VSTO/VSTA Power Tools

#### v1.0

# Overview

Visual Studio Tools for Office (VSTO) and Visual Studio Tools for Applications (VSTA) are released both with major Office releases and with mainstream Visual Studio releases. To meet the demands of frequent releases, we trim our feature set aggressively. Inevitably, this means that there are some features that we'd really like to release but are prevented from doing so by time constraints. This is where the VSTO/VSTA Power Tools come in. This is a set of tools that we’re releasing alongside Visual Studio to provide additional functionality. In order to release these tools in a timely manner, they undergo only the simplest of testing, and are therefore unsupported. They are freely downloadable from [here](http://www.microsoft.com/downloads/details.aspx?FamilyId=46B6BF86-E35D-4870-B214-4D7B72B02BF9&displaylang=en).

# Installing the Tools

The download consists of a zip file which includes three separate installer packages, as follows:

* **VSTO\_PTRibbonIDs.exe** installs the Ribbon IDs Tool Window.
* **VSTO\_PTExtLibs.exe** installs the Office Interop API Extensions and Custom UI Manager.
* **VSTO\_PT.exe** installs all other power tools.

All packages install their files under “%PROGRAMFILES%\Microsoft\Microsoft VSTO Power Tools 1.0”.

# The Power Tools v1.0 Toolset

There are nine tools in the first release of the VSTO Power Tools, including two runtime tools and seven design-time tools, as described in Table 1.

|  |  |  |
| --- | --- | --- |
| **Tool** | **Description** | **Type of Tool** |
| Open XML Package Editor | A graphical treeview-based editor for examining and editing Open XML Package files (including Word, Excel and PowerPoint documents). | Add-in to Visual Studio. |
| Ribbon IDs Tool Window | A custom tool window that displays all re-usable built-in Office Ribbon IDs. | Add-in to Visual Studio. |
| Office Custom UI Manager | A set of classes for coordinating multiple instances of Ribbon, custom task pane, and custom form region objects. | Runtime class library and sample source code. |
| Office Interop API Extensions | A set of C# classes for handling parameterized properties and optional/named parameters, as well as for LINQ-enabling Office collection objects. | Runtime class library. |
| SharePoint Feature Sweeper | A GUI tool that cleans up unwanted SharePoint features from the developer's environment. | Standalone Windows Forms application. |
| SharePoint Workflow Package Generator | A tool to package a SharePoint workflow project so that it can be deployed. | Scriptable command-line tool. |
| VSTO/VSTA Pipeline Verifier | A GUI tool that reflects over custom pipeline assemblies to validate the complete VSTA pipeline. | Standalone Windows Forms application. |
| VSTO Developer Cleaner | A GUI tool that removes leftover build artifacts from the developer's computer. | Standalone Windows Forms application. |
| VSTO Troubleshooter | A diagnostic tool that examines a computer for the necessary prerequisites for running VSTO solutions. | Standalone Windows Forms application. |

Table 1 - Summary of the tools

Each of these tools is described in more detail in the following sections.

# Open XML Package Editor

This is a Visual Studio 2008 add-in to allow parsing and editing of Open Packaging Conventions files, including Word, Excel and PowerPoint documents. Features include:

* Open any Office 2007 Open XML Package file or XPS Package file directly in Visual Studio.
* Intuitive, browsable tree view of the Package file.
* Open any XML part directly in Visual Studio's rich XML editor.
* Easy to use user interface for adding and removing parts and relationships.
* Import and export part contents to and from files.
* Detects when a Package file that is opened in Visual Studio is modified externally. Prompts user to reload without having to close any open XML part editors.
* Create new Office Packages from a set of templates using Visual Studio's File > New dialog.

Figure 1 shows the treeview that is provided when you open an Open XML Package file in Visual Studio:



Figure 1 – Open XML Package treeview in Visual Studio

From the treeview, if you double-click on any XML part in the file, that part will be opened in the standard Visual Studio XML editor, as shown in Figure 2:



Figure 2 - XML part open in the Visual Studio XML editor

# Ribbon IDs Tool Window

This tool provides a Visual Studio custom tool window that displays all the reusable Office images and their IDs. The add-in puts an “ImageMso Window” item on the Tools menu. Click this item to get the custom tool window. You can float or dock the tool window.

