

Welcome to Enterprise Library. The following sections of this guidance describe the ways that you can use Enterprise Library and the individual application blocks in your applications. The sections are:

- What Is Enterprise Library?
- About This Release of Enterprise Library
- Developing Applications with Enterprise Library
- Design of Enterprise Library
- The Caching Application Block

- The Cryptography Application Block
- The Data Access Application Block
- The Exception Handling Application Block
- The Logging Application Block
- The Policy Injection Application Block
- The Security Application Block
- The Validation Application Block
- <u>Unity Dependency Injection and Interception</u>

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **What Is Enterprise Library?**



Enterprise Library consists of a collection of application blocks and core infrastructure. All of these are reusable software components designed to assist developers with common enterprise development challenges.

Enterprise Library also provides many highly configurable features that make it much easier to manage repetitive tasks, known as crosscutting concerns, which occur in many places in your applications. These include tasks such as logging, validation, caching, exception management, and more. In addition, the dependency injection container it provides can help to simplify and decouple your designs, make them more testable and understandable, and help you to produce more efficient designs and implementations of all kinds of applications.

■ Three Simple Steps for Using Enterprise Library

■ Benefits of Enterprise Library

■ Goals for Enterprise Library

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

About This Release of Enterprise Library



This section contains the following topics that will help you to understand this release of Enterprise Library and how you use it alongside earlier versions or migrate your applications to this version. This section includes the following topics:

- Changes in This Release
- Target Audience and System Requirements
- Contents of Enterprise Library
- Migration and Side-by-Side Execution
- Related patterns & practices Links
- Copyright and Terms of Use

■ How to Use This Guidance

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Changes in This Release



Enterprise Library 5.0 is a new release of the Microsoft patterns & practices Enterprise Library. One of the major changes is the implementation of full dependency injection capabilities for instantiating and managing the lifetime of objects, which makes creation of Enterprise Library objects consistent across the entire library. Enterprise Library can now be used with different dependency injection containers. Unity is the default dependency injection container. For alternative DI container configuration plug-ins go to the patterns & practices: Enterprise Library Contrib Web site.

This release also includes additions in functionality to several of the existing application blocks. In addition, this release has been adapted to work with both Microsoft Visual Studio® 2008 and Microsoft Visual Studio 2010; and with the Microsoft .NET Framework versions 4.0 and 3.5 with Service Pack 1.

The following sections discuss these and other changes:

- Integration of Unity and Object Builder
- Breaking Changes
- Changes That Affect All Application Blocks
- Changes to the Configuration Tool
- Changes to the Caching Application Block
- Changes to the Cryptography Application Block
- Changes to the Data Access Application Block
- Changes to the Exception Handling Application Block
- Changes to the Logging Application Block
- Changes to the Policy Injection Application Block
- Changes to the Security Application Block
- Changes to the Validation Application Block

Go to CodePlex for information on **Known Issues**.

■ Integration of Unity and Object Builder

■ Breaking Changes

■ Changes That Affect All Application Blocks

■ Changes to the Configuration Tools

■ Changes to the Caching Application Block

■ Changes to the Cryptography Application Block

■ Changes to the Data Access Application Block

■ Changes to the Exception Handling Application Block

■ Changes to the Logging Application Block

■ Changes to the Policy Injection Application Block

■ Changes to the Security Application Block

■ Changes to the Validation Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Target Audience and System Requirements



This guidance is intended for software architects and software developers. To get the greatest benefit from this guidance, you should have an understanding of the following technologies:

- Microsoft Visual C#® or Microsoft Visual Basic® .NET
- Microsoft .NET Framework

■ System Requirements and Prerequisites

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Contents of Enterprise Library



Enterprise Library is a combination of reusable components, a supporting infrastructure, and guidance. It contains the following:

- **Binaries**. The Enterprise Library includes pre-compiled, strong-named assemblies for all the source code. For more information, see Deploying Enterprise Library.
- **Source code**. The Enterprise Library includes the source code for the application blocks, the core infrastructure, and the configuration tool.
- **Unit tests**. The Enterprise Library includes the unit tests that were created while the application blocks were being developed. For more information, see <u>Unit Tests</u>.
- Documentation. Enterprise Library includes documentation that can be viewed with the Visual Studio Help system. The documentation includes guidance about how to use the Enterprise Library and a class library reference.

A <u>Migration Guide</u>, a wide range of examples, <u>Hands-On-Labs</u>, and other learning materials are available from the <u>Enterprise Library community</u> Web site.

The following topics describe the contents of Enterprise Library within the context of their function:

- The Enterprise Library Application Blocks
- The Enterprise Library Core
- The Enterprise Library Configuration Tools
- The Instance Creation and Dependency Injection Mechanism
- Utilities, Tools, and Guidance

Additional features that you can use with Enterprise Library are available from the Enterprise Library Community Site.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Enterprise Library Application Blocks



The Enterprise Library application blocks help to address the common problems that developers face from one project to the next. They are designed to encapsulate the Microsoft recommended practices for .NET applications. In addition, they can be added to .NET applications quickly and easily. For example, the Data Access Application Block provides access to the most frequently used features of ADO.NET in simple-to-use classes, thus boosting developer productivity. It also addresses scenarios not directly supported by the underlying class libraries.

Different applications have different requirements, and you will not find that every application block is useful in every application that you build. Before using an application block, you should have a good understanding of your application requirements and of the scenarios that the application block is designed to address.

This release of Enterprise Library contains the following application blocks:

- The Caching Application Block. Developers can use this application block to incorporate a local cache in their applications.
- The Cryptography Application Block. Developers can use this application block to incorporate hashing and symmetric encryption in their applications.
- <u>The Data Access Application Block</u>. Developers can use this application block to incorporate standard database functionality in their applications.
- The Exception Handling Application Block. Developers and policy
 makers can use this application block to create a consistent strategy for
 processing exceptions that occur throughout the architectural layers of
 enterprise applications.
- The Logging Application Block. Developers can use this application block to include standard logging functionality in their applications and systems administrators can use the configuration tool to adjust the granularity of logging at run time.
- The Policy Injection Application Block. This block contains legacy code for backwards compatibility with existing applications. The new functionality is available by using the Unity interception mechanism and

call handlers located in the related application block assemblies.

- The Security Application Block. Developers can use this application block to incorporate authorization and security caching functionality in their applications.
- The Validation Application Block. Developers can use this application block to create validation rules for business objects that can be used across different layers of their applications.
- <u>Unity Dependency Injection and Interception</u>. Developers can use this to implement a lightweight, extensible dependency injection container with support for constructor, property, and method call injection; and to capture calls to target objects and add additional functionality to the object.

The Enterprise Library also includes a set of core functions, including configuration and instrumentation. All other application blocks use these functions. See <u>The Enterprise Library Core</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

The Enterprise Library Core



Enterprise Library contains a set of core features that integrate the application blocks with the configuration system, support alternative configuration systems, and provide common utilities used across all of the application blocks. The following are some examples of the contents of the Enterprise Library core:

- Run-time configuration classes and providers that expose configuration data to the library, the application blocks, and users' applications at run time
- Common utility functions for tasks such as serialization, used in many places throughout the library and the application blocks and available for developers to use in their code
- Instrumentation features that allow developers and administrators to monitor the behavior and performance of the application blocks at run time
- Design-time configuration classes that support the configuration tools and allow developers to specify and persist configuration information for the library and the application blocks

For more details about the design of the Enterprise Library core features, see <u>The Enterprise Library Core</u> in the <u>Design of Enterprise Library</u> section of this guidance.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Enterprise Library Configuration Tools



Application blocks are designed to be used in a variety of enterprise application development scenarios. This design approach enables you to easily adapt an application block to meet the needs of your situation. You do this with configuration settings, which you can easily change using the configuration tools. You can define configuration settings for an application block's central functionality and for each provider type.

You can use Visual Studio to create and modify configuration files or you can use the stand-alone Enterprise Library configuration console. Collectively, the Visual Studio integrated configuration editor and the stand-alone configuration console are referred to as the configuration tools. Both of these configuration tools have identical functionality. With these tools, you can change and validate application block settings without manually editing the XML configuration files where they are stored. The configuration tools display these settings and supply default values that you can change.

Each application block defines points of extensibility, where developers can include their own implementations (typically, these are providers) with specific functionality. For example, you can add your own custom log entry formatters to the Logging Application Block. These custom providers can be interchanged with the providers that are supplied with the application block. The configuration console lets you select the custom backing store and writes this information to the appropriate XML configuration file. This means that the application will use the custom store without any code changes and without being recompiled.

The Visual Studio integrated configuration tool is named the configuration editor. The configuration editor has the same functionality as the stand-alone configuration console, but it uses the Visual Studio Properties grid to display an application block's properties and uses the errors list to display configuration errors.

The following describes some of the activities the configuration tools help you with:

• You can use the configuration tools to create and modify standard .NET

Framework **<appSettings>** sections. For more information, see <u>Using</u> the appSettings Section.

- You can use the configuration tools to tailor an application block's configuration to a particular run-time environment. For more information, see Configuring a Deployment Environment.
- You can use the configuration tools to encrypt and decrypt the data contained in configuration sections. For more information, see Encrypting Configuration Data.

Using configuration settings to adapt an application block to a particular situation has two advantages:

- Different people can configure the characteristics of an application block at different times during the application life cycle. For example, a developer could configure a provider to access a particular database during application development, while a system administrator could decide during deployment to encrypt the database connection strings.
- You can change the application block configuration incrementally for increasingly complex situations. For example, you could initially configure an application block to use the default settings and providers. As your understanding of the scenario deepens, you can change the application block configuration without modifying its code, recompiling it, or redeploying it.

For more information on using the Visual Studio configuration editor and the configuration console see <u>Using the Configuration Tools</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

The Instance Creation and Dependency Injection Mechanism



Enterprise Library incorporates an easy-to-use mechanism for creating and wiring instances of objects (which may contain other dependent objects), and managing their lifetimes. This feature is the Unity container and the following design patterns are used:

- Dependency Injection
- Inversion of Control (IoC)
- Service Locator
- Service Container
- Factory
- Builder

■ Unity and Enterprise Library

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Utilities, Tools, and Guidance**



To help developers learn how to use Enterprise Library, and to make sure they get optimum results from it, the installation of Enterprise Library includes the following utilities, tools, and guidance:

- **Batch files** that build the Enterprise Library source code and copy the assemblies to the appropriate locations. For more information, see Building Enterprise Library from the Source Code.
- **Utilities** to install the instrumentation required by Enterprise Library. For more information, see **Enabling Instrumentation**.

In addition, the source code for Enterprise Library includes Visual Studio projects and unit tests that developers can use to extend and modify the library and the application blocks. Developers can make sure applications still meet the design requirements by running the unit tests and writing new tests. For information about the unit tests included with Enterprise Library, see <u>Unit Tests</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0 Migration and Side-by-Side Execution



In general, applications built for Enterprise Library for .NET Framework 2.0 in January 2006 and later will function with this release of Enterprise Library without the need for any code changes. It will be necessary to update the references to refer to the new assemblies and to update the configuration files to reference the correct version of the assemblies. To review the few breaking changes you may need to address go to the Breaking Changes section.

This version of Enterprise Library can also be installed side by side with earlier versions of Enterprise Library. You can deploy new applications written for this version of Enterprise Library along with applications written for earlier versions. In addition, you can choose to migrate existing applications, one assembly at a time, to the new version.

If you decide to use side-by-side execution, you must deploy the different Enterprise Library versions in different directories. In any specific directory, you cannot mix and match assemblies from different versions. For example, you cannot have Data Access Application Block version 5.0 in the same directory with Caching Application Block version 4.0.

The shipped project files use data in the AssemblyInfo.cs file to build assemblies that have different version information. This enables you to use strong names and to add different versions to the global assembly cache for side-by-side execution.

■ Partial Migration

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Related patterns & practices Links



For information related to Enterprise Library, and other tools and guidance for designing and building applications, see the patterns & practices Web site and guides:

- Microsoft patterns & practices Developer Center
- Developer's Guide to Microsoft Enterprise Library 5
- Microsoft Application Architecture Guide, 2nd Edition
- Solution Development Fundamentals
- patterns & practices Security Guidance for Applications Index
- .NET Data Access Architecture Guide
- Improving .NET Application Performance and Scalability
- Monitoring in .NET Distributed Application Design
- Deploying .NET Framework-based Applications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Copyright and Terms of Use



This document is provided "as-is". Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2010 Microsoft. All rights reserved.

Microsoft, Windows, Windows Server, Windows Vista, Visual C#, Visual Basic, and Visual Studio are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> <u>Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Developing Applications with Enterprise Library



Enterprise Library can be useful in a variety of situations; the following are some examples:

- Enterprise Library provides sufficient functionality to support many common scenarios that enterprise-level applications must address.
- Enterprise Library can serve as the basis for a custom library. You can take advantage of the extensibility points incorporated in each application block and extend the application block by supplying new providers. You can also modify the source code for the existing application blocks to incorporate new functionality. You can develop extensions for existing application blocks and new application blocks yourself, or you can use extensions and application blocks developed by others. Finally, you can add new application blocks to Enterprise Library.
- Enterprise Library is designed so that its application blocks can function independently of each other. You have to add only the application blocks that your application will use; you do not have to add the entire library.
- Enterprise Library includes the source code for the application blocks. This means you can modify the application blocks to merge into your existing library or you can use parts of the Enterprise Library source code in other application blocks or applications that you build.
- Enterprise Library includes comprehensive documentation. This means that you can use the library and the source code as tools for learning architectural, design, and coding best practices.
- A wide range of examples, Hands-On-Labs, and other learning materials are available from the Enterprise Library community Web site.

This section describes the general procedures for working with Enterprise Library in your applications. You can find details of how to use each of the application blocks in the relevant section for each. This section includes the following sets of related topics:

• Configuring Enterprise Library. These topics describe the basic and more advanced procedures for configuring Enterprise Library; including using the configuration tools, sharing and managing configuration for multiple applications, configuring Enterprise Library programmatically,

- encrypting configuration files, and enabling the built-in instrumentation.
- <u>Using Enterprise Library in Applications</u>. These topics describe how to add the Enterprise Library assemblies to your projects, import the required namespaces, and create instances of Enterprise Library objects that exercise the functionality of the application blocks.
- Deploying Enterprise Library. These topics discuss the issues that you should consider when deploying Enterprise Library and applications that use it. This includes versioning and strong naming assemblies if you modify the source code for Enterprise Library, and pointers to help you use Enterprise Library in partial trust scenarios.
- Administering Enterprise Library. This topic summarizes the techniques available to administrators and operators for running multiple versions of Enterprise Library, managing configuration, using the built-in instrumentation, and integrating with system management tools.
- Extending and Modifying Enterprise Library. These topics provide advice on extending and modifying Enterprise Library by changing the source code, and information about creating custom providers that integrate with Enterprise Library and the configuration tools.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Configuring Enterprise Library



Configuration information for the Enterprise Library application blocks is stored in XML format. Typically, the information exists in an XML file. By default, the XML file that contains the information is the application configuration file.

You can manually edit the XML data, but the Enterprise Library provides two configuration tools that greatly simplify this task. These tools are the standalone configuration console and the configuration editor that is integrated with Visual Studio®. If you choose to manually edit the XML, refer to the appropriate application block documentation for schema details.

✓ Note:

When creating a configuration file and adding application blocks, <u>Using the Configuration Tools</u> provides a less error prone experience and is the recommended process. Editing the XML in Visual Studio does not enforce the configuration file hierarchy at all levels and can result in invalid XML.

The configuration files are not encrypted, they are in clear text, by default. A configuration file may contain sensitive information about connection strings, user IDs, passwords, database servers, and catalogs. You should protect this information against unauthorized read/write operations by using encryption techniques. To provide security they must be encrypted or protected using access control lists (ACLs). It is recommended that the configuration store be in the same trust boundary and that decrypting the configuration is done in the same trust boundary after the configuration is read. For information about how to encrypt configuration files, see Encrypting Configuration Data.

This section includes the following topics:

- <u>Using the Configuration Tools</u>
- Advanced Configuration Scenarios
- Using Group Policy with Enterprise Library
- Configuring a Deployment Environment
- Using the Fluent Configuration API
- Using the appSettings Section

- <u>Updating Configuration Settings at Run Time</u>
- Encrypting Configuration Data
- Enabling Instrumentation
- Source Schema for Enterprise Library Core

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Using the Configuration Tools



The Enterprise Library configuration tools are graphical tools that allow you to create, change, and validate application block settings without having to manually edit the XML configuration files where they are stored. There is a configuration editor that integrates with Visual Studio 2008, plus standalone versions for different operating system and Microsoft® .NET Framework versions that you can launch from within Visual Studio or open from your Start menu.

These tools display the available configuration settings, the default values that you can change, and information about what each setting means.

This topic contains the following sections:

- Launching Configuration Editor from Visual Studio
- Setting the Visual Studio Configuration Editor as the Default Editor
- Launching the Standalone Configuration Tool
- Using Wizards in the Configuration Tool
- Using the Keyboard with the Configuration Tool
- The Configuration Type Selector
- Enterprise Library Configuration Schema
- Building the Configuration Console
- Specifying Different Assemblies
- Configuration Tool Usage Notes

Launching the	e Configuration	Editor from	Visual	Studio

■ Setting the Visual Studio Configuration Editor as the Default Editor

■ Launching the Stand-alone Configuration Tool

• Using	Wizards in the Configuration Tool	

■ Using the Keyboard with the Configuration Tool	

■ The Configuration Type Selector

■ Enterprise Library Configuration Schema

Building	the	Configuration	Console

■ Specifying Different Assemblies

■ Configuration Tool Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Advanced Configuration Scenarios



The Enterprise Library standalone configuration console and the Visual Studio integrated configuration editor allow you to satisfy a range of advanced configuration scenarios based on external configuration sources such as disk files. For example, you can:

- Read configuration information from a wide range of sources.
- Enforce common configuration settings across multiple applications.
- Share configuration settings between applications.
- Specify a core set of configuration settings that applications can inherit.
- Merge configuration settings that are stored in a shared location.
- Create different configurations for different deployment environments.

The default and simplest scenario for configuring Enterprise Library is to configure your application using the configuration tool without adding a **Configuration Sources** section or any configuration sources. This is the approach described in <u>Using the Configuration Tools</u>.

When you use the configuration tools without specifying a configuration source, they default to using the System Configuration Source to create a single configuration file that contains the entire configuration for the application. Your application will expect this to be named App.config file (which is copied to [exe-name].config when you compile your application) or Web.config (depending on the technology you are using), and will read it automatically.

You can select **Add Configuration Settings** on the **Blocks** menu to display the section that contains the default System Configuration Source. If you click the chevron arrow to the right of the **Configuration Sources** title to open the section properties pane you can see that the System Configuration Source is also, by default, specified as the **Selected Source**—the configuration source to which the configuration generated by the tool will be written. When an application that uses Enterprise Library reads the configuration, it uses the settings specified for the selected source.

By adding additional configuration sources, you can implement more advanced configuration scenarios. The following sections describe the scenarios that you can implement using the configuration tools:

- <u>Using a Non-default Configuration Store</u>
- Sharing the Same Configuration between Multiple Applications
- Managing and Enforcing Configuration for Multiple Applications
- Sharing Configuration Sections across Multiple Applications
- Applying a Common Configuration Structure for Applications
- Managing Configuration in Different Deployment Environments

You can also change the contents of a configuration file programmatically, which is useful when working with shared configuration stored as disk files. For more information, see <u>Updating Shared Configuration Settings</u>

<u>Programmatically</u>. For links to related topics and more details of how the configuration system works, see <u>More Information</u> at the end of this topic.

■ Using a Non-default Configuration Store

■ Sharing the Same Configuration between Multiple Applications

■ Managing and Enforcing Configuration for Multiple Applications

	Sharing	Configuration	on Sections	across Multip	ple Ap	plications
--	---------	---------------	-------------	---------------	--------	------------

• <i>F</i>	Applying a	Common	Configuration	Structure fo	r Ap	plications
	-PP-J8 a	Commi	Cominguitation	ou acture ro	P	piicacions

■ Managing Configuration in Different Deployment Environments

■ Updating Shared Configuration Settings Programmatically

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

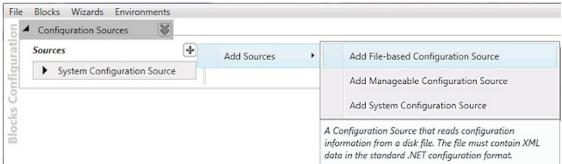
Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0 Using a Non-default Configuration Store

The Enterprise Library configuration tools allow you to specify the configuration source to be read at run time to configure your application. You can point to configuration files or other types of configuration sources from which to read configuration information. This topic explains how to add a non-default configuration source to your configuration. It uses the File-based Configuration Source, but the same approach is used with other configuration sources.

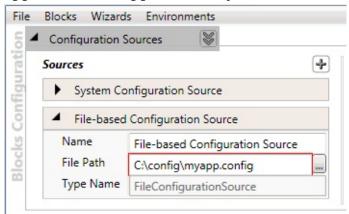
To add a configuration source

- 1. On the taskbar, click **Start**, point to **All Programs**, point to **Microsoft patterns & practices**, point to **Enterprise Library 5.0**, point to **Enterprise Library Configuration**, and then select the version of the configuration editor you require. Alternatively, right-click on a configuration file in Visual Studio Solution Explorer and click **Edit Enterprise Library V5 Configuration**.
- 2. Select **Add Configuration Settings** on the **Blocks** menu to display the section that contains the configuration sources.
- 3. Click the **Sources** plus-sign icon, point to **Add Sources**, and then click **Add File-based Configuration Source**.

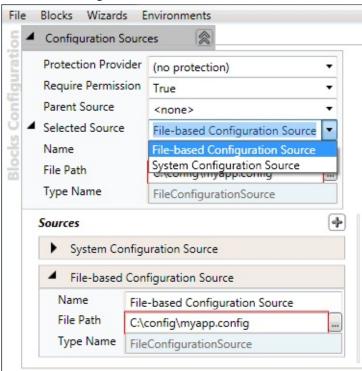


- 4. A new section is added to the **Sources** pane with the default name of the type of configuration source you added, for example **File-based Configuration Source**. If you add more than one of the same configuration source type, the default names increment by one each time.
- 5. Set the properties of the configuration source. For example, set the **File Path** property by typing the file path and name of the file, or by

clicking the ellipsis button (...) and navigating to the file you want to use. The file can be located on the local machine, or in a shared location if you want to use the same configuration for multiple applications or application layers.



- 6. Edit the **Name** property of the configuration source if required.
- 7. Click the **Configuration Sources** property expander chevron to open the properties pane for the section. Set the **Selected Source** property by selecting the configuration source you want to use for your application from the drop-down list.



8. On the **File**, menu click **Save** or **Save As**.

✓ Note:

A QuickStart sample named **SqlConfiguration** that implements an example **SQL Configuration Source** to demonstrate how you can store configuration information in a database is available from the <u>Enterprise Library Contrib</u> Web site. This provider uses the Data Access Application Block to read configuration settings from a SQL Server© database.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

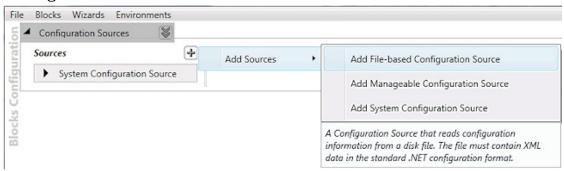
Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

You can use redirection to read the configuration for individual application blocks (individual configuration sections) from a shared location or other configuration source. This enables you to use a local application configuration for some application blocks, while reading and including the shared sections configuration. You create composite configurations like this by specifying additional sources in your application configuration file. The contents of the individual sections in the configuration source can be read and included with the contents of your default application configuration file at run time.

The following procedure explains how you can use the configuration tools to redirect Enterprise Library to read a specific configuration section's content from a specified source.

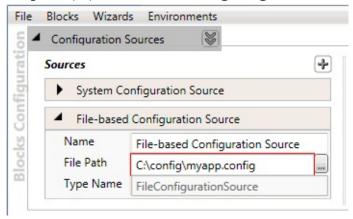
To use redirection to read a configuration section from a shared source

- 1. On the taskbar, click **Start**, point to **All Programs**, point to **Microsoft patterns & practices**, point to **Enterprise Library 5.0**, point to **Enterprise Library Configuration**, and then select the version of the configuration editor you require. Alternatively, right-click on a configuration file in Visual Studio Solution Explorer and click **Edit Enterprise Library V5 Configuration**.
- 2. On the **Blocks** menu click **Add Configuration Settings**.
- 3. In the **Sources** pane click the plus-sign icon, point to **Add Sources** and then click the type of configuration source you want to add to the configuration. This is the configuration source to which you will redirect configuration sections. In this example, we add a File-based Configuration Source.

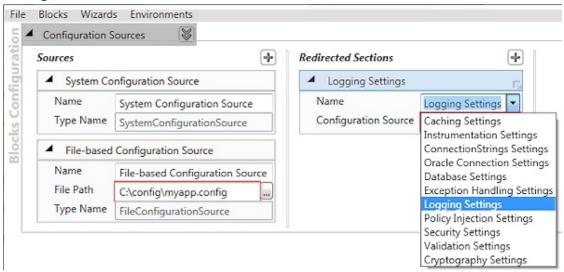


4. In the **Sources** column, enter the desired **File Path** property for the new

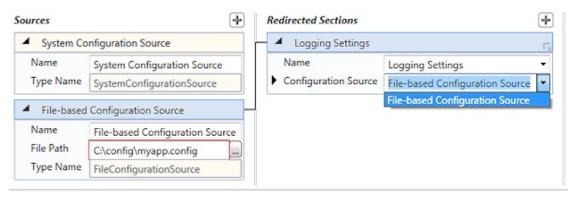
configuration source by typing the path and file name or by clicking the ellipses (...) button and navigating to the file.



- 5. You can now redirect sections from your local source to another using the **Redirected Sections** pane. Click the plus sign icon in the **Redirected Sections** column and click **Add Redirected Section**.
- 6. In the new **Redirected Section** that is displayed, select the configuration section that you want to redirect to a shared configuration source. After you select the section, the name of the redirected section changes to reflect this.



7. Select the configuration source that will expose the shared configuration section you want to use in the drop-down list for the **Configuration Source** property of the redirected section. A connecting line appears showing which configuration source will provide the information for the redirected section.



8. Click **Save** or **Save** As to save the configuration.

✓ Note:

In the Configuration Tool, when redirected sections are used with a **Selected Source** other than **System Configuration Source**, the redirected sections configuration information will still be saved to the **System Configuration Source**, which is the file opened in the configuration tool. To edit the contents of the shared configuration sections, open the configuration that contains them directly.

■ An Example of the XML Generated by Redirected Sections

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

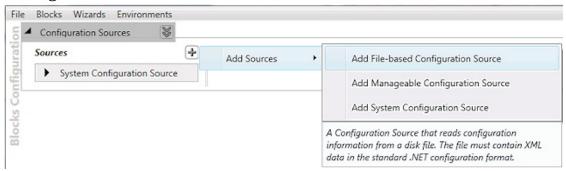
To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

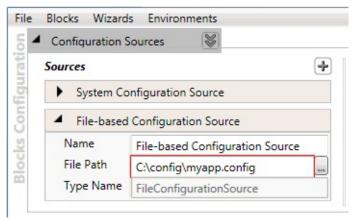
If you add a configuration source to your application configuration and specify that this configuration source is the parent source, Enterprise Library will read the contents of this configuration source and merge any local configuration settings you have specified with this parent configuration. Effectively, it inherits the shared configuration and then applies the local settings to override any that occur in both configurations. You can use the configuration tool to specify a parent configuration that will be inherited.

