



webMethods for Microsoft Package Installation and User's Guide

VERSION 6.5.2

webMethods, Inc.
South Tower
3877 Fairfax Ridge Road
Fairfax, VA 22030
USA
703.460.2500
<http://www.webmethods.com>

webMethods Access, webMethods Administrator, webMethods Broker, webMethods Dashboard, webMethods Developer, webMethods Fabric, webMethods Glue, webMethods Installer, webMethods Integration Server, webMethods Mainframe, webMethods Manager, webMethods Modeler, webMethods Monitor, webMethods Optimize, webMethods Portal, webMethods Servicenet, webMethods Trading Networks, and webMethods Workflow are trademarks of webMethods, Inc. webMethods and the webMethods logo are registered trademarks of webMethods, Inc.

Acrobat and Adobe are registered trademarks, and Reader is a trademark of Adobe Systems Incorporated. Amdocs is a registered trademark, and ClarifyCRM is a trademark of Amdocs. Ariba is a registered trademark of Ariba, Inc. BEA, BEA WebLogic Server, Jolt, and Tuxedo are registered trademarks, and BEA WebLogic Platform is a trademark of BEA Systems, Inc. Action Request System, BMC Software, PATROL, and Remedy are registered trademarks of BMC Software, Inc. BroadVision is a registered trademark of BroadVision, Inc. ChemeStandards and CIDX are trademarks of Chemical Industry Data Exchange. Unicenter is a registered trademark of Computer Associates International, Inc. PopChart is a registered trademark of CORDA Technologies, Inc. Kenan and Arbor are registered trademarks of CSG Systems, Inc. Data Connection and SNAP-IX are registered trademarks of Data Connection Corporation. DataDirect, DataDirect Connect, and SequeLink are registered trademarks of DataDirect Technologies. D&B and D-U-N-S are registered trademarks of Dun & Bradstreet Corporation. Entrust is a registered trademark of Entrust, Inc. papiNet is a registered trademark of the European Union and the United States. Financial Information eXchange, F.I.X., and F.I.X Protocol are trademarks of FIX Protocol Ltd. UCCnet and eBusinessReady are registered trademarks, and 1SYNC and Transora are trademarks of GS1 US. Hewlett-Packard, HP, HP-UX, OpenView, PA-RISC, and SNAplus2 are trademarks of Hewlett-Packard Company. i2 is a registered trademark of i2 Technologies, Inc. AIX, AS/400, CICS, DB2, Domino, IBM, Informix, Infoprint, Lotus, Lotus Notes, MQSeries, OS/390, OS/400, RACF, RS/6000, SQL/400, S/390, System/390, VTAM, z/OS, and WebSphere are registered trademarks; and Communications System for Windows NT, DB2 Universal Database, IMS, MVS, and SQL/DS are trademarks of IBM Corporation. InnoDB is a trademark of Innobase Oy. Itanium is a registered trademark of Intel Corporation. JBoss is a registered trademark, and JBoss Group is a trademark of Jboss, Inc. Linux is a registered trademark of Linus Torvalds. W3C is a registered trademark, and X Window System is a trademark of the Massachusetts Institute of Technology. MetaSolv is a registered trademark of Metasolv Software, Inc. ActiveX, Microsoft, Outlook, Visual Basic, Windows, and Windows NT are registered trademarks; and Windows Server is a trademark of Microsoft Corporation. Six Sigma is a registered trademark of Motorola, Inc. Firefox is a registered trademark, and Mozilla is a trademark of the Mozilla Foundation. MySQL is a registered trademark of MySQL AB. nCipher is a trademark of nCipher Corporation Ltd. TeraData is a registered trademark of NCR International, Inc. Netscape is a registered trademark of Netscape Communications Corporation. SUSE is a registered trademark of Novell, Inc. ServletExec is a registered trademark, and New Atlanta is a trademark of New Atlanta Communications, LLC. CORBA is a registered trademark of Object Management Group, Inc. JD Edwards, OneWorld, Oracle, PeopleSoft, Siebel, and Vantive are registered trademarks, and PeopleSoft Pure Internet Architecture and WorldSoftware are trademarks of Oracle Corporation. Infranet and Portal are trademarks of Portal Software, Inc. Red Hat is a registered trademark of Red Hat, Inc. PIP and RosettaNet are trademarks of RosettaNet, a non-profit organization. SAP and R/3 are registered trademarks of SAP AG. SWIFT and SWIFTNet are registered trademarks of Society for Worldwide Interbank Financial Telecommunication SCRL. SPARC and SPARCStation are registered trademarks of SPARC International, Inc. SSA is a registered trademark, and Baan and SSA Global are trademarks of SSA Global Technologies, Inc. EJB, Enterprise JavaBeans, Java, JavaServer, JDBC, JSP, J2EE, Solaris, Sun, and Sun Microsystems are registered trademarks; and Java Naming and Directory Interface, SOAP with Attachments API for Java, JavaServer Pages, and SunSoft are trademarks of Sun Microsystems, Inc. Sybase is a registered trademark of Sybase, Inc. VERITAS is a registered trademark, and VERITAS Cluster Server is a trademark of Symantec Corporation. UNIX is a registered trademark of The Open Group. Unicode is a trademark of Unicode, Inc. VeriSign is a registered trademark of Verisign, Inc.

All other marks are the property of their respective owners.

Copyright © 2004—2006 by webMethods, Inc. All rights reserved, including the right of reproduction in whole or in part in any form.

Contents

About This Guide	7
Document Conventions	7
Additional Information	8
Chapter 1. Understanding Microsoft .NET	9
What is Microsoft .NET?	10
Microsoft Windows 2000 Server and Windows Server 2003	10
Developer Tools and Technologies	10
The Microsoft .NET Framework	10
Common Language Runtime	10
The Class Library	11
Microsoft Visual Studio .NET	11
Type Library Importer	11
Assemblies and Their Roles	12
Facilities of the webMethods for Microsoft Package	13
Using webMethods Components with .NET	14
webMethods Developer and .NET	14
webMethods Administrator and .NET	15
webMethods Client API for .NET	15
webMethods and Visual Studio .NET	15
Chapter 2. Installing the .NET Package	17
Overview	18
Requirements	18
Supported Platforms and Operating Systems	18
webMethods for Microsoft Package	19
webMethods for Microsoft Code Generator Package	19
The webMethods for Microsoft Plug-in	19
webMethods add-in for Microsoft Visual Studio .NET	19
Required webMethods Components	20
Required Microsoft Products	20
Microsoft .NET Framework	20
Microsoft Visual Studio .NET	20
Supported Browsers	21
Hardware Requirements	21
Install the webMethods for Microsoft Package	22
Install the webMethods add-in for Microsoft Visual Studio .NET	23

Uninstall the webMethods for Microsoft Package	23
Uninstall the webMethods Add-In for Visual Studio .NET	24
Chapter 3. Using Developer with the .NET Package	25
What is the webMethods for Microsoft Plug-in	26
Importing .NET Methods into Developer	26
Input Variables	28
Output Variables	28
Importing COM Objects into Developer	28
Configuring .NET Assembly Information	29
Managing Class Lifetimes	30
Modifying Class Lifetime	32
The Class Timeout Value	32
Invoking a .NET Service from Developer	33
Application Domains	33
Using Services to Start and Destroy Application Domains	33
Practical ways to Start and Destroy Application Domains	33
Marshalling Data	34
Running a .NET Service in Developer	34
Chapter 4. Using the Add-In for Microsoft Visual Studio .NET	37
What is Microsoft Visual Studio .NET?	38
Using the Add-In for Microsoft Visual Studio .NET	38
Opening the Add-in and Generating Code	38
Connecting to an Integration Server	39
Disconnecting from an Integration Server	40
Chapter 5. Administering the .NET Package	41
Integration Server Administrator and Microsoft .NET	42
Accessing the webMethods for Microsoft Package	42
Pages for Administering the Microsoft .NET Package	43
Displaying Pages in Administrator	43
Refreshing the View	44
The Manage CLR Page	44
Values in the Manage CLR Page	44
Starting and Stopping the Microsoft .NET CLR	45
Tracing for the webMethods for Microsoft Package	46
Method Invocation Tracing	46
Assembly Introspection Tracing	47
Location of the Log File	47
Controlling Tracing from Administrator	47