In the tool window, when you hover over a button, the tooltip shows you the imageMso ID. When you click a button, the imageMso ID is copied to the clipboard. If you're using this with the VSTO Ribbon designer, you can then paste from the clipboard into the OfficeImageId property field, to give your control the specified standard Office icon image. You can also use drag-and-drop to copy the imageMso ID from a button to a text-based source file in Visual Studio (such as your ribbon XML if you're not using the VSTO ribbon designer).

Figure 3 shows Visual Studio open with a VSTO Excel 2007 add-in project, and the VSTO Ribbon designer. The custom tool window is docked to the right:

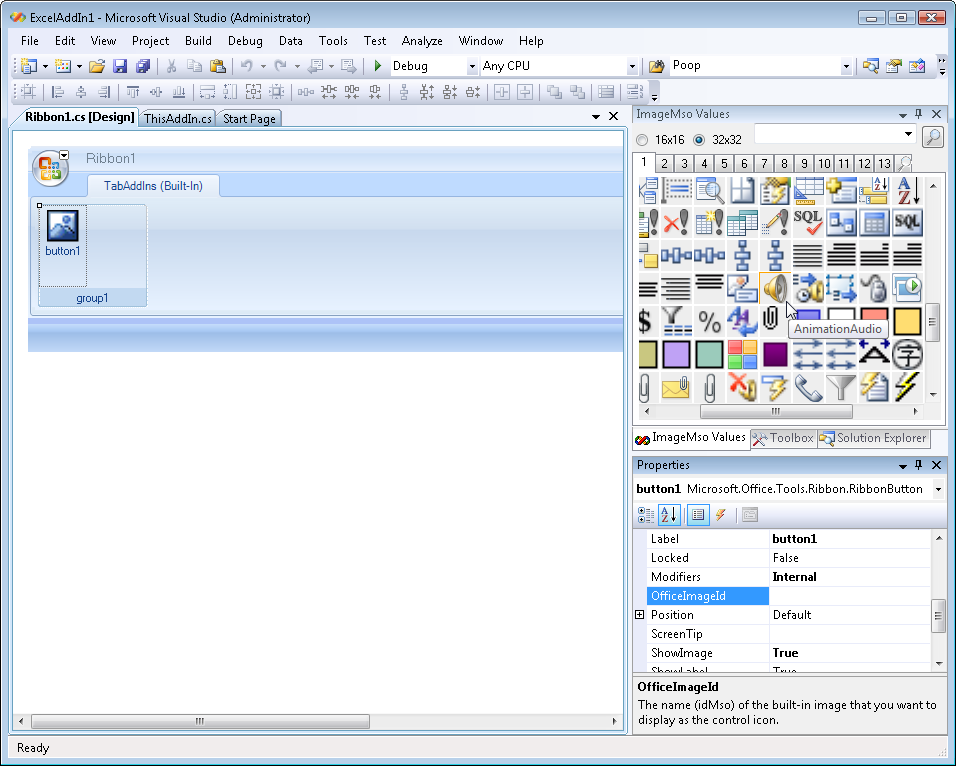


Figure 3 – Ribbon IDs custom tool window

Some Office images have multiple image sizes (typically 32x32 and 16x16). Other Office images are provided in only a single size ( 32x32, 24x24, or 16x16). This tool renders all images in either 32x32 or 16x16 format, regardless of the original size of the native image. You can choose whether your Ribbon customization uses large (32x32) or small (16x16) images, and see how each image would be rendered regardless of its original native size.

A simple search capability is also provided. You can type in any string, and the tool will return a set of images where the string is found within the image ID string. For example, Figure 4 shows the results of searching for the string "data", and displaying 16x16 images.