To inherit a shared configuration structure and settings from another configuration source

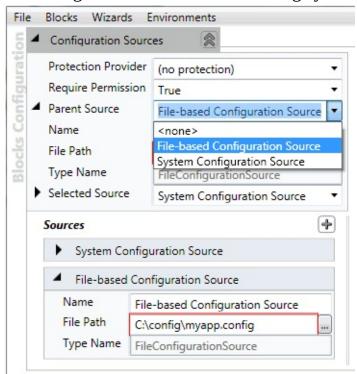
- 1. On the taskbar, click **Start**, point to **All Programs**, point to **Microsoft patterns & practices**, point to **Enterprise Library 5.0**, point to **Enterprise Library Configuration**, and then select the version of the configuration editor you require. Alternatively, right-click on a configuration file in Visual Studio Solution Explorer and click **Edit Enterprise Library V5 Configuration**.
- 2. On the **Blocks** menu click **Add Configuration Settings**.
- 3. In the **Sources** pane click the plus-sign icon, point to **Add Sources** and then click the type of configuration source you want to add to the configuration. This is the configuration source from which you will inherit configuration information. In this example, we add a File-based Configuration Source.



4. Set the properties of the configuration source you added. For example, enter the desired **File Path** property for the new configuration source by typing the path and file name or by clicking the ellipses (...) button and navigating to the file.



5. Click the **Configuration Sources** property expander chevron to open the properties pane for the section. Set the **Parent Source** property to the configuration source whose settings you want to inherit.



6. Click **Save** or **Save As** on the **File** menu to save your configuration.

Settings in the inherited configuration source will apply to your application unless you override them by configuring the same setting in your local configuration. For information about how settings are inherited and merged, see Merge Rules for Inherited Configuration.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to

pagdoc@microsoft.com.

Copyright $\ensuremath{\text{@}}$ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Merge Rules for Inherited Configuration



When you specify a parent source from which you inherit shared configuration information, all of that configuration information is read and the settings are merged at run-time at the level of individual setting elements. For example, you can add exception handlers to an existing policy, or add a property to an element in the local configuration that does not exist in the default settings. The run time merge basically behaves like a logical OR. If the element or property is in one of the configuration files, it will be present in the merged configuration.

If an element or property is present in both sources when you inherit a section, the element in the local configuration (the local source) is the one present in the resulting merged configuration. Inheritance can only be one level deep; you cannot define a parent source for a configuration source that is itself used as a parent source. This prevents the accidental creation of circular references and endless recursion.

✓ Note:

When you use **environmental overrides** to specify different configurations for different deployment environments in conjunction with settings inherited from a parent source, the merge rules described here do not apply to the inherited configuration settings. The merged configuration is specified by the combination of the common and the delta (differences) configuration files that you merge.

When you use **redirected sections**, and an element or property is present in both sources, the element from the shared configuration (the source you redirect the section to) is the one present in the resulting merged configuration. Local settings in a redirected section are ignored. If you specify a parent source as well as redirected sections, the merge behavior applies to the redirected sections.

To avoid issues when you use different tools to manage your configuration, you should avoid inheriting the standard .NET configuration sections that are also used by Enterprise Library, such as **appSettings** and **connectionStrings**.

When creating a composite configuration, there is a hierarchy to the components merged into your Enterprise Library configuration at run time, as described in <u>Additive Merge</u> below. In addition, there is a hierarchy for merging of the collections from the specified sources, described in the section <u>Merged Collections Order</u> later in this topic.

■ Additive Merge

■ Merged Collections Order

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

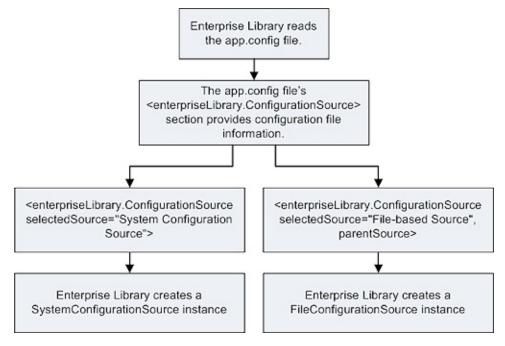
To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Enterprise Library first looks at the default configuration file (the current application domain configuration file). This will be either your App.config file (which is copied to [exe-name].config when you compile your application) or your Web.config file. This file may contain a configuration section that defines additional configuration sources. The core configuration classes that support reading configuration information first try to read the requested configuration section from the configured source location.

The **enterpriseLibrary.ConfigurationSource** section in the configuration file carries the **selectedSource** attribute, which specifies the configuration source that the application should use. The

<enterpriseLibrary.ConfigurationSource> section is retrieved from the current application domain's configuration file, and Enterprise Library uses the configuration information exposed by the configuration source specified by the selectedSource attribute.



The source defined by the **selectedSource** attribute could be any of the following sources:

• A **System Configuration Source**, which reads from the application's configuration file.

- A **File-based Configuration Source**, which reads from an arbitrary file.
- A **Manageable Configuration Source**, which reads from an arbitrary file and applies group policy overrides, or any custom configuration source.
- The sample **SQL Configuration Source** available from the **Enterprise Library Contrib Project** Web site, which reads configuration data from a database.
- A custom configuration source that you create and add to Enterprise Library.

If there is no **<enterpriseLibrary.ConfigurationSource>** section in the configuration file, an instance of the **SystemConfigurationSource** class is used by default and becomes the application configuration source.

✓ Note:

Configuration sources are an extensibility point in Enterprise Library.

The following XML fragment shows a configuration section that specifies Enterprise Library should read configuration information using the File Configuration Source.

XML

■ Redirection and Inheritance Configuration Property and Attribute Settings

■ Saved Configuration Settings

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Using Group Policy with Enterprise Library



Group Policy provides a centralized one-to-many management capability for Windows that allows administrators to define settings that apply to a group of computers, systems, services, or applications. Group Policy relies on the Active Directory® service to manage the settings applied at run time to members of the Active Directory forest, domain, or subgroup. For more information about Group Policy, see Group Policy on MSDN.

Enterprise Library includes a manageable configuration source. This allows you to use Group Policy to manage an Enterprise Library application. You do not need to write any application code to use these features. This topic provides information about the following tasks for using Group Policy:

- Adding a Manageable Configuration Source
- Generating and Installing Group Policy Templates
- Troubleshooting Group Policy

Adding	a	Manageal	ole (Configu	ration	Source

■ Generating	and Installing	Group	Policy Templat	es

■ Troubleshooting Group Policy

■ Administrator Enters Invalid Values

■ Application Is Modified Without Generating New Group Policy Template

Policies	Conflict to	Produce	Invalid	Configurat	ion

Other Errors

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Configuring a Deployment Environment



You can use the configuration tools at design time to customize the run-time settings of your configuration to suit a particular deployment environment. This feature is useful if you have multiple environments that share the same basic configuration but require different property settings. For example, you may have a development environment that uses one connection string for the Data Access Application Block and a test environment that uses a different one.

Every environment you configure maintains its own environment delta file (using the file name extension .dconfig), which is updated after you save the configuration. This environment delta file contains the differences between this environment and the standard environment, plus some additional metadata for merging the differences into a new complete configuration file that you can deploy. The main advantage is that you can distribute an environment delta file separately from the original configuration file and allow management by users or administrators who have access to all the passwords, server names, and other details of that environment.

Therefore, instead of maintaining multiple configuration files or having to change a file manually or programmatically, you can create a base configuration file (.config) and an environment delta file that contains the differences (.dconfig).

You can use environmental property overrides to override specific properties for a block and for the properties of the child elements for a block, and then create a deployment environment by saving the configuration delta. Then you can export the merged configuration file and copy or deploy it to production.

The lifetime of the main configuration and the lifetime of the environmental overrides are linked only through the structure of the files. The following operations can be performed on environment delta files:

- **File** | **Save**: Saving the main configuration also saves any related environments. The environment .dconfig file is saved using the path specified in the **Environment Delta File** property of the environment override.
- File | Save As: Specifying a new path or name only affects the main

configuration file. You must change the path and name specified in the **Environment Delta File** property of the environment override to change the environment name and path.

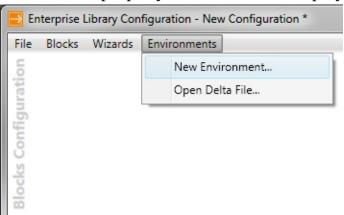
- **File** | **New**: Creating a new configuration clears the existing configuration and all open environments, creates a new empty configuration file, and clears the validation results. You are prompted to save the file being cleared if it has any changes.
- **Validation**: You can save configuration and environment files only if all files are valid. Either all are saved or none are saved.

To open or create a configuration file in order to create a deployment environment

- 1. Click on **Open** in the **File** menu to open an existing .config file, or click **New** on the **File** menu to create a new file, and then add the appropriately configured application blocks in the configuration tool.
- 2. Browse to the configuration file you want or the location where you want to save the new file.
- 3. Select a configuration file and view the configured application blocks.

To customize a block's properties for a run time environment

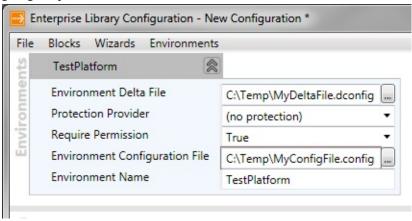
1. On the **Environments** menu, click **New Environment** to open a new environment configuration file. You must have an environment open in order for the property overrides to be displayed.



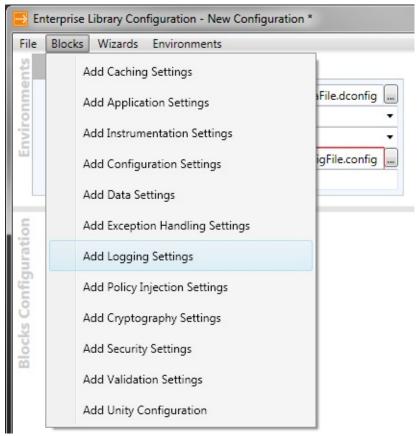
2. Click the **Environment** property expander chevron to open or close the environment properties edit pane.

✓ Note:

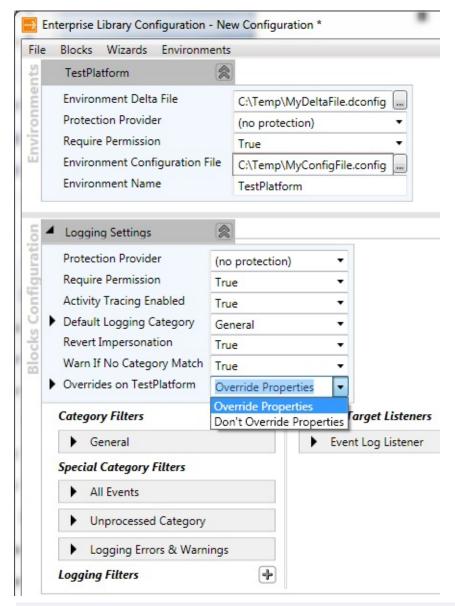
You can have multiple environment configuration files open simultaneously. Each is displayed by name. 3. In the edit pane, set the properties. The **Environment Delta File** property is the name of the delta file. If you want to encrypt the file, set the **Protection Provider** property by selecting a provider from the drop-down list. The **Environment Configuration File** property is the name of the merged configuration file. The **Environment Name** property is the name of the environment.



- 4. Repeat the previous steps for each environment you want to create.
- 5. To customize a block's properties for this environment, add the application block to the configuration. Open the **Blocks** menu and select the block you require by clicking on its settings; for example, **Add Logging Settings**.



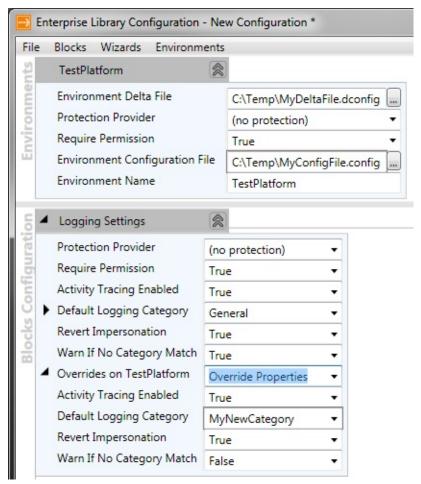
6. Click the blocks properties expander arrow to view the list of properties. In the **Overrides on** [environment name] drop-down box, click **Override Properties** to enable setting the block's properties for this environment.



✓ Note:

If you rename an environment, the updated name may not be displayed in the **Overrides on** [environment name] property. However, it will be shown the next time you open the configuration in the tool.

7. Click the expander arrow on the left of the named environment listed under the block's properties, in this screen shot to the left of **Overrides on TestPlatform**, to display the properties for that named environment. You can then use each property drop-down edit box to edit the properties appropriately for your requirements.



8. Repeat steps 4, 5 and 6 for each block you want to customize.

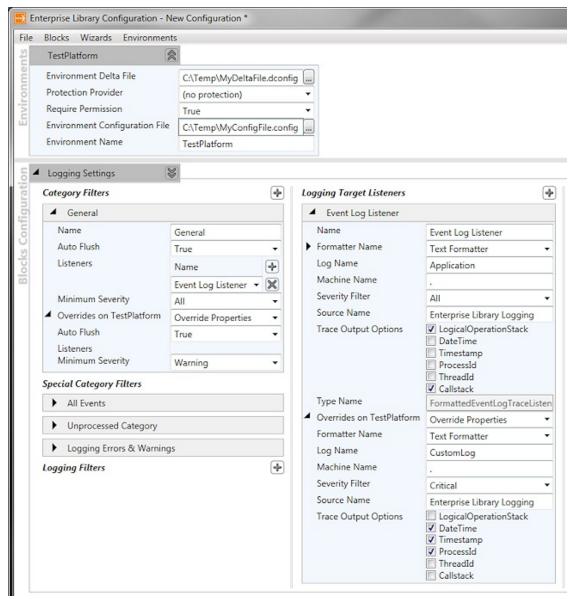
✓ Note:

The configuration tool allows you to add more than one additional environment. Each configured item then displays a set of properties for the base configuration file and a set of properties specific to each of the named environments (with the exception of properties that cannot be overridden). This enables you to edit the base configuration file properties or the delta file properties for a specific named environment.

To customize a block's child element properties for a run-time environment

- Click on the child expander arrow to the left of the block name
 —Logging Settings in this example—to display the child elements if they are not already visible.
- 2. In each of the child elements that you want to configure for different

- environments, click on the property expander for the environment you want to edit—**Overrides on TestPlatform** in this example.
- 3. Set the **Overrides on** [*environment name*] property to **Override Properties**, and then edit the properties you want to override for that environment. This screenshot shows how you can specify different property values for a Logging block category filter and a target listener in different environments.



To save a configuration (.config) file and a delta environment (.dconfig) file

1. Select **Save** or **Save As** on the **File** menu to save both the named environment configuration file and the delta file; if either file name is blank you will be prompted to select a path and file name.

- 2. Click **Save**. The configuration file is saved and you are prompted for a path and name to save the delta environment file.
- 3. Click **SaveAs** and you are prompted first for a path and name to save the configuration file. Then you are prompted for a path and name to save the delta environment file.

To create a merged environment configuration file

- 1. Right-click on the named **Environment** for which you want to create a merged deployment configuration file, then click **Export Merged Environment Configuration File**.
- 2. To save just the differences between the main and an environment override configuration, click **Save Environment Delta File**.

You can also merge your main configuration file and your environment delta file from the command line. This is useful if you want to use build scripts or automate deployment. To merge the configuration files, you must provide both the main configuration file and the environment delta file in your command. The following command shows the syntax.



MergeConfiguration.exe configFile deltaFile [mergedFi]

The parameters are the following:

- configFile. This is the main configuration file (.config). It is merged with the environment delta file.
- deltaFile. This is the environment delta file (.dconfig). It contains the information that is merged into the main configuration file.
- mergedFile. This is the output file. This file results from merging configFile with deltaFile. If you do not specify the mergedFile, the file name stored in deltaFile is used.

✓ Note:

The environment delta files (.dconfig) you create must be reloaded each time you close and reopen the configuration file.

To reload an environment delta configuration file

1. Start the Enterprise Library configuration console or open the configuration file in the Visual Studio configuration editor. For more

- information on the Enterprise Library configuration console see <u>Using</u> the <u>Configuration Tools</u>.
- 2. On the **File** menu, click **Open**.
- 3. In the **Open** dialog box, browse to and select the application .config file you want to reload.
- 4. In the Enterprise Library configuration tool, left-click **Environments**, and then click **OpenDelta File**.
- 5. In the **Open** dialog box, select the environment delta configuration (.dconfig) file you want to reload, and then click the **Open** button.
- 6. Save the changes through the **File** menu or the right-click shortcut menu on the environment item, as described in the previous procedure.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Using the Fluent Configuration API



It is possible to programmatically manipulate the default configuration classes used by Enterprise Library for the core, instrumentation, and all of the application blocks. The fluent interface exposed by Enterprise Library is designed to facilitate this process. The fluent interface can be used for all of the configurable features of instrumentation and for all of the Enterprise Library application blocks with the exception of the Validation and Policy Injection Application Blocks.

Using the fluent configuration API, you can:

- Create configuration sources to pass to Enterprise Library objects that you create in your applications.
- Create configuration sources for entire sections of the configuration, such as for the Exception Handling Application Block or the Logging Application Block.
- Create the complete configuration for Enterprise Library for an application.
- Create different environment-specific configurations for sections or for entire applications; this makes it easier to deploy an application to different run-time environments.
- Modify the configuration at run time; perhaps to take into account external factors or changes to the environment.

Effectively, the fluent configuration API allows you to specify a set of configuration sections programmatically and, if required, merge them with other configuration settings specified in other sources such as a file configuration source. It also allows you to programmatically define a base configuration that can be changed from another source, such as a file configuration source, which provides the ability to override parts of the base configuration.

Working With the Fluent Configuration API

Enterprise Library provides the **ConfigurationSourceBuilder** class that acts as the basis for creating configuration sources. Each of the blocks, and the core and instrumentation features of Enterprise Library, provides extensions to this class that expose a fluent interface and make it easy to create configuration sources.

The general approach is to create an instance of the

ConfigurationSourceBuilder class and then specify the section you want to configure. The extensions for each configuration section expose intuitive methods that allow you to add the appropriate policies, providers, or other types of extensions to the configuration and specify the required configuration settings for each one. In Visual Studio, the IntelliSense® feature will help you to locate and use the individual methods available for each configuration extension. For example, you can configure instrumentation for a configuration source using the following code.

The final line in this code creates a new container and assigns the configuration to it. Enterprise Library will use this configuration to resolve the objects it requires. You can also use the methods of the **IServiceLocator** interface to obtain instances of Enterprise Library objects on demand in your code by accessing the **IServiceLocator** implementation through the **EnterpriseLibraryContainer.Current** property.

The API also provides methods that allow you to check if a specific section already exists in the configuration, and add new sections to the configuration. For performance reasons, the fluent configuration API does not perform full validity checking of the settings you specify. It does check for null values, however. To obtain full validity checking, you can use the graphical configuration tools to set up and verify your configuration, and then translate it into calls to the configuration API.

All configuration source classes implement the **IConfigurationSource** interface. This interface allows your application code to subscribe to notifications of configuration changes. For more information, see <u>Updating Configuration Settings at Run Time</u>. By default, in Enterprise Library, only the Logging Application Block registers to receive notifications of configuration changes.

The following examples show how you can configure the other blocks in Enterprise Library using the fluent API:

- The Caching Application Block
- The Cryptography Application Block
- The Data Access Application Block
- The Exception Handling Block
- The Logging Application Block
- The Security Application Block

■ The Caching Application Block

■ The Cryptography Application Block

■ The Data Access Application Block

■ The Exception Handling Block

■ The Logging Application Block

■ The Security Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright @ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0 **Using the appSettings Section**



You can use the configuration tools to create and modify Microsoft .NET Framework **<appSettings>** sections. Note that the **<appSettings>** sections must be included in a single file; they should not be distributed across multiple files. Examples of files that you cannot edit include the Machine.config file and **<appSettings>** sections that include the optional **file** attribute that specifies a relative path to an external configuration file.

■ Configure an <appSettings> Section Using the Configuration Tool

■ Configure an <appSettings> Section by Editing the XML

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Updating Configuration Settings at Run Time



The default configuration system included in the Enterprise Library Core is, with one exception, read-only. When using application configuration files (App.config and Web.config), updates to these files do not affect existing instances of objects. However, new instances created after changes are made to the configuration file will reflect these changes. In Windows Forms applications, you can restart the application to cause it to read the all of the new configuration information. Web Forms (ASP.NET) applications will detect and reload the configuration information, but the standard behavior of ASP.NET causes the application to restart when you edit the configuration file, which causes all state to be lost for the application.

The one exception to this is the Logging Application Block, which is able to detect configuration changes and reload the configuration without restarting the application. This works for Web Forms and Windows Forms applications, though it will still automatically trigger the application to restart for Web Forms applications. This means that you cannot rely on maintenance of in-process session state in ASP.NET applications when you change the configuration file.

■ Detecting Configuration Changes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Encrypting Configuration Data



You can encrypt and decrypt the data in a configuration file's configuration sections. A configuration section contains the configuration information for an application block. The configuration tool allows you to select from the encryption providers that are included in the Machine.config file. Typically, these are the **DataProtectionConfigurationProvider**, which uses the Windows data protection API (DPAPI), and the **RsaProtectedConfigurationProvider**, which uses RSA.

If the encrypted configuration file is going to be on only a single server, you can use the **DataProtectionConfigurationProvider**. If you want to deploy the same encrypted configuration file on multiple servers in a Web farm, you should use the **RsaProtectedConfigurationProvider**. This provider makes it easy for you encrypt the data on one server computer and then export the RSA private key needed to decrypt the data. You can then deploy the configuration file and the exported key to the target servers, and then re-import the keys.

The user account used for encrypting the file when using the **RsaProtectedConfigurationProvider** must have the appropriate minimal permissions, which must include read permissions on the **NetFrameworkConfigurationKey** key container, in order to encrypt and decrypt sections when using the Enterprise Library configuration tools. By default, this includes only administrative accounts.

The appropriate minimal permissions, which must include read permissions, are also required for run time and configuration merges performed when using the configuration tools and working with configuration sections that have been encrypted by using the **RsaProtectedConfigurationProvider**.

For more information see <u>Creating and Exporting an RSA Key Container</u> on MSDN.

✓ Note:

Whenever you change security settings and permissions, be sure that you are aware of any security risks raised by giving elevated permissions.

To encrypt a configuration section

- 1. Open one of the Enterprise Library configuration tools.
- 2. Open an existing configuration file or create a new one.
- 3. Click the chevron arrow at the right of name of the application block whose configuration information you want to encrypt.
- 4. In the Properties pane, click the drop-down list for the **Protection Provider** property.
- 5. Select either **DataProtectionConfigurationProvider** or **RsaProtectedConfigurationProvider**.

All the settings for the providers, such as where keys are stored, are also in the Machine.config file. You cannot change this file with a configuration tool. Instead, you must modify the file using a text editor.

To decrypt a configuration file, simply open it in the configuration tool. The file is automatically decrypted.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Enabling Instrumentation



The term instrumentation refers to a system's built-in ability to monitor or measure performance and to diagnose errors. Enterprise Library incorporates the following instrumentation in the majority of the application blocks:

- **Event logs**. The application blocks inform users of key events, such as errors or warnings.
- **Performance counters**. The application blocks record key metrics—such as the number of critical events that occur per second or the average time it takes to complete key tasks—by writing to the Microsoft Windows® operating system performance counters.

For details of the design of the Enterprise Library instrumentation, see <u>Instrumentation</u>.