The Manage Application Domains Page	48
The View Assemblies Page	49
The Manage .NET Object Instances Page	49
The Domain Details Page	50
The Options Page	51
The Additional Installs Page	52
Glossary	53
Index	55

About This Guide

This guide describes the installation and use of the webMethods for Microsoft Package. This package facilitates use of the Microsoft .NET application platform in conjunction with webMethods components. Using the webMethods for Microsoft Package, you can administer and manage the webMethods for Microsoft Package from any standard browser. Using the webMethods for Microsoft Plug-in, you can create services from existing .NET assemblies. Using the webMethods add-in for Microsoft Visual Studio .NET, you browse services on Integration Server and create .NET assemblies to invoke those services.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
<i>Italic</i>	Identifies variable information that you must supply or change based on your specific situation or environment. Identifies terms the first time they are defined in text. Also identifies service input and output variables.
Narrow font	Identifies storage locations for services on the webMethods Integration Server using the convention <i>folder.subfolder.service</i> .
Typewriter font	Identifies characters and values that you must type exactly or messages that the system displays on the console.
UPPERCASE	Identifies keyboard keys. Keys that you must press simultaneously are joined with the "+" symbol.
\	Directory paths use the "\" directory delimiter unless the subject is UNIX-specific.
[]	Optional keywords or values are enclosed in []. Do not type the [] symbols in your own code.

Additional Information

The webMethods Advantage Web site at <http://advantage.webmethods.com> provides you with important sources of information about webMethods components:

- **Troubleshooting Information.** webMethods provides troubleshooting information for many webMethods components in the [webMethods Knowledge Base](#).
- **Documentation Feedback.** To provide documentation feedback to webMethods, go to the [Documentation Feedback Form](#) on the [webMethods Bookshelf](#).
- **Additional Documentation.** All webMethods documentation is available on the [webMethods Bookshelf](#).

Understanding Microsoft .NET

■ What is Microsoft .NET?	10
■ Microsoft Windows 2000 Server and Windows Server 2003	10
■ Developer Tools and Technologies	10
■ Assemblies and Their Roles	12
■ Facilities of the webMethods for Microsoft Package	13
■ Using webMethods Components with .NET	14

What is Microsoft .NET?

At the highest level, Microsoft defines .NET as a set of Microsoft software technologies that help connect information, people, systems and devices using Web services. Elements of Microsoft .NET are included in several products that make up the Microsoft platform. Microsoft .NET is made up of several components, technologies, languages, and tools, such as the Microsoft .NET Framework, the C# and VB.NET languages, Visual Studio .NET, Windows Server 2003, and so on.

Microsoft .NET consists of a number of components relevant to the implementation of the webMethods for Microsoft Package. These components provide the interfaces, services and tools necessary to provide a close coupling between webMethods components and the Microsoft .NET platform. The components relevant to the webMethods for Microsoft Package are described in the following sections.

Microsoft Windows 2000 Server and Windows Server 2003

Current releases of Windows operating systems such as Windows Server 2003 contain most of the .NET runtime components required by the webMethods for Microsoft Package. You can upgrade older releases of the Windows operating systems to support .NET by installing Microsoft .NET Framework Version 1.1 Redistributable Package. This redistributable package is available for download from the Microsoft Web site. See [“Microsoft .NET Framework” on page 20](#).

Developer Tools and Technologies

The development tools available for .NET include the .NET Framework, Visual Studio .NET and the Type Library Importer.

The Microsoft .NET Framework

The Microsoft .NET Framework is the programming model of the .NET platform for building and running XML Web services and applications. The .NET Framework consists of the Common Language Runtime and a set of base class libraries.

Common Language Runtime

The Common Language Runtime (CLR) is the execution engine for .NET Framework applications. The CLR provides core runtime services such as memory, process, and thread management, along with higher-level features such as cross-language exception handling. The CLR enforces type safety and other forms of code accuracy to promote security and robustness. The CLR is to .NET as the Java Virtual Machine (JVM) is to Java.

The CLR has some important differences from the JVM. While the JVM supports only the Java language, the CLR supports a variety of Microsoft-supported languages, including

Visual Basic, C, C++, and a Microsoft-specific language named C# (C Sharp). Derived from the C language, C# is similar to Java.

What these languages have in common is that they all adhere to the Microsoft Common Type Specification (CTS), a comprehensive specification for a language-independent object-oriented data type system that is capable of representing simple data types, such as integers, or more complex data types that are embodied as classes. By enforcing the CTS specification, the CLR allows programmers to write application software in any of the supported languages.

While the Java compiler compiles Java into machine-independent object code, or bytecode, the .NET Development Environment compilers compile the source code (Visual Basic, C, C++, and C#) into the Common Intermediate Language (CIL), a low-level instruction set that is similar to Java bytecode. At run time, the CLR compiles the CIL into machine code, unlike most Java implementations, which interpret the Java bytecode.

The Common Language Runtime can be integrated into any application requiring its services.

The Class Library

Because all languages in the .NET environment compile to the same CIL, they share a common .NET Library. The advantage of this feature is that the programmer can choose the language most appropriate to the task. For instance, for high-level applications programmers can implement the modules in Visual Basic .NET; for low-level tasks, they can use Visual C++ .NET.

The .NET Framework Library consists of several components designed to encapsulate system services. The library also contains facilities that permit loading and management of the CLR from code running in environments external to the .NET environment.

Microsoft Visual Studio .NET

Microsoft Visual Studio .NET provides the integrated development environment for those developing .NET components and applications. This IDE (Integrated Development Environment) provides a great number of features, chief among them are the editing of source code written in a .NET language of choice, the management of source files, class views, project build, solution debug, and deployment facilities. The IDE also provides the ability for third party vendors to extend it using components called add-ins.

Type Library Importer

The Type Library Importer (TLBIMP) provides interoperability with existing COM (Component Object Module) components. Code that executes within the CLR is referred to as managed code. Code that executes outside the CLR is referred to as unmanaged code. Existing COM components are unmanaged. COM components do not operate within the CLR because they were developed prior to the advent of the CLR. TLBIMP provides interoperability with these unmanaged COM components by generating what is

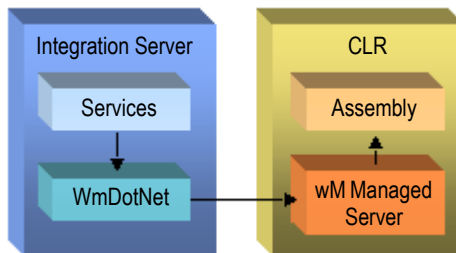
referred to as a Primary Interop Assembly (PIA). The PIA is a .NET assembly that leverages the COM interoperability facilities provided by .NET. As such, this managed code provides access to services provided by existing COM objects from other managed code running within the CLR.

Assemblies and Their Roles

Components designed to operate within the .NET environment are referred to as assemblies. Assemblies are of two types, DLLs (dynamically-linked library) and executables. The primary difference between DLLs and executables in the .NET environment, and the equivalent components in previous versions of Windows, is that the .NET components require the CLR to execute. Microsoft .NET assemblies, like components of other programming models, generally operate in one of two roles: a service provider or a service consumer.

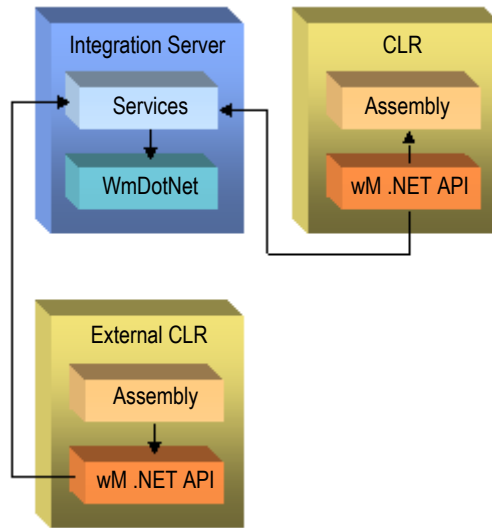
Service providers provide services to external components using the public methods and properties they expose. Service providers are generally (though not necessarily) DLLs.

Microsoft .NET assemblies operating in a service-provider role



Service consumers consume the services provided by service provider components. Service consumers are generally (though not necessarily) executables.

Microsoft .NET assemblies operating in a client role



Facilities of the webMethods for Microsoft Package

The webMethods for Microsoft Package provides .NET integration and management services within webMethods components in a tightly coupled implementation. The webMethods for Microsoft Package is an umbrella of solutions that encompasses several functional areas.