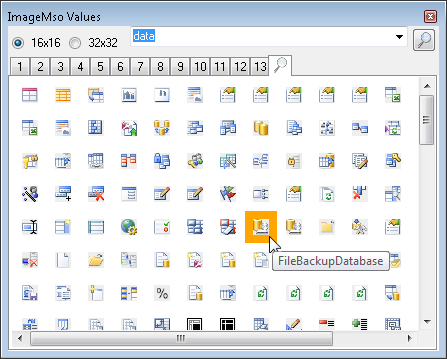


Figure 4 - Ribbon IDs tool window search results

# Office Custom UI Manager

In multiple document interface (MDI) applications like Excel and PowerPoint, there is only one main frame window, which is used by all workbooks or presentations. Custom task panes are associated with this main frame window. Therefore, by default, any custom task pane is always available to any workbook.

This is not the case with Word, InfoPath and Outlook. In Word/InfoPath there are multiple instances of this frame window, one for each document/form you open. So, when you create a custom task pane in Word/InfoPath, it gets attached to whichever is the active document frame, and it is not visible in any other document frame.

The situation is slightly different in Outlook because the user can have multiple Inspector windows open, each of which supports custom task panes. Outlook Explorer windows also support custom task panes. In addition, Outlook supports custom form regions. By default, there is no indication within a custom task pane or form region as to which specific window it is associated with.

This tool is a reusable set of classes that provide a mechanism to synchronize a window (whether it's a Word document window, an InfoPath form window, or an Outlook Inspector or Explorer window), and the custom task panes that are associated with that window. For Outlook, it also enables you to synchronize a window and the custom form region that is associated with it.

The tool provides a special UIContainer class instance associated with each window. You can use this instance as an entry point for all components associated with the window. The tool also provides another class, called UIManager. You can use this class to manage the list of available containers. UIManager also provides mapping between window instances and UIContainer classes.

# Office Interop API Extensions

The Office Interop API Extensions tool uses new features found in the .NET Framework 3.5 and Visual C# 3.0 to wrap the Office object model and provide a more productive environment for the C# developer. Specifically, it employs extension methods, object initializers, and nullable types to create a simplified, strongly-typed, and, in some cases, Visual Basic-like API. It is *not* a complete managed API for Office, but is designed to augment the raw object model in useful ways.

The Office object model was originally targeted at dynamic languages such as Visual Basic for Applications (VBA) and Visual Basic. As such, it makes extensive use of some of their capabilities such as late-binding and optional parameters. Being an early-bound and strongly-typed language, C# can be awkward, tedious, and error prone when used in this context. The Office Interop API Extensions, with its simplified and strongly-typed API, allows C# developers to be as productive in this context as Visual Basic developers.

For example, the library includes support for optional parameters, and for parameters that are typically passed by reference. Figure 5 shows examples of code both with and without using this library.

// Using the standard PIAs:

object fileName = "Foo.html";

object missing = Type.Missing;

object saveFormat = Word.WdSaveFormat.wdFormatHTML;

doc.SaveAs(ref fileName, ref saveFormat,

ref missing, ref missing, ref missing, ref missing, ref missing,

ref missing, ref missing, ref missing, ref missing, ref missing,

ref missing, ref missing, ref missing, ref missing);

// Using the Office Interop API extensions:

doc.SaveAs("Foo.html", Word.WdSaveFormat.wdFormatHTML);

Figure 5 – Optional parameters and reference parameters

The library includes similar time-saving support for named parameters, and for parameterized properties. In addition, the library includes extensive support for using LINQ-friendly syntax in your code when iterating over the various collections in the Office application object models. Figure 6 shows how you can make use of the LINQ to DASL implementation in the library in order to query Outlook data without constructing raw SQL-like query strings:

// Using the standard PIAs:

string filter = @"@SQL=(""urn:schemas:httpmail:subject"" LIKE '%" + subject.Replace("'", "''") + @"%' AND ""urn:schemas:httpmail:date"" <= '" + (DateTime.Now – new TimeSpan(7, 0, 0)).ToString("g") + @"')";