■ Enabling	and	Disabling	Instrumentation

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Source Schema for Enterprise Library Core
```

The Enterprise Library Core provides services, such as instrumentation and configuration, and all Enterprise Library application blocks except for Unity are dependent on the core. The core library functionality is contained in the assembly **Microsoft.Practices.EnterpriseLibrary.Common.dll**.

There are two configuration sections associated with the Enterprise Library Core. They are the **instrumentationConfiguration** section and the **enterpriseLibrary.ConfigurationSource** section. These sections define which types of instrumentation are enabled in the application (if any), and which configuration source should be used to access configuration information within the application. The following XML code shows the configuration section declarations that define the locations of these configuration sections. These declarations should be defined within the **<configSections>** section of the application configuration file.

If there is no **enterpriseLibrary.ConfigurationSource** section in the configuration file, an instance of the **SystemConfigurationSource** class becomes the configuration source for the application. This means that when the application creates Enterprise Library objects, it retrieves configuration information from the application configuration file.

■ enterpriseLibrary.ConfigurationSource Element

• sources Element

■ instrumentationConfiguration Child Element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Using Enterprise Library in Applications



Enterprise Library can be used in most types of applications and in the majority of scenarios where you require an easy-to-use yet highly flexible library to help you manage crosscutting concerns. For example, you can use Enterprise Library in application built with Windows Forms, Windows Presentation Foundation (WPF), Windows Communication Foundation (WCF), ASP.NET, Silverlight®, and console-based applications. You can also use Enterprise Library in both greenfield and brownfield scenarios.

When architecting new applications, Enterprise Library can be used to implement many of the common design patterns you incorporate in your design. You can plan for its use throughout the layers and components of your application. You can also use Enterprise Library successfully in existing applications, where it can help to simplify the task of updating or adding new functionality. The configuration tools do not affect existing information in your application configuration files, and the assemblies and code can be added to existing applications without fear of conflicts.

Enterprise Library includes source code for the application blocks, and compiled and signed versions of the application block assemblies. You can use these assemblies directly, compile the application blocks and use the compiled assemblies, or include the source code in your application. However, before you can use the assemblies in your application, you must add references to the relevant application block assemblies, and to the Common and Unity assemblies. Then, to write application code, you also must be aware of the Enterprise Library namespace conventions and object creation patterns.

This section contains the following topics that will help you to get started using Enterprise Library:

- Referencing Enterprise Library Assemblies. This topic explains how to add references to the required assemblies to your application, and import these into your projects.
- Dependencies in Enterprise Library. This topic explains how some of the blocks depend on others, and how all of the blocks depend on core features of Enterprise Library. It will help you to understand which assemblies and features you require depending on the blocks you use and

your own scenarios.

• <u>Creating and Referencing Enterprise Library Objects</u>. This topic describes the various ways of accessing Enterprise Library objects, what their advantages and disadvantages are, and details of how they work in the underlying code.

For information about how to compile the application blocks, see <u>Building</u> <u>Enterprise Library from the Source Code</u>. For information about the design of the dependency injection mechanism, see <u>The Dependency Injection Model</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Referencing Enterprise Library Assemblies



Before you can build applications that incorporate the Enterprise Library application blocks, you must add references to the application block assemblies and to the Common and Unity assemblies. This prepares your application to use the Enterprise Library. Take care to select the assemblies that reflect your decision to use the Microsoft strong-named assemblies, or the non-strong-named assemblies, or a customized set of Enterprise Library assemblies.

To prepare your application

- 1. Add a reference to the application block assembly. In Visual Studio, right-click your project node in Solution Explorer, and then click **Add Reference**. Click the **Browse** tab, and then find the location of the application block assembly (the script CopyAssemblies.bat copies all application block assemblies to the **bin** subdirectory). Select the assembly, and then click **OK** to add the reference. For example, to reference the Logging Application Block assembly, browse to the **bin** subdirectory, select the assembly
 - **Microsoft.Practices.EnterpriseLibrary.Logging.dll**, and then click **OK**.
- 2. Use the same procedure to set a reference to the following assemblies:
 - $\bullet \ \ Microsoft. Practices. Enterprise Library. Common. dll$
 - Microsoft.Practices.ServiceLocation.dll
 - Microsoft.Practices.Unity.dll
 - Microsoft.Practices.Unity.Interception.dll.

✓ Note:

You will also need the assembly Microsoft.Practices.Unity.Configuration.dll if you wish to reference specific Unity configuration classes in your code. However, in the majority of cases, you will not require this assembly.

■ Enterprise Library Namespaces

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Dependencies in Enterprise Library



This topic describes the dependencies between the blocks and on the core features of Enterprise Library. It contains the following sections that describe the inter-block dependencies for common scenarios:

- Dependencies for All Application Blocks
- Block Dependency Schematic
- Additional Dependencies for the Caching Application Block
- Additional Dependencies for the Exception Handling Application Block
- Additional Dependencies for the Policy Injection Application Block
- Additional Dependencies for the Security Application Block

■ Dependencies for All Application Blocks	

■ Block Dependency Schematic

Additional Dependent	dencies for the	Caching A	Application	Block
----------------------	-----------------	-----------	--------------------	--------------

■ Additional Dependencies for the Exception Handling Application Block

■ Additional Dependencies for the Policy Injection Application Block

■ Additional Dependencies for the Security Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Creating and Referencing Enterprise Library Objects



This topic discusses the ways you can instantiate and access Enterprise Library objects, the advantages and disadvantages of each approach, and more details about how they work in the underlying code. Enterprise Library offers a great deal of functionality, and many different ways to access it. This topic is intended to clarify the choices and to help you determine what will work best for your application.

Typically, you will create instances of Enterprise Library objects using one of the following two approaches:

- <u>Using the Unity Service Locator</u>. This is the simplest approach, and is recommended for simple applications that have few dependencies, and where you do not want to take advantage of contemporary architectural patterns such as dependency injection. It requires no initialization or setup. You simply configure your application to use Enterprise Library and then call the methods of the service locator to obtain instances of Enterprise Library types on demand.
- Accessing the Unity Container Directly. This more sophisticated approach allows you to obtain the full benefits of contemporary architectural patterns such as dependency injection for your layers, components, and custom types. It requires only minimal setup, but may require that you maintain a reference to the container in your application.

When you use either of these approaches, you will typically request and obtain references to one or more non-static objects and interfaces that are part of each application block, which allow you to access the functionality of the blocks and obtain instances of Enterprise Library objects in your code using both dependency injection and the service locator approach. For a list of these objects and interfaces, see Non-Static Instances and Instance Factories.

Other approaches to creating Enterprise Library objects that you may choose include:

• Using a different service locator if you decide to use a container other than Unity. For more information about how the service locator works, see Initializing and Setting the Current Container. For more information about replacing the default Unity container, see The Dependency

- <u>Injection Model</u> in the section "Design of Enterprise Library".
- Using the legacy static facades and factories that were the default approach in previous versions of Enterprise Library. For more information about these, see <u>Legacy Static Facades and Factories</u>.

■ Using the Unity Service Locator

■ Accessing the Unity Container Directly

■ Non-Static Instance	ces and Inst	ance Factories
-----------------------	--------------	----------------

■ Legacy Static Facades and Factories

■ Initializing and Setting the Current Container

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.



Typically, if you do not use constructor, property, or method call injection to resolve and populate all of your application dependencies during initialization, you will need to retain a reference to the container if you wish to access it directly to resolve objects. However, there are some other situations in which you may want to be able to reference the container after initialization. The following are some scenarios in which you may consider storing a reference to the container for use in your code:

- If you are using ASP.NET Web Forms or building a Web service. You must be able to resolve objects for each page or service request, and it is quite expensive in terms of resources and does not make sense to create the container and load the Enterprise Library configuration information or your own custom registrations every time. In this case, you will usually store the populated container in the ASP.NET Application dictionary or within your service implementation and use it to resolve instances as required for each request.
- If you are creating a console application or a component (rather than an application with a user interface or a Web service). If you are using Unity as your container, you can create the container and load the Enterprise Library container extension in your startup code and then use it to resolve the dependencies of other classes on demand using the Resolve method. Any dependencies defined in these classes will be populated automatically. You can hold it in a global variable throughout the life of the application. Note that disposing the container is not recommended.
- **If you want to store registrations for your own objects** within the container. You may choose to use a separate container for this, though you can just as easily make use of the default container that holds the Enterprise Library configuration registrations.
- If you want to be able to resolve instances of objects on demand rather that always having them resolved when a class is instantiated. For example, if an object is only required in certain cases (perhaps based on the run-time environment or some other factor), it may be useful to be able to call the methods that resolve instances from the container at run

time instead of using constructor, property, or method call injection to create them when the class is initialized.

The following table will help you to understand when and where you should hold a reference to the container in **forms-based and rich client applications** built using technologies such as Windows Forms, Windows Presentation Foundation, and Silverlight.

Task	When	Where
Create and configure container.	At application startup.	Main routine, startup events, application definition file, or as appropriate for the technology.
Obtain objects from the container.	At application startup, and later if required.	Where appropriate in the code.
Store a reference to the container.	At application startup.	Global application state.
Dispose the container.	When the application shuts down.	Where appropriate in the code or automatically when the application ends.

The following table will help you to understand when and where you should hold a reference to the container in **request-based applications** built using technologies such as ASP.NET Web applications and Web services.

Task	When	Where
Create and configure container.	At application startup.	HTTP Module (ASP.NET and ASMX), InstanceContext extension (WCF).
Obtain objects from the container.	During each HTTP request.	In the request start event or load event. Objects are disposed when the request ends.
Store a reference to the	At application startup.	Global application state or service context.

container.		
Dispose the container.	When the application shuts down.	Where appropriate in the code.

■ Resolving an Entire Object Graph at Application Startup

■ Resolving and Injecting Objects on Demand

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Windows Forms and WPF Applications



Although they are very different technologies, Windows Forms and Windows Presentation Foundation (WPF) share some fundamental characteristics. In particular, they both use window objects to implement the user interface and allow you to specify which code should run when the application starts up. By default, the startup code simply loads and shows the main window, but you can easily modify this to create a Unity container, populate it with the Enterprise Library configuration information, and resolve the objects used by the application.

For example, you may not wish to create all of the windows at startup to minimize startup times or memory usage. You can store a reference to the container and use it to resolve windows and other types on demand.

The following sections contain more details:

- Windows Forms Applications
- Windows Presentation Foundation Applications

■ Windows Forms Applications

+	Windows	Presentation	Foundation	Applications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

ASP.NET Web Forms Applications



In ASP.NET Web Forms applications, the recommended approach is to store the container in the global state provided by the **Application** dictionary object. You can then access the container when required or—even better—use an HTTP module to perform injection on all of the controls in the page automatically.

In general, you should use the **Application** dictionary object to store the single instance of the container. You may decide to create child containers of the main container and store them in each user **Session** object, or even for each request, to register your custom types and mappings in the child containers. However, this may reduce application performance, and you should generally avoid creating additional containers if possible.

The following sections describe the techniques and limitations for instantiating the container in ASP.NET applications. They include a basic and simple approach to using the **Application** object to store the container, followed by the recommended approach that—while more complicated—will perform injection on the controls in your page automatically at run time:

- The Basic Approach. This is the simplest approach, which may be suitable in small applications where discoverability and testability are less of a concern.
- Recommended Approach for Dependency Injection. This technique can perform injection on the controls in your page automatically at run time, and provide better discoverability and testability.
- <u>Limitations and Alternative Approaches</u>. This section describes some of the issues you should be aware of when using the container in ASP.NET applications.

■ The Basic Approach

Recommended	l Ap	proach	for I	Dependency	y In	jection
-------------------------------	------	--------	-------	------------	------	---------

■ Limitations and Alternative Approaches

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

ASP.NET Dependency Injection HTTP Module
```

You can use an HTTP module, an extension to the ASP.NET **HttpApplicationState** class, and code in Global.asax to force ASP.NET to automatically inject dependent objects at every page request, as discussed in the topic <u>ASP.NET Web Forms Applications</u>.

The following code shows a suitable HTTP module that captures the **PreRequestHandlerExecute** event to insert itself into the ASP.NET execution pipeline. On each page request, the handler resolves the current HTTP handler through the container using the **BuildUp** method and then captures the **OnPageInitComplete** executes, the module code walks the complete control tree, resolving each control through the container using the **BuildUp** method.

The **BuildUp** method takes an existing object instance, resolves and populates any dependencies specified for the class, and then returns the instance. If there are no dependencies, it simply returns the original instance.

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.UI;
using Microsoft.Practices.Unity;

namespace Unity.Web
{
   public class UnityHttpModule : IHttpModule
   {
     public void Init(HttpApplication context)
     {
        context.PreRequestHandlerExecute += OnPreRequest
     }
     public void Dispose() { }
```

```
private void OnPreRequestHandlerExecute(object ser
  IHttpHandler currentHandler = HttpContext.Currer
  HttpContext.Current.Application.GetContainer().E
                      currentHandler.GetType(), cu
  // User Controls are ready to be built up after
  var currentPage = HttpContext.Current.Handler as
  if (currentPage != null)
    currentPage.InitComplete += OnPageInitComplete
}
// Build up each control in the page's control tre
private void OnPageInitComplete(object sender, Eve
{
  var currentPage = (Page)sender;
  IUnityContainer container = HttpContext.Current.
  foreach (Control c in GetControlTree(currentPage
  {
    container.BuildUp(c.GetType(), c);
  context.PreRequestHandlerExecute -= OnPreRequest
// Get the controls in the page's control tree exc
private IEnumerable<Control> GetControlTree(Control
  foreach (Control child in root.Controls)
  {
    yield return child;
    foreach (Control c in GetControlTree(child))
      yield return c;
 }
}
```

```
}
Visual Basic
Imports System
Imports System.Collections.Generic
Imports System.Web
Imports System.Web.UI
Imports Microsoft.Practices.Unity
Namespace Unity.Web
  Public Class UnityHttpModule
    Implements IHttpModule
    Public Sub Init(ByVal context As HttpApplication)
               Implements System.Web.IHttpModule.Init
      AddHandler context.PreRequestHandlerExecute, Add
    End Sub
    Public Sub Dispose() Implements System.Web.IHttpMc
    End Sub
    Private Sub OnPreRequestHandlerExecute(ByVal sende
      Dim currentHandler As IHttpHandler = HttpContext
      HttpContext.Current.Application.GetContainer().E
                          currentHandler.[GetType](),
      ' User Controls are ready to be built up after p
      Dim currentPage = TryCast(HttpContext.Current.Ha
      If Not currentPage Is Nothing Then
        AddHandler currentPage.InitComplete, AddressOf
      End If
```

' Build up each control in the page's control tree Private Sub OnPageInitComplete(ByVal sender As Obj Dim currentPage = DirectCast(sender, Page) Dim container As IUnityContainer = _ HttpContext

End Sub

```
For Each c As Control In GetControlTree(currentF
        container.BuildUp(c.[GetType](), c)
      Next
      RemoveHandler context.PreRequestHandlerExecute,
    End Sub
     Get the controls in the page's control tree excl
    Private Function GetControlTree(ByVal root As Cont
      Dim controlList As New List(Of Control)
      For Each child As Control In root. Controls
        If child Is Nothing Then
          Exit For
        Else
          controlList.Add(child)
        End If
        For Each c As Control In GetControlTree(child)
          If child Is Nothing Then
            Exit For
          Else
            controlList.Add(child)
          End If
        Next
      Next
      Return controlList
    End Function
  End Class
End Namespace
```

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

You can use an HTTP module, an extension to the ASP.NET **HttpApplicationState** class, and code in Global.asax to force ASP.NET to automatically inject dependent objects at every page request, as discussed in the topic <u>ASP.NET Web Forms Applications</u>.

The following shows a suitable implementation of an application state extension that exposes a static **GetContainer** method. The method creates a new Unity container in the **Application** state if one does not already exist, or returns a reference to the existing instance.

```
C#
using System.Web;
using Microsoft.Practices.Unity;
namespace Unity.Web
  public static class HttpApplicationStateExtensions
    private const string GlobalContainerKey = "EntLib(
    public static IUnityContainer GetContainer(this Ht
      appState.Lock();
      try
        var myContainer = appState[GlobalContainerKey]
        if (myContainer == null)
        {
          myContainer = new UnityContainer();
          appState[GlobalContainerKey] = myContainer;
        return myContainer;
      finally
```

```
{
          appState.UnLock();
     }
}
```

Visual Basic

```
Imports System.Web
Imports Microsoft.Practices.Unity
Namespace Unity.Web
 Public Module HttpApplicationStateExtensions
    Private Const GlobalContainerKey As String = "Entl
    <System.Runtime.CompilerServices.Extension> _
    Public Shared Function GetContainer(appState As Ht
      appState.Lock()
      Try
        Dim myContainer = TryCast(appState(GlobalConta)
        If myContainer Is Nothing Then
          myContainer = New UnityContainer()
          appState(GlobalContainerKey) = myContainer
        End If
        Return myContainer
      Finally
        appState.UnLock()
      End Try
    End Function
  End Module
End Namespace
```

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

WCF and ASP.NET Web Service Applications



To initialize the container and populate dependencies in a Web service application requires a different approach from the types of applications that expose a user interface (such as Windows Forms, WPF, and ASP.NET Web Forms). This topic describes a possible solution for ASP.NET Web services (ASMX), and points to resources that will help you implement the process in a Windows Communication Foundation (WCF) application.

■ ASP.NET Web Service Applications

WCF Applications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Creating Application Block Objects



You can resolve instances of both Enterprise Library objects and your own custom classes using the container. To understand how to resolve the appropriate Enterprise Library objects, you must be aware of the way that the Enterprise Library configuration information for your application maps to the registrations in the container.

Default and Named Object Registrations

Most of the Enterprise Library application blocks use plug-in providers to accomplish their tasks, while providing the flexibility required for use in the widest range of scenarios and environments. For example, you might add two cache managers to the Caching Application Block; one that accesses a database backing store, and one that uses the local Isolated Storage mechanism. If you use the Data Access Application Block, you might define several connections to different databases, depending on the requirements of your application.

Each provider is identified by a name, and the mappings between the provider interface or base class and the concrete implementations of the providers are differentiated by this name. An application configuration may, for example, define two cache managers, **DBStore** and **LocalStore**, while the Data Access Application Block configuration may contain mappings named **CustomerDB**, **SalesDB**, and **ProductsDB**.

When you create instances of these objects using any of the approaches shown in the related topics listed below, you must provide the name to select the appropriate object. However, Enterprise Library also has the concept of a default provider for most of the blocks. This is specified in the configuration, and defines the provider that the block will use if you do not specify a named provider. When you resolve a type without providing a name, the container will return an instance of the default provider if it is specified for the block. This is a useful feature that makes it easy to switch the application to use a different provider by just changing the default for the block in the configuration file.

For information about configuring default and named providers, see the topic "Entering Configuration Information" in the section "Developing Applications" for each of the application blocks.

Creating and Resolving Objects

The following topics describe in more detail the three main scenarios for creating or injecting instances of objects:

- Injecting Resolved Types into Other Classes
- Resolving Instances of Types Using Unity
- Creating Application Block Objects Directly

For details on how to create and reference objects see <u>Creating and Referencing Enterprise Library Objects</u>. For information about initializing the Enterprise Library container and—if required—storing a reference to it, see <u>Storing a Reference to the Container</u>.

✓ Note:

Previous versions of Enterprise Library used static factory methods to create application block objects. The static facades and static types are included in this version of Enterprise Library, and existing code that uses them will continue to work. For more information about using the static facades and static types, see the online guidance at http://msdn.microsoft.com/entlib/.

One point to note if you are familiar with the provider factory approach for generating instances of objects used in versions of Enterprise Library prior to version 5.0 is that the static facades used in these earlier versions cannot be injected. New non-static facades are included in this version of Enterprise Library.

For more information about Unity, see <u>Unity Dependency Injection and Interception</u>. For details of how Unity integrates with the underlying configuration mechanism in Enterprise Library, see <u>The Dependency Injection Model</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Injecting Resolved Types into Other Classes



One of the major advantages provided by dependency injection mechanisms such as Unity is the ability to automatically inject resolved instances of types into other classes at run time. This is the preferred approach, as it provides several advantages over other techniques (for details of these advantages, see Creating and Referencing Enterprise Library Objects.)

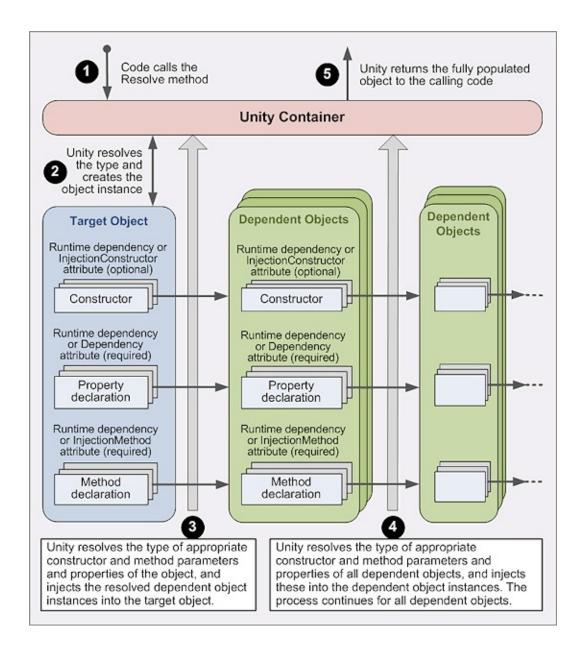
Unity supports injection into the parameters of constructors and methods, and to set the value of properties. This means that you can easily inject instances of the Enterprise Library objects and your own components and services into custom business objects and other components. The following sections of this topic provide more information:

- <u>Using Constructor Injection</u>
- Using Property (Setter) Injection
- <u>Using Method Call Injection</u>

✓ Note:

Other topics in this section show how you can resolve instances of Enterprise Library objects on demand, and how you can create instances of Enterprise Library objects directly. For more information, see <u>Resolving Instances of Types Using Unity</u> and <u>Creating Application Block Objects Directly</u>.

Using constructor, method, and property injection allows you to inject instances of resolved types not only into the target type (the type you are actually resolving), but it also populates dependencies in all resolved types. The following schematic shows an overview of the process.



For more detailed information about the dependency injection features of Unity, see <u>Unity Dependency Injection and Interception</u>.

■ Using Constructor Injection

■ Using Property Injection

■ Using Method Call Injection

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Resolving Instances of Types Using Unity



This topic discusses the methods that Unity provides for resolving types and creating instances of types. It also describes how you can resolve instances of Enterprise Library objects and your own custom types. The topics are the following:

- The Unity Resolve Method
- Resolving Instances of Enterprise Library Types
- Resolving Instances of Your Own Custom Types

✓ Note:

Other topics in this section show how you can inject resolved instances into your own custom classes, and how you can create instances of Enterprise Library objects directly. For more information, see <u>Injecting Resolved Types into Other Classes</u> and <u>Creating Application Block Objects Directly</u>.

■ The Unity Resolve Method

■ Resolving Instances of Enterprise Library Types

■ Resolving Instances of Your Own Custom Types

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

Creating Application Block Objects Directly



There are times when the configuration information for your application scenario does not reside in a configuration source. For example, your system may dynamically create configuration information from data entered by a user. For these situations, you can directly construct application block objects, passing the required configuration information to the constructor.

✓ Note:

Other topics in this section show how you can inject resolved instances into your own custom classes, and how you can resolve instances of Enterprise Library objects on demand. For more information, see <u>Injecting Resolved</u> <u>Types into Other Classes</u> and <u>Resolving Instances of Types Using Unity</u>.

When you construct an application block object, you must construct a specific provider implementation type using the appropriate constructor parameters and pass the required arguments to the constructor. The following code shows how to construct the Data Access Application Block **SqlDatabase** object.

C#

String connString = @"server=(local)\SQLEXPRESS;databaseSqlDatabase db = new SqlDatabase(connString);

Visual Basic