- Expose .NET assemblies as integration services

The webMethods for Microsoft Package exposes, as integration services, the services provided by .NET assemblies operating in a service provider role. Those integration services can be exposed to the outside world as Web services just like any other webMethods service. This feature simplifies integration of business logic developed on the Microsoft .NET platform into applications external to the Microsoft environment. Additionally, it simplifies the development of integration logic using a .NET language of choice.

- Allow .NET assemblies to invoke integration services

Microsoft .NET assemblies operating in a client role can invoke services exposed by Integration Server. The webMethods for Microsoft Package provides a native .NET API to Integration Server for this purpose. The API simplifies the integration of legacy or new business logic into Microsoft .NET applications.

- Host .NET executable assemblies

The webMethods for Microsoft Package provides hosting of .NET executable assemblies or applications that may be operating in an Integration Server client role or in a role unrelated to Integration Server.

- Manage .NET assembly lifetimes

The webMethods for Microsoft Package provides management of .NET assembly lifetimes for assemblies operating in a service provider role where multiple invocations of one or more methods on a given assembly instance is required.

- Manage the .NET environment

The webMethods for Microsoft Package allows you to manage the .NET environment and view metrics pertaining to its operation and the operation of assemblies executing in the .NET environment.

- Generate code in Microsoft Visual Studio .NET to invoke services

The webMethods add-in for Microsoft Visual Studio .NET provides the ability to enumerate services on Integration Server and generate code within Visual Studio .NET to invoke those services. Installation of the add-in is separate from the rest of the webMethods for Microsoft Package. See [“Install the webMethods add-in for Microsoft Visual Studio .NET” on page 23](#).

- Integrate with existing COM objects

The Primary Interop Assemblies generated by TLBIMP are .NET assemblies and can therefore be exposed as integration services just like any other .NET assembly. Once exposed as integration services, they can also be exposed to the outside world as Web services if required.

Using webMethods Components with .NET

In much the same way that IIS or the Windows operating system loads a Microsoft CLR to provide the execution environment for .NET assemblies, Integration Server loads a CLR to provide the execution container for .NET assemblies. When Integration Server starts, it loads the CLR into its address space. After the CLR is loaded, the webMethods Managed Server assumes control over it, providing access to the .NET environment from Integration Server.

webMethods Developer and .NET

With the webMethods for Microsoft Plug-in installed, You can use Developer to generate services from an existing .NET assembly. This process is referred to as introspection. Developer provides the ability to locate assemblies by browsing the Integration Server directory or UNC (Uniform Naming Convention) names accessible from the Integration Server. When you select an assembly, all public methods of public classes in that assembly are presented. You can select which services provided by the assembly are exposed as integration services, and subsequently as Web services if required.

After the services of assemblies are exposed as integration services, you can invoke them as you would any other integration service.

webMethods Administrator and .NET

Through the webMethods for Microsoft Package, you can use Administrator to view application domain statistics, load assemblies, start and stop the .NET environment, and set diagnostic tracing. As service invocations occur, the package collects metrics for execution times and invocation counts on a method and assembly instance basis. You can retrieve these metrics using Administrator.

See [Chapter 5, “Administering the .NET Package”](#) for more information.

webMethods Client API for .NET

The webMethods .NET API provides access to Integration Server and its services from the .NET environment. Client assemblies using this API may reside within Integration Server’s locally hosted CLR, another CLR on the same machine, or a CLR on another machine within the network. The code required to invoke integration services using the .NET API can be created manually, generated by Developer, or by the webMethods add-in for Microsoft Visual Studio .NET.

webMethods and Visual Studio .NET

The webMethods add-in for Microsoft Visual Studio .NET permits you to connect to a given Integration Server and list the packages, folders and services configured on the server. After a service is selected, you can use Visual Studio .NET to generate the code required to invoke that service. The Visual Studio user is not required to know what the target service type is. The target service could be any type of integration service.

See [Chapter 4, “Using the Add-In for Microsoft Visual Studio .NET”](#) for more information.

Installing the .NET Package

■ Overview	18
■ Requirements	18
■ Install the webMethods for Microsoft Package	22
■ Install the webMethods add-in for Microsoft Visual Studio .NET	23
■ Uninstall the webMethods for Microsoft Package	23
■ Uninstall the webMethods Add-In for Visual Studio .NET	24

Overview



Important! The information in this chapter might have been updated since the guide was published. Go to the webMethods Advantage Web site at <http://advantage.webmethods.com> for the latest version of the guide.

If you are installing the webMethods for Microsoft Package with webMethods components such as Integration Server, see the *webMethods Installation Guide* for instructions on installing those components.

The webMethods component known as the webMethods for Microsoft Package contains four elements that can be installed individually. Install each one on the same host as the component with which they interact:

This element...	Needs to be installed with...
webMethods for Microsoft Package	Integration Server
webMethods for Microsoft Code Generator Package*	Integration Server
webMethods for Microsoft Plug-in	Developer
webMethods add-in for Microsoft Visual Studio .NET	Microsoft Visual Studio .NET

*The webMethods for Microsoft Code Generator Package is called by Developer and by the webMethods add-in for Microsoft Visual Studio .NET to generate code, but does not require a Windows environment to run.

You cannot use the webMethods installer for the webMethods add-in for Microsoft Visual Studio .NET. You must install this element separately, as described in “[Install the webMethods add-in for Microsoft Visual Studio .NET](#)” on page 23.

The following sections describe the system requirements for the webMethods for Microsoft Package, how to install the package, and how to uninstall it.

Requirements

Supported Platforms and Operating Systems

Platform support is dependent on which element of the webMethods for Microsoft Package you are installing.

webMethods for Microsoft Package



Note: You can only run the webMethods for Microsoft Package if it is installed with an instance of Integration Server on a computer having a Microsoft Windows operating system.

Platform and Operating System

Microsoft Windows XP Professional

Microsoft Windows Server 2003

The webMethods for Microsoft Package uses the JRE you install for Integration Server. The package requires the Microsoft .NET Framework version 1.1. See [“Microsoft .NET Framework” on page 20](#).

webMethods for Microsoft Code Generator Package

The webMethods for Microsoft Code Generator Package runs on any platform supported by Integration Server and uses the JRE you install for Integration Server.

The webMethods for Microsoft Plug-in

The webMethods for Microsoft Plug-in runs on any platform supported by Developer and uses the JRE you install for Developer.

webMethods add-in for Microsoft Visual Studio .NET

Platform and Operating System

Microsoft Windows XP Professional

Microsoft Windows Server 2003

Required webMethods Components

For each element of the webMethods for Microsoft Package, the table below lists the webMethods components you must install before or at the same time you install the element. The table also lists the webMethods components you must install at some point for each element to operate fully.

Element	Required for Installation	Required for Operation
webMethods for Microsoft Package	Integration Server 6.5	Integration Server 6.5
webMethods for Microsoft Code Generator Package	Integration Server 6.5	Integration Server 6.5
webMethods for Microsoft Plug-in	Developer 6.5	Developer 6.5 Integration Server 6.5
webMethods add-in for Microsoft Visual Studio .NET		Integration Server6.5

Required Microsoft Products

Various elements of the webMethods for Microsoft Package require Microsoft products for operation, as described in the following sections.

Microsoft .NET Framework

The webMethods for Microsoft Package requires that the Microsoft .NET Framework be installed on the same computer as Integration Server. You can obtain the .NET Framework from the Microsoft download site at <http://www.microsoft.com/downloads>.

Microsoft Redistributable Packages
Microsoft .NET Framework Version 1.1 Redistributable Package.

Microsoft Visual Studio .NET

The webMethods add-in for Microsoft Visual Studio .NET requires that Visual Studio .NET be installed on the same computer.

Software	Version
Microsoft Visual Studio .NET	7.1 or later

Supported Browsers

All elements of the webMethods for Microsoft Package require one of the Internet browsers listed in the table below. (The webMethods for Microsoft Plug-in and the webMethods add-in for Microsoft Visual Studio .NET require a browser to display help files.)

Browser	Version
Microsoft Internet Explorer	6.x
Mozilla Firefox	1.x

Hardware Requirements

The table below lists the minimum hardware requirements for each element of the webMethods for Microsoft Package.