Items restrictedItems = folder.Items.Restrict(filter);

foreach (MailItem item in restrictedItems)

{

Console.WriteLine("Body: {0}", item.Body);

}

// Using the Office Interop API extensions:

var results = from item in folder.Items.AsQueryable<Mail>()

where item.Subject.Contains(subject)

&& item.Sent <= DateTime.Now – new TimeSpan(7, 0, 0)

select item.Body;

foreach (var result in results)

{

Console.WriteLine("Body: {0}", result);

}

Figure 6 – Office collections made LINQ-friendly

# SharePoint Feature Sweeper

Each time a VSTO SharePoint Workflow solution is debugged in Visual Studio 2008, the SharePoint feature is installed in the SharePoint file system and data is added to the SharePoint database. If the same document library is used for debugging many feature projects, these artifacts begin to clutter the SharePoint environment.

Further, if all of the workflow features have been configured to start automatically when a new item is saved or when an item is changed, they will all attempt to start. If there are very many workflow features, this may significantly hamper your experience – especially if features that are not being currently debugged fail or if they have resource intensive operations. You can remove or disable the associated workflows manually through the SharePoint user interface, but that can be time-consuming.

The SharePoint Feature Sweeper tool is designed to alleviate this problem. When the tool starts, it searches the local SharePoint server for all installed workflow features. These are then listed in a list control, identified by workflow and feature. Each entry is accompanied by a checkbox control, which will be checked by default. You can use these checkboxes to indicate which installed workflow features you want to remove from SharePoint. When you click the Remove button, the workflow features selected in the list are removed. The Remove button is enabled only if there are items in the list that are selected.

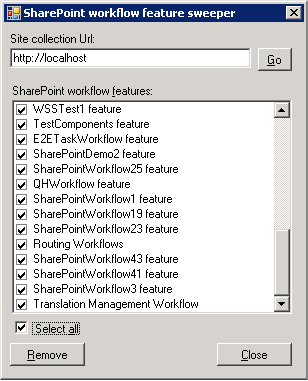


Figure 7 – SharePoint Feature Sweeper

# SharePoint Workflow Package Generator

VSTO greatly improves the developer experience for creating, debugging, and modifying workflow projects for Microsoft Office SharePoint Server 2007. A SharePoint workflow customization is one type of a SharePoint feature. Once you complete creating, debugging, and testing a SharePoint workflow feature, you need to hand it over to the SharePoint server system administrators along with all of the artifacts they need to in order to install the completed solution on a staging or production server. Typically, this is done using a Windows SharePoint Solution Package (WSP).

A WSP is essentially a cabinet (CAB) file with a very specific set of contents. The administrator can run the SharePoint administration command-line utility (stsadm.exe) on the WSP to install and activate the solution on the server.

This power tool builds a WSP file that contains relevant artifacts and metadata from the current VSTO SharePoint workflow project. You can use the tool from a command prompt (including as a post-build step), or you can add a script to the Visual Studio build tasks to cause the tool to run automatically after Visual Studio builds a project. An example command prompt entry is shown below:

“c:\Program Files\Microsoft VSTO Power Tools 1.0\workflowpackagegen.exe” /featureManifest:feature.xml

|  |  |  |
| --- | --- | --- |
| **Command-line switch** | **Required/Optional** | **Purpose** |
| /featureManifest:value | Required | Specifies the location of the feature manifest file. |
| /outputPath:value | Optional | Specifies the output path for the WSP file (the default location is the same directory as the feature manifest). |
| /?, /help | Optional | Shows command usage message. |

An example of incorporating the tool into the project as an MSBuild task is shown below:

<Target Name="AfterBuild">

<Exec Command="&quot;c:\Program Files\Microsoft VSTO Power Tools 1.0\workflowpackagegen.exe&quot; /featureManifest:feature.xml" ContinueOnError="true">

</Exec>

</Target>

You specify a feature manifest file as an input, and the tool will generate a manifest.xml file, a DDF file and the WSP file as output. You have the option to specify the location where the outputs will be generated.