```
Dim connString As [String] = "server=(local)\SQLEXPRES
"Integrated Security=true"
Dim db As New SqlDatabase(connString)
```

Configuration Information for New Objects

Some object constructors have an overload that accepts an instance of a configuration source that implements the **IConfigurationSource** interface. This allows you to supply configuration information directly to the new object.

Dependency Injection for Existing Objects

When you create instances of objects without using the Unity dependency injection mechanism, dependent objects are not automatically injected into your new object. However, you can force Unity to resolve and populate dependencies by using the **BuildUp** method of the container. Keep in mind that constructor injection does not take place when you use the **BuildUp** method because the object already exists and so the constructor does not execute.

The following example shows how you can use the **BuildUp** method to apply dependency injection to an existing object instance named **myDataService** that implements the interface **IMyService**. For more information about the **BuildUp** method, see <u>Using BuildUp</u> to <u>Wire Up Objects Not Created by the Container</u> in the Unity documentation.

C#

IMyService myDataService = new DataService(); IMyService builtupDataService = container.BuildUp<IMyS</pre>

Visual Basic

Dim myDataService As IMyService = New DataService()
Dim builtupDataService As IMyService = container.Builc

✓ Note:

Enterprise Library includes code to enable instrumentation. If you directly construct application block objects, instrumentation will not be enabled for those objects. However, in most cases, you can bind the appropriate instrumentation providers to the application providers. For more information about instrumentation listeners and instrumentation providers, see Enabling Instrumentation.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> <u>Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to

pagdoc@microsoft.com.

Copyright $\ensuremath{\text{@}}$ 2010 by Microsoft Corporation. All rights reserved.

Deploying Enterprise Library



The Enterprise Library Application Blocks are comprised of multiple assemblies. Each assembly that belongs to the Enterprise Library (excluding Unity, which is a generic utility) has a file name that begins with *Microsoft.Practices.EnterpriseLibrary*. Additionally, the application blocks may depend on the Enterprise Library common assemblies. An application that uses one or more of the application blocks may have dependencies on other application blocks. For example, some applications that use the Caching Application Block also require the Data Access Application Block assemblies. For information about the dependencies between the application blocks and the Enterprise Library Core, see Dependencies in Enterprise Library.

An application that uses the Enterprise Library application blocks can be deployed in one of two configurations:

- As private assemblies in the application folder hierarchy.
- As shared assemblies in any file system location or in the global assembly cache.

Specific deployment recommendations are included in the documentation for each application block. For more information, see the deployment topic for the individual application block. For general information about preparing and versioning Enterprise Library and using the global assembly cache, see Preparation and Versioning.

✓ Note:

If you decide to locate the Enterprise Library assemblies in the global assembly cache, there are some extra steps you must take if you use the Data Access Application Block and the Validation Application Block. These are described in the section "Using the Global Assembly Cache" in the topic Preparation and Versioning, and at http://entlib.codeplex.com/WorkItem/View.aspx?WorkItemId=26903.

Enterprise Library includes pre-compiled strong-named assemblies for all the source code. The assemblies are signed with a Microsoft strong-naming key that

is not included with the source code. This means that you cannot build a compiled version from the source code that uses the same public key. However, you can use your own key pair to create strong-named assemblies. If you believe that you may customize the Enterprise Library source code, you should use the binaries that you compile from the source code and sign with your own key instead of using the pre-compiled binaries signed with the Microsoft key.

For more information, see <u>Building Enterprise Library from the Source Code</u> and <u>Strong Naming the Enterprise Library Assemblies</u>.

If you update any of the application blocks, or if you want to install an updated version of an assembly, you can install the new version and have all applications use the updated assembly. Alternatively, you can install the new version in the global assembly cache and configure some applications to use the updated version, while others continue to use the earlier version. For more details, see Updating Application Block Assemblies.

If you intend to run your application in partial trust environments, in particular using a customized ASP.NET Medium Trust mode, see <u>Partial Trust</u> Environments.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Preparation and Versioning



When you compile the installed version of the Enterprise Library source code, the assemblies produced will not be strong named. As a result, they cannot be installed in the global assembly cache, nor will they have the other benefits associated with strong-named assemblies.

■ Using XCopy

■ Using the Global Assembly Cache

Versioning

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Building Enterprise Library from the Source Code



Although the Enterprise Library includes binaries for all the source code, you may want to customize the source code. This means you will need to build the Enterprise Library yourself. The following sections describe how to do this. After you have the binaries, you may want to strong name them. For more information about this topic, see Strong Naming the Enterprise Library Assemblies.

This topic contains the following subsections:

- Installing the Source Code
- Enterprise Library Visual Studio Solution Files
- Building the Enterprise Library Application Blocks and Tools
- Building the Enterprise Library Using Microsoft .Net Framework 4.0
- Additional Notes for Building and Using the Source Code

■ Installing the Source Code

■ Enterprise Library	Visual	Studio	Solution	Files

■ Building the Enterprise Library Application Blocks and Tools

■ Additional Notes for Building and Using the Source Code

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Strong Naming the Enterprise Library Assemblies



If you build Enterprise Library from the source code, you may decide to apply strong naming to the assemblies. A strong name consists of the assembly's identity—the simple text name, version number, and culture information (if provided)—plus a public key and a digital signature. The strong name is generated from an assembly file (the file that contains the assembly manifest, which in turn contains the names and hashes of all the files that make up the assembly), using the corresponding private key. Signing an assembly with a strong name ensures that its name is globally unique. Assemblies with the same strong name are expected to be identical.

For example, if you intend to share the Enterprise Library assemblies among several applications, you can install them into the global assembly cache. Each assembly in the GAC must have a globally unique name. You can use a strong name to ensure this. Even if you use the assemblies within only a single application, you can strong name them to ensure that your application uses their correct versions.

Strong names satisfy the following requirements:

- Strong names guarantee name uniqueness by relying on unique key pairs. No one can generate the same assembly name that you can because an assembly generated with one private key has a different name than an assembly generated with another private key.
- Strong names protect the version lineage of an assembly. A strong name can ensure that no one can produce a subsequent version of your assembly. Users can be sure that a version of the assembly they are loading comes from the same publisher that created the version originally provided with the application.
- Strong names provide a strong integrity check. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built. However, note that strong names alone do not imply a level of trust such as that provided by, for example, a digital signature and supporting certificate.

For information about deploying assemblies into the global assembly cache, see Working with Assemblies and the Global Assembly Cache.

■ Using Visual Studio to Strong Name Enterprise Library Assemblies

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Updating Application Block Assemblies



If an upgraded version of an Enterprise Library Application Block becomes available, you can install the new version and have all applications use the updated assembly. However, if the new version introduces compatibility problems for certain applications, you can install the new version in the global assembly cache and configure some applications to use the updated version, while others continue to use the earlier version.

■ Updating Private Assemblies

• Updating Shared Assemblies

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright @ 2010 by Microsoft Corporation. All rights reserved.

Partial Trust Environments



Applications that use versions of the Enterprise Library prior to version 3.0 require enough permissions so that the only security level they can use is full trust. With later versions of Enterprise Library, including this version, you can run applications under partial trust. A common example is an ASP.NET application that runs in a hosted environment. Typically, these types of applications require only enough permissions to run under medium trust. Depending on the Enterprise Library features that your application uses, you may need to grant additional permissions beyond those granted by a default partial-trust policy.

This topic contains the following sections:

- Overview of Partial Trust
- Enterprise Library and Partial Trust
- Enterprise Library Code Access Security and the SecurityTransparent Attribute

■ Overview of Partial Trust

■ Enterprise Library and Partial Trust

■ Enterprise Library Code Access Security and the SecurityTransparent Attribute

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Customizing the Medium Trust Policy



The tables in the following sections show the additional permissions—beyond those granted by default in the medium trust policy—that may be required by your application. You need to grant these additional permissions only if you want to use these specific features. Unless otherwise noted, make these modifications in the custom policy file.

For extended examples of how to modify a custom policy file, see <u>How To: Use Medium Trust in ASP.NET 2.0</u> on MSDN. If you are using a partial-trust policy other than medium trust, other restrictions and permissions may apply. For a table that lists the different permissions and the trust policies that allow them, see <u>ASP.NET Code Access Security</u> on MSDN.

These additional permissions are the following:

- General Permissions
- Caching Application Block Permissions
- Cryptography Application Block Permissions
- Data Access Application Block Permissions
- Exception Handling Application Block Permissions
- Logging Application Block Permissions
- Security Application Block Permissions
- Policy Injection Application Block Permissions
- Validation Application Block Permissions

The next sections describe these permissions.

■ General Permissions

■ Caching Application Block

■ Cryptography Application Block

■ Data Access Application Block

■ Exception Handling Application Block

■ Logging Application Block

■ Security Application Block

■ Policy Injection Application Block

■ Validation Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Limitations When Using Partial Trust



There may be some limitations regarding how you use partial trust with an Enterprise Library application block. These limitations include the following:

- Enterprise Library throws a Security Exception if it cannot obtain the mandatory permissions.
- Some calls to Logging Application Block trace listener classes fail.
- ASP.NET application directories require specific permissions.
- The AzMan provider is not available with partial trust.

■ Enterprise Library Throws Security Exception

Limitations on Log	ging Application	Block Trace	Listeners
--------------------------------------	------------------	--------------------	-----------

■ ASP.NET Application Directories Require Permissions

■ AzMan Provider Not Available

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Administering Enterprise Library



This topic describes some of the main scenarios for system administrators and operators who manage applications that use Enterprise Library. In general, the processes and techniques are no different from those involved in managing any .NET application. However, Enterprise Library does provide several useful features designed to make administration and management easier. This section covers the following topics:

- Running Multiple Versions of Enterprise Library
- Run Time Configuration
- Changes to the Run Time Environment
- Instrumentation within the Application Blocks
- Integration with Enterprise Management Tools
- Debugging Using the Source Code

■ Running Multiple Versions of Enterprise Library

■ Run Time Configuration

■ Changes to the Run Time Environment

■ Instrumentation within the A	Application	Blocks
--------------------------------	--------------------	---------------

■ Integration with Enterprise Management Tools

■ Debugging Using the Source Code

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending and Modifying Enterprise Library



Enterprise Library can serve as the basis for a custom library. You can take advantage of the extensibility points incorporated in each application block and extend the application block by supplying new providers. You can also modify the source code for the existing application blocks to incorporate new functionality. Use the guidelines in this topic when you extend the Enterprise Library.

There are three ways to extend the Enterprise Library. You can:

- **Write custom providers**. For detailed information about creating providers and extensions for each application block, see the "Extending and Modifying" topic in the sections of this documentation for each of the blocks. For general information about creating custom providers, and integrating with the configuration tools, see <u>Creating Custom Providers for Enterprise Library</u>.
- **Modify the source code of an application block**. For more information about the design of each block, see the topic "Design of the Application Block" in the sections of this guidance devoted to each of the application blocks.
- Write a new application block. You can use the existing blocks as guidance to create a completely new block. You will also find additional information and resources on creating and extending the blocks on the Enterprise Library community contributions site at http://www.codeplex.com/entlibcontrib/.

The following sections explain some of the factors to keep in mind when extending or modifying Enterprise Library.

■ Guidelines for Extending Enterprise Library

■ Guidelines for Modifying the Application Blocks

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Creating Custom Providers for Enterprise Library



Enterprise Library incorporates an easy-to-use extensible mechanism for you to add your own custom providers if you require specialized behavior. You can plug in custom providers that you create without needing to recompile Enterprise Library.

In addition, the configuration mechanism makes it easy to add a full design-time experience to your custom providers so that they can be configured within the configuration tools in exactly the same way as the built-in providers. The configuration tools are metadata driven, and read information about available providers and extensions from assemblies located in their run-time folder on startup. This means that you do not need to recompile the configuration tool to add a design-time experience for your custom providers.

The following topics describe the general process for creating a custom provider for Enterprise Library, and include an example of a simple provider:

- Enterprise Library Extension Points. This topic lists the extension points for Enterprise Library, and the classes you can use when creating a custom provider and specifying configuration information for it.
- Enterprise Library Configuration Integration. This topic describes the two ways that you can integrate custom providers into the Enterprise Library configuration system.
- <u>Creating a Custom Provider</u>. This topic provides the information required for creating a provider that you add to the application configuration using the **Add Custom** [provider type] menu commands in the configuration tools, and explains how you can create a fully integrated design-time experience for a custom provider.

You can download a Visual Studio project containing the example provider described in this section. The project includes a custom exception handler in both basic and full configuration integration modes, and a simple console application that uses both versions of the handler. To download the sample, go to http://www.codeplex.com/entlib/.

✓ Note:		

This section does not cover extending the Unity dependency injection and interception mechanism. For information about how you can extend Unity, see the documentation available on the Unity Web site at http://www.codeplex.com/unity/.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Enterprise Library Extension Points



The following tables list the extension points that Enterprise Library provides for each of the application blocks. For each type of provider or extension, the tables show:

- The interface you can implement and/or the base class you can inherit.
- The type of the configuration element to apply to your custom class to enable basic integration with the configuration system.
- The base class that you must inherit in your custom configuration element if you want to implement full design-time configuration integration. Your custom provider or extension will then use your custom configuration element class as its configuration element type, instead of the default shown in the Configuration Element (basic integration) column.

Providers and extensions that cannot be integrated with the configuration tools have no entry in the Configuration Element (basic integration) column.

Caching Application Block

Custom Provider or Extension	Interface or Base Class	Configuration Element (basic integration)	Config Class (integra
Backing Store	IBackingStore BaseBackingStore	CustomCacheStorageData	Caches
Cache Manager	ICacheManager	CustomCacheManagerData	Cachel
Expiration Policy	ICacheItemExpiration	n/a	n/a
Storage Encryption Provider	IStorageEncryptionProvider	n/a	Storage

Cryptography Application Block

Custom Provider or Extension	Interface or Base Class	Configuration Element (basic integration)
Hash Algorithm Provider	IHashProvider	CustomHashProviderData
Symmetric Encryption Algorithm Provider	ISymmetricCryptoProvider	CustomSymmetricCryptoProviderDat

Data Access Application Block

Custom Provider or Extension	Interface or Base Class	Configuration Element (basic integration)	Configuration Element Base Class (design time integration)
Database Provider	Database	n/a	n/a

Exception Handling Application Block

Custom Provider or Extension	Interface or Base Class	Configuration Element (basic integration)	Configuration Element Base Class (design time integration)
Exception Handler	IExceptionHandler	CustomHandlerData	ExceptionHandlerData
Exception Formatter	ExceptionFormatter	n/a	n/a

Logging Application Block

Custom Provider or Extension	Interface or Base Class	Configuration Element (basic integration)	Configuration Element Base Class (design time integration
Log Entry Formatter	ILogFormatter	CustomFormatterData	FormatterData
Trace Listener	CustomTraceListener	CustomTraceListenerData	TraceListenerD
Log Filter	ILogFilter LogFilter	CustomLogFilterData	LogFilterData

If you implement a custom Log Filter by implementing the **ILogFilter** interface or by extending the **LogFilter** base class, you must be aware of an issue that can prevent application code from resolving the configured name of the provider. However, this is only an issue when you wish to query the collection of filters when checking if a log entry will be specifically blocked by this filter. The **ILogFilter** interface defines a **Name** property that should return the name of the instance of the custom log filter from the configuration. However, there currently is no way to retrieve that name from within your custom log filter. Instead, you can pass a key/value pair in the **NameValueCollection** received by the constructor, and use this to set the **Name** property of the filter. When configuring your custom log filter, you will have to duplicate the name: once for the actual name of that instance of the custom log filter, and again in the named property collection that is passed to the constructor.

Policy Injection Application Block

Policy injection is a feature driven by the Unity interception mechanism. You can create custom behaviors, call handlers, call handler attributes, and matching rules for use with the Unity interception mechanism. For more information, see http://www.codeplex.com/unity/.

Security Application Block

Custom Provider or Extension	Interface or Base Class	Configuration Element (basic integration)
Authorization Provider	AuthorizationProvider	CustomAuthorizationProviderData
Security Cache Provider	ISecurityCacheProvider	CustomSecurityCacheProviderData

Validation Application Block

Custom Provider or Extension	Interface or Base Class	Configuration Element (basic integration)	Configuration Element Base Class (design time integration)
Validator	Validator <t> Validator</t>	CustomValidatorData	ValidatorData
Validator Attribute	ValueValidatorAttribute	n/a	n/a

For more information about creating custom providers and extensions for Enterprise Library, see <u>Enterprise Library Configuration Integration</u> and <u>Creating a Custom Provider</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Enterprise Library Configuration Integration



Enterprise Library allows you to easily integrate custom providers and extensions into the library in such a way that you can configure them using the Enterprise Library configuration tools. There are two types, or levels, of configuration integration.

■ Basic Configuration Integration

■ Full Configuration Integration

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Creating a Custom Provider



The following steps describe the process for creating a custom provider:

- Preparing Your Project
- Implementing the Interface or Extending the Base Class
- Specifying the Configuration Element Type
- Adding Full Design-time Integration
- Summary of Steps

■ Preparing Your Project

■ Implementing the Interface or Extending the Base Class

■ Specifying the Configuration Element Type

■ Adding Full Design-time Integration

■ Summary of Steps

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design of Enterprise Library



Enterprise Library incorporates a set of best practices in its overall design. Among these are the following:

- It uses common application block functionality (the Enterprise Library Core).
- It uses uniform conventions for naming and versioning.
- It incorporates instrumentation into all application blocks.
- It uses unit tests written during the design phase.

The section describes the design of Enterprise Library and includes the following topics:

- <u>Design Patterns</u>. This topic describes the use of design patterns within Enterprise Library.
- <u>The Enterprise Library Core</u>. This topic describes the Enterprise Library Core, including the configuration system.
- <u>Providers</u>. This topic describes the use of providers to implement extensibility.
- <u>Design Time Configuration</u>. This topic describes the design-time configuration features of Enterprise Library.
- <u>The Dependency Injection Model</u>. This topic describes the dependency injection mechanism used by Enterprise Library to instantiate objects and manage their lifetimes.
- <u>Instrumentation</u>. This topic describes the implementation of instrumentation within Enterprise Library.
- <u>Group Policy Support</u>. This topic describes Group Policy support using the Manageable Configuration Source.
- <u>Unit Tests</u>. This topic describes the use of unit tests within Enterprise Library.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design Patterns



In software architecture and development, a pattern is a description of a recurring problem that occurs in a specific context and, based on a set of guiding forces, suggests a solution. The solution is usually a simple mechanism because it is a collaboration between two or more classes, objects, services, processes, threads, components, or nodes that work together to solve the underlying architecture or development challenge.

Patterns are useful to developers and architects because they do the following:

- They document simple mechanisms that work.
- They provide a common vocabulary and taxonomy for developers and architects.
- They allow solutions to be described concisely as combinations of patterns.
- They enable reuse of architecture, design, and implementation decisions.

The Enterprise Library application blocks use the following patterns (among others):

- **Plug-in pattern**. This pattern extends the behavior of a class by allowing extensions to plug into an abstract class that, in turn, plugs into a core class. This creates a new subclass that includes only the capabilities required in the specific context.
- **Dependency Injection pattern**. With this pattern, you can inject objects into a class, instead of relying on the class to create the object.

For more information about patterns, see the <u>Microsoft patterns & practices</u> Web site.

■ Plug-in Pattern

■ Dependency Injection Pattern

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

The Enterprise Library Core



Many tasks that the Enterprise Library application blocks perform are common across more than one application block and are also useful in application code outside of Enterprise Library. Examples are routines that serialize data or access configuration information. To promote usability, these routines reside in a common assembly named the Enterprise Library Core.

In addition, all the application blocks are designed to have a limited number of dependencies so that they can be used individually as well as with other application blocks. All application blocks except Unity depend on the Enterprise Library Core, which is a logical grouping made up of the following subsystems:

- The Common assembly
- Instrumentation for the application blocks
- Configuration helper classes and design-time configuration components

For information about the dependencies between application blocks and the Enterprise Library Core, see <u>Dependencies in Enterprise Library</u>.

■ The Common Assembly

■ Instrumentation

■ Configuration Helper Classes and Design-time Components

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Providers**



A provider in the Microsoft.NET Framework is an intermediary piece of code; with it, your application can connect to a service or data source and then retrieve or modify an object or data from that service or source. The Enterprise Library includes many providers. In addition, you can create your own provider to supply information that you need for your specific application.

A provider type defines an interface that is associated with a capability an application block must have to perform correctly. A provider is a specific implementation of a provider type. Each application block includes one or more providers for each provider type. You can also write custom providers for application blocks. Separating the application block's functionality from specific implementations of its capabilities achieves the following goals:

- **Variability**. This allows you to choose from multiple implementations of the same capability, according to the requirements of a specific application.
- **Extensibility**. This allows you to use the application block in environments where the capability in question has a mandatory implementation. For example, an application can require a specific encryption algorithm when deployed to a particular environment.
- **Encapsulation**. This allows you to react to changes in the environment in which the application block is used. With providers, functionality that is not a part of the application block's core functionality can be replaced or upgraded without affecting other areas of the application block.
- **Portability across environments**. This allows you to deploy the application block in a new environment with providers specific to that environment. You can also create providers that run in one environment and simulate behavior from a different environment.
- Minimized coupling between application blocks. Application blocks that are dependent on other application blocks can encapsulate this dependency in a provider. This means that the application block is less vulnerable to revisions in the application block on which it depends. For example, the Exception Handling Application Block includes the logging exception handler. This handler is dependent on the Logging Application Block and is included as a provider. A new version of the Logging

Application Block would require only a new logging handler provider; the rest of the Exception Handling Application Block can remain unchanged.

For information about creating your own providers for the application blocks, see the "Extending and Modifying" section of the documentation for each application block.

To provide feedback, get assistance, or download additional content, please visit the Enterprise Library Community Web site.

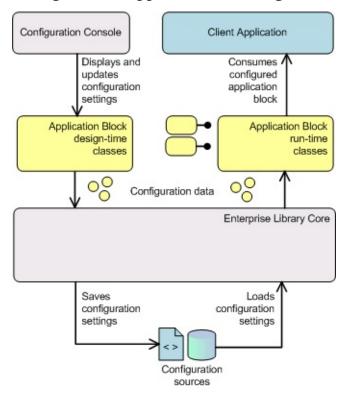
To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Design Time Configuration**

All application blocks include both run-time support and design-time support for configuration settings. The run-time support includes classes that represent the configuration settings. The Configuration Application Block uses these class definitions when it loads configuration settings. It reads the configuration settings from storage and returns objects that contain the configuration data to the application block.

The design-time configuration support includes classes that allow you to change the configuration settings by using the Enterprise Library configuration tools. These classes define a visual representation of the different configuration settings, specify the actions that can be performed based on the current configuration state, and provide the ability to validate the configuration settings.

The following figure illustrates the relationship between the run-time configuration support and the design-time configuration support.



The design-time classes depend on the configuration run-time classes because

they obtain the current configuration settings from the configuration run-time objects. When you change these settings and save the changes, the design-time objects update the run-time objects, which are then saved in storage. However, the run-time classes have no dependency on the design-time classes. There is a single lightweight design-time assembly that contains design-time core and some block-specific classes. This is separate from the assemblies containing the run-time implementations. The design-time assembly is not required for running an application that uses the application blocks. However, it is required when you use the configuration console to change the configuration of an application block.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

The Dependency Injection Model



This topic describes the mechanism used by Enterprise Library to create and manage the lifetime of objects within the core and the application blocks. Enterprise Library uses a dependency injection mechanism to create and manage the lifetime of all the Enterprise Library objects it creates. Depending on the style of your application, it may use the same dependency injection container that Enterprise Library uses. This topic contains the following sections:

- Application and Container Initialization
- Using an Alternative Dependency Injection Container
- More Information

To learn how to use the Unity dependency injection approach when writing application code, see <u>Using Enterprise Library in Applications</u>.

Application and Container Initialization

■ Using an Alternative Dependency Injection Container

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Instrumentation
```

In Enterprise Library, the interfaces and classes responsible for logging events to the Windows Event Log or updating Windows Performance Counters are separated from those that indicate an instrumentation-worthy activity has occurred.

Each application block contains one or more classes responsible for translating activities from within the block into Event Log messages or Performance Counter updates. These classes are generally known as instrumentation providers because they provide instrumentation services within that block. Classes within an application block use the instrumentation providers to indicate that an activity has occurred.

Each block is responsible for a different set of activities and so the exact interface of the provider varies for each block. For example, the instrumentation provider interface for the **CachingInstrumentationProvider** class is as follows:

```
public interface ICachingInstrumentationProvider
{
  void FireCacheUpdated(long updatedEntriesCount, long
  void FireCacheAccessed(string key, bool hit);
  void FireCacheExpired(long itemsExpired);
  void FireCacheScavenged(long itemsScavenged);
  void FireCacheCallbackFailed(string key, Exception expoid FireCacheFailed(string errorMessage, Exception)
}
```

The Cache class in the Caching Block uses an

ICachingInstrumentationProvider each time it must indicate the cache was accessed. When the cache is accessed it calls the **FireCacheAccessed** method of the instrumentation provider.

The instrumentation providers are typically connected to other classes within

the block through a constructor parameter. The **Cache** class, for example, takes an **ICachingInstrumentationProvider** as shown here.

Because the implementations for the instrumentation provider interfaces are registered within the dependency injection container, they are injected into the classes that require them.

■ How Instrumentation Providers Work

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Group Policy Support



This reference supplements the information in <u>Using Group Policy with</u> <u>Enterprise Library</u>. It is divided into the following sections:

- The Manageable Configuration Source Class. This section describes the design of the class that provides Group Policy configuration support in Enterprise Library.
- Integration of Group Policy with Enterprise Library Applications. This section provides information about how instances of the **ManageableConfigurationSource** are integrated with Group Policy and includes information about Group Policy template structure.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

The Manageable Configuration Source Class



This topic describes the classes that make up the **ManageableConfigurationSource** class.

■ ManageableConfigurationSource

■ ManageabilityHelper

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Integration of Group Policy with Enterprise Library Applications



The **ManageableConfigurationSource** class reads configuration overrides from the registry and applies those overrides to an application configuration file. By reading the registry in this way, this configuration source allows you to use Group Policy to specify the settings. A Group Policy template defines the configuration options for each application that is configured in Group Policy and defines the default options in Group Policy for each configuration option. One Group Policy template is required for each application because a separate registry key represents each application. The key is based on the application name.

To simplify the creation of Group Policy templates, you can use the **AdministrativeTemplateGenerator** class to generate an **AdmContent** object, and then write this content to a stream in order to generate ADM files from the Enterprise Library configuration tools. These templates use the settings defined in the application configuration file to determine the default settings for each option. They also determine the structure of the ADM template, which must match the contents of the configuration file.

In this version of Enterprise Library, templates and Group Policy support are available for all the application blocks, except for the Validation Application Block, the Policy Injection Application Block, and Unity. Group Policy is a fairly static and flat mechanism, and the configuration for these application blocks is quite dynamic and usually complex. Therefore, it would be almost impossible to provide a useable user interface implementation for these application blocks.

✓ Note:

Applications that use Group Policy to specify their settings must define and use a **ManageableConfigurationSource** in the configuration sources section, and set the **EnableGroupPolicy** flag on that source to **true**. You can use the configuration tools to configure this setting.

This section contains the following topics:

- Group Policy Template Structure
- General Settings for the Application Blocks
- Exceptions to the General Template Structure
- Applying Group Policy Settings
- Limitations of Group Policy Support

■ Group Policy Template Structure

Genera	l Settings f	or the	Application	Blocks
--------------------------	--------------	--------	--------------------	--------

• Exceptions to the C	General Temp	late Structure
-----------------------	--------------	----------------

■ Applying Group Policy Settings

■ Limitations of Group Policy Support

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Unit Tests



Enterprise Library includes Visual Studio® solution files that include Visual Studio Team System unit tests with the application block projects. You can use the main Enterprise Library solution file (EnterpriseLibrary.VSTS.sln) to build the entire Enterprise Library with the unit tests. This solution contains the entire set of application block projects and the Enterprise Library core projects, and includes all unit tests for execution with the Visual Studio Team System.

The Enterprise Library solution files include two build configurations: Release and Debug. All projects within a solution file are compiled in both configurations, including the unit test projects. For more information, see Building Enterprise Library from the Source Code.

✓ Note:

This release of Enterprise Library does not include solutions that use the NUnit test framework.

■ Software Requirements

Organization

■ Using the Unit Tests

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Caching Application Block



The Enterprise Library Caching Application Block lets developers incorporate a local cache in their applications. It supports both an in-memory cache and, optionally, a backing store that can either be the database store or isolated storage. The Caching Application Block can be used without modification; it provides all the functionality needed to retrieve, add, and remove cached data. Configurable expiration and scavenging policies are also part of the block.

✓ Note:

Caching Application Block functionality is built into .NET Framework 4.0; therefore the Enterprise Library Caching Application Block will be deprecated in releases after 5.0. You should consider using the .NET 4.0 System.Runtime.Caching classes instead of the Caching Application Block in future development.

The Enterprise Library Caching Application Block includes the following features:

- You can use the graphical Enterprise Library configuration tools to manage configuration settings.
- You can configure a persistent storage location, using either isolated storage or the Enterprise Library Data Access Application Block, whose state is synchronized with the in-memory cache.
- Administrators can manage the configuration using Group Policy tools.
- You can extend the block by creating custom expiration policies and storage locations.
- You are assured that the block performs in a thread-safe manner.

This section includes the following topics that will help you to understand and use the Caching Application Block:

• What Does the Caching Application Block Do? This topic provides a brief overview that will help you to understand what the block can do, and explains some of the concepts and features it incorporates. It also provides a simple example of the way that you can write code to use the

block.

- When Should I Use the Caching Application Block? This topic will help you to decide if the block is suitable for your requirements. It explains the benefits of using the block, and any alternative techniques you may consider. It also provides details of any limitations of the block that may affect your decision to use it.
- <u>Developing Applications Using the Caching Application Block</u>. This topic first explains how to configure the Caching Application Block and add it to your application. It then explains how to select a backing store.
- <u>Key Scenarios</u>. This section demonstrates how to use the block to perform typical caching operations.
- Design of the Caching Application Block. This topic explains the decisions that went into designing the Caching Application Block and the rationale behind those decisions.
- Extending and Modifying the Caching Application Block. This topic explains how to extend the block by adding your own backing store and your own expiration policies. It also explains how to modify it by changing the source code.
- <u>Deployment and Operations</u>. This topic explains how to deploy and update the Caching Application Block assemblies.

More Information

For related information, see the following patterns & practices guides and documents:

- Microsoft Application Architecture Guide, 2nd Edition
- Caching Architecture Guide for .NET Framework Applications
- Enterprise Library home page on MSDN®
- Enterprise Library on the CodePlex Web site

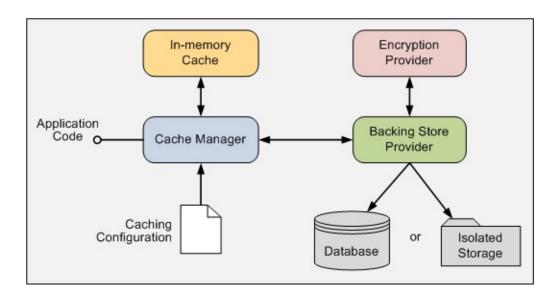
For links to external caching providers, see the "More Information" section in <u>The Caching Application Block</u> on MSDN.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.



The Caching Application Block provides an in-memory cache that your application can manipulate through a simple API to store and retrieve items, and to obtain information about the stored items. In addition, you can configure a persistent or custom backing store for your cache, and—if required—encrypt the information stored there. You can configure more than one cache for your application, and specify a partition for each one so that data from multiple caches can be stored in separate containers within the same backing store location, such as a database. The following schematic shows the main elements of the Caching Application Block.



At application startup, the block loads the in-memory cache from the backing store (if configured). Alternatively, you can also load the cache yourself if you want to implement a delayed loading pattern. As the application runs, the block checks the expiration of cached items and removes them from the cache. Expiry can be configured based on sliding or absolute time values; or through dependencies on files, other cached items, or external resources. The block also manages the cache in conjunction with memory availability, based on cached item priorities.

✓ Note:

For information about the types of expiration you can use, and the default expiration for new items, see <u>Design of the Expiration Process</u>.

As application code interacts with the Cache Manager, it updates the in-memory cache and—if a backing store is configured—updates the backing store. You can configure an encryption provider for the backing store, which is implemented by the Cryptography Application Block, to encrypt the items that are cached in the backing store (note that the block does not encrypt items stored in the in-memory cache).

The block includes providers that store data in a database or in Isolated Storage on the local machine. It does not provide a distributed caching mechanism. Additional providers, including providers that support distributed caching, may be available from third parties and the Enterprise Library community Web site. For more information, see the CodePlex Community and Enterprise Library Contributions Web sites.

■ Example Application Code

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

When Should I Use the Caching Application Block?



The Caching Application Block is suitable if you encounter any of the following situations:

- You must repeatedly access static data or data that rarely changes.
- Data access is expensive in terms of creation, access, or transportation.
- Data must always be available, even when the source, such as a server, is not available.

You can use the Caching Application Block with any of the following application types:

- Windows Forms
- Windows Presentation Foundation (WPF)
- Windows Communication Foundation (WCF)
- Console application
- Windows service
- ASP.NET Web application or Web service if you need features not included in the ASP.NET cache

■ Scenarios for the Caching Application Block

■ Benefits of the Caching Application Block

■ Limitations of the Caching Application Block	
------------------------------------------------	--

Alternatives	to	Using	the	Caching	Application	Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Developing Applications Using the Caching Application Block



This section describes how to use the Caching Application Block to develop applications. It explains how to enter configuration information for the block, incorporate it into your solution, and select a backing store. This section includes the following topics:

- Entering Configuration Information
- Adding Application Code
- Selecting a Backing Store

All blocks ship as binary assemblies and as source code. If you want to use the source code, you must compile it. To learn how to compile the Enterprise Library source code, see <u>Building Enterprise Library from the Source Code</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Entering Configuration Information



These procedures explain how to configure the Caching Application Block. If you add a Database Cache Storage provider to the configuration of the Caching Application Block, the configuration tool automatically adds the Data Access Application Block. You must configure that block before you configure the Database Cache Storage provider in the Caching Application Block configuration.

For details of the schema for the Caching Application Block configuration, see <u>Source Schema for the Caching Application Block</u>. You can also configure the block in code by using an alternate configuration source. For more information, see <u>Advanced Configuration Scenarios</u> and <u>Using the Fluent Configuration API</u>.

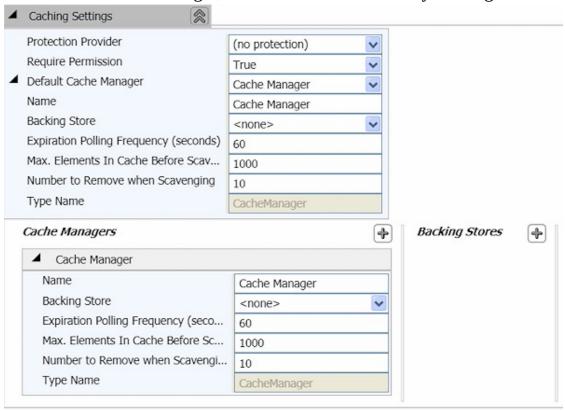
To add the Caching Application Block

- 1. Open the configuration file. For more information, see <u>Configuring Enterprise Library</u>.
- 2. Open the **Blocks** menu and then click **Add Caching Settings**.
- 3. The configuration tool automatically adds the **Caching Settings** section with default values, and a default **Cache Manager** item.

To configure cache managers

- 1. Click the properties expander arrow in the **Caching Settings** section to open the list of properties.
- 2. (Optional) Change the **Default Cache Manager** property name. The default cache manager is used if the code does not specify a cache manager. Either enter a new name or select one from the drop-down list. The default name is **CacheManager**.
- 3. (Optional) If you want to encrypt the configuration, make a selection from the Protection Provider drop-down list. You can select the RsaProtectedConfigurationProvider or the DataProtectedConfigurationProvider. See Encrypting Configuration Data for information about the restrictions on using the RsaProtectedConfigurationProvider.
- 4. (Optional) If you want to run your application in partial trust mode, change the **Require Permission** property to **False**. The default is **True**.
- 5. To access the default **Cache Manager** properties, click the section

expander arrow to the left of the default **Cache Manager** title. If you renamed the cache manager, the title will be the name you assigned it.



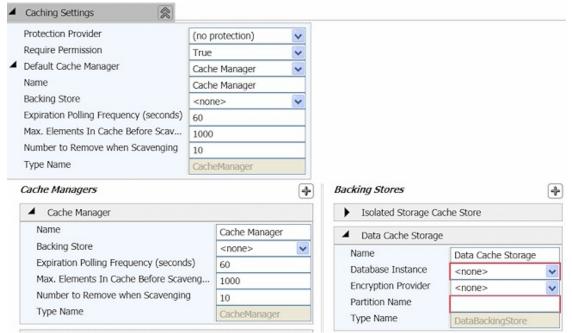
- 6. (Optional) Rename the **CacheManager** node. The default name is **CacheManager**.
- 7. (Optional) Set the **Backing Store** property. The default is <**none**> which means that the cache manager only stores data in memory. The drop-down list shows you the available backing stores you can choose from. You can add backing stores in the **Backing Stores** pane.
- 8. (Optional) Set the **Number to Remove when Scavenging** property. This is the number of elements to remove after scavenging begins. The default setting is **10** elements.
- 9. (Optional) Set the **Max. Elements in Cache Before Scavenging** property. This is the maximum number of elements that can be in the cache before scavenging. The default setting is **1000** elements.
- 10. (Optional) Set the **Expiration Polling Frequency** property. This is the frequency of the timer that regulates how often the background scheduler checks for expired items. The unit is seconds, and the default setting is **60**.

