1

Visual Studio Tools for Office

Visual Studio Tools for Office is the new kid on the block for harnessing the power and functionality of Microsoft Office technology. And Microsoft is investing heavily in getting the word out. The marketing push forms part of an overall strategy to secure the dominance of Office technology and Microsoft Office products on the desktop while also providing a solid platform for developing enterprise level applications based on Microsoft Office technology.

At its core, Visual Studio Tools for Office (VSTO, pronounced “visto”) is simply a platform that allows Microsoft Office documents to execute code wrapped in a .NET assembly. It is certainly not the only technology available for developing applications based on Microsoft Office. In fact, web developers have been using classic ASP and regular COM Interop to build Office based software for years. But VSTO has been designed to make development of such applications easier and more reliable than the current approaches available today.

What’s New in VSTO?

VSTO has been completely reworked for .NET 2005. The rework includes tighter integration with Excel as well as the promotion of native Excel and Word binaries to bona fide .NET objects. These new objects expose more functionality than what is currently available through the aging Microsoft Office Application Programming Interface (API). Also, these new .NET objects resolve many of the memory management issues that plagued the previous Microsoft Office Component Object Modeler (COM) components.

VSTO sports a new development model, a novel approach to building Office-based applications that offers significant advantages when compared to previous models. The Classic ASP model that predates VSTO was based on a platform where content and presentation were mangled together to construct applications.

One advantage is that there is more room to scale because business logic can be cleanly separated from the presentation of data. Another advantage is that Office automation can occur on the web application server in a thread-safe way without the need to write code to adjust the presentation of the data. A third advantage is that the new model allows executing code to exploit the rich feature set of the .NET Framework Class Library. The downside, and there always is one, is that these improvements come at the expense of an increased learning curve, and the need to write code to target a new model.

To help drive the acceptance of VSTO, Microsoft is using the .NET framework as the vehicle to move the new Office development platform into the midst of the .NET developer population. The marketing strategy is geared toward enticing .NET developers into building applications based on Microsoft Office technology. Hopefully, if .NET developers find it easy to write Office applications, then they will.

Unfortunately, the marketing picture may not always present the whole story. This book will peel back the marketing hype baked into Visual Studio Tools for Office to show you what works well and what does not. The book also intends to show you, in a very practical way, how to build and deliver industrial strength applications based on Microsoft Office technology. The material in the next few sections will compare and contrast VSTO with some current approaches for developing Microsoft Office based applications so you can decide whether or not VSTO is the right tool for the job.

VSTO Architecture

VSTO is not entirely new. The first .NET tool suite appeared with Visual Studio 2003. Before that, savvy developers simply used COM Interop and Visual Basic for Applications (VBA) to etch out Office based software. The effort required was considerable due in part to the limitations of the COM based VBA programming model. The chosen few who succeeded in building Office applications took on the crown of "Office Developers".

Office developers used any means necessary — macros, hacks, Interop — to coax performance and stability out of these crude, unforgiving environments. Along the way, the approach to building Office applications got a bad rap because hacks and macros cultivated an environment for malicious software to flourish even today, embedded macros in applications still cause all sorts of security concerns. Microsoft is hoping to turn a new page with Visual Studio Tools for Office. VSTO is safer and more secure than previous technologies used to build Microsoft Office applications. In fact, VSTO can build fully operational applications without macros.

VSTO.NET addresses the safety and security issues in a couple of ways. Firstly, the new document/view architecture allows code to be separated from presentation. The code that runs behind the Office document can have security restrictions applied to it so that it is guaranteed safe and its authenticity can be verified before being run. Contrast this to the macro approach where embedded macros are permitted to execute on the user system unbounded. The new VSTO approach provides developers with more freedom to build secure, scalable applications than what was possible before. In that sense, VSTO carves out new ground, and it does so in a way that eliminates the need to write trick code.

Office 2003 sports a new surface for embedding objects into the document — the actions pane. The actions pane reduces the effort that is required to host controls on the document surface. Controls can now be hosted cleanly and efficiently on the document. This new approach leads to well-behaved applications and side-steps the need to implement hacks to achieve the impossible.

VSTO also hosts the Excel and Word objects inside the Microsoft Visual Studio designer. When you open a new Excel project in Visual Studio 2005, the Excel spreadsheet appears in the Microsoft Visual Studio designer. The Excel object is able to respond to design-time settings and is fully customizable. By hosting Excel and Word on the document surface, debugging the application is a lot more manageable.

VSTO exposes bona fide .NET objects that seamlessly integrate with the .NET framework. These objects expose more functionality and are cleaner to incorporate in managed software development efforts. The approach also stems the tide of memory exhaustion issues and component instability that plagued the previous generation of Office software.

VSTO offers n-tier architecture for Microsoft Office document development. The model is tiered into the user interface, client interface, server components, and backend data.

User Interface

The user interface tier functions as the front-end of the document. It handles user input, validation, and document navigation. This is all handled through the same Excel, Word, InfoPath, or Outlook interface that users are accustomed to. By basing the user interface on popular, proven front-end, very little end-user training is required for those already familiar with Microsoft Office functionality.

Client Interface

The client component is made up of a combination of the .NET assembly developed for the new VSTO-based application and the VSTO.NET engine provided by Visual Studio Tools for Office System. The engine floats on top of the Common Language Runtime infrastructure and allows the Microsoft Office document to execute managed code inside the .NET assembly.

Server Component

Every VSTO-based application has direct access to VSTO's new server component class. This server component piece functions as a document processing layer that is able to create and manipulate Microsoft Office documents without the need to automate Microsoft Office or Microsoft Excel on a web server. In case you missed it, server-side processing of Microsoft Office documents have traditionally been the weakest link in software applications that run in a web-server environment. The new server component strengthens this link by allowing software applications based on Microsoft Office to scale correctly while making prudent use of web-server resources.

The server component architecture incorporates the use of XML in its design and implementation. The data that forms part of the Microsoft Office document being manipulated on the web-server is now stored independently in XML data islands. This is a major departure from all previous Microsoft Office based implementations where the application data was stored inside the Microsoft Office document itself.

The XML used to store the application data is based on the W3C XML open standard. Data storage is no longer proprietary as in some previous technologies such as the Microsoft Office Web Components. The business implication is that data sources can be as diverse as the corporate entity allows it to be. And, data can now flow freely between Microsoft Office and third party applications built to take advantage of open standards for data exchange such as web services and remoting platforms.