Element	Hard Drive Space (MB)	RAM (MB)	Virtual/Swap (MB)	CPUs
webMethods for Microsoft Package	Minimal space beyond Integration Server.			
webMethods for Microsoft Code Generator Package	Minimal space beyond Integration Server.			
webMethods for Microsoft Plug-in	Minimal space beyond Developer.			
webMethods add-in for Microsoft Visual Studio .NET	Minimal space beyond Microsoft Visual Studio .NET.			

Install the webMethods for Microsoft Package



Important! This section provides only instructions that are specific to installing the webMethods for Microsoft Package, the webMethods for Microsoft Code Generator Package, and the webMethods for Microsoft Plug-in. For complete instructions on using the webMethods Installer, see the *webMethods Installation Guide*.

Installation for the webMethods for Microsoft Package, the webMethods for Microsoft Code Generator Package, and the webMethods for Microsoft Plug-in is similar. Install each element on the appropriate computer:

Install this element...	On the same computer as...
webMethods for Microsoft Package	Integration Server
webMethods for Microsoft Code Generator Package	Integration Server
webMethods for Microsoft Plug-in	Developer

The following instructions describe installation of the webMethods for Microsoft Package, the webMethods for Microsoft Code Generator Package, and the webMethods for Microsoft Plug-in.



To install elements of the webMethods for Microsoft Package

- 1 Shut down all webMethods components.
- 2 Download webMethods Installer 6.5.2 from the webMethods Advantage Web site at <http://advantage.webmethods.com> and start the installer.
- 3 Specify the installation directory as the webMethods 6.5.2 installation directory (webMethods6 by default).
- 4 In the component selection list, move to the appropriate location and choose to install one or more elements:

For this element...	Choose to install...
webMethods for Microsoft Package	Integration Server ▶ webMethods for Microsoft Package 6.5.2
webMethods for Microsoft Code Generator Package	Integration Server ▶ webMethods for Microsoft Code Generator Package Package for Microsoft .NET 6.5.2
webMethods for Microsoft Plug-in	Developer ▶ webMethods for Microsoft Plug-in 6.5.2

- 5 Follow the installation instructions in the *webMethods Installation Guide*.

Install the webMethods add-in for Microsoft Visual Studio .NET

Install the webMethods add-in for Microsoft Visual Studio .NET on the same machine as Visual studio .NET.

- 1 Shut down Microsoft Visual Studio .NET.
- 2 Use the **Additional Installs** page in webMethods Administrator, as described in “[The Additional Installs Page](#)” on page 52, to access the webMethodsAddIn.msi file.

You can also download the file from the webMethods Advantage Web site at <http://advantage.webmethods.com>.

- 3 Run the executable file and follow the instructions.

Uninstall the webMethods for Microsoft Package



Important! This section provides only instructions that are specific to uninstalling the webMethods for Microsoft Package and the webMethods for Microsoft Plug-in. For complete instructions on uninstalling webMethods components, see the *webMethods Installation Guide*.

Uninstallation for the webMethods for Microsoft Package and the webMethods for Microsoft Plug-in is similar.

To uninstall elements of the webMethods for Microsoft Package

- 1 Shut down the webMethods component associated with the element:

For this element...	Shut down this component...
webMethods for Microsoft Package	Integration Server
webMethods for Microsoft Code Generator Package	Integration Server
webMethods for Microsoft Plug-in	Developer

- 2 Start the webMethods Uninstaller, as described in the *webMethods Installation Guide*.

- 3 Choose to uninstall one or more elements of the webMethods for Microsoft Package:

For this element...	Choose to install...
webMethods for Microsoft Package	Integration Server ► webMethods for Microsoft Package 6.5.2
webMethods for Microsoft Code Generator Package	Integration Server ► webMethods for Microsoft Code Generator Package Package for Microsoft .NET 6.5.2
webMethods for Microsoft Plug-in	Developer ► webMethods for Microsoft Plug-in 6.5.2

The uninstaller removes all Integration Server package-related files that were installed into the *Integration Server_directory\packages\WmDotNet* directory or *WmMSCodegen* directory. Likewise, the uninstaller removes all Developer plug-in-related files that were installed into the *Developer_directory\plugins\dotNETplugin* directory.

The uninstaller does not delete files created after you installed the component (for example, user-created or configuration files), nor does it delete the directory structure that contains the files.

- 4 If you do not want to save the files, do one or both of the following:

Traverse to...	And delete the directory...
<i>Integration Server_directory\packages</i>	WmDotNet
<i>Integration Server_directory\packages</i>	WmMSCodegen
<i>Developer_directory\plugins</i>	dotNETplugin

Uninstall the webMethods Add-In for Visual Studio .NET

- 1 Shut down Microsoft Visual Studio .NET.
- 2 Use the Windows Add/Remove Programs utility, available from the Control Panel, to start the uninstaller.
- 3 Select **webMethods add-in for Microsoft Visual Studio .NET** as the program to uninstall.

The uninstaller removes all webMethods add-in for Microsoft Visual Studio .NET files that were installed.

Using Developer with the .NET Package

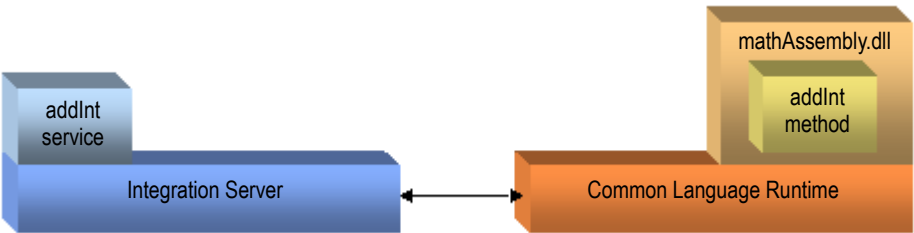
■ What is the webMethods for Microsoft Plug-in	26
■ Importing .NET Methods into Developer	26
■ Importing COM Objects into Developer	28
■ Configuring .NET Assembly Information	29
■ Managing Class Lifetimes	30

What is the webMethods for Microsoft Plug-in

With the webMethods for Microsoft Package installed, you can use the webMethods for Microsoft Plug-in to scan existing .NET assemblies to determine what methods they contain and then import the methods into Developer. In Developer, you can create services that call those methods. Once a .NET service exists within Developer, it can become part of a flow, just like any other service.

To take advantage of this feature, you must have the Microsoft .NET Framework, which includes the Common Language Runtime (CLR) installed on the same computer as the instance of Integration Server to which Developer is connected (see [“Microsoft .NET Framework” on page 20](#)). In addition, the location of the .NET assembly must be known to Integration Server. For example, assume that you have used Developer to create the service addInt to call the addInt method in mathAssembly.dll. When you invoke the addInt service on Integration Server, the service calls the addInt method through the CLR.

A service calls a .NET method



Using Developer, you can perform the following actions:

You can do this...	Described here...
Import .NET assembly methods into Developer	“Importing .NET Methods into Developer” on page 26
Modify information about a .NET method associated with a service	“Configuring .NET Assembly Information” on page 29
Manage the way the CLR maintains instance data	“Managing Class Lifetimes” on page 30

Importing .NET Methods into Developer

You can import methods from Microsoft .NET assemblies and use them to create webMethods services.



Tip! From a practical standpoint, it is easiest to import and maintain all .NET services from an assembly into the same folder.

To create a webMethods service from a .NET method

- 1 The .NET Common Language Runtime (CLR) should start automatically, but if it is not yet loaded, use Integration Server Administrator to start it.

For more information, see [“Starting and Stopping the Microsoft .NET CLR” on page 45](#).

- 2 On the **File** menu of Developer, click **New**.
- 3 In the **New** dialog box, select the **.NET Service** option and click **Next**.
- 4 Select the folder in which the services are to be created and click **Next**.
- 5 In the **Open Source Assembly(s) for Automatic Conversion** dialog box, locate a NET assembly on a drive where the CLR can access it.

This dialog box depicts drives mapped on the machine where Integration Server is running. If Integration Server is running as a service, only the local drives are recognized. The only files displayed in this dialog box are assembly DLLs or EXEs. You can select multiple .NET assemblies for importation.

- 6 Optionally, in the **Add UNC path** field of the **Open Source Assembly(s) for Automatic Conversion** dialog box, type the UNC (Uniform Naming Convention) path to that directory and press ENTER.