By default, the cache stores items only in memory and assigns the value of the

backing store to **NullBackingStore**. You can add caching stores and then configure the Caching Application Block to use any of the stores you have added. You can configure the Caching Application Block to use database cache storage, isolated storage, or custom cache storage. Database cache storage uses the Data Access Application Block.

To add database cache storage and configure the Caching Application Block to use it

- 1. Click the plus sign icon in the **Backing Stores** pane, point to **Add Backing Stores** and click **Add DataCacheStorage**.
- 2. The configuration tool automatically adds the **Database Settings** section. For information about configuring this section, see <u>The Data Access Application Block</u> documentation.
- 3. Click the properties expander arrow in the new **DataCacheStorage** section to open the list of properties.



- 4. (Optional) Set the **Name** property by renaming the **DataCacheStorage** node.
- 5. Set the **Database Instance** property by making a selection from the text box drop-down list. This is the name of the database connection string. It must correspond to the name of a connection string in the **Database Settings** section.
- 6. If you want to encrypt the information stored in the database, you must have configured an encryption provider. To add a new encryption

- provider, click the plus sign icon in the **Encryption Providers** pane of the **Caching Settings** section, point to **Add Encryption Providers** and then click **Add Symmetric Crypto Provider** (this is the only encryption provider offered). The configuration tool automatically adds the **Cryptography Settings** section.
- 7. In the **Cryptography Settings** section (not the **Caching Settings** section), add a symmetric cryptography provider to the configuration. For information about configuring this block, see <u>The Cryptography</u> Application Block documentation.
- 8. In the **Caching Settings** section, select the new symmetric encryption provider you configured in the **Cryptography Settings** section or an existing encryption provider in that section, in the drop-down list for the **Symmetric Crypto Provider** property.
- 9. In the **Backing Stores** pane, in the **Data Cache Storage** section, set the **Encryption Provider** property by selecting the **Symmetric Crypto Provider** item you just configured.
- 10. In the **Cache Managers** section, select the new backing store you added in the drop-down list for the **Backing Store** property of the cache manager that will use this backing store.

To add isolated storage and configure the Caching Application Block to use it

- 1. Click the plus sign icon in the **Backing Stores** pane, point to **Add Backing Stores** and click **Add Isolated Storage Cache Store**.
- 2. (Optional) In the new **Isolated StorageCacheStore** section, set the **Name** property.
- 3. Set the **Partition Name** property. This identifies the portion of isolated storage that the cache manager will use.
- 4. If you want to encrypt the information stored in isolated storage, you must have configured an encryption provider. To add a new encryption provider, click the plus sign icon in the Encryption Providers pane of the Caching Settings section, point to Add Encryption Providers and then click Add Symmetric Crypto Provider (this is the only encryption provider offered). The configuration tool automatically adds the Cryptography Settings section.
- 5. In the **Cryptography Settings** section (not the **Caching Settings** section), add a symmetric cryptography provider to the configuration.

- For information about configuring this block, see <u>The Cryptography</u> <u>Application Block</u> documentation.
- 6. In the **Caching Settings** section, select the new symmetric encryption provider you configured in the **Cryptography Settings** section or an existing encryption provider in that section, in the drop-down list for the **Symmetric Crypto Provider** property.
- 7. In the **Backing Stores** pane, in the **Isolated Storage Cache Store** section, set the **Encryption Provider** property by selecting the **Symmetric Crypto Provider** item you just configured.
- 8. In the **Cache Managers** section, select the new backing store you added in the drop-down list for the **Backing Store** property of the cache manager that will use this backing store.

To add a custom cache storage provider

- 1. Click the plus sign icon in the **Backing Stores** pane, point to **Add Backing Stores** and click **Add Custom Cache Storage**.
- 2. The **Type Selector** dialog is displayed. Navigate to the assembly containing your custom backing store and click on it. The store will be added and displayed in the **Backing Stores** pane.
- 3. (Optional) In the new custom store section, set the **Name** property.
- 4. If the custom backing store requires any other configuration values to be provided, add these as key/value pairs to the configuration by typing them into the **Key** and **Value** text boxes. As you enter a value, the configuration tool displays a new row in this section. Click the cross button to remove a name/value pair.

If you want to add another cache manager to your application configuration, click the plus sign icon in the Cache Managers pane, point to **Add Cache Managers** and then click on the manager you wish to add. Repeat the preceding procedures. There can be only one default cache manager. Each instance of the cache manager must have a unique name.

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Source Schema for the Caching Application Block
```

This topic lists the XML elements and attributes used to configure the Caching Application Block. You can manually edit the XML data, but the Enterprise Library configuration tools greatly simplify this task. If you choose to manually edit the XML, use the schema information contained in this topic.

The configuration file has the following section handler declaration.

The section handler declaration contains the name of the configuration settings section and the name of the section handler class that processes configuration data in that section. The name of the configuration settings section is **cachingConfiguration**. The name of the section handler class is **Microsoft.Practices.EnterpriseLibrary.Caching.Configuration.CacheManag**

■ cachingConfiguration Element

• encryptionProviders Element

■ backingStores Element

■ cacheManagers Element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Adding Application Code



The Caching Application Block is designed to support the most common situations for storing data in a cache. When adding your application code, refer to the scenarios in the Key Scenarios sections and select the ones that best suit your situation. Use the code that accompanies the scenario either as it is, or revise it as required.

To prepare your application

- 1. Add a reference to the Caching Application Block assembly. In Microsoft Visual Studio®, right-click your project node in Solution Explorer, and then click **Add Reference**. Click the **Browse** tab and find the location of the
 - **Microsoft.Practices.EnterpriseLibrary.Caching.dll** assembly. Select the assembly, and then click **OK** to add the reference.
- 2. Follow the same procedure to set a reference to the following assemblies:
 - Microsoft.Practices.EnterpriseLibrary.Common.dll
 - Microsoft.Practices.ServiceLocation.dll
 - Microsoft.Practices.Unity.dll
 - Microsoft.Practices.Unity.Interception.dll
- 3. If you are using the database backing store, add a reference to **Microsoft.Practices.EnterpriseLibrary.Caching.Database.dll** and **Microsoft.Practices.EnterpriseLibrary.Data.dll**.
- 4. If you are using the Cryptography Application Block to encrypt data in the cache, add references to
 - Microsoft.Practices.EnterpriseLibrary.Security.Cryptography.dll and
 - Microsoft.Practices.EnterpriseLibrary.Caching.Cryptography.dll.
- 5. (Optional) To use elements from the Caching Application Block without fully qualifying the element reference, add the following **using** statements (C#) or **Imports** statements (Microsoft Visual Basic®) to the top of your source code file.

C#	

using Microsoft.Practices.EnterpriseLibrary.Cach using Microsoft.Practices.EnterpriseLibrary.Cach

Visual Basic

Imports Microsoft.Practices.EnterpriseLibrary.Ca Imports Microsoft.Practices.EnterpriseLibrary.Ca

✓ Note:

For Visual Basic projects, you can also use the References page of the Project Designer to manage references and imported namespaces. To access the References page, select a project node in Solution Explorer, and then click [project name] Properties on the Project menu. When the Project Designer appears, click the References tab.

Next, add the application code. Generally, there are two steps to create code that uses the Caching Application Block:

- 1. Resolve a **CacheManager** instance.
- 2. Call the appropriate methods.

Each key scenario demonstrates how to incorporate these steps into an application.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Selecting a Backing Store



Each cache manager can be configured to store data only in memory, which means that it uses the <u>null backing store</u>, or each cache manager can be configured to store data both in memory and in persistent storage. The type of persistent storage is specified when you configure the backing store. Backing stores let cached data survive if the application must be restarted. In its original state, the Caching Application Block supports two types of persistent backing stores, each of which is suited to particular situations:

- Isolated storage see <u>Using the Isolated Storage Backing Store</u>.
- Database cache storage see <u>Using the Data Access Application Block</u> <u>Backing Store</u>.

If you intend to perform caching in a multiple-server environment, such as a Web farm, see <u>Considerations for Server Scenarios</u>.

To protect external data stores from unauthorized access, consider this list of <u>Usage Notes</u>.

Developers can extend the Caching Application Block to support additional types of backing stores. For more information about this topic, see Extending and Modifying the Caching Application Block.

✓ Note:

An application can use more than one cache; each cache will be represented by a cache manager in the application's configuration. The Caching Application Block does not support the use of the same persistent backing store location by multiple cache managers in an application. However, multiple cache managers in an application can have the same partition name.

■ Usage Notes

■ Using the Null Backing Store

■ Using the Isolated Storage Backing Store

■ Using the Data Access Application Block Backing Store

+	Consid	lerations	for	Server	Scenarios	

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Key Scenarios



This section describes the most common situations developers must address when storing data in a cache. Each scenario explains the task, gives a real-world situation for the task, and includes code demonstrating how to use the Caching Application Block to complete the task.

- Adding Items to the Cache. This topic describes the basic caching operations such as how to add an item to the cache using the **Add** method, setting the item's expiration policy (for the expiration process) and its priority (for the scavenging process).
- Removing Items from the Cache. This topic describes how to remove an item from the cache using the **Remove** method.
- Retrieving Items from the Cache. This topic describes how to obtain an item from the cache using the **GetData** method.
- <u>Flushing the Cache</u>. This topic describes how to flush the cache, which empties it, using the **Flush** method.
- <u>Loading the Cache</u>. This topic demonstrates proactive and reactive loading.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Adding Items to the Cache**



Caches store items that are either expensive to create or expensive to transport. For example, in a retail application, a list of products must be passed from the data access components to the user interface components so that the product list can be displayed to the users. The data represents real-world business entities, such as products or orders. To increase performance, some of these items may be added to the cache.

■ Typical Goals

■ Solution

■ Using the Add Method

■ Refreshing Removed Items

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Removing Items from the Cache



The scavenging and expiration processes automatically remove items from the cache according to the priorities and expiration policies of the items. You can also remove specific items from the cache. For example, in a retail application, some data may no longer be applicable, depending on selections the customer makes.

■ Typical Goals

■ Solution

■ Using the Remove Method

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Retrieving Items from the Cache



Data stored in the cache must be retrieved so that it can be displayed or processed. For example, in a retail application, you may want to display a list of products from a catalog.

■ Typical Goals

■ Solution

■ Using the GetData Method

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Flushing the Cache



Flushing lets you manage cached items to make sure that storage, memory, and other resources are used efficiently. Flushing removes all items in the cache, including those that have not yet expired. For example, in a retail application, cached data may no longer be valid because of selections made by the customer or because the customer has logged off.

■ Typical Goals

■ Solution

■ Using the Flush Method

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Loading the Cache



Before you can use cached data, you must first load data into the cache. For example, in a retail application, you may want to load data about various products, or all products, into the cache.

■ Typical Goals

■ Solution

■ Caching Data Proactively

■ Caching Data Reactively

■ Loading Examples

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design of the Caching Application Block



The Caching Application Block is designed specifically so that:

- It provides a set of APIs that are manageable in size.
- It enables developers to incorporate the standard caching operations into their applications without having to learn the internal workings of the block.
- It uses the Enterprise Library configuration tools for easy configuration.
- It performs efficiently.
- It is thread safe. Code is considered to be thread safe when it can be called from multiple programming threads without unwanted interaction among those threads.
- It ensures that the backing store remains intact if an exception occurs while it is being accessed.
- It ensures that the states of the in-memory cache and the backing store remain synchronized.

This topic describes the design of the caching system, describing the <u>highlights</u> and specific <u>design details</u>. Other topics in this section include <u>Design of the Expiration Process</u> and <u>Design of the Scavenging Process</u>.

■ Design Highlights

■ Design Details

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design of the Expiration Process



The Caching Application Block's expiration process is performed by the **BackgroundScheduler**. It periodically examines the cached items in the hash table to see if any items have expired. You control how frequently the expiration cycle occurs when you configure an instance of the **ICacheManager** interface default implementation **CacheManager** by using the configuration tools.

The expiration policies provided with the Caching Application Block are these:

- **Absolute**. This means the item expires at a specific time.
- **Sliding**. This means the item expires after the specified time has elapsed from when the item was last accessed. The default time is 2 minutes.
- **Extended format**. This allows you to specify very detailed expiration conditions. For example, you can specify that an item expires every Saturday night at 10:03 PM or on the third Tuesday of each month. Extended formats are listed in the ExtendedFormat.cs file.
- **File dependency**. This means the item expires when a specific file is modified.
- **Never expired.** This means the item will never expire, although it may still be removed if the block detects a lack of available memory.

The first three policies, absolute, sliding, and extended format, are referred to as time-based expirations. You should use time-based expiration for volatile cache items, such as those that have regular data refreshes or those that are valid for only a specified time. Time-based expiration lets you set policies that keep items in the cache only as long as their data remains current. For example, if you are writing an application that tracks currency exchange rates by obtaining the data from a frequently updated Web site, you can cache the currency rates for the time that those rates remain constant on the source Web site. In this situation, you would set an expiration policy that is based on the frequency of the Web site updates.

The fourth policy, file dependency, is referred to as a notification-based expiration. It defines the validity of a cached item based on a particular file. If the file is modified, the cached item is invalidated and removed from the cache.

The **Add** method has two overloads. One overload assumes the default

expiration policy, which is **NeverExpired**. The other overload lets you set the expiration policies yourself. You can use as many policies as you want, including policies that you create yourself. (For more information about extending the Caching Application Block by adding your own expiration policies, see Extending and Modifying the Caching Application Block.) If you have an item with multiple policies, the item will expire if any one of the policy's criteria is met.

■ Marking and Sweeping

Callbacks

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design of the Scavenging Process



The Caching Application Block's scavenging process is performed by the **BackgroundScheduler** object. It checks the cache every time an item is added to see if the number of items in the cache has reached a predetermined limit. You use the configuration tools to set this limit when you configure an instance of a cache manager. You also set how many items are removed from the cache after scavenging begins.

When an item is added to the cache, the code can set one of four priority settings: **Low**, **Normal**, **High**, or **NotRemovable**. The **BackgroundScheduler** object determines which items should be scavenged by doing a major sort based on priority and a minor sort based on the last time the item was accessed. For example, an item with a **Low** priority that has just been used will be scavenged before something with a **High** priority that has not been accessed for three years. The default value is **Normal**.

The **NotRemovable** priority is used when you want an item to remain in the cache until it expires. However, the cache should not be used as the only location where an item of data exists. A cache should be used to improve performance; it should not be used as a form of permanent storage.

Unlike the expiration process, the scavenging process performs marking and sweeping in a single pass. For more information about marking and sweeping, see <u>Design of the Expiration Process</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending and Modifying the Caching Application Block



In its original state, the Caching Application Block works well for typical caching situations. However, there may be times when you have to customize some of the block's behavior to better suit your application's particular requirements. There are two ways to do this. You can extend the Caching Application Block using the built-in extension points. You can also modify the block by making changes to its source code. For more details, see the following topics:

- Extending the Caching Application Block
- Extending and Modifying Enterprise Library

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

Extending the Caching Application Block



You extend the block through designated extension points. Typically, these are custom classes, written by you, that implement a particular interface or derive from an abstract class. Because these custom classes exist in your application space, you do not have to modify or rebuild the block. Instead, you designate your extensions using configuration settings.

You can extend the block by adding a new type of backing store and storage encryption provider, by adding new expiration policies, or by replacing the default Cache Manager. The following table lists the interfaces and base classes that you can use to extend the block.

Custom Provider or Extension	Interface or Base Class
Backing Store	IBackingStore or BaseBackingStore
Cache Manager	ICacheManager
Expiration Policy	ICacheItemExpiration
Storage Encryption Provider	IStorageEncryptionProvider

For detailed information about how to integrate custom providers with the Enterprise Library configuration system and configuration tools see <u>Creating Custom Providers for Enterprise Library</u>.

■ Adding a New Backing Store

■ Adding a New Expiration Policy

■ Replacing the Default Cache Manager

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Deployment and Operations



Two of an administrator's main tasks will be to make sure that the initial deployment of the Caching Application Block is planned and managed and to make sure that subsequent updates are deployed with minimal impact to existing applications that use the block. For details of deploying and updating Enterprise Library and the blocks, see <u>Deploying Enterprise Library</u>.

In addition, administrators must decide whether they want to use the instrumentation exposed by the block. For details of how to enable and disable instrumentation, see Enabling Instrumentation. For information about the instrumentation contained within the Caching Application Block, see the following topics:

- Caching Application Block Performance Counters
- Caching Application Block Event Log Entries

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Caching Application Block Performance Counters



The following table describes the Caching Application Block performance counters.

Performance Counter	Description
Cache Expiries/sec	The rate at which items were expired from the cache.
Cache Hit Ratio	The ratio of cache hits to reads. A cache hit occurs when the cache contains the requested item.
Cache Hits/sec	The rate at which the cache received requests for items it contained.
Cache Misses/sec	The rate at which the cache received requests for items it did not contain.
Cache Scavenged Items/sec	The rate at which items were scavenged from the cache.
Total # of Cache Access Attempts	The total number of reads from the cache.
Total Cache Entries	The total number of items in the cache.
Total Cache Expiries	The total number of items expired from the cache.
Total Cache Hits	The total number of requests for existing items received by the cache.
Total Cache Misses	The total number of requests for non-existing items received by the cache.
Total Cache Scavenged Items	The total number of items scavenged from the cache.
Total Updated	The total number of items updated in the cache. An

Entries	update can be either an "add" or a "remove" action.
Updated Entries/sec	The rate at which items in the cache were updated. An update can be either an "add" or a "remove" action.

A rate counter samples an increasing count of events over time and divides the values by the change in time to display a rate of activity. For more information about performance counters, see Overview of Performance Monitoring in the .NET Framework Class Library on MSDN.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Caching Application Block Event Log Entries



This topic lists the Caching Application Block event log entries. The listener is the class that raised the event.

■ Cache Failed Event

■ Cache Callback Failed Event

■ Configuration Error Event

■ Configuration Changed Event

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Cryptography Application Block



Developers frequently write applications that require encryption and hashing capabilities to meet the security requirements of their organization. Data that is created and maintained by applications, as well as configuration information, often needs to be encrypted. Additionally, passwords that are used to access application functionality or data need to be hashed.

The Enterprise Library Cryptography Application Block simplifies the way developers incorporate cryptographic functionality in their applications. Applications can use the block for a variety of tasks, such as encrypting information, creating a hash from data, and comparing hash values to verify that data has not been altered. In addition, you can change the underlying providers through configuration without changing the underlying application code.

The Cryptography Application Block includes support for the following features:

- Encryption algorithms
- Hashing algorithms
- Multiple cryptography providers
- Additional implementations of cryptography providers
- Key protection with the data protection API (DPAPI)

This section includes the following topics that will help you to understand and use the Cryptography Application Block:

- What Does the Cryptography Application Block Do? This topic provides a brief overview that will help you to understand what the block can do, and explains some of the concepts and features it incorporates. It also provides a simple example of writing code to use the block.
- When Should I Use the Cryptography Application Block? This topic will help you to decide if the block is suitable for your requirements. It explains the benefits of using the block, and any alternative techniques you may consider. It also provides details of any limitations of the block that may affect your decision to use it.
- <u>Developing Applications Using the Cryptography Application Block</u>. This topic describes how to install the Cryptography Application Block so that you can use it in your applications. It also describes how to

- configure the block for common operations.
- <u>Key Scenarios</u>. This topic then shows how to use the block to perform most cryptography tasks.
- Design of the Cryptography Application Block. This topic explains the decisions that went into the design of the block and the rationale behind those decisions.
- Extending and Modifying the Cryptography Application Block. This topic explains how to extend the Cryptography Application Block by creating your own providers and how to modify the source code.
- <u>Deployment and Operations</u>. This topic explains how to deploy and update the block's assemblies. It also contains information about configuration.

More Information

For more information, see the following resources:

- Improving Web Application Security: Threats and Countermeasures
- How To: Use Authorization Manager (AzMan) with ASP.NET 2.0
- patterns & practices Security How To Index

To provide feedback, get assistance, or download additional content, please visit the Enterprise Library Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

What Does the Cryptography Application Block Do?



The Cryptography Application Block consists of a facade that allows you to access the two types of providers contained within the block. These two types of providers are:

- **Hashing providers**. These providers can be used to generate a hash from a value you supply, and compare two hash values. The block includes hash providers that use a range of common hashing algorithms.
- **Cryptography providers**. These providers can be used to encrypt and decrypt values that you supply. The block includes cryptography providers that use a range of common encryption algorithms.

When you use the cryptography providers to encrypt a value, you can specify the value as an array of bytes, and the method will return the result as an array of bytes. Alternatively, you can specify the input value as a string, and the methods will return the result as a base-64 encoded string. The methods that decrypt values work the same way, except that string values returned by the methods are not base-64 encoded.

When you use the hashing providers to create a hash, you can specify the value to hash as an array of bytes, and the method will return the result as an array of bytes. Alternatively, you can specify the input value as a string, and the methods will return the result as a string. The methods that compare hash values accept either an array of bytes or a string, and return either true or false.

■ Example Application Code

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

When Should I Use the Cryptography Application Block?



You should use the Cryptography Application Block when you need hashing and/or symmetric encryption functionality. You can use these functions in conjunction with the cryptographic providers included with the block or with your own custom cryptographic providers. If the data only needs to be encrypted, and it does not need to be decrypted (for example, a password), you can use hashing. If the data needs to be both encrypted and decrypted (for example, to transmit sensitive customer data), you can use symmetric encryption.

■ Scenarios for the Cryptography Application Block

Benefits of the	Cryptography	Application	Block
-----------------------------------	--------------	--------------------	-------

■ Limitations of the Cryptography Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Developing Applications Using the Cryptography Application Block



This topic describes how to develop applications using the Cryptography Application Block. It <u>explains</u> how to modify the configuration of the block to perform particular tasks and how to use the block for particular scenarios, such as encrypting data. This topic assumes that you are using the block in its original state, without extending it. (To learn how to add functionality, see <u>Extending and Modifying the Cryptography Application Block</u>.) This section includes the following topics:

- Entering Configuration Information
- Adding Application Code

All application blocks ship as binary assemblies and as source code. If you want to use the source code, you must compile it. To learn how to compile the Enterprise Library source code, see <u>Building Enterprise Library from the Source Code</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

Entering Configuration Information



These procedures explain how to configure the Cryptography Application Block. Also see <u>Using the Cryptographic Key Wizard</u>. For details of the configuration schema, see <u>Source Schema for the Cryptography Application Block</u>. You can also configure the block in code by using an alternate configuration source. For more information, see <u>Advanced Configuration Scenarios</u> and <u>Using the Fluent Configuration API</u>.

To add the Cryptography Application Block

- 1. Open the configuration file. For more information, see <u>Configuring Enterprise Library</u>.
- 2. Open the **Blocks** menu and then click **Add Cryptography Settings**.
- 3. The configuration tool automatically adds the **Cryptography Settings** section, and the **Hash Providers** and **Symmetric Cryptography Providers** sub sections, with default settings.
- 4. To edit the properties of the **Cryptography Settings** section, click the property expander chevron.

To configure a hash algorithm provider

- 1. Click the **Hash Providers** plus sign icon, point to **Add Hash Providers**, and then click **Add Hash Algorithm Provider**.
- 2. In the type selector dialog, select the hash algorithm provider type you want to use. If the hash provider is not included, click **Add from File** or **Add from GAC** to locate the assembly that contains the required type.
- 3. (Optional) In the **Name** property textbox, change the name of the hash algorithm provider. The default name is the type name you selected in step 2.
- 4. Set the **Salt Enabled** property by clicking **True** or **False** in the drop-down list. The default is **True**.

✓ Note:

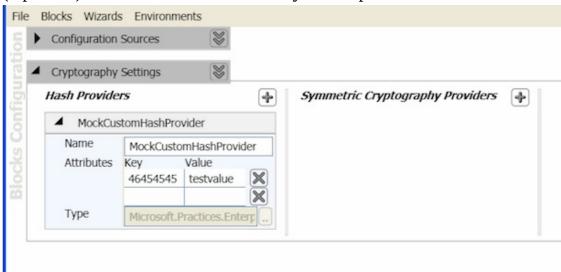
The Cryptographic Key Wizard appears for keyed hash algorithm providers. For information about generating and importing keys, see the section <u>Using</u>

To configure a custom hash provider

1. Click the **Hash Providers** plus sign icon, point to **Add Hash Providers**, and then click **Add Custom Hash Provider**.



- 2. In the type selector dialog either type the full path name for a custom hash provider or navigate to it in the list. You can filter the classes displayed by typing in the text box. To add a custom provider from another assembly, click **Add from File** and navigate to the assembly file. To add a provider stored in the global assembly cache (GAC), click **Add from GAC**.
- 3. (Optional) Add custom **AttributesKey/Value** pairs in the edit box.



4. (Optional) Set the **Name** property by typing the name in the edit box. The default name is the name of the type selected by using the **Type Selector** tool.

To configure a DPAPI symmetric cryptography provider

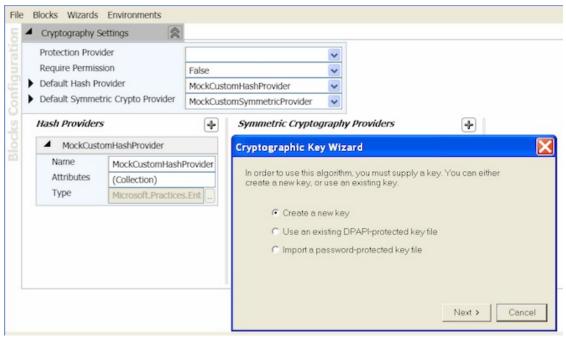
1. Click the **Symmetric Cryptography Providers** plus sign icon, point to **Add Symmetric Cryptography Providers**, and then click **Add DPAPI Symmetric Crypto Provider**.

- 2. (Optional) Change the **Name** property of the DPAPI symmetric cryptography provider. The default name is **DPAPI Symmetric Crypto Provider**.
- 3. Set the **Protection Scope** property. In the drop-down list, click **CurrentUser** or **LocalMachine**.

The **CurrentUser** value means that DPAPI uses a loaded user profile to generate the key. Only that particular user account can decrypt the encrypted data. The **LocalMachine** value means that that any code running on the machine has access to the protected key; therefore, it can decrypt any secret encrypted in **LocalMachine** mode. To counteract this, your application code can pass an entropy value when it calls the **Encrypt** or **Decrypt** methods. Entropy makes it more difficult for one application, running on the same computer, to compromise another application's encryption key. However, you must protect the entropy value. If it is simply saved to an unprotected file, attackers can access the file, retrieve the entropy value, and use it to decrypt an application's data. The Cryptography Application Block configuration does not include the entropy value. This means that you cannot use the configuration tools to create or save an entropy value.

To configure a symmetric algorithm provider

- 1. Click the **Symmetric Cryptography Providers** plus sign icon, point to **Add Symmetric Cryptography Providers**, and then click **Add Symmetric Algorithm Provider**.
- 2. In the type selector dialog, select the symmetric algorithm provider type you want to use. You can filter the classes displayed by typing in the text box. To add a custom provider from another assembly, click **Add from File** and navigate to the assembly file. To add a provider stored in the global assembly cache, click **Add from GAC**.
- 3. After selecting the algorithm provider, the **Cryptographic Key Wizard** will run to either import or generate a key. For more information on using the Wizard, see <u>Using the Cryptographic Key Wizard</u>.



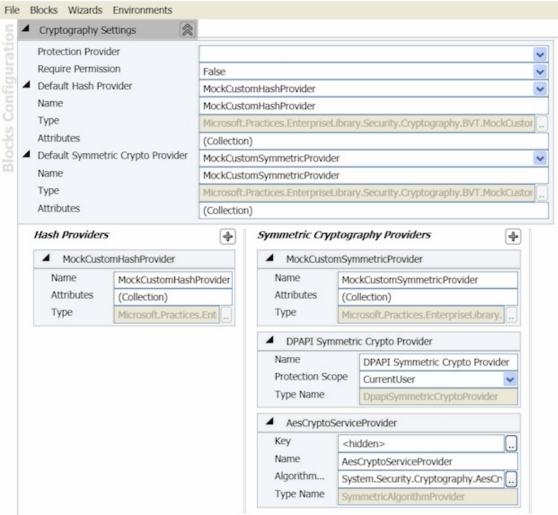
4. On the final page of the Wizard click on **Finish** to add the new provider.

To configure a custom symmetric cryptography provider

- 1. Click the **Symmetric Cryptography Providers** plus sign icon, point to **Add Symmetric Cryptography Providers**, and then click **Add Custom Symmetric Crypto Provider**.
- 2. In the type selector dialog either type the full path name for the custom symmetric cryptography provider or navigate to it. You can filter the classes displayed by typing in the text box. To add a custom provider from another assembly, click **Add from File** and navigate to the assembly file. To add a provider stored in the global assembly cache, click **Add from GAC**.
- 3. (Optional) Add custom AttributesKey/Value pairs in the edit box.
- 4. (Optional) In the **Name** property textbox, change the name of the custom symmetric cryptography provider. The default name is the name of the type selected by using the **Type Selector** tool.

To configure the default providers for the block

1. Open the **Cryptography Settings** properties by either right clicking on the **Cryptography Settings** section or by clicking on the property expander chevron.



- 2. (Optional) In the properties pane, set the **Default Hash Provider** property. This sets the instance of the hash provider that the Cryptography Application Block uses if the application code does not specify another provider. In the drop-down list, click the hash provider. The default is **none**.
- 3. (Optional) In the properties pane, set the **Default Symmetric Crypto Provider** property. This sets the instance of the symmetric cryptography provider that the Cryptography Application Block uses if the application code does not specify another provider. In the dropdown list, click the symmetric provider. The default is **none**.

■ Using the Cryptographic Key Wizard

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Source Schema for the Cryptography Application Block**

٧

This topic lists the XML elements and attributes used to configure the Cryptography Application Block. You can manually edit the XML data, but the Enterprise Library configuration tools greatly simplify this task. If you choose to manually edit the XML, use the schema information contained in this topic.

The configuration file has the following section-handler declaration.

XML

The section-handler declaration contains the name of the configuration settings section and the name of the section-handler class that processes configuration data in that section. The name of the configuration settings section is **securityCryptographyConfiguration**. The name of the section-handler class is **CryptographySettings**. It is in the **Microsoft Practices Enterprise Library Security Cryptography Configuration**.

 ${\bf Microsoft. Practices. Enterprise Library. Security. Cryptography. Configuration names pace.}$

■ securityCryptographyConfiguration Element

■ hashProviders Child Element

■ symmetricCryptoProviders Child Element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Maximizing Security**



Two important points you should consider when you use the Cryptography Application Block are how you are going to manage symmetric encryption keys and which hashing algorithm or symmetric encryption algorithm you are going to use.

■ Managing and Distributing Keys

■ Selecting an Encryption Algorithm

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Adding Application Code



The Cryptography Application Block is designed to support common scenarios for symmetric encryption and hashing. When you add your application code, refer to the scenarios in Key Scenarios and select the ones that best suit your situation. Use the code that accompanies the scenario either as it is shown here or adapt it as needed.

To prepare your application

- 1. Add a reference to the Cryptography Application Block assembly. In Visual Studio®, right-click your project node in Solution Explorer, and then click **Add Reference**. Click **Browse** to locate the **Microsoft.Practices.EnterpriseLibrary.Security.Cryptography.dll** assembly. Select the assembly, and then click **OK**.
- 2. Following the same procedure, set a reference to the following assemblies:
 - Microsoft.Practices.EnterpriseLibrary.Common.dll
 - Microsoft.Practices.Unity.dll
 - Microsoft.Practices.ServiceLocation.dll
 - Microsoft.Practices.Unity.Interception.dll
- 3. (Optional) To use elements from the Cryptography Application Block without fully qualifying the element reference, you can add the following **using** statement (C#) or **Imports** statement (Visual Basic) to the top of your source code file.

C#

using Microsoft.Practices.EnterpriseLibrary.Secu

Visual Basic

Imports Microsoft.Practices.EnterpriseLibrary.Se

4. Next, add the application code.

✓ Note:

For Microsoft® Visual Basic® projects, you can also use the References page of the Project Designer to manage references and imported namespaces. To access the References page, select a project node in Solution Explorer, and then click **Properties** on the **Project** menu. When the Project Designer appears, click the **References** tab.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Key Scenarios



This topic describes the most common situations that developers must address when providing cryptography functionality in their applications. Each scenario explains the task, describes a real-world situation where such a task might occur, and includes code that demonstrates how to use the Cryptography Application Block to complete the task. The scenarios are the following:

- Encrypting Data Using a Symmetric Provider. This scenario illustrates how you can use a symmetric algorithm provider to encrypt a secret.
- <u>Decrypting Data Using a Symmetric Provider</u>. This scenario illustrates how you can use a symmetric algorithm provider to decrypt a secret that has been encrypted.
- Obtaining a Hash Value. This scenario illustrates how you can generate a hash value from data.
- Checking Whether a Hash Value Matches Some Text. This scenario illustrates how you can compare plaintext data with a hash value previously generated from the data. By doing this, you can verify that the data has not been changed since the hash was originally generated.

✓ Note:

The non-static facade named **CryptographyManager** replaces the static **Cryptographer** facade used in previous versions of Enterprise Library. However, code that uses the **Cryptographer** facade will continue to work in this release. For information about resolving Enterprise Library objects in your applications, see <u>Creating and Referencing Enterprise Library Objects</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Encrypting Data Using a Symmetric Provider



A common cryptography task is to encrypt data using a symmetric provider. You may want to do this when an application has data you want to keep secure.

■ Typical Goals

■ Solution

■ Using EncryptSymmetric

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Decrypting Data Using a Symmetric Provider



If you encrypt data by using a symmetric encryption provider, you usually have to decrypt the data using the same provider.

■ Typical Goals

■ Solution

■ Using DecryptSymmetric

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Obtaining a Hash Value



An example of when you might want to obtain a hash value is when you have a password that you do not want to pass over the network.

■ Typical Goals

■ Solution

■ Using CreateHash

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Checking Whether a Hash Value Matches Some Text



An example of when you would check whether a hash matches some text is when you have to verify that data has not been changed in transit over a network. In this case, the hash value for the data would be stored at the server, and when the data arrives at the server, it is compared against the hash value.

■ Typical Goals

■ Solution

■ Using CompareHash

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design of the Cryptography Application Block



This topic describes the design goals, design highlights, and the implementation of the Cryptography Application Block.

■ Design Goals

■ Design Highlights

■ Key Management Model

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Designing for Simplified Cryptography Functionality



Cryptography in applications can be implemented in many ways. Typically, developers must duplicate code to perform common tasks. To meet the needs of their organization, they may have to familiarize themselves with many different ways of implementing cryptography. The Cryptography Application Block is designed to simplify and abstract the implementation of cryptography in applications.

■ Design Implications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Designing for Algorithm Abstraction



With the Cryptography Application Block, developers can refer to the algorithm to be used by the various methods by using logical names, such as "hash provider" or "password encryption."

■ Design Implications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending and Modifying the Cryptography Application Block



In its original state, the Cryptography Application Block works well for typical cryptography scenarios. However, there may be times when you have to customize some of the block's behavior to better suit your application's particular requirements. There are two ways to customize the block, extension and modification.

■ Extending the Cryptography Application Block

■ Modifying the Cryptography Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending the Cryptography Application Block



The Cryptography Application Block is designed to be used in a variety of applications and to be a general-purpose block. Extension points let you adapt the block to suit the needs of any particular application. You can extend the capabilities of the block by adding custom cryptography providers. Typically, these custom providers are third-party cryptography providers. The following table lists the interfaces that you can use to extend the block.

Custom Provider or Extension	Interface
Hash Algorithm Provider	IHashProvider
Symmetric Encryption Algorithm Provider	ISymmetricCryptoProvider

To extend the Cryptography Application Block

- 1. Create a new custom class and add it to your project.
- 2. Make sure the class implements the required interfaces, constructors, and methods.
- 3. Configure the generic provider in the Enterprise Library configuration tools:
 - Specify your custom class as the type name.
 - Specify any custom configuration properties by modifying the attributes of the object.

To create a custom hash algorithm provider

- 1. Create a new class, and then add it to your project.
- 2. (Optional) To use elements without fully qualifying the element reference, you can add the following using statement (C#) or Imports statement (Visual Basic) to the top of your source code file.

using Microsoft.Practices.EnterpriseLibrary.Secuusing Microsoft.Practices.EnterpriseLibrary.Secu

Visual Basic

Imports Microsoft.Practices.EnterpriseLibrary.Se Imports Microsoft.Practices.EnterpriseLibrary.Se

✓ Note:

For Visual Basic projects, you can also use the References page of the Project Designer to manage references and imported namespaces. To access the References page, select a project node in Solution Explorer, and then click [project name] Properties on the Project menu. When the Project Designer appears, click the References tab.

- 3. Specify that the class implements **IHashProvider**.
- 4. Add the class attribute **ConfigurationElementType**. Specify the type **CustomHashProviderData** as the attribute parameter.