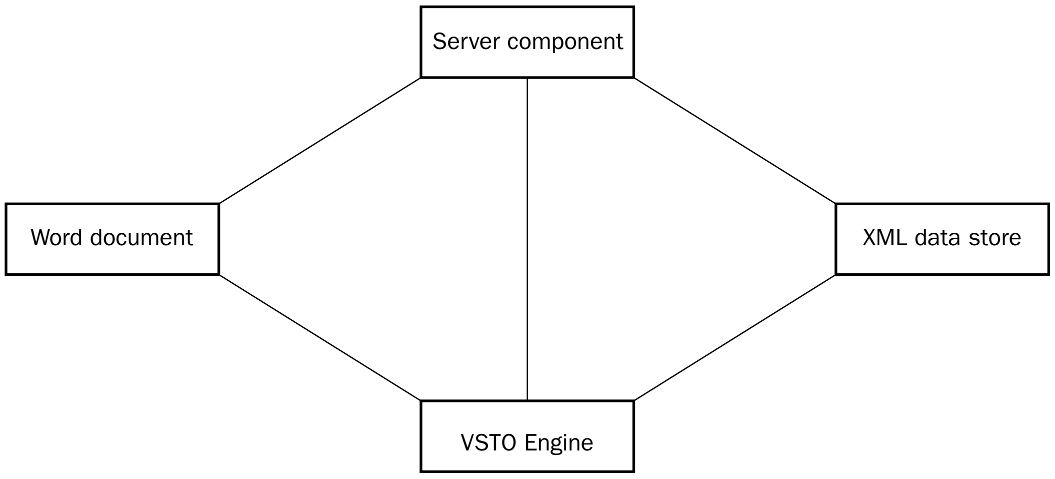


Figure 1-1

From Figure 1-1, you can see that the new architecture is flexible enough to leverage the VSTO engine for new applications based on Visual Studio Tools for Office. In other cases, the architecture allows the developer the flexibility to build applications that perform a legacy role; that is, documents may be opened without firing code housed in the .NET assembly. The new architecture also allows VSTO based applications to be written and run with embedded macros. Although this is discouraged, you should understand that there are no architectural fences that prevent such an implementation.

The VSTO Package

VSTO offers Excel, Access, Word, InfoPath, and Outlook functionality. The functionality targets the windows platform to include smart client and remoting scenarios. Although VSTO allows users to publish functionality such as spreadsheets and pivot tables on the web, the tool suite isn't specifically designed for full–blown, interactive web applications. Part of the reason lies in the fact that VSTO is primarily built for Windows-based applications.

Still, this book will explore the concept of server-side automation of Microsoft Office applications that provide ways to build code that is scalable and well-behaved when compared with other methods of building application software. Since more and more real world applications are blurring the line between desktop and web platforms, the material presented in later chapters will shed some light on these gray areas so that you can make informed choices when building new application software.

The Visual Studio Tools for Office suite also includes Visual Basic standard edition. The standard edition allows developers to write code to target the Office system. Microsoft Access and SQL server developer edition are also included. These products allow you to develop solutions based on Access and SQL server for the Office system. Finally, Microsoft also offers a separate stand-alone VSTO version that encompasses all the power and productivity of Visual Studio.

VSTO is implemented as an add-in to Visual Studio. The VSTO engine runs in-process in the Visual Studio address space. The VSTO engine depends on Office PIA's in order to communicate with Microsoft Office objects.

About Microsoft Office PIAs

Primary Interop Assemblies (PIA) are .NET wrappers built around existing COM components that allow .NET code to communicate with the COM component. Microsoft strongly recommends that Primary Interop Assemblies are to be used instead of your own auto-generated assemblies. The literature indicates that PIAs can make certain optimizations that may be beneficial to the application. While this may be true in some cases, it certainly is not always true. In fact, the dominant reason for using PIAs is that it helps combat an insidious assembly version issue in applications built with Microsoft .NET.

Microsoft .NET run-time cannot communicate directly with COM components. When communication with COM components is required, the .NET Common Language Runtime (CLR) requires an adapter or proxy to sit between the .NET assembly and the COM component. This adapter translates .NET requests into COM requests making the automation possible. It's important to understand that the proxy or Runtime Callable Wrapper (RCW) does not re-implement functionality in the COM component. It simply makes the type information inside the COM object available to the calling assembly so communication or marshalling can take place.

Visual Studio is automatically configured to generate RCWs when a reference to a legacy COM component is detected in the Integrated Development Environment (IDE). For small projects that remain in-house, these extra RCWs do not provide a cause for alarm. However, if this software must be deployed to customers, you need to package all the RCWs along with the application for it to function correctly.

Consider a scenario where one vendor makes their RCW available to a customer who already has a wrapper available for a home grown assembly of the same name. The final result will be run-time errors for the mismatched types since the assemblies used to create the wrappers are different but both of the wrappers have the same name.

To put this in perspective, consider a practical example. If company A creates an RCW for a Microsoft Outlook application for third party software called `Outlooklib` that is used by company C, the `Outlooklib` RCW must be deployed to company C. However, company C may be running another application from company B who also has an RCW for Microsoft Outlook called `Outlooklib`. This creates an ugly situation where company C's software starts to misbehave since it may be using the wrong RCW for Microsoft Outlook automation. Bugs of this caliber can be extremely difficult to resolve to say the least.

To prevent such chaos, Microsoft recommends that only one RCW be used per Microsoft Office product. This RCW or Interop assembly would be the primary source of type information for the automation call hence the term Primary Interop Assembly. In the theoretical example provided, company C's software would not misbehave since the RCW provided by company A and company B would be the same; that is, both company A and company B would be using a Microsoft Outlook PIA. Finally, note that this is a recommendation only. PIAs cannot possibly be enforced in the real world.

System Requirements

VSTO offers several benefits, but there is a cost associated with reaping the rewards of this new architecture. VSTO.NET requires the Professional version of Office 2003 Service Pack 1 on the development machine. In the absence of this requirement, the VSTO projects will not work. The requirement also makes it impossible to bind a .NET assembly to third party Excel or Office interfaces. However, Microsoft insists that applications built today based on VSTO.NET will be forward compatible with Office version 12 due out sometime in 2006. The guarantee is based in part on the fact that the data storage model is XML.