Use two slashes (//) or backslashes (\\) to indicate the machine name and one slash or backslash to indicate the path or shared directory on the machine. For example,

```
//test1/MyFolder/MyAssemblies
```

provides the UNC path to the MyAssemblies directory on the machine test1.

- 7 After you have selected assemblies, click **Next**.
- 8 In the **Select Specific .NET Services** dialog box, select the methods you want to import into Developer.

By default, all methods in all selected assemblies are selected. You can select or clear whole assemblies, whole classes, or individual methods as needed.

- 9 Click **Finish**.

Developer creates a service for each method and places it in the specified folder.



Note: For assemblies that are not located in the same windows domain or on the same machine as the CLR, the CLR may fail to load an assembly and issue a security error if the assembly was compiled with the unsafe option or if user permissions for the remote directory do not permit access. To resolve either condition, copy the assembly to the machine where the CLR resides, set directory permissions appropriately, or configure a trust relationship between the domains.

You can view the variables for the resulting service in the **Input/Output** tab of the editor. In addition to the variables supported by a specific method, there are variables that are part of each .NET service.

Input Variables

The `trace` variable allows you to perform method invocation tracing for a specific service. For more information, see [“Method Invocation Tracing” on page 46](#).

The `marshallingType` variable identifies the marshalling type to be used for a specific service, as described in [“Marshalling Data” on page 34](#). To use a remote reference ID to marshal data, choose `refid`. To actually send data in the form of an XML file, choose `xml`.

The `instanceID` variable identifies a particular session for object instances using a Session class lifetime, as described in [“Managing Class Lifetimes” on page 30](#). On the first invocation, the `instanceID` variable should be set to a value zero. If a Session lifetime is configured for an object instance, on subsequent invocations, you should set the variable to the `instanceID` returned from the first invocation of the method.

Output Variables

As mentioned in [“Input Variables”](#) `instanceID` is present as an output variable. This variable identifies the instance of the session object and can be used as input for subsequent invocations of the object.

A `return` variable is also present. The `return` variable contains the value returned by the method upon completion of its invocation.

Importing COM Objects into Developer

The Microsoft Type Library Importer (TLBIMP) generates a Primary Interop Assembly, which is a .NET assembly, from a COM object. Once an assembly is generated using TLBIMP, you can import the methods from that assembly into Developer as you would any .NET assembly.

In broad overview, these are the steps to import COM objects into Developer as services:

- 1 If you have not already done so, use the `regsvr32` command to register the COM object on the computer where you will make the conversion.
- 2 Use TLBIMP to generate the assembly, as in this simple example:

```
tlbimp COM_object_name /out:output_name
```
- 3 Use the output file as though it were a .NET assembly and import it into Developer, as described in [“Importing .NET Methods into Developer” on page 26](#).

Configuring .NET Assembly Information

When you create a service to call a Microsoft .NET method, Integration Server stores information needed by the .NET CLR to load the method into its processing space. If you change the location or name of the .NET assembly in which the method resides, you need to use Developer to modify that information; otherwise, any attempt to call that method from Integration Server will fail.

Information about a particular .NET method resides with the service you have created to call it. In Developer, you can see this information in the **.NET Properties** tab for the specified service.



Note: When you create multiple .NET services from an assembly, as described in [“Importing .NET Methods into Developer” on page 26](#), all of those services share information about the assembly. If you change shared information for one .NET service, it is changed for all.

The **Shared Properties** panel of the **.NET Properties** tab information that is common to all .NET services created from the same assembly:

Property	Description
Domain Name	The name of the application domain in which the service is to run.
Assembly Path	The location of the directory that holds the .NET assembly in which the method called by this service resides. Editable
Assembly Name	The name of the .NET assembly in which the method called by this service resides. Editable
Class Name	The name of the class of which the method called by this service is a part. Not editable.
Class Lifetime	The lifetime setting assigned to the class that owns the method called by this service. For information about valid class lifetime settings, see “Managing Class Lifetimes” on page 30 .
Class Timeout (Mins)	The timeout setting assigned to the class that owns the method called by this service. Valid only for a Session lifetime. See “The Class Timeout Value” on page 32 .

Another property in the **.NET Properties** tab is unique to the specific .NET service:

Property	Description
Method Name	The name of the method called by this service.

The **Assembly Path** and **Assembly Name** properties are both editable. Changes to either of these properties are applied to all .NET services that call methods in the same class definition.



To change information about a .NET method

- 1 In the Navigation panel, double-click the service for which you want to change information about its associated .NET method.
- 2 Click the **.NET Properties** tab to bring it to the front.
- 3 Perform one or more of the following actions:

<u>To do this...</u>	<u>Do this...</u>
Change the assembly pathname	In the Assembly Path field, type the new location of the directory that holds the .NET assembly in which the method resides.
Change the assembly name	In the Assembly Name field, type the new name of the .NET assembly in which the method resides.

- 4 On the **File** menu, click **Save**.
- 5 Stop and restart the CLR to clear the cache and make sure the correct assembly is loaded.

For more information, see [“Starting and Stopping the Microsoft .NET CLR” on page 45](#).

You will see that these changes are applied to all .NET services that call methods in the same assembly.

Managing Class Lifetimes

.NET classes running on the CLR can maintain instance data, which are variables containing information used across multiple invocations of methods on the class. A class instance can maintain instance data according to one of four Class Lifetime settings, described in the following table. All .NET services associated with a class have the same lifetime. When you change the class lifetime for a particular .NET service, Developer modifies the lifetimes of all services associated with the same class.

In Developer, you can manage the class lifetime for a particular .NET service in the **.NET Properties** tab for that service. The **Class Lifetime** property has the following valid settings:

Lifetime Setting	Description
Global	The webMethods for Microsoft Package creates a single instance of the class, or object, which has an unlimited lifetime. The class shares instance data among all sessions. Use this setting where it is appropriate to gather data to be retrieved by multiple users. You can create only one instance of a global object of a given type.
Session	The webMethods for Microsoft Package creates a separate object for each user. The object exists until the user session is closed or until the object times out. The default is three minutes. See “The Class Timeout Value” on page 32 .
Single-Use	The webMethods for Microsoft Package creates and destroys an object each time a .NET service calls a method in the class. Data may be kept during the lifetime of the object.
Static	The .NET service calls a method that does not require any session data to be kept. All methods of the class are static and the webMethods for Microsoft Package does not create an object.



Note: If multiple services are using a given global object or a session object at the same time, those objects need to be thread safe.

Modifying Class Lifetime

The following procedure describes how to modify the Class Lifetime setting for a .NET service.

To modify the Class Lifetime for a .NET service

- 1 In the Navigation panel, double-click the service for which you want to change the **Class Lifetime** setting.
- 2 Click the **.NET Properties** tab to bring it to the front.
- 3 In the **Class Lifetime** list, select the setting to be used for this service.
- 4 If you select the **Session** setting, determine if the current **Class Timeout** value, is correct.
For more information, see [“The Class Timeout Value”](#) next in this chapter.
- 5 On the **File** menu, click **Save**.

The Class Timeout Value

The Class Timeout value applies only to .NET services having a Class Lifetime setting of **Session**. The clock starts when the webMethods for Microsoft Package creates an instance of the class. The clock is reset whenever a .NET service accesses the class. If time expires, the object is destroyed. The default value is three minutes. You should choose a different value if you know the default timeout value would cause the object to be destroyed prematurely under normal usage.

To modify the Class Timeout value for a .NET service

- 1 In the Navigation panel, double-click the service for which you want to change the **Class Timeout** setting.
- 2 Click the **.NET Properties** tab to bring it to the front.
- 3 In the **Class Lifetime** list, make sure the **Session** setting is selected.
- 4 In the **Class Timeout** field, type the correct timeout value, in minutes.
- 5 On the **File** menu, click **Save**.

Invoking a .NET Service from Developer

To test a .NET service using Developer as the client, you use the **Run** command on the **Test** menu. Before you attempt to test .NET services, you should understand application domains and the forms of data marshalling available for .NET services.

Application Domains

An application domain is a boundary between objects in the same application scope, which is established by the CLR. Application domains provide the ability to isolate multiple applications running within a process. You can have multiple application domains loaded onto the CLR and destroy each of them independently without stopping and restarting Integration Server.

For information on managing application domains in Integration Server, see [“The Manage Application Domains Page” on page 48](#).