```
C#

[ConfigurationElementType(typeof(CustomHashProvipublic class MyHashProvider : IHashProvider

Visual Basic
```

<ConfigurationElementType(GetType(CustomHashProv
Public Class MyHashProvider
 Implements IHashProvider</pre>

5. Add a constructor that has a parameter of type **NameValueCollection**.

```
public MyHashProvider(NameValueCollection attrib
{
}
```

Visual Basic

Public Sub New(ByVal attributes As NameValueColl End Sub

6. Add the **CreateHash** and **CompareHash** methods to your class, and then implement the required behavior.

```
public byte[] CreateHash(byte[] plaintext)
{
}

public bool CompareHash(byte[] plaintext, byte[]
{
}

Visual Basic

Public Function CreateHash(ByVal plaintext As By
End Function
Public Function CompareHash(ByVal plaintext As B
End Function
```

To create a custom symmetric encryption algorithm provider

- 1. Create a new class, and then add it to your project.
- 2. (Optional) To use elements without fully qualifying the element reference, you can add the following using statement (C#) or Imports statement (Visual Basic) to the top of your source code file.

```
using Microsoft.Practices.EnterpriseLibrary.Secuusing Microsoft.Practices.EnterpriseLibrary.Secu
```

Visual Basic

Imports Microsoft.Practices.EnterpriseLibrary.Se Imports Microsoft.Practices.EnterpriseLibrary.Se

✓ Note:

For Visual Basic projects, you can also use the References page of the Project Designer to manage references and imported namespaces. To access the References page, select a project node in Solution Explorer, and then click [project name] Properties on the Project menu. When the Project Designer appears, click the References tab.

- 3. Specify that the class implements **ISymmetricCryptoProvider**.
- 4. Add the class attribute **ConfigurationElementType**. Specify the type **CustomSymmetricCryptoProviderData** as the attribute parameter.

C#

[ConfigurationElementType(typeof(CustomSymmetric public class MyCustomEncryptionProvider : ISymme

Visual Basic

<ConfigurationElementType(GetType(CustomSymmetri
Public Class MyCustomEncryptionProvider
 Implements ISymmetricCryptoProvider</pre>

5. Add a constructor that has a parameter of type **NameValueCollection**.

C#

public MyCustomEncryptionProvider (NameValueColl
{
}

Visual Basic

```
Public Sub New(ByVal attributes As NameValueColl End Sub
```

6. Add the **Encrypt** and **Decrypt** methods to your class, and then implement the required behavior.

```
public byte[] Encrypt(byte[] plaintext)
{
}
public byte[] Decrypt(byte[] ciphertext)
{
}
```

Visual Basic

Public Function Encrypt(ByVal plaintext As Byte(
End Function
Public Function Decrypt(ByVal ciphertext As Byte
End Function

For detailed information about how to integrate custom providers with the Enterprise Library configuration system and configuration tools see <u>Creating Custom Providers for Enterprise Library</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Modifying the Cryptography Application Block



The Cryptography Application Block was designed to be used in a variety of applications and to be a general-purpose cryptography application block. Extension points let you adapt the block to suit the requirements of any particular application. However, if you want to add new features to the block, you can do so by modifying the source code (the block includes both the source code and the binaries).

✓ Note:

When modifying the source code, you should follow good practices described in the topic Extending and Modifying Enterprise Library.

■ Modifying the Key Management Code

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Deployment and Operations



Two of an administrator's main tasks will be to make sure that the initial deployment of the Cryptography Application Block is planned and managed and to make sure that subsequent updates are deployed with minimal impact to existing applications that use the block. For details, see Deploying the Cryptography Application Block.

In addition, administrators must decide if they want to use the instrumentation exposed by the block. For details of how to enable and disable instrumentation, see <u>Enabling Instrumentation</u>. For information on the instrumentation contained within the Cryptography Application Block, see the following topics:

- Cryptography Application Block Performance Counters
- Cryptography Application Block Event Log Entries

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Deploying the Cryptography Application Block



The Cryptography Application Block is comprised of multiple assemblies. Each assembly that belongs to the Cryptography Application Block has a file name that begins with *Microsoft.Practices.EnterpriseLibrary.Security.Cryptography*. Additionally, the block depends on the common assembly and on the Unity subsystem. For details of deploying and updating Enterprise Library and the blocks, see <u>Deploying Enterprise Library</u>.

■ Distributing Keys

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Cryptography Application Block Performance Counters



The following table describes the Cryptography Application Block performance counters.

Performance Counter	Description
Hash Comparisons/sec	The rate at which hash comparisons were performed.
Hash Mismatches/sec	The rate at which hash comparison mismatches were detected.
Hash Operations/sec	The rate at which plaintext was hashed.
Symmetric Decryptions/sec	The rate at which symmetric decryptions were performed.
Symmetric Encryptions/sec	The rate at which symmetric encryptions were performed.
Total Hash Comparisons	The total number of hash comparisons performed.
Total Hash Mismatches	The total number of hash comparison mismatches detected.
Total Hash Operations	The total number of plain text hashing operations performed.
Total Symmetric Decryptions	The total number of symmetric decryptions performed.
Total Symmetric Encryptions	The total number of symmetric encryptions performed.

A rate counter samples an increasing count of events over time and divides the values by the change in time to display a rate of activity. For more information about performance counters, see Overview of Performance Monitoring in the .NET Framework Class Library on MSDN.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Cryptography Application Block Event Log Entries



This topic lists the Cryptography Application Block event log entries. The listener is the class that raised the event.

■ Cryptographic Operation Failed Event (Symmetric Algorithm)

■ Cryptographic Operation Failed Event (Hash Algorithm)

■ Configuration Failure Event

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Data Access Application Block



The Enterprise Library Data Access Application Block simplifies the development of tasks that implement common data access functionality. Applications can use this application block in a variety of situations, such as reading data for display, passing data through application layers, and submitting changed data back to the database system.

The application block includes support for both stored procedures and inline SQL statements. Common housekeeping tasks, such as managing connections and creating and caching parameters, are encapsulated in the application block's methods. In other words, the Data Access Application Block provides access to the most often used features of ADO.NET in simple-to-use classes and provides a corresponding boost in developer productivity.

This section includes the following topics that will help you to understand and use the Data Access Application Block:

- What Does the Data Access Application Block Do? This topic provides a
 brief overview that will help you to understand what the block can do,
 and explains some of the concepts and features it incorporates. It also
 provides a simple example of the way that you can write code to use the
 block.
- When Should I Use the Data Access Application Block? This topic will help you to decide if the block is suitable for your requirements. It explains the benefits of using the block, and any alternative techniques you may consider. It also provides details of any limitations of the block that may affect your decision to use it.
- Developing Applications Using the Data Access Application Block. This
 describes how to configure the application block, how to prepare your
 application to use the Data Access Application Block, and contains
 details of specific features of the application block, such as how you
 create a database, work with transactions, and handle parameters and
 exceptions.
- <u>Key Scenarios</u>. This section demonstrates how to use the application block to perform the most typical data access operations.
- Design of the Data Access Application Block. This section describes the

- decisions that went into designing the application block and the rationale behind those decisions.
- Extending and Modifying the Data Access Application Block. This section describes how to extend the application block by adding your own database provider and gives suggestions for modifying the source code.
- <u>Deployment and Operations</u>. This section describes how to deploy and update the application block assemblies. It also contains information about configuration and Microsoft® SQL Server® security.

More Information

For more information, see the following resources:

- <u>Microsoft Application Architecture Guide, 2nd Edition Chapter 8: Data Layer Guidelines</u>
- .NET Data Access Architecture Guide
- INFO: Microsoft Guide for Designing Data Tier Components and Passing Data Through Tiers
- Improving Web Application Security: Threats and Countermeasures
- Improving .NET Application Performance and Scalability

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

What Does the Data Access Application Block Do?



The Data Access Application Block includes a small number of methods that simplify the most common techniques for accessing a database. Each method encapsulates the logic required to retrieve the data and manage the connection to the database. The methods exposed by the block allow you to execute queries, return data in a range of different formats, populate a **DataSet**, update the database from a **DataSet**, perform data access asynchronously (against SQL Server databases), and return data as objects in a suitable format for use with client-side query technologies such as LINQ.

ADO.NET provides classes such as the **DbCommand** class and the **DbConnection** class; these classes help to abstract the data provider from any particular database implementation. The Data Access Application Block takes advantage of these classes and provides a model that further supports encapsulation of specific features of each database type, such as parameter discovery and type conversion. As a result, applications can often be ported from one database type to another without modifying the client code. The Data Access Application Block includes an abstract base class that defines a common interface and provides much of the implementation required by the data access methods available in ADO.NET.

The application block also includes classes designed to work with Microsoft SQL Server, Microsoft SQL Server CE, and Oracle. These classes perform operations that are specific to the database type. The code for applications written for one type of database, such as SQL Server, looks much the same as the code for applications written for another type of database such as Oracle.

Another feature of the Data Access Application Block is that application code can refer to particular databases by an ADO.NET connection string name, such as "Customer" or "Inventory." The application code can specify a named instance of a database when creating the **Database** implementation it uses as a facade for the majority of the methods. Each named database has its connection information stored in a configuration file. By changing the settings in the configuration file, developers can use their applications with different database types without recompiling their code.

■ Example Application Code

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

When Should I Use the Data Access Application Block?



You should consider using the Data Access Application Block if your application uses standard data access techniques. The block is particularly suited to querying, retrieving, and updating data through implementations based on the **ExecuteReader**, **ExecuteXmlReader**, and **ExecuteNonQuery** methods commonly used in ADO.NET—including the asynchronous versions of these methods. It also provides methods to populate **DataSet** instances, and flush the changes back to the database.

You should also consider using the block if you want to work with data using client-side querying mechanisms such as Language Integrated Query (LINQ). It contains support for these programming techniques by exposing methods that return data as sequences of objects.

A third reason for using the Data Access Application Block is its capability to abstract the database type. This makes it easier to change your application to use a different type of database if required, although some of the more advanced data access methods do rely on the specific capabilities of the underlying ADO.NET Data Provider.

■ Scenarios for the Data Access Application Block

■ Benefits of the Data Access Application Block

■ Limitations of the Data Access Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Developing Applications Using the Data Access Application Block



This section describes how you can use the Data Access Application Block to develop applications. It first explains how to configure the application block and incorporate it into your applications. Next, it explains how to use the application block for specific scenarios, such as retrieving a single item or using a **DataSet** object to retrieve multiple rows. Lastly, it provides more information about topics such as connection management, parameter handling, and handling exceptions.

This section assumes that you are using the application block in its original state, without extending it. (To learn how to add functionality, see Extending and Modifying the Data Access Application Block.) This section includes the following topics:

- Entering Configuration Information
- Adding Application Code
- Creating a Database Object
- Creating a DbCommand Object
- Managing Connections
- <u>Using the TransactionScope Class</u>
- Using the Asynchronous Data Access Methods
- Returning Data as Objects for Client Side Querying
- Creating Portable Database Applications
- Handling Exceptions
- Handling Parameters

All application blocks ship as binary assemblies and as source code. If you want to use the source code, you must compile it. To learn how to compile the Enterprise Library source code, see <u>Building Enterprise Library from the Source Code</u>.

✓ Note:

The Data Access Block cannot be used in applications that target the **.NET Framework 4.0 Client Profile**. It can only be used in applications that target the full .NET Framework 4.0 profile. For information about profiles in the

.NET Framework 4.0, see .<u>NET Framework Client Profile</u> on MSDN. For information about changing the targeted profile, see <u>How to: Target a Specific .NET Framework Version or Profile</u> on MSDN.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The following procedures describe how to configure the Data Access Application Block. Properties associated with the nodes appear in the right pane of the Configuration Console or the Properties window of the Visual Studio® Configuration Editor. For details of the schema for the Data Access Application Block configuration, see Source Schema for the Data Access Application Block. You can also configure the block in code by using an alternate configuration source. For more information, see Advanced Configuration Scenarios and Using the Fluent Configuration API.

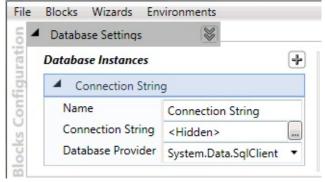
To add the Data Access Application Block

- 1. Open the configuration file. For more information, see <u>Configuring Enterprise Library</u>.
- 2. Open the **Blocks** menu and then click **Add Data Settings**.
- 3. The configuration tool automatically adds the **Database Settings** section with default settings. Click the properties expander chevron in the **Database Settings** section to view the settings for this section.

The next procedure explains how to configure an instance of the default database that is added automatically to the configuration. This instance is used if the application resolves a **Database** with no instance name.

To configure the default database

1. In the **Database Instances** pane, click the expander arrow for the **Connection String** section.



- 2. (Optional) Set the **Name** property by typing a new name. The default name is **ConnectionString**.
- 3. Use the **Database Provider** property to change the provider name as

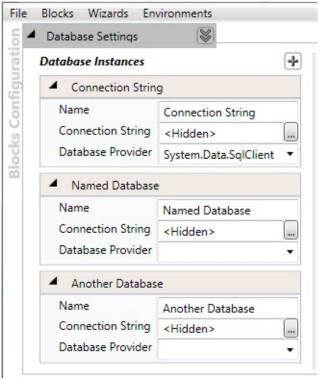
required. Enter the name of the provider or select it from the drop-down list. The default provider name is **System.Data.SqlClient**. You can choose a different provider if you wish, such as **System.Data.OleDb**. The **Database Provider** property must be a provider name specified in a **DBProviderFactory** class.

4. Click the ellipses (...) button in the **Connection String** property to show the connection string in a pop-up window. Edit it as required.

The next procedure explains how to create additional database instances for other databases.

To configure additional database instances

1. Click the plus sign icon in the **Database Instances** pane and then click **Add Database** to add a new connection string section.



- 2. (Optional) Set the **Name** property by typing a new name. This is the name you will use to refer to the database.
- 3. In the **Database Provider** property section, enter the name of the database provider or select it from the drop-down list. The **Database Provider** property must be a provider name specified in a **DbProviderFactory** class. If you are using the Visual Studio integrated configuration editor and you do not see the provider type you require in the list, you must add a reference to the assembly containing the

- provider to your project.
- 4. Click the ellipses (...) button in the **Connection String** property to show the connection string in a pop-up window. Type the connection string that this database will use. For example, the following connection string specifies the database named CorpData1 on the SQL Server instance named DBSERVER1 using integrated Windows® security:

```
Server=DBSERVER1; Initial Catalog=CorpData1; Int
```

The next procedure describes how to change which of the currently configured databases is the default that will be used if the application resolves a **Database** with no instance name.

To change the default database

- 1. Click the properties expander chevron in the **Database Settings** section to view the settings for this section if it is not already open.
- 2. Select the database that you want to use as the default in the drop-down list for the **Default Database Instance** property.

The next procedure describes how to configure a SQL Server CE database. These steps are appropriate if your application always uses a single file that you name during configuration. For more information about SQL Server CE, see Creating a Database Object.

To configure SQL Server CE

- 1. Click the plus sign icon in the **Custom Databases** pane and then click **Add Custom Database Provider**.
- 2. Set the **Name** property of the custom database by typing a new name. This name is used to link the SQL Server CE database provider to the connection string you will define for it.
- 3. In the **Type** property, click the ellipses (...) button, fully expand the Microsoft.Practices.EnterpriseLibrary.Data.SqlCe node, and select **SqlCeDatabase**. Then click **OK**.
- 4. Click the plus sign icon in the **Database Instances** pane and then click **Add Database Connection String**. This adds a new connection string to the configuration, which you will use to specify the connection string

- for the SQL Server CE database. Alternatively, you can configure the default **Connection String** item that the configuration tool adds if you do not require any other connection strings in your configuration.
- 5. (Optional) Set the **Name** property of the connection string item by typing a new name. This is the name you will use to refer to the database in your code.
- 6. Click the ellipses (...) button in the **Connection String** property to show the connection string in a pop-up window. Enter the appropriate value for the **Connection String** property; for example:

Copy Code

Data Source='C:\MyApp\MyDatabase.sdf'

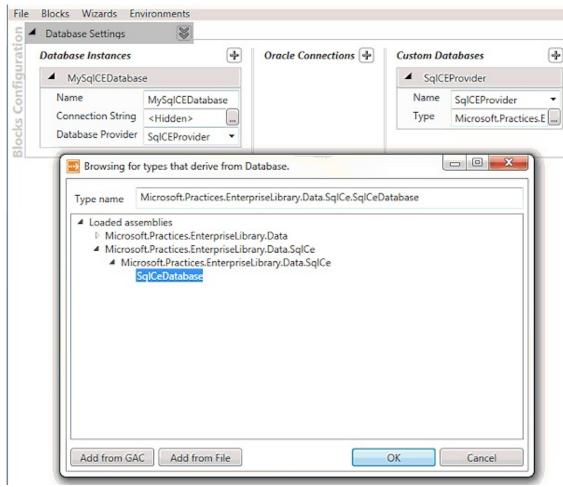
✓ Note:

The SQL CE database is created by the Enterprise Library **SQLDatabase** provider, which does not enforce password protection or encryption on the database during creation. To ensure security of yoru data, you should consider setting the connection string to protect the file and create it with the correct permissions, and encrypt the file. For more information see <u>SqlCEConnection</u> <u>ConnectionString Property</u> on MSDN.

7. The following example enables encryption on the database:

"Data Source=MyData.sdf;Encrypt Database=True;Pa

8. Set the **Database Provider** property of your SQL Server CE connection string item to the name you entered for the custom database provider in step 2. The following screenshot shows details of the configuration for a SQL Server CE database in the configuration tools.



9. For information on using SQL Server CE, see the section "Using SQL Server CE" in the topic <u>Creating a Database Object</u>.

The next procedure describes how to add Oracle packages. An Oracle package serves as a way to group stored procedures into common groups, typically based on their functionality. When an application calls an Oracle stored procedure located in a package, the code must prefix the stored procedure name with the package name. For example, to call a procedure named **GetEmployeeName** that is in a package named **Employee_pkg**, you would call **Employee_pkg.GetEmployeeName**.

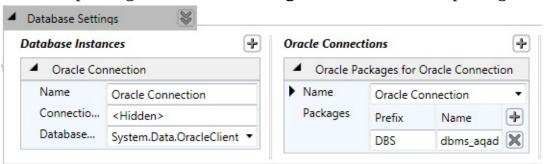
Incorporating this code into the application makes it less portable because this syntax is specific to Oracle. Instead, the Data Access Application Block can prefix the stored procedure with the package name. This means your client code does not need to specify the package name to call a stored procedure. To do this, the application block uses information in the configuration file. The **OraclePackage** node stores a name/prefix pair. The name is the name of the

package. The prefix is a string that is associated with the package. All stored procedures that start with that prefix are assumed to be in the associated package.

When the application calls a stored procedure, the Data Access Application Block checks to see if it begins with any of the prefixes in the configuration file. If it does, the application block prefixes the stored procedure with the associated package name. (The first match it finds is the one the application block uses.) If you specify an asterisk ("*") as the prefix, the associated package is used for all stored procedure calls.

To configure an Oracle package

- 1. Click the plus sign icon in the **Oracle Connections** pane, and click **Add Oracle Connection**.
- 2. An **Oracle Packages for Oracle Connection** section is added to the **Oracle Connections** pane and its properties are displayed.
- 3. (Optional) Set the **Name** property by typing a new name.
- 4. Click the plus sign icon in the **Packages** row to add a new package.



- 5. Change the **Name** property by entering the name of the Oracle package. The default name is **Package**.
- 6. Enter a value for the **Prefix** property.

✓ Note:

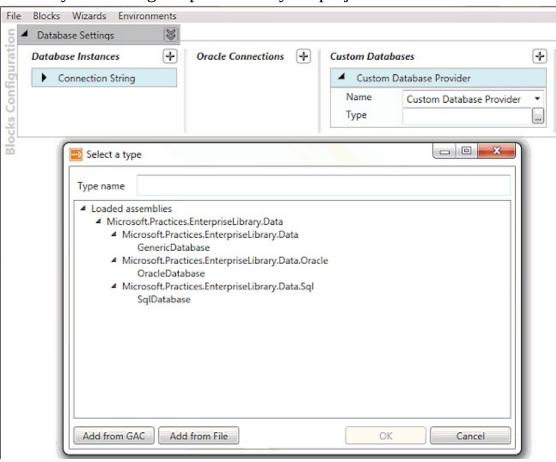
The OracleClient data provider is deprecated in version 4.0 of the .NET Framework, although it is still supported by the Enterprise Library 5.0. For future development, consider choosing a different Oracle driver, such as that available from the Enterprise Library Contrib site.

The next procedure describes how to add custom provider mappings by

associating a provider with the fully qualified name of a database.

To configure a custom provider

- 1. Click the plus sign icon in the **Custom Databases** pane, and then click on **Add Custom Database Provider**.
- 2. (Optional) Set the **Name** property by typing a new name or select it from the drop-down list. The default provider name is **Custom Database Provider**.
- 3. In the **Type** property text box, click the ellipsis button (...) and use the type selector to select the fully qualified name of the Enterprise Library database type. If the provider is registered in the global assembly cache (GAC), click the **Add from GAC** button to locate it. If you are using the Visual Studio integrated configuration editor and you do not see the provider type you require in the list, you must add a reference to the assembly containing the provider to your project.



■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

This topic lists the XML elements and attributes used to configure the Data Access Application Block. You can manually edit the XML data, but the Enterprise Library configuration tools greatly simplify this task. If you choose to manually edit the XML, use the schema information contained in this topic.

The configuration file has the following section-handler declaration.