The installation packages must be installed in a specific order because there are tight dependencies between these packages. Install Microsoft Office 2003 Professional Edition first. Microsoft Office 2003 Professional Edition is an absolute requirement; VSTO 2005 cannot work with any other Office edition. Follow this install with any updates or service packs that are available. You may download and install the latest updates and service packs for Microsoft Office 2003 by visiting the Microsoft Office website. Then, install Visual Studio.NET 2005 followed by Visual Studio Tools for Office. Finally, install MSDN library so that help documentation is available. Unless you require customized features, you can simply install the products with the default settings.

The total install package can range in size from 1 to 3 gigabytes of space requirements depending on the options you choose to install. If you are challenged for space on your hard drive, you may elect to use the online help documentation instead. It is automatically configured to retrieve help documentation from the F1 short-cut key. After installing the help documentation, you should use the "check for updates" feature of the Microsoft Visual Studio 2005 installation to check for updates and patches. Normally, the entire process may take a couple of hours.

The framework must be installed first in order to create the Global Assembly Cache. Visual Studio Tools for Office uses that Global Assembly Cache to house the Primary Interop Assemblies.

You also need to have the .NET framework installed on the end-user system. While the framework offers significant advantages over legacy COM based frameworks, the requirement is a serious hindrance today. Eventually, as more end-users adopt the framework, the hindrance will fade away.

Though .NET languages are fairly numerous today, VSTO only supports C# and Visual Basic — formerly known as Visual Basic.NET. You may use either language to build applications based on Microsoft Office technology. Visual Studio Tools for Office does not currently support assemblies built from a combination of Visual Basic and C#. To be fair, this limitation is actually imposed by the Common Language Runtime. However, for those not familiar with the underpinnings of the Common Language Runtime (CLR) this lack of flexibility will be blamed on VSTO.

Of the two programming languages, Visual Basic is better for building VSTO applications. There are several reasons for this due in part to the history of the Microsoft Excel and Word components.

C# has no support for optional parameters in method calls. However, the objects that form the core part of the Office system are built on COM and tuned for VBA code. These API's expect optional parameters. Because of the lack of support in C#, special place holders must be passed in for each call. Using these placeholders for each method call can be tedious and demanding. For instance, the document's open method accepts 15 optional parameters. Consider a Visual Basic example.

```
Me.CheckSpelling()
```

Now consider a C# example.

```
this.CheckSpelling(ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing);
```

The PIA's that are installed during VSTO setup enforce a pass by reference semantic on the caller for Microsoft Word development. For C#, the `ref` keyword must be included to satisfy the requirement or a compilation error will be generated. There is no workaround other than to add the `ref` keyword. The arguments also must be variable references. Literals or strings cannot be passed in because these do not satisfy the compiler requirement.

C# does not allow multiple arguments to properties. However, the Excel object model supports such expressions. This is not a problem for Visual Basic because the support has been continued for historic reasons. Range manipulation in particular makes heavy use of multiple arguments to properties. In such instances, you can use the optional methods provided by the Excel PIA in C#.

Visual Basic

```
ThisApplication.Range("A1")
```

C#

```
ThisApplication.get_Range("A1");
```

Although the PIA's have been updated to add substitute properties for C# developers, these properties often do not show up in the intellisense engine of Microsoft Visual Studio.NET 2005. Without solid documentation, C# developers are left holding the short end of the stick.

If you care for a vigorous exercise in typing, you may choose to develop VSTO-based applications using C#. However, if you don't particularly care for the exercise or are running on a lean productivity budget, Visual Basic is the most appropriate choice. This book will present code in both languages for the benefit of all.

Alternatives to the VSTO Office Systems

As mentioned before, VSTO is not the only approach to harnessing the power of Microsoft Office technology in next generation application software, but it offers significant advantages over other approaches as we shall see. In fact, some of these existing approaches have only been tolerated because there was no suitable alternative available to justify the expense of adopting a new approach to building Office applications. Today, with VSTO, the expense of adopting a new approach can be justified in terms improved security, scalability, and reliability.

VBA

Visual Basic for Applications (VBA) was the de facto standard for building applications based on Microsoft Office technology. VBA made it easy to build Office technology because the user interface could be designed and built easily. Underneath the hood, VBA connected directly to the Office API infrastructure. Developers could quickly build and release Office based application software using VBA.

However, one major drawback was that security was an afterthought in the VBA approach. VBA macros can run embedded inside a document, often underneath the security blanket of the application. Executing code cannot usually determine the origin of the macros. For instance, the macro can be embedded by the program or it may be embedded by malicious code.

VBA falls underneath the umbrella of unmanaged code. Unmanaged code extracts a performance penalty when interacting with managed code. VBA also cannot lessen this performance hit by routing calls through a Run-time Callable Wrapper, so the performance penalty is dead weight.

VBA is also a procedural programming language. It cannot take advantage of some of the niceties such as inheritance, interfaces, and polymorphism. These terms aren't just buzzwords, they enable the creation of large, scalable application software.

VBA exposes arcane approaches to error handling. VSTO uses the mature exception handling mechanism in .NET. The transition from clunky error handling constructs in VBA to a polished exception handling surface in .NET enables applications to scale properly and remain well-behaved in abnormal circumstances. For instance, exceptions that are thrown in a remoting call contain the full context of the exception. A VBA implementation would typically involve reading an error code when some exception occurred.

VBA does provide certain advantages that VSTO cannot provide. For instance, VBA allows code to be executed in worksheet functions. VSTO is unable to perform this task. The VBA Office model is more mature than VSTO. Maturity offers certain benefits such as better documentation, developer familiarity and end-user acceptance that tend to make development in VBA a more natural choice than VSTO. VSTO has not yet achieved that comfort level because it is the new kid on the block.

Another important advantage of VBA is that VBA is available on virtually every machine running Microsoft Office 97 or later. VSTO requires installation of some key infrastructure pieces, such as the .NET Framework.

Office Web Components

The Office web components (OWC) may be used both on the desktop and on the web. They exactly mirror the fit and finish of Microsoft Excel. However, this functionality comes at a price. The components are more strictly licensed than VSTO and are only intended to be used in intranet web application scenarios. A suitable book to help you negotiate the pitfalls of Office development for the web is my recent book The Microsoft Office Web Components Black Book with .NET.