# VSTO/VSTA Pipeline Verifier

This tool is used to verify the integrity of a given VSTO/VSTA pipeline. Starting in Visual Studio 2008, the VSTO runtime is layered on top of the VSTA runtime, and uses the default VSTA pipeline. The tool can be used in GUI mode or in command prompt mode.

To run the tool in GUI mode, double click the PipelineVerifier.exe. If the VSTO/VSTA runtime is installed (which lays down the default VSTO/VSTA pipeline) the default pipeline path is displayed, as shown in Figure 8.

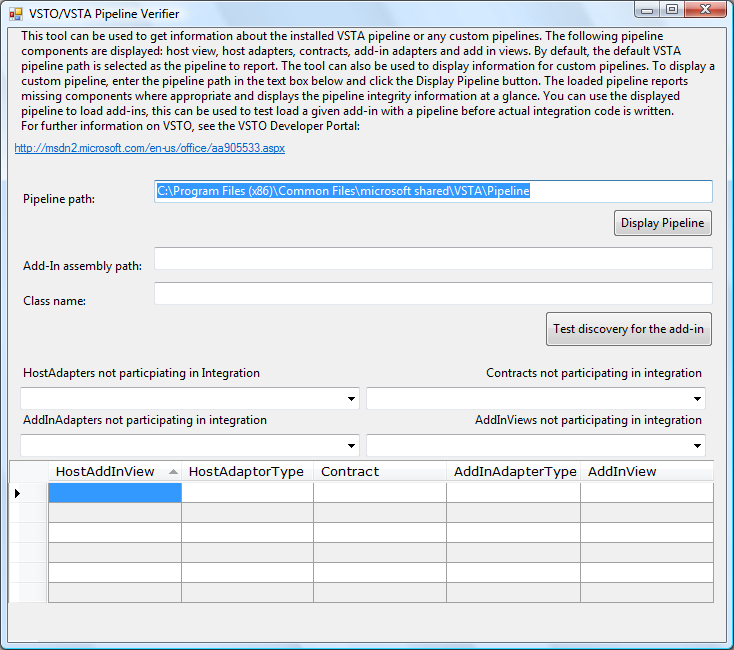


Figure 8 - VSTA Pipeline Verifier showing the default pipeline

You can either accept the default pipeline or specify a custom pipeline. Then, click Display Pipeline to display the details on the specified pipeline.

The tool also displays information on non-participating types in the pipeline assemblies. These types may be used in add-in load scenarios but they don’t directly contribute to creating the pipeline. The combo boxes at the bottom of the tool window display lists of non-participating host adapters, add-in adapters, contracts and views.

To test load an add-in with the displayed pipeline, specify the add-in path and the add-in class, and then click on the “Test discovery for the add-in” button. If the load succeeds, a success message is displayed, otherwise an error is shown with the appropriate stack trace. To load the add-in, all possible views discovered in the pipeline are attempted.

Figure 9 shows an example of a test Excel add-in assembly built as ExcelAddIn1.dll, with an add-in class named ExcelAddIn1.ThisAddIn.