```
XML
```

The section-handler declaration contains the name of the configuration settings sections and the names of the section-handler classes that process configuration data in that section. The name of the first configuration settings section is **dataConfiguration**. The name of the section-handler class is **DatabaseSettings** (in the **Microsoft.Practices.EnterpriseLibrary.Data.Configuration** namespace).

The name of the second configuration settings section is **oracleConnectionSettings**. The name of the section-handler class is **OracleConnectionSettings** (in the **Microsoft.Practices.EnterpriseLibrary.Data.Oracle.Configuration** namespace).

• connectionStrings Element

■ dataConfiguration Element

■ providerMappings Child Element

• oracleConnectionSettings

■ packages Child Element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Adding Application Code

The Data Access Application Block is designed to support the most common scenarios for accessing a database. When you add your application code, refer to the scenarios in the Key Scenarios section and select the ones that best match your situation. Use the code that accompanies the scenario as-is or adapt it as necessary.

First, you must prepare your application to use the Data Access Application Block. After you do that, you can create the **Database** object and call the appropriate method overloads.

To prepare your application

- 1. Add a reference to the Data Access Application Block assembly. In Visual Studio, right-click your project node in Solution Explorer, and then click **Add Reference**. Click the **Browse** tab, and then navigate to the location of the **Microsoft.Practices.EnterpriseLibrary.Data.dll** assembly. Select the assembly, and then click **OK** to add the reference.
- 2. Following the same procedure, add references to the following assemblies:
 - Microsoft.Practices.EnterpriseLibrary.Common.dll
 - Microsoft.Practices.Unity.dll
 - Microsoft.Practices.ServiceLocation.dll
 - Microsoft.Practices.Unity.Interception.dll.
- 3. (Optional) To use elements from Data Access Application Block without fully qualifying the element reference, you can add the following **using** statement (C#) or **Imports** statement (Visual Basic®) to the top of your source code file.



Imports Microsoft.Practices.EnterpriseLibrary.Da Imports System.Data

✓ Note:

For Visual Basic projects, you can use the References page of the Project Designer to manage references and imported namespaces. To access the References page, select a project node in Solution Explorer. On the **Project** menu, click **Properties**. When the Project Designer appears, click the **References** tab.

For information on how to create a Database object and advice on using the Database types, see <u>Creating a Database Object</u>.

For information about the key scenarios for using the block, see **Key Scenarios**.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{^{\odot}}$ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Creating a Database Object



All data access methods are executed against a **Database** object. You can use the Enterprise Library container to create a **Database** object. The specific type of **Database** object it returns is determined by the application configuration information. By changing the default configuration, the unmodified application can be run against different databases. The connection string information for each database you define is stored in the **<connectionStrings>** section in the application configuration file.

This topic includes the following sections that describe how to create instances of different types of databases, and some things you must consider when working with specific types of databases:

- Creating a Default Database Instance
- Creating a Named Database Instance
- Creating a Specific Database Type
- Creating a Database Instance Directly
- Creating a GenericDatabase Instance
- Writing Code to Use the Database Classes
- <u>Using SQL Server CE</u>
- <u>Using Oracle with the TransactionScope Class</u>
- <u>Using the OracleDataReaderWrapper Class</u>

■ Creating a Default Database Instance

■ Creating a Named Database Instance

■ Creating a Specific Database Type

■ Creating a Database Instance Directly

■ Creating a GenericDatabase Instance

■ Writing Code to Use the Database Classes

■ Using SQL Server CE

■ Using	Oracle	with th	ie Tran	sactionS	cope Clas	S

■ Using the OracleDataReaderWrapper Class

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Creating a DbCommand Object



The Data Access Application Block provides a consistent way to retrieve ADO.NET **DbCommand** objects. The data access methods of the application block include overloads that accept a **DbCommand** object. If you use the overloads with **DbCommand** objects, you have more control when you call stored procedures. For example, if you use a **DbCommand** object, you can have a stored procedure that returns several results in the output parameters. In addition, a **DbCommand** object allows you to specify the stored procedure's timeout value.

The methods that create **DbCommand** objects are separated into two types:

- Methods that represent stored procedure calls (for example, GetCustomers)
- Methods that represent SQL text commands (for example, Select CustomerID, Fullname From Customers)

The method you call to retrieve a **DbCommand** object is determined by whether you want to execute inline SQL or call a stored procedure. The method that creates a **DbCommand** object for a stored procedure also provides parameter caching. For more information about parameter caching, see Handling Parameters.

All **DbCommand** objects are created using methods on the **Database** class. These methods are the following:

- **GetStoredProcCommand**. This method is for stored procedures commands.
- **GetSqlStringCommand**. This method is for SQL text commands.

Both methods return a **DbCommand** object.

✓ Note:

SQL Server CE does not support stored procedures. Instead, use inline SQL statements. For more information, see the section "Using SQL Server CE" in Creating a Database Object.

■ DbCommand Objects for SQL Statements

■ DbCommand Objects for Stored Procedures

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Managing Connections

Database connections are a limited resource, and proper management of them is essential for scalable applications. It is good practice to keep connections open only as long as they are needed and to close them as soon as practical. By design, most of the **Database** class methods handle the opening and closing of connections to the database on each call. Therefore, the application code does not need to include code for managing connections. (By default, and for performance reasons, ADO.NET returns connections to the connection pool without closing them. Therefore, you do not need to cache your **Database** objects.)

For example, the **ExecuteDataSet** method returns a **DataSet** object that contains all the data. This gives you your own local copy. The call to **ExecuteDataSet** opens a connection, populates a **DataSet**, and closes the connection before returning the result.

The following code demonstrates the use of the **ExecuteDataSet** method. It assumes that you have resolved the **Database** class you require and stored a reference in the variable named **db**.

For more information on instantiating objects, see <u>Creating and Referencing Enterprise Library Objects</u>.

```
string sql = "Select ProductID, ProductName From ProductDocommand cmd = db.GetSqlStringCommand(sql);

// No need to open the connection; just make the call.
DataSet customerDataSet = db.ExecuteDataSet(cmd);

Visual Basic

Dim sql As String = "Select ProductID, ProductName From ProductName From ProductID, ProductName From ProductName From ProductID, ProductName From ProductName From Prod
```

' No need to open the connection; just make the call. Dim customerDataSet As DataSet = db.ExecuteDataSet(cmc

However, there are other cases where it is unclear when to close the connection. An example is the **ExecuteReader** method. This method returns an object that implements the **IDataReader** interface. The **Database** base class has a default implementation that returns a **DbDataReader** object. **DbDataReader** objects are designed to read specific portions of the data as needed, which requires an open connection. In other words, it is unknown when the application no longer needs the **DbDataReader**.

If the Data Access Application Block methods close the connection before returning the **DbDataReader**, the **DbDataReader** becomes useless to the client code. Instead, the **DbDataReader** methods indicate to the underlying ADO.NET call to automatically close the connection when the **DbDataReader** is disposed.

✓ Note:

If you fail to close the **DbDataReader**, you can cause random and hard-to-find errors in your code—particularly when you are operating under an implicit transaction created within a **TransactionScope** context. You should always ensure that your application closes the **DbDataReader** in a timely fashion, either by explicitly closing the reader using the **DbDataReader.Close** method or by forcing the disposal of the **DbDataReader**, which results in the **Close** method being called.

The following code demonstrates a call to the **ExecuteReader** method. The **using** statement (**Using** in Visual Basic) ensures that the **DbDataReader** object is disposed, which closes the **DbDataReader** object. It assumes that you have resolved the **Database** class you require and stored a reference in the variable named **db**.

For more information on instantiating objects, see <u>Creating and Referencing</u> <u>Enterprise Library Objects</u>.

-		-	۰
		-	
	$\overline{}$	11	

```
DbCommand cmd = db.GetSqlStringCommand("Select Name, A
using (IDataReader reader = db.ExecuteReader(cmd))
{
    // Process results
}
```

Visual Basic

Dim cmd As DbCommand = db.GetSqlStringCommand("Select
Using reader As IDataReader = db.ExecuteReader(cmd)
 ' Process results
End Using

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Using the TransactionScope Class
```

Some of the **Database** class methods take advantage of the .NET Framework **TransactionScope** class. This class automatically enlists database calls into an ambient transaction. This is useful for enlisting business objects in a transaction without passing a transaction to those business objects. Here is the basic model for using the **TransactionScope** class. It assumes that you have resolved the **Database** class you require and stored a reference in the variable named **db**.

For more information on instantiating objects see <u>Creating and Referencing</u> <u>Enterprise Library Objects</u>.

```
using (TransactionScope scope = new TransactionScope(1
{
  int dRows = db.ExecuteNonQuery(CommandType.Text, ins
  dRows = db.ExecuteNonQuery(CommandType.Text, insertS
}
```

Visual Basic

Using scope As New TransactionScope(TransactionScopeOr Dim dRows As Integer = db.ExecuteNonQuery(CommandTyr dRows = db.ExecuteNonQuery(CommandType.Text, insertS End Using

The two **ExecuteNonQuery** methods insert the rows within the transaction that you define when you create a new **TransactionScope** instance.

The **TransactionScope** class creates a local, lightweight transaction. It assumes that you will use a single connection for all of the database calls that occur within the transaction. This means that, instead of passing the **DbTransaction** instance, you simply pass the connection, and the .NET Framework automatically sets the transaction for each command that you execute.

Enterprise Library, on the other hand, normally opens and closes a connection

for each request. This approach is incompatible with the way the **TransactionScope** class works. If there are multiple connections, the **TransactionScope** class considers the transaction to be a distributed transaction. Distributed transactions have a significant performance and resource overhead compared with a local transaction.

To avoid this, the **Database** class methods, such as **ExecuteDataSet**, recognize when a **TransactionScope** instance is active and they enlist database calls in this transaction. If a transaction is currently active as a result of using a **TransactionScope** instance, the **Database** class methods use a single connection.

In particular, the **GetOpenConnection** method replaces the **OpenConnection** method within the **Database** methods. The **GetOpenConnection** method returns a connection inside a wrapper. The method disposes the wrapper if there is no transaction in progress. However, when a transaction is in progress, the method keeps the connection open.

✓ Note:

Multiple threads sharing the same transaction in a transaction scope will cause the following exception: "Transaction context in use by another session."

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Using the Asynchronous Data Access Methods



One of the most common factors that can affect performance in applications is accessing a database, and most system designs aim to minimize database round trips. Simply calling across a network to a database can cause significant delays, and the additional delay while the database processes a query can be particularly troublesome in interactive applications where the delay can cause the UI to become unresponsive. To resolve this, you may consider using asynchronous methods to access your database.

The Data Access Application Block contains versions of four of the data access methods that you can execute asynchronously. These methods rely on the features of the underlying ADO.NET data provider, and are therefore available only for the **SqlDatabase** implementation of the **Database** class in this release. The **Database** class exposes a method named **SupportsAsync** that each database-specific class (such as **SqlDatabase**, **OracleDatabase**, and **SqlCeDatabase**) implements to return true or false. The three pairs of asynchronous methods available are the following:

- BeginExecuteReader and EndExecuteReader. These methods are used to obtain a DataReader populated with data from the database.
- **BeginExecuteScalar** and **EndExecuteScalar**. These methods are used to obtain a single value from the database.
- **BeginExecuteNonQuery** and **EndExecuteNonQuery**. These methods are used to execute a query that does not return data, but returns only a count of the number of rows affected.

In addition, when using **SqlDatabase** directly, an additional pair of asynchronous methods is available:

• **BeginExecuteXmlReader** and **EndExecuteXmlReader**. These methods are used to obtain an **XmlReader** populated with data from the database.

The **Begin** and **End** versions of the methods operate in the same way as those available when you code directly against the ADO.NET provider. When using the Data Access Application Block, you pass to the methods the same parameters as for the non-asynchronous versions, such as a **DbCommand** and (optionally) a transaction reference. The methods return an instance of the **IAsyncResult** interface, which you then use as the parameter in a call to the

End version of the method to obtain the result. Your code waits until the method completes. However, you can use an array of the standard Windows **WaitHandle** class to start multiple data access operations, and the code will return when they all complete or time out.

If you want to execute code in your application while the data access call is executing, you can use a lambda expression or a separate callback that is executed when data access is complete. If you use a separate callback, you pass a reference to your callback method to the **Begin** method, and optionally an object containing state information that you want to make available in the callback method. A common use of this object is to pass a reference to the **Database** object so that you can call the appropriate **End** method inside the callback handler. If you use a lambda expression, it will have access to the objects (such as the **Database** instance) that you use in the data access code.

As the methods for performing asynchronous execution match those available for the ADO.NET **SqlClient** class, the full range of options and approaches to performing asynchronous data access are not included in this topic. For more information, see <u>Asynchronous Command Execution in ADO.NET 2.0</u>.

■ Asynchronous Accessor Execution

■ Tips for Using the Asynchronous Data Access Methods

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

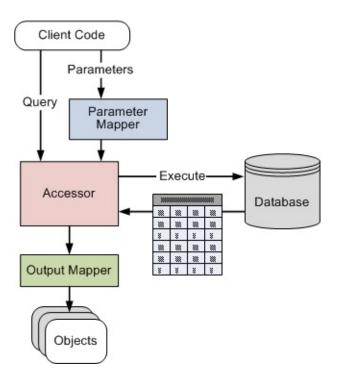
To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Returning Data as Objects for Client Side Querying



Developers often want to retrieve data in a format that better matches the real-world objects that the data represents, rather than in the rows and columns format of a **DataReader** or **DataSet**. This is typically the case when implementing common design patterns such as Domain Model, Table Module, and Repository.

The Data Access Application Block includes support for this scenario through a class called an accessor. Accessors accept information that is required to extract the data, and mappings that indicate how the input parameters correspond to the parameters of the underlying query. They use output mappings that define how the returned columns map to the properties of the objects the developer wants to work with, and return a sequence of objects of the specified type. The following schematic shows the high level process when using an accessor to retrieve data as an enumerable sequence of objects.



You can execute both stored procedures and SQL statements using the accessor approach. The block includes the **SprocAccessor** class for stored procedures

and the **SqlStringAccessor** class for executing SQL statements. Accessors also provide methods for asynchronous data retrieval where the database you are using supports asynchronous query execution.

The mapping mechanism is flexible and extensible. For stored procedures executed against SQL Server and Oracle databases, the accessor will attempt to resolve the parameters automatically if you do not specify a parameter mapper that defines the correlation. However, default parameter mapping is not available when using SQL statements, or for other databases and providers. In these cases, you must specify a custom parameter mapper that can resolve the parameters.

✓ Note:

Keep in mind that creating stored procedure accessors with the default mapper may be resource intensive and affect performance. Consider caching the accessor and/or the mapper.

When you execute an accessor, you can provide an output mapping that indicates how the accessor should map the values returned from the database to the properties of the objects it returns to the caller. If you do not specify an output mapper, the block uses a default map builder class that maps the column names of the returned data to properties of the objects it creates. Alternatively, you can create a custom mapping to specify the correlation between columns in the row set and the properties of the objects.

The accessor returns data as a sequence of objects in the form **IEnumerable**<**TResult**>, where each object represents one row of data in the data source and exposes properties that map to the columns in each row. You can handle the results in your code as objects that are part of your data model, or query them using client-side techniques such as Language Integrated Query (LINQ).

The following shows a simple example of executing a stored procedure that takes no parameters and then querying the results returned from the accessor. The code assumes you have defined the **Customer** class elsewhere, and you have resolved an instance of the **Database** class you want to use and stored it in the variable named **db**.

Order By customer.Name _

```
Select Name = customer.Name
' Display the results
For Each customer In results
   Console.WriteLine("{0} is a top customer in Washingt
Next
```

There are also methods that allow you to pass parameters or a parameter mapper to the accessor, and execute a query without creating an accessor directly. For more information about using accessors to retrieve data as objects, see the following topics:

- Defining Parameter Mappers
- Building Output Mappers
- Creating and Using Accessors
- Executing Queries without Creating an Accessor
- Executing Accessor Queries Asynchronously
- Additional Information for Accessors and Client-side Queries

✓ Note:

The accessor feature in the Data Access Application Block is not an Object/Relational Mapping (OR/M) mechanism, and should not to be confused with LINQ to SQL, or any other implementation that performs optimization of queries. It does not provide support for updates, identity maps, foreign keys, joins, or automatic SQL generation. The technique for client-side queries more closely resembles that of LINQ to Objects. However, there is no assumption on how you will (or can) use the returned object graph in your applications.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Defining Parameter Mappers



A parameter mapper takes the set of objects you want to pass to a query and converts each one into a **DbParameter** object. The accessor assigns these parameters to the command object it uses to access the data source. For information about using parameter mappers with an accessor, see <u>Creating and Using Accessors</u> and <u>Executing Queries without Creating an Accessor</u>.

The Data Access Application Block includes a default parameter mapping mechanism. This mechanism maps values you provide as query parameters to the **DbParameter** instances the query will use when executing against the database.

✓ Note:

The default parameter mapping mechanism is only available for the **SqlDatabase** and **OracleDatabase** classes, or for custom database classes you create where the database supports parameter discovery. The **Database** class exposes a **Boolean** property named **SupportsParemeterDiscovery** that your code can test (note that, for backward compatibility reasons, the misspelling of this name is preserved in the current version). If you use any other database provider, you must create a suitable implementation of the **IParameterMapper** interface that assigns the parameter values to the **DbCommand**. For more information, see <u>Creating Custom Parameter Mappers</u> later in this topic.

This default mapping uses the position of the parameters you provide in the object array, and executes the ADO.NET **DeriveParameters** method to discover the parameters required by the procedure. It maps your **parameter(0)** to the first parameter of the procedure, **parameter(1)** to the second parameter of the procedure, and so on. It converts the CLR types you specify into the appropriate database types for each **DbParameter** it populates.

The ADO.NET **DeriveParameters** call that resolves parameters for a stored procedure requires a round-trip to the database. The application block provides parameter information caching to mitigate the performance hit that this incurs.

After the first call to a stored procedure that requires parameter discovery, the information about each parameter is saved in the parameter cache. This means that subsequent calls to the same stored procedure will not require a round- trip to the database.

■ Creating Custom Parameter Mappers

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Building Output Mappers



An output mapper takes the result set returned from a database (in the form of rows and columns) and converts the data into a sequence of objects. There are two different types of output mappers that you can use to transform the returned data. *Row mappers* transform each row into an object, so that the accessor can return a sequence of these objects. *Result set mappers* take the entire result set and generate a complete object graph that represents the result set as objects.

The following sections of this topic provide more information about the default mapping capabilities and the two types of output mapper:

- Using the Default Row Mapper
- Defining Custom Row Mappings
- Creating Result Set Mappers

For information about using output mappers with an accessor, see <u>Creating and Using Accessors</u> and <u>Executing Queries without Creating an Accessor</u>.

■ Using the Default Row Mapper

■ Defining Custom Row Mappings

■ Creating Result Set Mappers

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Creating and Using Accessors



Accessors execute the query you specify using the parameter values and a parameter mapper (if provided), and transform the result into a series of objects using the output mapper you specify. You can create an accessor by calling a method on your chosen implementation of the **Database** class, such as **SqlDatabase** or **OracleDatabase**, or you can create an accessor directly using the **new** operator or through dependency injection.

✓ Note:

Because of the overhead associated with setting up Accessors and their associated mappings, you gain improved performance by creating the accessor in advance, maintaining a reference to it, and reusing the instance. Using this technique you only incur the setup cost once instead of every time you call the database.

The difference between using the default mappers and providing your own implementation of the mapper interfaces, **IRowMapper** and **IResultSetMapper**, is that when providing your own implementation you can get better performance through special case code, or do more sophisticated data transformations than the default column-to-property direct mapping. The cost here is the time to implement that special case code and the future maintenance burden.

The Data Access Application Block provides two accessors that you can use to retrieve data as objects. The following sections of this topic describe each one in detail:

- Stored Procedure Accessor
- <u>SQL String Accessor</u>

See the following topics for more information about using accessors in your applications:

- <u>Defining Parameter Mappers</u>
- Building Output Mappers

- Executing Queries without Creating an Accessor
- Executing Accessor Queries Asynchronously
- Additional Information for Accessors and Client-side Queries

■ Stored Procedure Accessor

■ SQL String Accessor

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Executing Queries without Creating an Accessor



Instead of creating an accessor first, and then executing it, you can use the methods of the **Database** class to create an accessor and execute it as one operation. You can do this with both the **SprocAccessor** and **SqlStringAccessor**. The main difference between this approach and calling the **Execute** method explicitly on an existing accessor is that you must also pass any required parameters into the **ExecuteSprocAccessor** method. The following sections show the technique for the two types of accessor:

- Stored Procedure Accessor
- SQL String Accessor

See the following topics for more information about using accessors in your applications:

- Defining Parameter Mappers
- Building Output Mappers
- Creating and Using Accessors
- Executing Accessor Queries Asynchronously
- Additional Information for Accessors and Client-side Queries

■ Stored Procedure Accessor

■ SQL String Accessor

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Executing Accessor Queries Asynchronously



You can execute accessors asynchronously if you wish, in much the same was as you can execute many of the other methods of the Data Access Application Block. However, this is only possible where the underlying database supports asynchronous operations (in other words, the **Database.SupportsAsync** property is **true**).

The **SprocAccessor** and **SqlStringAccessor** classes expose the **BeginExecute** method, which takes as parameters a reference to a callback method that will execute when the operation ends, a parameter that contains state you want to pass to the callback method, and an object array containing the parameter values the accessor will use when it executes the query. Typically you will pass a reference to the accessor as the asynchronous state parameter so that you can call the **EndExecute** method on it within the callback method.

The accessors expose only a single overload of the **BeginExecute** method. You can create the accessor using any of the techniques described in the topic <u>Creating and Using Accessors</u>.

When the query operation completes, the code in the callback handler executes and you can retrieve the accessor and call the **EndExecute** method to retrieve the **IEnumerable** sequence as the result. The following code shows an example of using asynchronous operations with an accessor. Notice that the connection string specifies a connection that supports asynchronous operations. In addition, keep in mind that the connection is not closed until the entire list is processed. You must force complete evaluation of the result sequence to extract all of the data when you call the **EndExecute** method. For example, call the **ToList** method, as shown in the following code extract.

```
C#
String connectionString
    = @"server=(local); database=Northwind; Integrated S
SqlDatabase db = new SqlDatabase(connectionString);
DbCommand cmd = db.GetStoredProcCommand("Some Procedur
try
```

```
// Create the accessor. This example uses the simple
 var accessor = db.CreateSprocAccessor<Customer>("Tor
 // Execute the accessor asynchronously, passing in t
 // the existing accessor as the AsyncState, and the
 IAsyncResult async = accessor.BeginExecute(MyEndExec
}
catch
  // ...
 // handle any execution initiation errors here
// callback handler that executes when call completes
public void MyEndExecuteCallback(IAsyncResult async)
{
 try
  {
    // obtain the results from the accessor
    DataAccessor accessor = async.AsyncState as DataAc
    var customers = accessor.EndExecute(async).ToList(
    // ...
    // use the results here
    // ...
  catch
    // ...
   // handle any execution completion errors here
}
```

Visual Basic

Dim connectionString As String _

```
= "server=(local); database=Northwind; Integrated Sε
Dim db As New SqlDatabase(connectionString)
Dim cmd As DbCommand = db.GetStoredProcCommand("Some F
Try
  ' Create the accessor. This example uses the simples
  Dim accessor = db.CreateSprocAccessor(Of Customer)("
  ' Execute the accessor asynchronously, passing in th
  ' the existing accessor as the AsyncState, and the r
  Dim async As IAsyncResult _
      = accessor.BeginExecute(AddressOf MyEndExecuteCa
Catch
  ' handle any execution initiation errors here
End Try
' callback handler that executes when call completes
Public Sub MyEndExecuteCallback(async As IAsyncResult)
  Try
      obtain the results from the accessor
    Dim accessor As DataAccessor = TryCast(async.Async
    Dim customers = accessor.EndExecute(async).ToList(
    ' use the results here
  Catch
    ' handle any execution completion errors here
  End Try
End Sub
```

An alternative approach to coding the callback is to use a lambda function declared within the call to the **BeginExecute** method of the Accessor.

✓ Note:

There are some limitations on using asynchronous data operations, and several issues you should be aware of. For more details, see <u>Using the Asynchronous Data Access Methods</u>.

In addition, when you execute an accessor asynchronously, you can only iterate over the result set once. If you attempt to iterate over it again, the block will raise an exception. For this reason, you may need to use an approach such as the **ToList** method to force complete evaluation of the result sequence.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Additional Information for Accessors and Client-side Queries



This topic describes some additional factors and issues related to using accessors to retrieve data as a sequence of objects. It covers the following:

- Creating Accessors through Dependency Injection
- Accessing and Iterating the Results Set
- Querying the Results Set Using LINQ
- Deferred Loading and Multiple Iterations of the Result Set

■ Creating Accessors through Dependency Injection

■ Accessing and Iterating the Results Set

■ Querying the Results Set Using LINQ

■ Deferred Loading and Multiple Iterations of the Result Set

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Creating Portable Database Applications



There are issues that you must consider if your application must work with multiple database types.

■ Working with Oracle Databases

■ Suggestions for Creating Portable Database Applications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Handling Exceptions
```

Strategies for handling exceptions are essential in any enterprise application. The following information will help you incorporate the Data Access Application Block into your approach to managing exceptions:

- The application block uses configuration information, which may result in configuration-related exceptions.
- The **Database** methods use both ADO.NET and the underlying database provider. Exceptions thrown by ADO.NET are caught by the Data Access Application Block for instrumentation purposes, and then they are re-thrown.
- Adequately processing an exception often requires access to the specific exception type. You can include a **catch** statement for a specific database provider exception such as **SqlException**. However, database provider—specific exception types are not portable across different providers.

If you use **ExecuteReader** within a **try** block, you should add a **finally** statement and close the returned **DataReader** object, as shown in the following example. It assumes that you have resolved the **Database** class you require and stored a reference in the variable named **db**. For more information on instantiating objects, see <u>Creating and Referencing Enterprise Library Objects</u>.