The OWC is based on unmanaged code so the .NET framework is not a requirement. It may be downloaded for free and installed on demand. The OWC is light enough to fit inside a browser as an ActiveX control. The components provide the full power and functionality of Microsoft Excel. Other Microsoft Office functionality is not provided.

Excel COM Interop Libraries

COM Interop packages are inherently difficult to program and require deep knowledge of COM implementation. These COM libraries do not always play well together and present all sorts of issues when scalability increases. The COM libraries are limited in the functionality they expose when compared to VSTO and are notably weak in exposing the event model of Excel and Word.

One serious issue with Excel COM automation is that it possesses serious problems in concurrent environments. In fact, instability in these environments most often leads to application failure, memory exhaustion issues, and very unpleasant end-user experiences.

Third Party Products

A number of independent software vendors make products that target Excel and Word. The common theme of these third party application software addresses the short fall of what is available to the customer through Microsoft channels. Usually, these products are not free, contain licensing restrictions, and have shaky support that limits the use and effectiveness of these products in industry.

On the other hand, some third party products such as SoftArtisan's Excel writer are very stable and mature, have a wide customer base, and perform well under load. If you are considering adopting an Office technology, it is important to become familiar with the choices that are available. Build some test applications based on these products and expose them to a variety of real world conditions such as load and stress to see how well they perform before deciding on one product over another.

Disadvantages of VSTO

As with any sizeable piece of real world software, there are advantages and disadvantages. The following sections describe in detail the major drawbacks of VSTO. This is not a complete list. But it represents the thorniest issues that decision makers must understand before making a final decision on adopting a new technology.

.NET Framework Required

One major drawback to VSTO is that it requires the .NET framework to execute. In most corporate environments, end-users simply cannot install application software, much less run-time frameworks. This requirement alone puts VSTO-based applications at a serious disadvantage. In fact, it practically rules out off-the-shelf applications based on VSTO.

Independent Software Vendors are more likely to build application software based on one of the alternatives presented previously simply because these applications are light and are guaranteed to work on Microsoft Windows systems. Even with the framework packaged along with the software, end-users may be skeptical about installing it due to the size of the .NET framework.

Security

VSTO does not implement security natively. Instead, it out-sources its security interests to the .NET framework. As usual, out-sourcing presents a number of obstacles. For the average developer, this is more than a passing inconvenience because some detailed knowledge about the security aspects of the .NET framework is required. This represents another hurdle in the learning process. Code Access Security and its related subtopics that help define .NET security are sufficiently complicated to discourage even some seasoned developers from adopting VSTO as an Office development technology.

The need for security cannot be overstated. The idea is always to prevent malicious applications from subverting the integrity of the user system. Security policies in effect may also protect the user system from misbehaving applications. Web applications and smart clients in particular should be guaranteed to execute in a controlled manner since the user cannot always guarantee that the application is friendly. If you intend to secure your VSTO applications, it is worth your while to invest some time and effort in understanding .NET security. If you must build real world software based on VSTO, you will need to learn about .NET security.

Performance

Much of the hype around VSTO includes claims that VSTO out-performs VBA and COM approaches. That argument is without merit! While performance is not a serious cause for concern, the performance of a VSTO application does lag behind VBA and COM approaches. There are several factors that influence VSTO performance, and it is important to put these factors in perspective if you intend to draw any meaningful conclusion

The .NET start up cost is inherently expensive. Applications written with .NET must incur the overhead of Just-In-Time (JIT) compilation. JIT compilation is a necessary evil and cannot be avoided. However, steps may be taken to reduce the performance expense of JIT compilation.

One approach is to use the `NGEN.exe` application that ships with the .NET framework. This utility forces a JIT compilation of all the methods in the assembly. Since the methods are Jitted, JIT compilation is no longer necessary when the application first starts up. There are several disadvantages to using NGEN. For one, NGEN files can get out-of-sync. Another issue is that the JIT code is no longer optimized for the underlying platform, since it is done offline. A third issue is that NGEN offers no performance gains in server application code.

Another performance factor influencing VSTO-based applications has to do with the expense of calling through the thick layers of automation skin that wrap the Microsoft Office COM objects. VBA, built and optimized to interact with Microsoft Office, has a shorter distance to travel than .NET.

Finally, hosting the Excel and Word objects in the Visual Studio IDE is expensive in terms of resources. VSTO applications have a larger memory footprint than VBA applications. This resource expense bleeds into the application run-time in noticeable ways.

In essence, if you are considering adopting VSTO as a development platform for Microsoft Office technology, it is worth your while both in time and resources to become familiar with the advantages, disadvantages, and idiosyncrasies of this relatively new technology. Otherwise, the investment may not be worth it.

VSTO Automation

Strictly speaking, automation refers to the process of controlling one object with another. In that regard, VSTO is an automation controller whose primary function is to automate the objects that form part of the Microsoft Office System.

While VSTO is mainly concerned with automating Office objects, it is not necessarily limited to just those objects. Automation can occur with any object as long as it exposes the required automation interface. For instance, it is quite feasible for a VSTO based application to consume components developed by third party vendors. However, VSTO automation is particularly well-suited to objects that belong under the umbrella term of Office products. Usually, these objects contain internal optimizations that make them particularly well-disposed to VSTO automation. Unfortunately, products such as Microsoft Outlook Express are not part of the Microsoft Office System and contain no internal support for VSTO.

Products such as Microsoft Access and InfoPath are supported in VSTO as Add-ins. Add-ins can take advantage of special internal pathways that offer some gains in performance over regular COM approaches. However, add-ins are not first-class VSTO citizens.

Automation problems have been addressed in VSTO but there are still steps that need to be taken to ensure that an automation environment is well-behaved. The dominant reason for these extra steps has to do with the fact that the underlying Microsoft Office legacy code is still based on a COM platform. Since these memory models and resource allocations differ dramatically from managed environments, you need to make sure that resource de-allocation has occurred. Later chapters will show exactly how to build these checks and balances into your code approaches.

Office XML Schemas

When developing applications for the internet, Microsoft Excel or Office is not needed on the web application server. The reason for this magic lies in the divorced document/view architecture. This separation, amicable as divorces go, actually allows processing code to operate on the data contained in the document while leaving the view untouched. It is this clever separation that provides opportunities for scaling out applications based on VSTO.