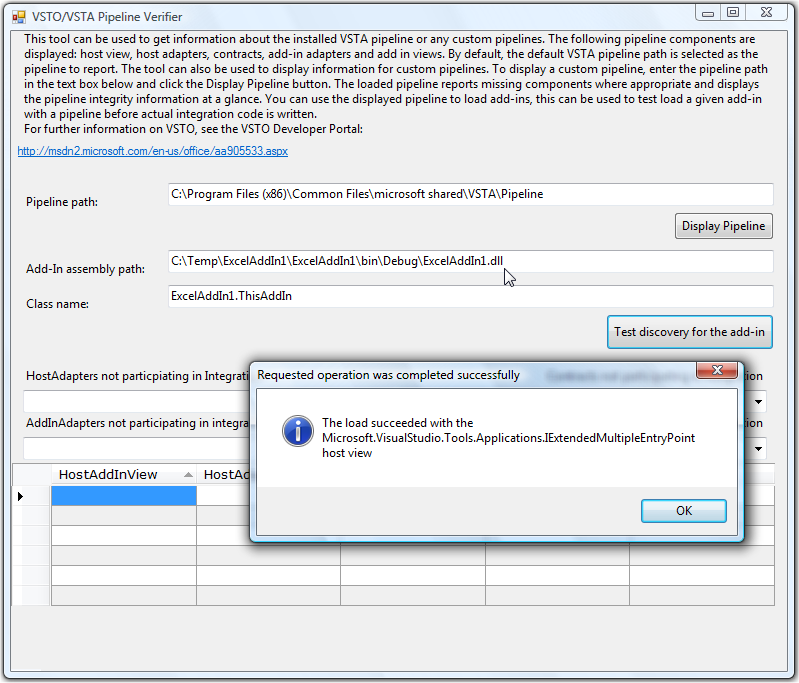


Figure 9 - Testing the discoverability of an Excel add-in

To run the tool in command prompt mode, execute PipelineVerifier.exe at the command prompt using the /silent switch. This disables the GUI display and logs the pipeline information as an XML file in the current working directory. Command prompt mode is more useful for build tasks and scripting. For further information on the functionality of this tool, please see the help document installed with the tool.

# VSTO Developer Cleaner

Visual Studio 2008 includes options to "Clean" a project or solution – that is, to remove the build artifacts left by a project, but this does not remove all build artifacts. The standard clean option removes most of the output files from the target folder, and some registry entries, but not all. You typically create multiple Visual Studio projects on a computer, including test projects, prototypes and production projects, which can result in many unwanted build artifacts over time. Many of these artifacts are not particularly discoverable, and it is not always obvious which items can be safely deleted. This tool provides a simple way to identify and remove these artifacts.

The tool allows you to list and remove either single instances or multiple instances of the following artifacts:

* Add-in registry entries
* Temporary certificates in the current user's personal store
* The current user's inclusion list entries created for debugging purposes.

The tool offers three tabs, to correspond with these three categories of artifact, as shown in Figures 10 to 13. In each case, you can remove selected entries or all entries.

### Add-in Registry Entries

When a VSTO add-in is built, Visual Studio registers it for debugging purposes. This registration includes the standard Manifest key. This key includes an additional “|vstolocal” tag that prevents the VSTO runtime from installing it into the ClickOnce cache. Deleting these entries will cause the add-in to not load. Simply calling build in the project may not recreate these entries unless a recompile occurs (you must either clean and rebuild the project or change the project in a way that causes it to rebuild). The tool finds all Manifest keys in the registry which have the “|vstolocal” tag.

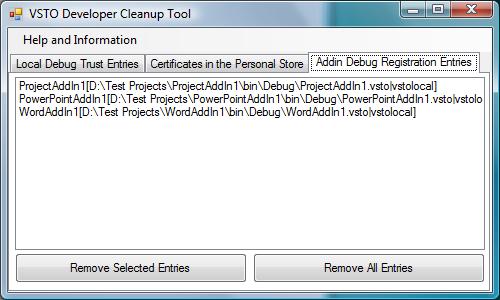


Figure 10 - Office add-in registry entries

### Temporary Certificates in the Personal Store

When a VSTO solution is built, a certificate must be used to sign the solution. If a valid certificate is not provided, a temporary “self cert” certification is created, added to the project, and then copied into the current user's personal store. The tool compares the issuer and the subject of the certificates in the store, and reports self certs in the list. If you accidently delete a self cert to a project you are still working with, next time the solution is built the self cert will be added back into the personal store. Removing these certificates has no effect on the runtime behavior of VSTO customizations.