The webMethods for Microsoft Package provides a default application domain with the domain name webmDomain. When you run a DLL in any application domain, it becomes locked, meaning that you cannot rename modify, or delete it at the file system level. If the DLL is in the default webmDomain, you have to stop and restart Integration Server to unlock it. By creating a separate application domain running a .NET service in it, you can unlock the DLL by destroying the application domain rather than stopping Integration Server.

Using Services to Start and Destroy Application Domains

There are two ways to create and destroy application domains. The first way is through the startDomain and destroyDomain services, which you can find in the WmDotNet package in the Navigation panel of Developer:

WmDotNet/wm/dotnet/Runtime

One advantage of using the startDomain service is that you can assign a .NET configuration file to modify the behavior of the application domain.

Practical ways to Start and Destroy Application Domains

At the time you run a .NET service in Developer, you can specify an application domain in which to run it. If the application domain does not already exist, Integration Server creates it. This dynamic creation feature makes it easy to add application domains as you need them. For more information, see [“Running a .NET Service in Developer” on page 34](#).



Note: If you do not specify an application domain, the service runs in the default webmDomain application domain.

Using Administrator, you can destroy an application domain on the Manage Application Domain page. This method is easier than using the `destroyDomain` service because you can find the application domain within a table and click the **Destroy** link. For more information, see [“The Manage Application Domains Page” on page 48](#).

Marshalling Data

Marshalling is the process of packing one or more items of data into a message buffer prior to transmitting that message buffer over a communication channel.


If Integration Server were to call a .NET method and it returned an object, such as a SQL connection, Integration Server could not process the object because it has meaning only in within .NET. There are two ways to marshal data across the division between Integration Server and .NET. The first way is to use a *remote reference ID*, or an instance ID. Instead of bringing the object over, Integration Server assigns an ID. If it is necessary to make another method call that uses that object as an input, Integration Server uses the remote reference ID. If you want to keep using the same object, such a SQL connection, you should use the remote reference ID.

The other way to marshal data is by passing XML. .NET has a built-in XML serializer that takes that object, examines it, and turns it into an XML string. Integration Server can manipulate the XML string and modify data, and then use that XML string as input to another method. If you need to manipulate the object in Java in some way, use XML, but you should be aware of the possible drawbacks. The use of XML is inefficient because you are passing a large string instead of a single ID. Also you need to be careful in designing the manipulation of data within Integration Server because the .NET XML serializer may create an invalid object upon its return. For information on how to choose the marshalling type, see [“Running a .NET Service in Developer”](#) next in this chapter.

Running a .NET Service in Developer

To run a .NET service in Developer, do the following:

To run a .NET service in Developer

- 1 In the Navigation panel, double-click the .NET service you want run.
- 2 Do one of the following:
 - In the **Test** menu, click **Run**.
 - OR —
 - In the Developer toolbar, click the  **Run** icon.
- 3 If there is an **Input** field, type a value to test the service.

- 4 In the **domainName** field, type the domain name of a new or existing application domain name on Integration Server.



Note: If you do not specify an application domain, the service runs in the default webmDomain application domain.

- 5 In the **marshallingType** list, select one of the following:

Value	Marshalling type
refid	Remote reference ID
xml	XML marshalling

- 6 Click **OK**.

Developer runs the .NET service on Integration Server and returns data as appropriate.

Using the Add-In for Microsoft Visual Studio .NET

- What is Microsoft Visual Studio .NET? 38
- Using the Add-In for Microsoft Visual Studio .NET 38

What is Microsoft Visual Studio .NET?

Microsoft Visual Studio .NET is a set of development tools for building a variety of applications. The Visual Studio integrated development environment (IDE) supports several languages, including Visual C# .NET and Visual Basic .NET.

With the webMethods add-in for Microsoft Visual Studio .NET you can browse for and capture services supported by an instance of webMethods Integration Server. You can then generate a client assembly or a service provider assembly that invokes a particular webMethods service. As needed, you can also manually create webMethods client or server assemblies.

Using the Add-In for Microsoft Visual Studio .NET

With the webMethods add-in for Microsoft Visual Studio .NET, you can introspect services on Integration Server and use them to generate client code in the C# or Visual Basic language.

Opening the Add-in and Generating Code



To create a C# or Visual Basic file from a webMethods service

- 1 Open Microsoft Visual Studio .NET.
- 2 Open an existing .NET project or create a new one.
- 3 On the **Tools** menu, click **webMethods add-in for Microsoft Visual Studio .NET**.
If this menu item is not visible, make sure you have installed the add-in.
- 4 In the **webMethods add-in for Microsoft Visual Studio .NET** dialog box, provide the following information about the instance of Integration Server to which you want to connect.

In this field...	Type this...
webMethods Integration Server	The name and port number of Integration Server in the format <i>ServerName:PortNum</i> . The default is localhost:5555.
UserID	The name of a valid user account on this server.
Password	The password for the user account in UserID . Passwords are case-sensitive.

5 Click **Connect**.

The **webMethods add-in for Microsoft Visual Studio .NET** window is displayed. This window contains a tree view of packages and services on the Integration Server to which you are connected.



Tip! The add-in opens as a floating window. You can dock the window by dragging it to the Visual Studio .NET toolbar.

6 Right-click a service and click **Generate Client Code**, and then click either of the following:

Click this...	To create code in...	With this extension...
C# Code	C# .NET	.cs
VB.NET Code	Visual Basic .NET	.vb

The file takes the name of the service from which it was generated and has an extension based on the type of code generated.

Connecting to an Integration Server

To connect to an additional Integration Server, do the following:

To connect to an Integration Server


- 1 With the webMethods add-in for Microsoft Visual Studio .NET displayed, right-click the **webMethods Integration Servers** node (or other nodes as well) and click **Connect to Integration Server**.
- 2 In the **webMethods add-in for Microsoft Visual Studio .NET** dialog box, provide the following information about the instance of Integration Server to which you want to connect.

In this field...	Type this...
webMethods Integration Server	The name and port number of Integration Server in the format <i>ServerName:PortNum</i> . The default is localhost:5555.
UserID	The name of a valid user account on this server.
Password	The password for the user account in UserID . Passwords are case-sensitive.

- 3 Click **Connect**.

Disconnecting from an Integration Server

To disconnect from an Integration Server, do the following:



To disconnect from an Integration Server

- 1 With the webMethods add-in for Microsoft Visual Studio .NET displayed, right-click the node for the Integration Server you want to disconnect.
- 2 Click **Disconnect from *server_name***.

The Integration Server is disconnected and the node is removed from the tree.

Administering the .NET Package

■ Integration Server Administrator and Microsoft .NET	42
■ Accessing the webMethods for Microsoft Package	42
■ Pages for Administering the Microsoft .NET Package	43
■ The Manage CLR Page	44
■ The Manage Application Domains Page	48
■ The View Assemblies Page	49
■ The Manage .NET Object Instances Page	49
■ The Domain Details Page	50
■ The Options Page	51
■ The Additional Installs Page	52

Integration Server Administrator and Microsoft .NET

Through the webMethods for Microsoft Package, you can use webMethods Administrator to administer and manage the webMethods for Microsoft Package from any standard browser. Among the activities you can perform are:

- View general status for the webMethods for Microsoft Package
- View status for (and destroy) application domains
- Start and stop the underlying .NET Common Language Runtime (CLR)
- Unload .NET assembly dynamically-linked libraries (DLL) by destroying the application domain that loaded them
- View runtime status and statistics about individual assembly DLLs
- View and manage .NET objects managed by the .NET CLR

Accessing the webMethods for Microsoft Package

You can access the webMethods for Microsoft Package through Administrator.

To access the webMethods for Microsoft Package from a connection to Integration Server

- 1 Start your browser.
- 2 Point your browser to the host and port for an instance of Integration Server where the webMethods for Microsoft Package is installed. Use the following syntax:

http://hostname:port_number

For example, if Integration Server is running on the default port on a machine named myhost, you would type:

<http://myhost:5555>

If Integration Server is running, you are prompted for a user name and password.

- 3 Log on with a user name that has administrator privileges.

For information on maintaining administrator privileges, see the *webMethods Integration Server Administrator's Guide*.

- 4 In the **Solutions** menu of the navigation frame, click **Microsoft Package**.