To further enhance application development, Microsoft has released the XML reference schema for Office applications. This release makes the Office XML available to developers. The implication here is that developers are now free to create customized XML documents that are guaranteed compatible with Microsoft Office applications.

In case you missed it, this is a very huge step forward coming from Redmond. Don't get it confused with generosity though, it is a calculated move to ensure the dominance of Microsoft Office products and the stifling of external competition from open-source initiatives nipping at the Office products heels.

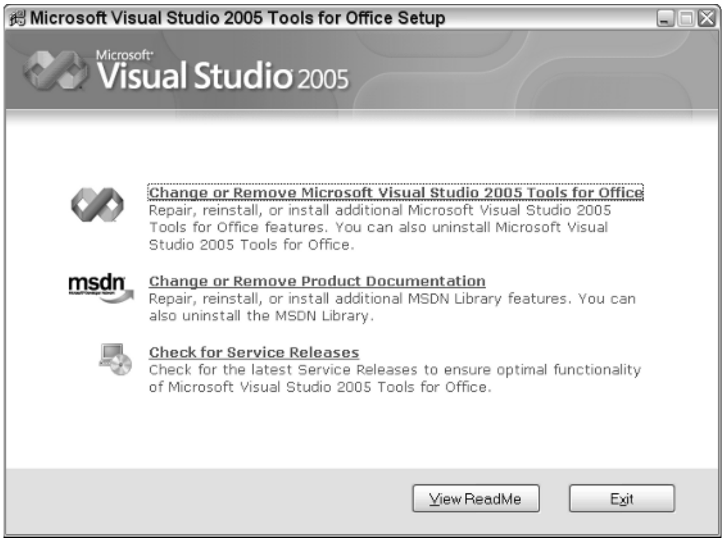
The Microsoft Office XML schema is available for Word, Excel, and Infopath as WordprocessingML, SpreadsheetML, and FormTemplateML, respectively. There is obvious merit to this initiative. It means that client software can now read and write data in a platform independent way taking full use of the advantages and platform affinity of the XML standard. Microsoft hopes that this initiative will have broad adoption releasing developers from the tedium of customized storage access for Office integration.

Installation and deployment

Before installing VSTO, ensure that you have first installed the .NET framework, followed by Visual Studio.NET in that specific order. The following sections show how to install the tools suite on your system.

1. From the install media containing Visual Studio Tools for Office Microsoft System, run the setup application in the root folder, as shown in Figure 1-2.
2. Choose the installation option to install VSTO.

3. Examine the installation log for possible setup problems that occurred during the installation process.
4. Open the Visual Studio Integrated Development Environment.
5. Examine the project types window pane in the left window to make certain that the VSTO templates are available.


Microsoft Visual Studio 2005 Tools for Office Setup

Change or Remove Microsoft Visual Studio 2005 Tools for Office
Repair, reinstall, or install additional Microsoft Visual Studio 2005 Tools for Office features. You can also uninstall Microsoft Visual Studio 2005 Tools for Office.

Change or Remove Product Documentation
Repair, reinstall, or install additional MSDN Library features. You can also uninstall the MSDN Library.

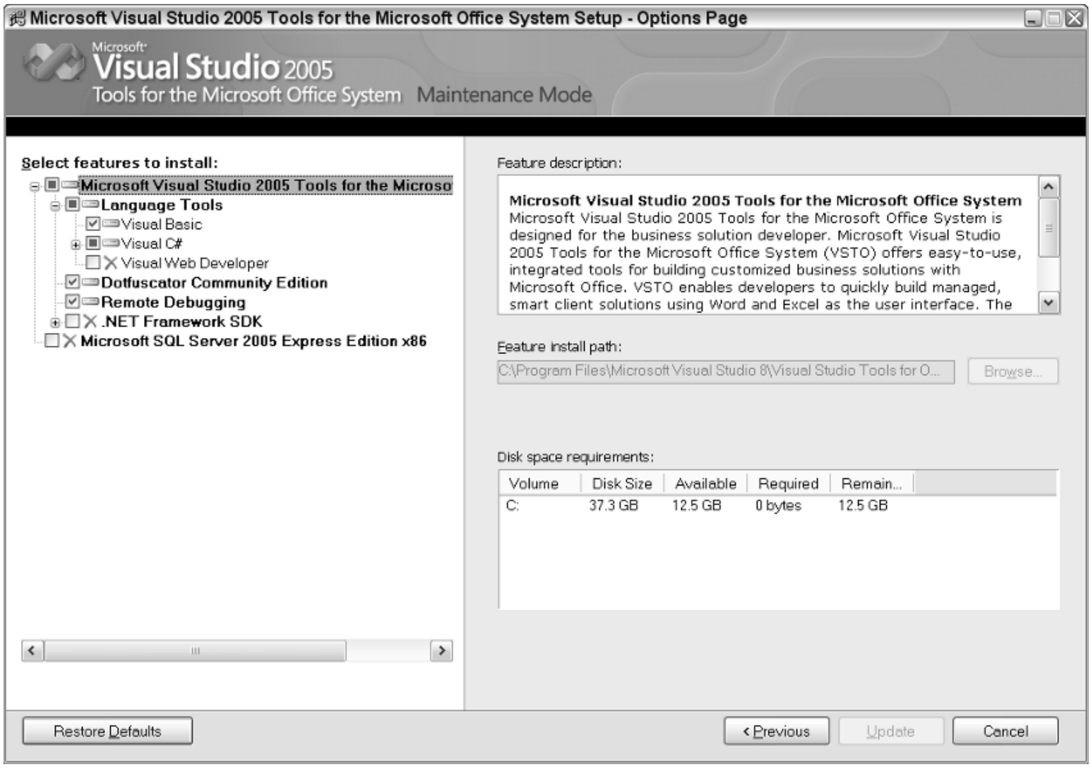
Check for Service Releases
Check for the latest Service Releases to ensure optimal functionality of Microsoft Visual Studio 2005 Tools for Office.

Figure 1-2

Visual Studio Tools for Office depends on the Microsoft Office Primary Interop Assemblies (PIA). During a typical installation, the Office PIA's are automatically installed. However, if you have chosen a custom installation, you may need to manually install the Microsoft Office PIA's.

Follow these few steps to install the necessary PIA's on your system.