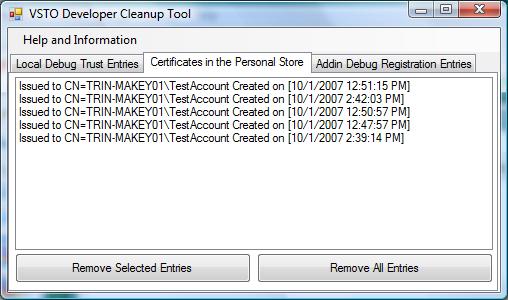


Figure 11 - Temporary certificates in the user's personal store

### Inclusion List Entries

When a VSTO solution is published and installed, if the solution's certificate is not explicitly trusted, a trust prompt that asks the user if they want to install the solution will occur. When the user responds to this prompt, the trust decision is persisted in the VSTO inclusion list. The VSTO inclusion list is a special set of registry keys. Visual Studio 2008 also automatically creates inclusion list entries for locally-built VSTO customizations on your development computer. Deleting these entries means the solution will not be trusted, and a trust prompt will occur if the solution is still registered when the customized application/document is next opened.

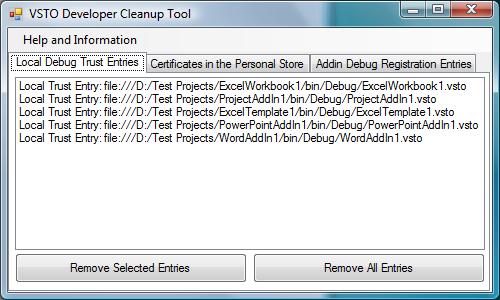


Figure 12 - User inclusion list entries created for debugging purposes

# VSTO Troubleshooter

The VSTO Troubleshooter tool is a stand-alone application designed to help diagnose potential client-side issues when you deploy a VSTO solution. The tool scans the client computer for the prerequisites required to run a VSTO solution and reports the state of the prerequisites in an HTML report. The tool also provides links to information about how to resolve the issue when a prerequisite is not found on the client computer. This is an update to a tool that was previously available for Visual Studio 2005 Tools for Office SE solutions, described here: <http://msdn2.microsoft.com/en-us/library/bb510371.aspx>.

The tool checks for the correct installation of the supported Office products, the Office primary interop assemblies (PIAs), the VSTO runtime, the .NET Framework, and a number of registry and configuration settings, as shown in Figure 16.

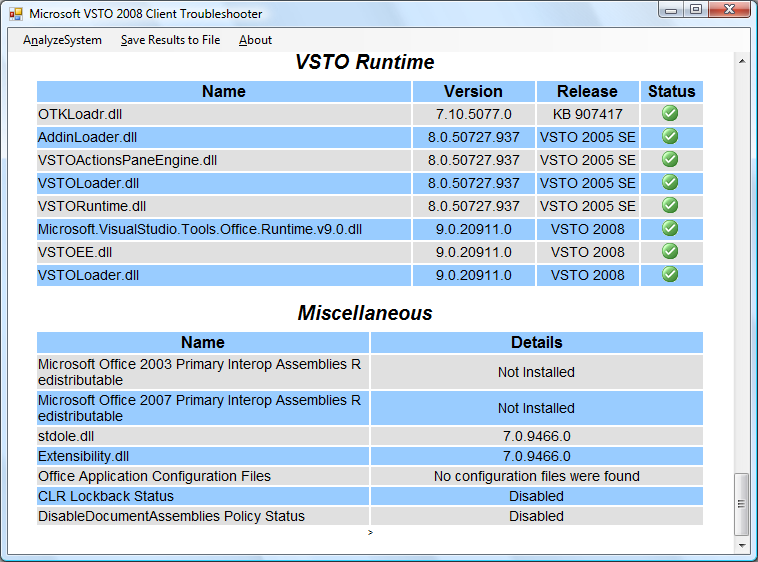


Figure 16 - VSTO Troubleshooter

# Conclusion

The VSTO team is dedicated to providing a rich professional toolset and runtime framework. We also recognize the need for additional support tools, and are committed to releasing these tools alongside the main product. Although the tools are not supported, we welcome any feedback, and we intend to post updates to the download periodically. These updates will likely include bug fixes and added features, as well as additional new tools, over time.