Pages for Administering the Microsoft .NET Package

Administrator contains a variety of pages for administering the webMethods for Microsoft Package, depending on the menu item selected in the navigation frame. The pages in Administrator are described in the following places:

Page	Description
Manage CLR	Starts and stops the Microsoft .NET CLR, and displays information and statistics. See “The Manage CLR Page” on page 44.
Manage Application Domains	Displays information about the application domains loaded onto the Microsoft .NET CLR. See “The Manage Application Domains Page” on page 48.
View Assemblies	Displays information about the assemblies loaded onto a specific application domain. See “The View Assemblies Page” on page 49.
Manage .NET Object Instances	Displays information about object instances created and currently running in a specific application domain. See “The Manage .NET Object Instances Page” on page 49.
Options	Controls monitoring options for the webMethods for Microsoft Package. See “The Options Page” on page 51.
Additional Installs	Makes available some installation packages needed by users of the webMethods for Microsoft Package. See “The Additional Installs Page” on page 52.

Displaying Pages in Administrator

To display pages for the webMethods for Microsoft Package

- 1 If you have not already done so, open Administrator and display the webMethods for Microsoft Package pages, as described in [“Accessing the webMethods for Microsoft Package” on page 42.](#)
- 2 In the **Administration** menu of the navigation frame, click the link to the page you want to display.

The exception to this step is the Additional Installs page, which is available in the **Resources** menu of the navigation frame.

Refreshing the View

Administrator polls Integration Server at regular intervals for information about the Microsoft .NET CLR. In addition to automatic polling, you can poll at any time using the **Refresh** command.



To refresh the information in Administrator

- 1 Make sure Administrator is displaying one of the pages you can select from the navigation frame.
- 2 Click **Refresh**, commonly found in the upper-left portion of the page.

To change the interval for automatic polling, see [“The Options Page” on page 51](#).

The Manage CLR Page

Using the Manage CLR page, you can start and stop the Microsoft .NET CLR and display information about it.

To display the Manage CLR page, open Administrator as described in [“Accessing the webMethods for Microsoft Package” on page 42](#) and, in the **Administration** menu, click **Manage CLR**.

Values in the Manage CLR Page

The **CLR Info** panel contains the following values:

Value	Description
Status	Current state of the .NET CLR. Values are: <div><div>■ Loaded — The CLR is running</div><div>■ Not Loaded — The CLR is not running</div></div>
Version	The version number of the CLR (Microsoft .NET Framework)
Start Time	If the CLR is running (Loaded), contains the date and time the CLR was started

The **CLR Statistics** panel contains the following values:

Value	Description				
Method Invoke Count	The total number of methods invoked since the CLR was started.				
Application Domain Count	The number of application domains loaded onto the CLR.				
Physical Thread Count	The number of operating system threads being used by the CLR.				
Logical Thread Count	The number of logical threads as allocated within the CLR. This number is typically larger than the Physical Thread Count, but an excessive difference, several times the physical count, indicates an error.				
Total Process Memory Used	The amount of memory in use by Integration Server and the CLR combined.				
CLR Memory Used	The amount of memory in use by the CLR.				
Tracing	<p>An action you can use to enable or disable tracing (see “Tracing for the webMethods for Microsoft Package” on page 46)</p> <table> <tr> <td>Enabled</td><td>Tracing is currently enabled. Click to disable.</td></tr> <tr> <td>Disabled</td><td>Tracing is currently disabled. Click to enable.</td></tr> </table>	Enabled	Tracing is currently enabled. Click to disable.	Disabled	Tracing is currently disabled. Click to enable.
Enabled	Tracing is currently enabled. Click to disable.				
Disabled	Tracing is currently disabled. Click to enable.				

Starting and Stopping the Microsoft .NET CLR

You can start and stop the Microsoft .NET CLR in the Manage CLR page of the webMethods for Microsoft Package.

To start or stop the .NET CLR

- 1 If you have not already done so, open Administrator and display the webMethods for Microsoft Package pages, as described in [“Accessing the webMethods for Microsoft Package” on page 42](#).
- 2 In the **Administration** menu of the navigation frame, click **Manage CLR**.

- 3 In the Manage CLR page, determine the current status of the CLR by looking at the **Status** field of the **CLR Info** panel:

<u>If the status is...</u>	<u>The CLR is...</u>
Not Loaded	Not running
Loaded	Running

- 4 Perform one of the following actions:

<u>To do this...</u>	<u>Click this link...</u>
Start the CLR	Start the CLR
Stop the CLR	Stop the CLR

Tracing for the webMethods for Microsoft Package

The webMethods for Microsoft Package provides two kinds of tracing for services created from .NET assembly methods: .NET assembly method invocation tracing and .NET assembly introspection tracing. Both kinds of tracing information are written to the same log file. See [“Location of the Log File” on page 47](#).

Method Invocation Tracing

Method invocation tracing writes trace statements at all points: Idata decode, assembly load, module location, class location, method location, parameter conversion, return parameter conversion and Idata modifications. You can set method invocation tracing for individual .NET services or for all .NET services.

You can set method invocation tracing for individual services created from .NET assembly methods. This type of tracing is useful if you want to collect information on a specific service.

To set method invocation tracing for an individual service

- 1 In the Navigation panel, double-click the .NET service for which you want to enable method invocation tracing.
- 2 On the **Test** menu, click **Run**.
- 3 In the **Trace** field of the input dialog box, type `true`.

Tracing is enabled for this service every time it is run until you clear the **Trace** field or replace `true` with `false`.

Using the webMethods for Microsoft Package, you can enable method invocation tracing for all services created from .NET assembly methods, overriding any services for which the trace input parameter is set to `false`. To enable or disable all tracing, see [“Controlling Tracing from Administrator” on page 47](#).

Assembly Introspection Tracing

Assembly introspection tracing writes trace statements at all introspection points: assembly load, module discovery, class discovery, method discovery, parameter discovery, and XML document generation. You can set introspection tracing only in the webMethods for Microsoft Package. To enable or disable all tracing, see [“Controlling Tracing from Administrator” on page 47](#).

Location of the Log File

The log file for all tracing performed by the webMethods for Microsoft Package is:

```
webmethods6\IntegrationServer\logs\MicrosoftPackage-domain.log
```



Note: When tracing is enabled, the webMethods for Microsoft Package creates a large number of log messages for each service, resulting in slower processing and the consumption of large amounts of disk space. It is not recommended that this log be enabled during normal webMethods for Microsoft Package usage.

Controlling Tracing from Administrator

Use the webMethods for Microsoft Package to enable or disable tracing for assembly introspection and for all services created from .NET assembly methods. To set method invocation tracing for individual services, see [“Method Invocation Tracing” on page 46](#).

To enable tracing for all services created from .NET assembly methods

- 1 If you have not already done so, open Administrator and display the webMethods for Microsoft Package pages, as described in [“Accessing the webMethods for Microsoft Package” on page 42](#).
- 2 In the **Administration** menu of the navigation frame, click **Manage CLR**.
- 3 In the Manage CLR page, determine the current status of tracing by looking at the **Tracing** action of the **CLR Statistics** panel:

<u>If the Tracing action is...</u>	<u>Tracing is...</u>
enable	Disabled
disable	Enabled

4 Perform one of the following actions:

To do this...	Click this link...
Enable tracing	enable
Disable tracing	disable

The Manage Application Domains Page

An application domain is a boundary between objects in the same application scope, which is established by the CLR. Application domains provide the ability to isolate multiple applications running within a process. You can have multiple application domains loaded onto the CLR and destroy each of them independently without stopping and restarting Integration Server.

To display the Manage Application Domains page, open Administrator as described in [“Accessing the webMethods for Microsoft Package” on page 42](#) and, in the **Administration** menu, click **Manage Application Domains**.

The Manage Application Domains page displays the following information about the application domains loaded into the Microsoft .NET CLR.

Value or link	Description
Domain Name	The name of the application domain.
View Assemblies	A link to the View Assemblies page for the application domain. See “The View Assemblies Page” on page 49 .
View Objects	A link to the Manage .NET Object Instances page for the application domain. See “The Manage .NET Object Instances Page” on page 49 .
Method Invoke Count	The number of times methods in the assembly have been invoked since the CLR started.
Last Invoke Time	The most recent date and time a method in the assembly was invoked.
Action	An action you can use to destroy the application domain. Click Destroy .