1. Run the setup file for the Primary Interop Assembly package.
2. Select the programmability support that you require, as shown in Figure 1-3. The Language Tools node contains the currently supported languages that may be used to build Office applications. The Dotfuscator Community Edition and Remote Debugging selections are optional. The .NET Framework SDK provides tools for working with the .NET framework. Although it is optional, you should install it.
3. Click OK to install the PIA.


Microsoft Visual Studio 2005 Tools for the Microsoft Office System Setup - Options Page
Microsoft Visual Studio 2005
Tools for the Microsoft Office System Maintenance Mode
Select features to install:
Feature description:
Feature install path:
Browse...
Disk space requirements:
Restore Defaults
< Previous
Update
Cancel

- Microsoft Visual Studio 2005 Tools for the Microso
  - Language Tools
    - ✓ Visual Basic
    - Visual C#
    - Visual Web Developer
  - ✓ Dotfuscator Community Edition
  - ✓ Remote Debugging
  - .NET Framework SDK
- Microsoft SQL Server 2005 Express Edition x86

Microsoft Visual Studio 2005 Tools for the Microsoft Office System
Microsoft Visual Studio 2005 Tools for the Microsoft Office System is designed for the business solution developer. Microsoft Visual Studio 2005 Tools for the Microsoft Office System (VSTO) offers easy-to-use, integrated tools for building customized business solutions with Microsoft Office. VSTO enables developers to quickly build managed, smart client solutions using Word and Excel as the user interface. The

C:\Program Files\Microsoft Visual Studio 8\Visual Studio Tools for O...

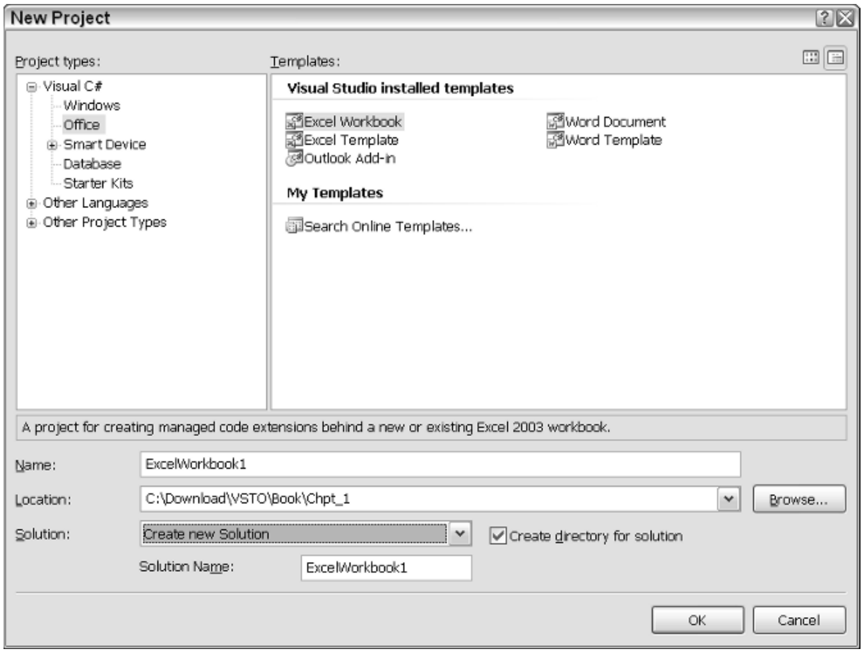
Volume	Disk Size	Available	Required	Remain...
C:	37.3 GB	12.5 GB	0 bytes	12.5 GB

Figure 1-3

Creating VSTO Projects

Once Visual Studio Tools for Office is installed correctly, you may create a new project based on VSTO. Follow these few steps to install the necessary PIA's on your system.

1. Open Visual Studio 2005.
2. From the File menu, create a new project. The action launches the Microsoft Visual Studio Project wizard.
3. In the project types window pane, choose Office from the language tree. Notice that Visual Studio has tailored your environment to your language preference. For instance, if your language preference is C#, as in Figure 1-4, the first language will be Visual C#. Otherwise it will be Visual Basic.


New Project

Project types:
- Visual C#
  - Windows
  - Office
  - Smart Device
  - Database
  - Starter Kits
- Other Languages
- Other Project Types

Templates:

Visual Studio installed templates
- Excel Workbook
- Excel Template
- Outlook Add-in
- Word Document
- Word Template

My Templates
- Search Online Templates...

A project for creating managed code extensions behind a new or existing Excel 2003 workbook.

Name: ExcelWorkbook1

Location: C:\Download\VSTO\Book\Chpt_1

Solution: Create new Solution

Create directory for solution ✓

Solution Name: ExcelWorkbook1

Figure 1-4

4. In the right window pane labeled Templates, select Excel workbook. You may choose to accept the default name at the bottom of the window – ExcelWorkbook1, or you may add a more meaningful name. In any event, the project will be assigned the name from the Name property text box in the wizard and the project will be created in the Visual Studio Project folder. Notice the checkbox off to the right. It allows you to keep projects in their own directory.
5. Click OK to generate a new project. Behind the scenes, VSTO first verifies that the minimum system requirements are at least present. Then, the internal plumbing is laid for the new project. If there is no misstep in the process, the wizard generates a project template and opens to a Microsoft Excel spreadsheet nested inside the Integrated Development Environment, as shown in Figure 1-5. If you had selected a word template, the application would open to a Microsoft Word document hosted inside the Integrated Development Environment.

Notice that the templates section contains the possible templates that are available in VSTO. It does not mean that these templates are configured on your system. For instance, if you do not have Microsoft Outlook professional edition installed, the template will still be available. However, it will fail to open. The purpose of the template is two-fold. It allows .NET code to automate the Excel object and it allows .NET code to be linked to the Microsoft Office document.

On the next page of the wizard, you may choose to create a new document or to copy an existing document, as shown in Figure 1-6. Click OK to complete the wizard. Notice that the name of the document in Figure 1-5 is used to title the solution in Figure 1-6. Also, the default name is used for the Excel workbook . The property window on the far right now contains a single workbook project made up of properties, references, and a workbook containing three Excel worksheets. From this point, you are ready to develop.

