one of the following occurs:

- The **Microsoft.Office.Tools.Outlook.Application.Quit** method is called.
- The **Close** event of an **Microsoft.Office.Interop.Outlook.Explorer** or **Microsoft.Office.Interop.Outlook.Inspector** is raised.

When the application domain is unloaded, all outstanding references to other Outlook objects are cleaned up, and Outlook 2003 can then shut down the add-in and close.

If you are migrating an existing Outlook 2003 add-in that implements the **IDTExtensibility2** interface to Visual Studio Tools for Office, and your add-in contains code that works around the potential shutdown problem and ensures that Outlook unloads your add-in, you should remove this code from your add-in. Otherwise, this code might conflict with the shutdown process for Outlook 2003 add-ins created by using Visual Studio Tools for Office, or your add-in might be unloaded prematurely.

See Also

Concepts

Custom Document Properties Overview
Data Model Overview
Runtime Storage Control Overview

Other Resources