The View Assemblies Page

To view the assemblies in an application domain

- 1 If you have not already done so, open Administrator and display the webMethods for Microsoft Package pages, as described in [“Accessing the webMethods for Microsoft Package” on page 42](#).
- 2 In the **Administration** menu of the navigation frame, click **Manage Application Domains**.
- 3 In the **View Objects** column for the application domain, click **Assemblies**.

The View Assemblies page displays the following information about the assemblies loaded into the Microsoft .NET CLR.

Value	Description
Assembly Name	The name of the assembly.
Version	The version number of the assembly, if applicable.
Culture	The locale of the assembly. A value of neutral means the assembly has not been localized.
Method Invoke Count	The number of times methods in the assembly have been invoked since the CLR started.
Last Invoke Time	The most recent date and time a method in the assembly was invoked.

The Manage .NET Object Instances Page

To view the .NET objects in an application domain

- 1 If you have not already done so, open Administrator and display the webMethods for Microsoft Package pages, as described in [“Accessing the webMethods for Microsoft Package” on page 42](#).
- 2 In the **Administration** menu of the navigation frame, click **Manage Application Domains**.
- 3 In the **View Objects** column for the application domain, click **Objects**.

An object is an instance of a class. Objects have the ability to contain data, making it possible to store data. An object with a Session lifetime allows a user session to store data needed by a later method; an object with a Global lifetime makes it possible for multiple user sessions to share data. The Manage .NET Object Instances page displays the

following information about objects with a Session or Global lifetime currently running on the Microsoft .NET CLR.

Value	Description
Action	An action you can use to destroy an object. Click Destroy .
Object Class	The class to which the object belongs.
ID	The value that identifies this particular instance.
Invoke Count	The number of methods that have been invoked on this object since it was created.
TTL/Timeout (sec)	The time to live (TTL) for this object and the timeout value established for it, both in seconds. When the TTL value equals zero, the object is destroyed. Applicable to only the Session lifetime.
Avg Response (ms)	The average response time, in milliseconds, of all methods invoked on the object since the object was created.
Last Method	The name of the method most recently invoked on the method.
Last Method Response (ms)	The response time, in milliseconds, of the method most recently invoked on the object.

The Domain Details Page



To view the details of an application domain

- 1 If you have not already done so, open Administrator and display the webMethods for Microsoft Package pages, as described in [“Accessing the webMethods for Microsoft Package” on page 42](#).
- 2 In the **Administration** menu of the navigation frame, click **Manage Application Domains**.
- 3 In the **View Details** column for the application domain, click **Details**.

The Domain Details page displays the following information about the application domain.

Value	Description
Name	The name of the application domain.
Method Invoke Count	The number of times methods in the application domain have been invoked since the CLR started.
Last Invoke Time	The most recent date and time a method in the application domain was invoked.

The Options Page

Use the Options page to change the refresh interval, which determines how often Administrator polls Integration Server for information about the webMethods for Microsoft Package.

To change the refresh interval

- 1 If you have not already done so, open Administrator and display the webMethods for Microsoft Package pages, as described in [“Accessing the webMethods for Microsoft Package” on page 42](#).
- 2 In the **Administration** menu of the navigation frame, click **Options**.
- 3 In the **Refresh Interval** field of the Options page, type a value for the polling interval, in seconds.
- 4 Click **Save Changes**.

The change in polling interval takes place immediately.

The Additional Installs Page

Use the Additional Installs page to access the webMethods add-in for Microsoft Visual Studio .NET.

To install the webMethods add-in for Microsoft Visual Studio .NET

- 1 Open Administrator on a machine where Microsoft Visual Studio .NET is installed.
- 2 In the **Resources** menu of the navigation frame, click **Additional Installs**.
- 3 Right-click the link **webMethodsAddIn.msi** and, in the pop-up menu, click **Save Link As** or **Save Target As**, depending on your browser, to save a copy on your machine.



Note: Make sure the file extension is .msi before you save the file to your machine.

- 4 Run the executable file and follow the instructions.

Glossary

application domain

A boundary between objects in the same application scope, which is established by the Common Language Runtime. Application domains provide the ability to isolate multiple applications running within a process.

assembly

A component designed to operate within the Microsoft .NET environment. Assemblies are of two types, DLLs (dynamically-linked library) and executables.

CIL

Common Intermediate Language. See [MSIL](#).

CLR

Common Language Runtime. The execution engine for .NET Framework applications.

COM

Component Object Module. Defines how object linking and embedding (OLE) objects and their clients interact within processes or across process boundaries.

CTS

Microsoft Common Type Specification. A comprehensive specification for a language-independent object-oriented data type system that is capable of representing simple data types, such as integers, or more complex data types that are embodied as classes.

DLL

Dynamically-linked library. A library that is linked to an application at run time, rather than during compilation. A DLL is a module that contains functions and data.

Framework

See [Microsoft .NET Framework](#).

IDE

Integrated development environment. A system for software development, typically including a syntax-directed editor and integrated support for compiling the program and relating errors back to the source code.

IL

Intermediate Language. See [MSIL](#).

JVM

Java Virtual Machine. Compiles Java into machine-independent object code, or bytecode.

managed code

Code executed and managed by the [CLR](#).

Microsoft .NET Framework

The programming model of the .NET platform for building and running XML Web services and applications.

MSIL

Microsoft Intermediate Language. A high-level object-oriented language that is similar to Java bytecode. The Microsoft .NET environment compiles source code into MSIL. At run time, the [CLR](#) compiles MSIL into machine code.

TLBIMP

Type Library Importer. Provides interoperability with existing [COM](#) components.

UNC

Uniform (or Universal) Naming Convention. A convention for specifying directories, servers, and other resources on a network.

Unmanaged code

Code that is executed outside the [CLR](#).

Visual Studio .NET

Microsoft Visual Studio .NET is a set of development tools for building a variety of applications. The Visual Studio [IDE](#) supports several languages, including Visual C# .NET.

Index

A

- accessing the webMethods Microsoft .NET Package 42
- add-in for Visual Studio .NET
 - connecting to Integration Server 39
 - disconnecting from Integration Server 40
 - generating code 38
 - opening 38
- application domains
 - in Developer 33
 - managing 48
 - starting and destroying
 - practical ways for 33
 - services for 33
- Assembly Name property 29
- Assembly Path property 29

C

- class library, .Net Framework 11
- Class Lifetime property 29
- Class Lifetime setting, modifying 32
- Class Name property 29
- Class Timeout property 29
- Class Timeout value 32
- Common Language Runtime (CLR) 10
- Common Type Specification (CTS) 11
- connecting to Integration Sever Administrator 42
- conventions used in this document 7
- create service from
 - .NET method 26
 - COM object 28

D

- destroying application domains
 - in Integration Server 48
 - using a service 33
- Developer plug-in for Microsoft .NET, overview 26

- documentation
 - additional 8
 - conventions used 7
 - feedback 8
- Domain Name property 29

G

- Global value, Class Lifetime property 31

I

- import
 - .NET methods 26
 - COM objects 28
- input variables in .NET services 28
- instanceID variable 28
- Integration Server Administrator, connecting to 42
- Integration Server, connecting with Visual Studio .NET 39
- Integration Server, disconnecting with Visual Studio .NET 40

J

- Java Virtual Machine (JVM) 10

M

- managing application domains 48
- Method Name property 29
- Microsoft .NET Framework 10
- Microsoft .NET Framework class library 11
- Microsoft .NET Package, accessing 42
- Microsoft Visual Studio .NET 11
- Microsoft Windows 2000 Server 10
- Microsoft Windows 2003 Server 10
- modifying Class Lifetime setting 32

O

- output variables in .NET services 28

P

program code conventions in this document 7

property

- Assembly Name 29

- Assembly Path 29

- Class Lifetime 29

- Class Name 29

- Class Timeout 29

- Domain Name 29

- Method Name 29

R

return variable 28

S

Session value, Class Lifetime property 31

Single-Use value, Class Lifetime property 31

starting application domains

- using a service 33

- when you run a service 33

Static value, Class Lifetime property 31

T

TLBIMP 11

trace variable 28

troubleshooting information 8

Type Library Importer 11

typographical conventions in this document 7

V

variables

- input 28

- output 28

Visual Studio .NET 11

- connecting to Integration Server 39

- disconnecting from Integration Server 40

- generating code 38

- opening the add-in 38