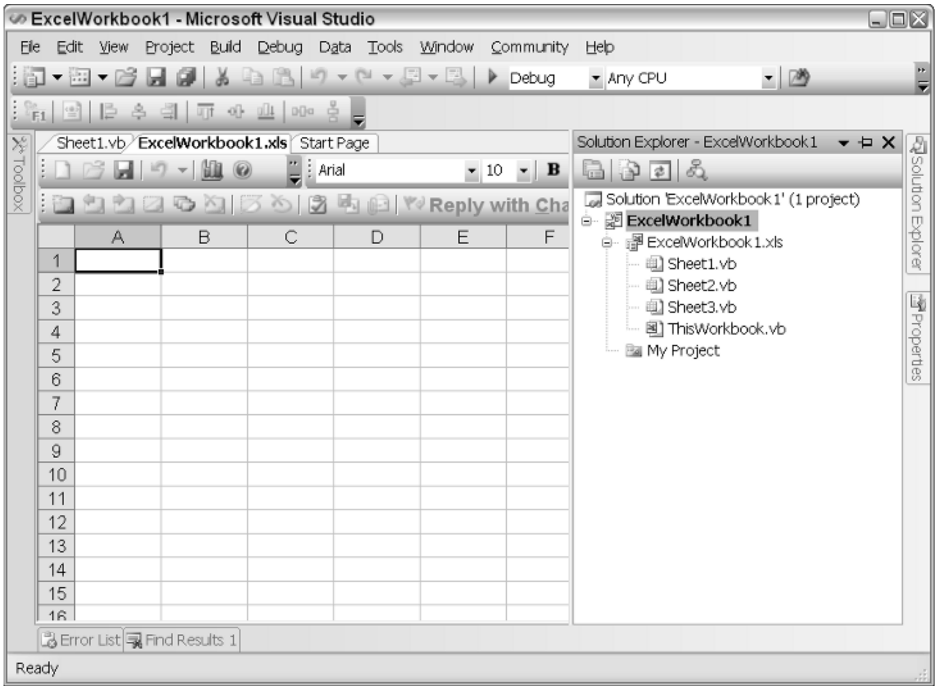
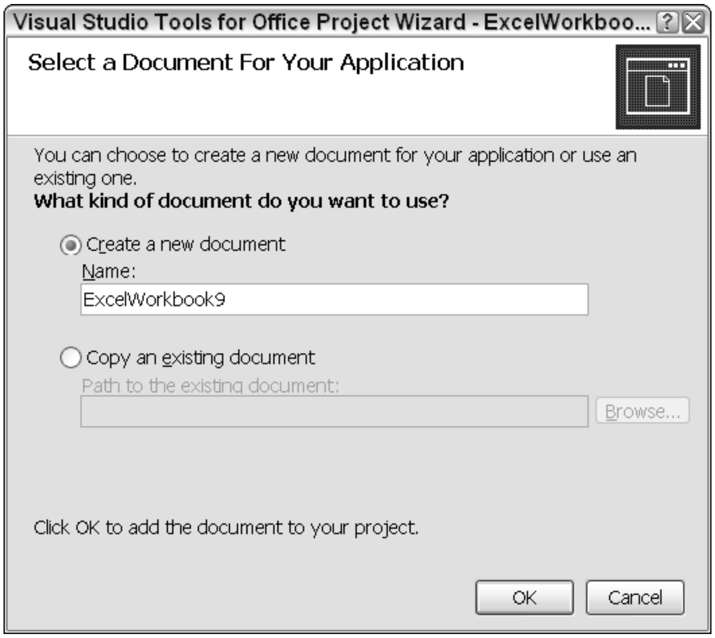


Figure 1-5


Visual Studio Tools for Office Project Wizard - ExcelWorkboo...

Select a Document For Your Application

You can choose to create a new document for your application or use an existing one.

What kind of document do you want to use?

(●) Create a new document

Name: ExcelWorkbook9

( ) Copy an existing document

Path to the existing document:

Click OK to add the document to your project.

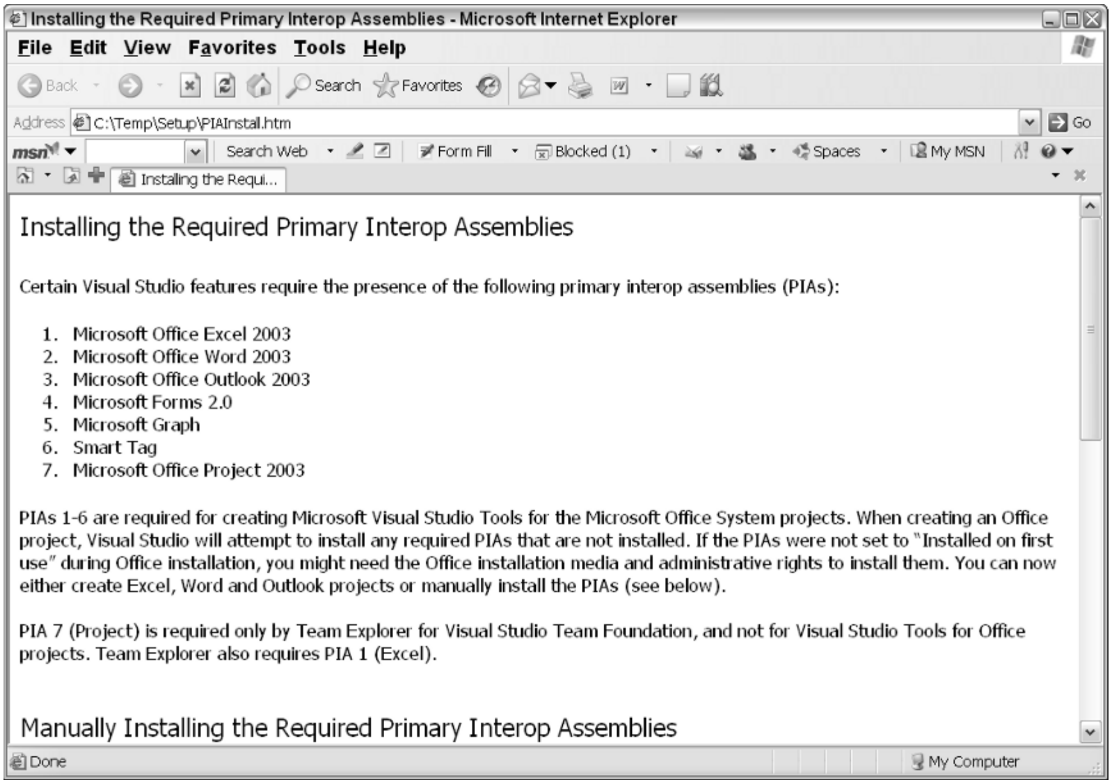
Figure 1-6

VSTO Installation Issues

In some instances, VSTO installations may be problematic. This section contains some tips for figuring out how to repair broken installations.

When you run VSTO for the first time, a security dialog box appears informing you to explicitly enable access to the Microsoft Office Visual Basic for Applications project. While this isn't an installation problem per se, selecting Cancel will cause VSTO projects to fail. Simply click OK to dismiss the dialog. Access to the Microsoft Office VBA project system is required for VSTO to run correctly. If you choose Cancel, you will need to re-open the project again so that the option can be corrected.

In certain instances, the Microsoft Office Primary Interop Assemblies may fail to install correctly. The remedy is to manually re-install the PIA's from the installation media. Locate the `PiaInstall.htm` file on the installation media. This file opens in a web browser and provides instructions for PIA installation, as shown in Figure 1-7. The PiaInstall.htm file resides a few folders deep in the install media. Figure 1-7 shows a typical install path F:\vs\setup.


Installing the Required Primary Interop Assemblies - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Temp\Setup\PIAInstall.htm
Go
Search Web Form Fill Blocked (1) Spaces My MSN
Installing the Requi...
Done
My Computer

Installing the Required Primary Interop Assemblies

Certain Visual Studio features require the presence of the following primary interop assemblies (PIAs):

1. Microsoft Office Excel 2003
2. Microsoft Office Word 2003
3. Microsoft Office Outlook 2003
4. Microsoft Forms 2.0
5. Microsoft Graph
6. Smart Tag
7. Microsoft Office Project 2003

PIAs 1-6 are required for creating Microsoft Visual Studio Tools for the Microsoft Office System projects. When creating an Office project, Visual Studio will attempt to install any required PIAs that are not installed. If the PIAs were not set to "Installed on first use" during Office installation, you might need the Office installation media and administrative rights to install them. You can now either create Excel, Word and Outlook projects or manually install the PIAs (see below).

PIA 7 (Project) is required only by Team Explorer for Visual Studio Team Foundation, and not for Visual Studio Tools for Office projects. Team Explorer also requires PIA 1 (Excel).

Manually Installing the Required Primary Interop Assemblies

Figure 1-7

The Office PIAs are not COM components, so you cannot manually register them at a command prompt using the Windows `regsvr32` executable. However, they do incorporate COM Callable Wrappers (CCW) that allow you to call them using a COM interface. To do this, you must register the CCW interfaces using the `regasm` utility that ships with the .NET framework.

In some versions, the application may install without errors but fail to create Excel projects. To remedy this annoyance, make sure the following key is accessible in the windows registry. If it is not present, back up the registry and perform the following steps.

1. Navigate to the following registry hive.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio \8.0\Setup\VSTO]
```

2. Add the following key

```
String Value
Name - ProductDir
Data - C:\Program Files\Microsoft Visual Studio 8\Visual Studio Tools for
Office
```

The Path in the Data field must point to your actual Visual Studio Tools for Office folder. Use Internet Explore to find the path to your Visual Studio Tools for Office folder. For more help editing the windows registry, please consult the Microsoft Windows help system.

A list of known issues is summarized in `VSknownIssues.rtm`. This file may be found on the installation media. Be sure to check the Visual Studio setup log for any installation failures. Issues that arise must be addressed in order for VSTO to function correctly.

The most recent update to VSTO ships with Visual Studio 2005. As of this writing, there are no patches or upgrades for the tool suite. However, you should always ensure that Microsoft Office is current and up to date by using the Windows Update site. You can find more information about Microsoft Office updates by visiting the Microsoft Office website or MSDN.

Before re-installing Visual Studio.NET to repair broken installations, run the `Vstor.exe` application manually. This application can be found on the installation media of Visual Studio. The application fixes a number of pesky issues inherent in the installation process.

Summary

While it is certainly romantic to think that VSTO was born out of innovative technology, in reality VSTO was born out of a need to stem the tide of open source competitors like OpenOffice and the like whose open source product offerings steadily erode the bottom line of Microsoft Office profitability. Even so, VSTO does bring a lot to the table where Office development is concerned.

VSTO adds productivity because applications based on the Microsoft Office System can tap into the vast reservoir of the .NET framework. Though it is still possible to accomplish all this with VBA and COM Interop, VSTO makes these tasks a lot easier. For instance, incorporating web-services into VSTO applications is a snap. The end result of these and other features really make it possible to develop applications that span corporate boundaries in a safe and secure way. VSTO is focused on increasing developer productivity.

The language support for Visual Studio is also in its infancy; that is, only Visual Basic and C# are supported with Visual Basic being the language of choice for historical reasons. However, both these languages are powerful enough to build commercial-grade software. The intellisense engine in both C# and Visual Basic offer full support for VSTO.

We also covered the unpolished side of VSTO to include performance concerns. While this is certainly not a popular route and is less likely to be traveled in other resources, it is very necessary if you are to form an educated opinion of Visual Studio Tools for Office. You may recall that some of these performance problems have nothing to do with VSTO at all. In fact, much of the startup costs have to do with the .NET JIT compilation process. However, .NET and VSTO are titled tangled so that VSTO performance perception is influenced negatively through guilt by association.

Much of the power and flexibility of VSTO comes from the new architecture model. If you have ever spent time developing server-side applications, you will recall that these applications failed to scale well because, as load increased, the COM components started to misbehave. Part of the reason for this malfeasance has to do with the inter-twining of logic and presentation code. The new model offers a new approach so that data can be manipulated without the need to touch the code that presents the data. The implication here is that Excel is no longer needed on the web-server for document processing. And that, by itself, is one compelling reason to make the switch to VSTO.

Whether or not you have developed software for the Microsoft Office System, VSTO is a very appealing choice. Developer productivity is increased and more functionality is available through the .NET interface. VSTO is also well-designed and implemented. The end result is a very well-behaved piece of machinery that delivers Microsoft Office technology in an effective way while reducing investments in time and effort. VSTO has a long way to go, but its first few steps are well-guided and deserve applause.