```
DbCommand cmd = db.GetStoredProcCommand("GetProductsBy

IDataReader reader = null;

try
{
    //...
    reader = db.ExecuteReader(cmd);
}
catch(Exception ex)
{
    // Process exception
}
```

```
finally
{
  if (reader != null)
    reader.Close();
}

Visual Basic

Dim cmd As DbCommand = db.GetStoredProcCommand("GetProc Dim reader As IDataReader = Nothing

Try
  ' ...
  reader = db.ExecuteReader(cmd)
Catch ex As Exception
```

Alternatively, you can include the **using** statement to dispose of the **DataReader** object, which causes it to close, as shown in the following example.

' Process exception

reader.Close()

If (Not reader Is Nothing) Then

Finally

End Try

Visual Basic

End If

```
C#

DbCommand cmd = db.GetStoredProcCommand("GetProductsBy
using (IDataReader reader = db.ExecuteReader(cmd))
{
    // Process results
}
```

```
Dim cmd As DbCommand = db.GetStoredProcCommand("GetProus Using reader As IDataReader = db.ExecuteReader(cmd)
   ' Process results
End Using
```

For design and implementation guidelines for exception management in .NET, see the <u>Design Guidelines for Exceptions</u>.

■ Handling Asynchronous Data Access Exceptions

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Handling Parameters



Most stored procedures accept parameters whose values are either used as input to the stored procedure or are set during output. As with ADO.NET, the Data Access Application Block allows developers to explicitly specify all of the attributes of a parameter. These attributes can include direction, data type, and length. This approach is named *explicit parameter handling*. However, as a convenience, you can specify only the values when using input parameters. In this case, the application block will look up and supply the parameter attributes. This approach is named *parameter discovery*.

■ Explicit Parameter Handling

■ Using Column Values as Parameter Inputs

■ Parameter Discovery

■ Optional Parameters

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Key Scenarios



This topic describes the most common situations developers must address when accessing a database. Each scenario explains the task, describes a real-world situation where such a task might arise, and includes code demonstrating how to use the Data Access Application Block to complete the task. The scenarios are as follows:

- <u>Using a DbDataReader to Retrieve Multiple Rows</u>. This scenario illustrates how you can use the **ExecuteReader** method to retrieve multiple rows of data from a database for display in tabulated form—without explicitly caching the data, using a **DataSet** object to manipulate it, or passing it to other components within your application. In other words, it illustrates how to display the results as quickly as possible.
- <u>Using a DataSet to Retrieve Multiple Rows</u>. This scenario illustrates how you can use the **ExecuteDataSet** method to pass data between the components and the tiers of a multi-tier application. The data consists of one or more data tables and, optionally, the relationships that link the tables together.
- Executing a Command and Accessing Output Parameters. This scenario illustrates how you can use the **ExecuteNonQuery** method to retrieve a single row that contains multiple column values.
- Executing a Command and Accessing a Single Item Result. This scenario illustrates how you can use the **ExecuteScalar** method to perform a single-item lookup.
- Performing Multiple Updates Within a Transaction. This scenario illustrates how you can use the **ExecuteNonQuery** method from within a transaction to perform multiple operations against a database, where it is essential that either all operations succeed or none succeed.
- <u>Using a DataSet to Update a Database</u>. This scenario illustrates how, after changing a **DataSet** object, you can use the **UpdateDataSet** method to update the database and make your changes permanent.
- Retrieving Multiple Rows as XML. This scenario illustrates how you can use the **ExecuteXmlReader** method to retrieve data from a SQL Server and have that data returned in XML format.
- Retrieving Data as Objects. This scenario demonstrates how you can use

- the data accessors included in the block to retrieve data as a sequence of objects of a specified type from the data store.
- Performing Asynchronous Data Access. This scenario demonstrates how you can use the asynchronous version of the **ExecuteReader** method to read data from a data store asynchronously, and then access the results using a callback that indicates when the method completes.

This topic helps you implement your approach by using the Data Access Application Block. It does not help you choose the correct approach for your particular situation (for example, it does not help you choose between a **DataSet** and a **DbDataReader**). For guidance on approaches to data access, see the following Microsoft patterns & practices guides:

- .NET Data Access Architecture Guide
- <u>INFO</u>: <u>Microsoft Guide for Designing Data Tier Components and Passing Data Through Tiers</u>
- Improving .NET Application Performance and Scalability

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Using a DbDataReader to Retrieve Multiple Rows



A common database task is to retrieve and display information. For example, an online retail application may need to display a list of products within a specified category.

■ Typical Goals

■ Solution

■ Using ExecuteDataReader

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Using a DataSet to Retrieve Multiple Rows



In a multi-tier system, you may need to pass data from a data access component to a middle-tier business component. The data is retrieved from the database and sent back, through the data access layer, to the business layer. The information is contained in the **DataSet** object.

■ Typical Goals

■ Solution

■ Using ExecuteDataSet

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Executing a Command and Accessing Output Parameters



A common database task is to retrieve specific, multiple-column values. For example, in a Web-based online retail application, you may want to retrieve full product details for a certain product in response to a user request.

■ Typical Goals

■ Solution

■ Using ExecuteNonQuery

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Executing a Command and Accessing a Single Item Result



There are many situations in which you have to perform a single-item lookup. For example, an online retailer may want to use a product ID to retrieve a product name or use a customer ID to retrieve a credit rating.

■ Typical Goals

■ Solution

■ Using ExecuteScalar

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Performing Multiple Updates Within a Transaction



When an application executes multiple operations against a database, a common requirement is that all of the operations must succeed or the database must roll back to its original state (that is, its state before the operations began). This allor-nothing requirement is referred to as a *transaction*. Transactions ensure the integrity of a database system's state. For example, in a classic banking scenario, an application must debit one account and credit another with a particular amount of money. For proper accounting, it is essential that either both operations succeed or neither operation succeeds. This means that both operations should be performed in the context of a single transaction.

■ Typical Goals

■ Solution

■ Using ExecuteNonQuery in a Transaction

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Using a DataSet to Update a Database**



Databases must be periodically updated with new information. For example, in a Web-based online retail application, you may want to add a new customer to the database, modify the name associated with a customer ID, or delete a customer record entirely.

■ Typical Goals

■ Solution

■ Using UpdateDataSet

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Retrieving Multiple Rows as XML



An example of where you may want to use XML data is within an e-commerce application that allows clients to request a product catalog that is in XML format.

■ Typical Goals

■ Solution

■ Using ExecuteXmlReader

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Retrieving Data as Objects**



This scenario demonstrates one of the ways that you can use the data accessors provided with the block to read data from a data store and return it as a sequence of objects of the type you specify. The Data Access Application Block provides two types of accessors, for stored procedures or for use with SQL statements, in addition to a range of classes that help you to map parameters to the query and map the returned data to the type of object you require. For information about these classes, see Returning Data as Objects for Client Side Querying.

■ Typical Goals

■ Solution

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Performing Asynchronous Data Access



This scenario demonstrates how you can use the asynchronous version of the **ExecuteReader** method to read data from a data store asynchronously, and then access the results using a callback that indicates when the method completes. You do this using the **BeginExecutereader** and **EndExecuteReader** methods of the **Database** class.

✓ Note:

Asynchronous data access is not supported by all ADO.NET data providers. The **Database** class exposes a Boolean property named **SupportsAsync** that you can test to check at run time if asynchronous operations are supported. If this property returns false, any asynchronous methods you call will throw an **InvalidOperationException**. In Enterprise Library 5.0, the only database type that supports asynchronous operation is the **SqlDatabase** class.

■ Typical Goals

■ Solution

■ Using BeginExecuteReader and EndExecuteReader with a Callback

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Design of the Data Access Application Block



The Data Access Application Block includes the following features:

- A simple and efficient way of working with different database systems (see Designing for Simplified Data Access)
- A way of developing database-agnostic applications (see <u>Designing for Database-Agnostic Applications</u>)
- An easy way to adjust and validate the database configuration settings

■ Design Goals

■ Design Highlights

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Designing for Simplified Data Access



Developers face many implementation choices and requirements when they build data access solutions. They must access the data in a variety of ways, and their solutions must work with different types of databases, each of which handles data access differently. As a result, developers may find themselves duplicating code that performs common tasks, such as managing connections and assigning parameters to commands.

Another challenge is maintaining a consistent approach in how data access operations are implemented. It may be necessary to maintain this consistency across single projects, multiple projects, or enterprise-scale solutions. Uniform methods of data access make the code easier to understand, more predictable, and easier to maintain.

The Data Access Application Block simplifies data access by encapsulating the logic that performs common database operations. These methods also handle common housekeeping tasks such as opening and closing connections. They are database-agnostic, which means that they work with SQL Server and Oracle databases and do not require modification to do so. Applications written for one type of database use the same methods as those written for another type of database. This means that applications are consistent in the ways that they access data. In addition, the **GenericDatabase** class supports many of these same features across ADO.NET data providers.

■ Design Implications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Designing for Database-Agnostic Applications



The Data Access Application Block provides an extensible framework for supporting multiple types of relational databases. Applications that use the application block are portable across different database systems.

There are a number of general data access tools—such as Open Database Connectivity (ODBC) or OLE DB—that can provide access to a variety of data sources. One drawback to these tools is that how they are used depends on the target database. That means that programmers need to understand various programming models to access different database types. Moving an application to a different database could require a significant amount of recoding.

Another drawback to the ODBC or OLE DB approach is that performance may suffer. Generic data providers are slower than those optimized for a particular data source. The Data Access Application Block provides an implementation that features both portability and optimized performance.

Abstraction of the Database System

The Data Access Application Block builds on the capabilities provided by ADO.NET to create a database-agnostic provider model. The following are some of the features it provides:

- It standardizes parameter names. For example, it supplies the "@" character for SQL parameter names.
- It configures the stored procedure package name mapping for Oracle databases.
- It uses ADO.NET static methods for SQL Server and Oracle to support agnostic parameter discovery.
- It adds the cursor parameter for results returned by an Oracle stored procedure.

The majority of the data access methods are available through the abstract **Database** class. Client code can refer to these methods in their code regardless of the actual **Database**-derived object used. For example, the following code shows how to use the **ExecuteDataSet** method.

```
C#

Database db;
...
db.ExecuteDataSet(cmd);

Visual Basic

Dim db As Database
...
db.ExecuteDataSet(cmd)
```

The block creates the specific **Database**-derived object. It returns an object of type **Database**, thus allowing the client code to remain generic regarding the actual database type returned.

The methods available on the **Database** class require information about the command to be executed as well as any associated parameters. Different

database systems handle commands and parameters in different ways. **Database-**derived classes provide methods that accept parameter information; the specific database systems provide their own derived implementations to handle parameter parameters.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending and Modifying the Data Access Application Block



In its original state, the Data Access Application Block works well for typical data access scenarios. However, there may be times when you need to customize some of the application block's behavior to better suit your application's requirements. You can extend the Data Access Application Block using the built-in extension points. You can also modify the application block by making changes to its source code. For more details, see the following topics:

- Extending the Data Access Application Block
- Extending and Modifying Enterprise Library

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending the Data Access Application Block



You extend the application block through designated extension points. Typically, these are custom classes that you must write. The custom classes implement a particular interface or derive from an abstract class. Because these custom classes exist in your application space, you do not have to modify or rebuild the application block. Instead, you use configuration settings to designate your extensions.

Currently, you can extend the application block by adding a new database type. You would do this if your application needs database-specific features from a database other than the SQL Server database or the Oracle database. You only need to extend the application block if the application block's **GenericDatabase** class and the ADO.NET data provider for the database system do not provide the features you require. For example, the **GenericDatabase** object does not support special parameter prefixes for stored procedures or invoke methods that are not on the **DbCommand** class or the **DbConnection** class. In addition, the **GenericDatabase** class does not support parameter discovery.

You could extend the application block by creating a new database class that supports features not exposed by the ADO.NET data provider. For example, your database class could support parameter discovery. You could also extend the application block to allow your client code to remain database-agnostic through type conversions or by managing SQL syntax conversions. Building a custom database class allows your application to support the entire Data Access Application Block API set and to be more compatible with the syntaxes of other databases. To learn more, see Adding a New Application Block Database Provider.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Adding a New Application Block Database Provider



If the relational database system you are using does not already have an application block database provider, you can create your own. This is only necessary if the application block's **GenericDatabase** class does not meet the requirements of your application. (This assumes that there is an ADO.NET **DbProviderFactory** type for the database system you are using.) To create a new database provider, you must create a new database class that derives from the **Database** class. Additionally, if you want your client code to remain database-agnostic, you might have to write additional code to perform such tasks as type conversion.

■ Creating a New Database Class

Configuring	Your Ap	plication	to Use	the Ne	w Provide	r

■ Recommendations for Creating a Database Provider

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Deployment and Operations



Two of an administrator's main tasks will be to see that the initial deployment of the Data Access Application Block is planned and managed and that subsequent updates are deployed with minimal impact to existing applications that use the application block. For details of deploying and updating Enterprise Library and the application blocks, see <u>Deploying Enterprise Library</u>.

In addition, administrators must decide whether they want to use the instrumentation exposed by the application block. For details of how to enable and disable instrumentation, see Enabling Instrumentation. For information about the instrumentation contained within the Data Access Application Block, see the following topics:

- Data Access Application Block Performance Counters
- Data Access Application Block Event Log Entries

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Data Access Application Block Performance Counters



The following table describes the Data Access Application Block performance counters.

Performance Counter Name	Description
Commands Executed/sec	The rate at which database commands were executed.
Commands Failed/sec	The rate at which database commands failed to execute.
Connections Failed/sec	The rate at which database connections failed to open.
Connections Opened/sec	The rate at which database connections were opened.
Total Commands Executed	The total number of database commands that were executed.
Total Commands Failed	The total number of database commands that failed to execute.
Total Connections Failed	The total number of database connections that failed to open.
Total Connections Opened	The total number of database connections that were opened.

A rate counter samples an increasing count of events over time and divides the values by the change in time to display a rate of activity. For more information about performance counters, see Overview of Performance Monitoring on TechNet.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to

pagdoc@microsoft.com.

Copyright $\ensuremath{\text{@}}$ 2010 by Microsoft Corporation. All rights reserved.

Data Access Application Block Event Log Entries



The Data Access Application Block is instrumented to log entries to the application event log for a variety of events. This topic lists the Data Access Application Block event log entries.

■ Connection Failed Event

■ Command Executed Event

■ Command Failed Event

■ Data Configuration Failure Event

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Exception Handling Application Block



The Enterprise Library Exception Handling Application Block helps developers and policy makers implement common design patterns and create a consistent strategy for processing exceptions that occur in all architectural layers of an enterprise application. It is designed to support the typical code contained in **catch** statements in application components. Instead of repeating this code (such as code that logs exception information) in identical catch blocks throughout an application, the Exception Handling Application Block allows developers to encapsulate this logic as reusable exception handlers. The Exception Handling Application Block includes four exception handlers:

- **Wrap handler**. This exception handler wraps one exception around another.
- **Replace handler.** This exception handler replaces one exception with another.
- **Logging handler**. This exception handler formats exception information, such as the message and the stack trace. Then the logging handler passes this information to the Enterprise Library Logging Application Block so that it can be published.
- **Fault Contract exception handler**. This exception handler is designed for use at Windows® Communication Foundation (WCF) service boundaries, and generates a new Fault Contract from the exception.

You can extend the Exception Handling Application Block by implementing custom handlers, and administrators can manage the configuration of the block to, for example, turn on additional debugging instrumentation or change the behavior of the block in line with changes to business requirements. The configuration can even be managed using Group Policy tools.

This section includes the following topics that will help you to understand and use the Exception Handling Application Block:

• What Does the Exception Handling Application Block Do? This topic provides a brief overview that will help you to understand what the block can do, and explains some of the concepts and features it incorporates. It also provides a simple example of the way that you can write code to use the block.

- When Should I Use the Exception Handling Application Block? This topic will help you to decide if the block is suitable for your requirements. It explains the benefits of using the block, and any alternative techniques you may consider. It also provides details of any limitations of the block that may affect your decision to use it.
- Developing Applications Using the Exception Handling Application Block. This topic explains how to configure the Exception Handling Application Block, how to add the block to your applications, how to determine appropriate exception handling policies, how to specify different handling actions, and how to send an exception to the Exception Handling Block.
- <u>Key Scenarios</u>. This topic shows different ways to use the Exception Handling Application Block in your own applications.
- Design of the Exception Handling Application Block. This topic explains the decisions that went into designing the block and the rationale behind those decisions.
- Extending and Modifying the Exception Handling Application Block.
 This topic explains how to extend the application block by adding custom handlers and formatters. It also gives some advice about how to modify the source code.
- <u>Deployment and Operations</u>. This topic explains how to deploy and update the block assemblies.

More Information

For more information, see the following Microsoft® patterns & practices guides:

- Application Architecture for .NET: Designing Applications and Services
- Design Guidelines for Exceptions

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

What Does the Exception Handling Application Block Do?



A robust and well-planned exception handling strategy is a vital feature of your application design and implementation that will help you avoid risks such as exposing error messages containing sensitive information, or leaving your application and systems in an inconsistent state and open to attack when an error occurs.

An exception handling strategy consists of a series of policies that define how you will present clear and appropriate messages to users and how you can provide assistance for operators, administrators, and support staff. A comprehensive exception-handling strategy will usually:

- Notify the user with a friendly message
- Store details of the exception in a production log or other repository
- Alert the customer service team to the error
- Assist support staff in cross-referencing the exception and tracing the cause

The Exception Handling Application Block can help you to define and create consistent exception management strategies by implementing three well-known design patterns:

- **Exception Shielding**. This pattern ensures that your application does not leak sensitive information, no matter what run-time or system event may occur to interrupt normal operation. And on a more granular level, it can prevent your assets from being revealed across layer, tier, process, or service boundaries.
- **Exception Logging**. This pattern can help you to diagnose and troubleshoot errors, audit user actions, and track malicious activity and security issues by logging details of exceptions and errors that occur.
- **Exception Translation**. This pattern describes how you can wrap exceptions within other exceptions specific to a layer to ensure that they actually reflect user or code actions within the layer at that time, and not some miscellaneous details that may not be useful.

The Exception Handling Application Block lets you associate exception types with named policies. You do this by using the configuration tools. Policies specify the exception handlers that execute when the block processes a

particular exception type. Exception handlers are .NET classes that encapsulate exception handling logic and implement the Exception Handling Application Block interface named **IExceptionHandler**.

You can chain these handlers together so that a series of them execute when the associated exception type is handled. The following are some examples of named policies and descriptions of what they might provide:

- **Base policy**. This policy logs the exception and re-throws the original exception.
- **Secure policy**. This policy logs the exception, replaces the original exception with a custom exception, and throws the new exception.
- **Expressive policy**. This policy wraps the original exception inside another exception and throws the new exception.

The following schematic illustrates examples of cross-layer and single-application component exception handling.

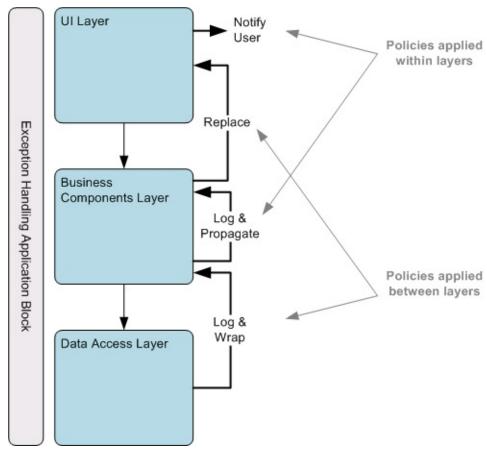


Figure 1
Examples of exception handling policies

In this example, exceptions that occur in the data access layer are logged and then wrapped inside another exception that provides more meaningful information to the calling layer. Within the business component layer, the exceptions are logged before they are propagated. Any exceptions that occur in the business component layer and that contain sensitive information are replaced with exceptions that no longer contain this information. These are sent to the user interface (UI) layer and displayed to the user.

For information about how to develop exception management strategies, see Determining Appropriate Exception Policies and Actions.

■ Example Application Code

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

When Should I Use the Exception Handling Application Block?



The Exception Handling Application Block is best used in situations that require uniform and flexible procedures for handling exceptions. For example, you might want consistent exception handling procedures for all components in a particular tier of an application's architecture. In addition, because of changing security or other operational issues, you might want the ability to change policies as needed, without requiring changes to the application source code. The Exception Handling Application Block, in conjunction with the Enterprise Library configuration tools, lets you accomplish both tasks.

For example, you could use the configuration tools to define a policy that uses handlers to replace exceptions that contain sensitive information with versions that do not include that information. The block then implements this policy across the components that contain code that specifies this policy should be used.

The Exception Handling Application Block is not limited to cross-tier applications. It can also be used within a particular application. For example, you can define policies that log exception information or display exception information to the user.

In either case, policies are configured without changing the application's code. This makes them easy to maintain or change when new situations occur. Note that, in all cases, you should use the block to perform only those tasks that are specific to exception handling and that do not intersect with the application's business logic. For example, you can remove the handlers that log an exception or wrap one exception in another without affecting such basic capabilities as updating a customer database.

■ Scenarios for the Exception Handling Block

Benefits	of the	Exception	Handling	App	lication	Block	<

■ Limitations of the Exception Handling Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

Developing Applications Using the Exception Handling Application Block



This topic describes how to develop applications by using the Exception Handling Application Block. It first explains how to configure the block and reference it in your applications. Next, it describes how to develop exception-handling strategies for your applications, and explains how to use the block in specific scenarios such as logging and propagating exceptions. This section includes the following topics:

- Entering Configuration Information
- Adding Application Code
- Determining Appropriate Exception Policies and Actions
- Specifying Different Handling Actions Based on Exception Type and Policy
- Sending an Exception to the Exception Handling Application Block
- Handling and Throwing Exceptions

All application blocks ship as binary assemblies and as source code. If you want to use the source code, you must compile it. To learn how to compile the Enterprise Library source code, see <u>Building Enterprise Library from the Source Code</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Entering Configuration Information



These procedures explain how to configure the Exception Handling Application Block. For information about using the configuration tools, see <u>Using the Configuration Tools</u>. For details of the schema for the Exception Handling Application Block configuration, see <u>Source Schema for the Exception Handling Application Block</u>. You can also configure the block in code by using an alternate configuration source. For more information, see <u>Advanced Configuration Scenarios</u> and <u>Using the Fluent Configuration API</u>.

To add the Exception Handling Application Block

- 1. Open your configuration file in the configuration editor. For more information, see <u>Configuring Enterprise Library</u>.
- 2. Open the **Blocks** menu and then click **Add Exception Handling Settings**.
- 3. This creates an **Exception Handling Settings** section with one exception handling policy that by default is called **Policy**, and a single item named **All Exceptions** in the **Exception Types** column that defines all exceptions of type **System.Exception** or that inherit the **System.Exception** class.
- 4. To edit the properties of the **Exception Handling Settings** section, click the chevron arrow the right of the **Exception Handling Settings** heading.
- 5. (Optional) If you want to encrypt the configuration, make a selection from the Protection Provider drop-down list. You can select the RsaProtectedConfigurationProvider or the DataProtectedConfigurationProvider. See Encrypting Configuration Data for information about the restrictions on using the RsaProtectedConfigurationProvider.
- 6. (Optional) If you want to run your application in partial trust mode, change the **Require Permission** property to **False**. The default is **True**.

To configure an exception handling policy

1. To add an exception handling policy, click the plus sign icon in the **Policies** column and click **Add Policy**. This adds a new exception handling policy item. Click the expander arrow on the left of the policy

- heading if the properties are not visible.
- 2. (Optional) Rename the policy. In the **Name** text box, type the required policy name.
- 3. To add an exception type to a policy, right-click the policy item and click **Add Exception Type** to display the type selector dialog.
- 4. Select the exception type in the type selector dialog box. To filter the list, type the filter string in the **Filter** edit box. For example, type "string" to filter for all classes containing the word **string**. If the type you want is not listed, click **Add from File** or **Add from GAC** (from the global assembly cache) and find the assembly containing the type you want, and then click **OK**. This creates the new policy and a default **All Exceptions** item.
- 5. Click the expander arrow to the left on the new **All Exceptions** node in the **Exception Types** column to display the properties if they are not visible.
- 6. Select the **Post handling action** you want. The post-handling action determines what action will occur after the exception handling chain completes. By default, the post-handling action is set to **NotifyRethrow**. In general, when you use the **Process** method, you will configure the exception handling policy with a post-handling action of **ThrowNewException** unless you want your code to continue to execute after the block handles the exception. Valid values are the following:
 - NotifyRethrow. The block executes all handlers for this exception and returns true to the application at the point at which the policy was invoked. Applications checking this value re-throw the original exception. However, if you use the Process method you cannot detect the value returned by the Exception Handling Application Block. Internally, the Process method calls the HandleException method, and throws the exception if this method returns true. Therefore, typically your code will just throw any exception that is raised.
 - ThrowNewException. The block executes all handlers for this exception and throws the exception that exists after the final handler runs. However, if you use the overload of the HandleException method that takes an out parameter that returns the final exception from the handlers, it does not

- automatically throw the exception; you must do this in your code.
- None. The block executes all handlers for this exception and returns false to the application at the point at which the policy method was invoked. Applications checking this value resume execution.
- 7. To add additional exception types to the policy, repeat steps 3 to 6.
- 8. To add an exception handler, right click the **All Exceptions** item or an exception item you added, point to **Add Handlers**, and then click the exception handler type that you want:
 - **Replace** exception handler. This exception handler replaces one exception with another.
 - **Wrap** exception handler. This exception handler wraps the exception that occurred with another exception.
 - **Logging** exception handler. This exception handler formats exception information and uses the Logging Application Block to log exception information. The Logging Application Block is automatically added to the application configuration when you select a logging handler. For more information, see The Logging Application Block.
 - **Fault Contract** exception handler. This exception handler is designed for use at Windows Communication Foundation (WCF) service boundaries, and generates a new Fault Contract from the exception.
 - Custom exception handler. This option allows you to configure custom exception handlers. A custom handler is a type that implements the IExceptionHandler interface and has a ConfigurationElementType of CustomHandlerData. Click Add from File in the type selector to add a custom handler that is located in a separate assembly.
- 9. To add additional exception handlers to the policy, repeat step 8.
- 10. If required, change the order of the exception handlers by right-clicking the handler item heading and clicking either **Move Up** or **Move Down**. Handlers are executed in the order you specify. Typically, you will place a Logging handler first in the list, followed by a Wrap handler or a Replace Handler.
- 11. See the following section of this topic for details of how to configure

exception handlers.

■ Configuring Exception Handlers

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

This topic lists the XML elements and attributes used to configure the Exception Handling Application Block. You can manually edit the XML data, but the Enterprise Library Configuration Console greatly simplifies this task. If you choose to manually edit the XML, use the schema information contained in this topic.

The configuration file has the following section-handler declaration.

The section-handler declaration contains the name of the configuration settings section and the name of the section-handler class that processes configuration data in that section. The name of the configuration settings section is **exceptionHandling**. The name of the section-handler class is **ExceptionHandlingSettings**. This class is in the **Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Configuration** namespace.

■ exceptionHandling Element

■ exceptionPolicies Child Element

■ exceptionTypes Child Element

■ exceptionHandlers Child Element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Adding Application Code



The Exception Handling Application Block is designed to support the most common scenarios for handling exceptions. When you add your application code, refer to the scenarios in Key Scenarios and select the ones that best suit your situation. Use the code that accompanies the scenario either as it is shown here or adapt it as required.

To prepare your application to use the Exception Handling Application Block

- Add a reference to the Exception Handling Application Block assembly. In Visual Studio®, right-click your project node in Solution Explorer, and then click Add Reference. Click the Browse tab, select the Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.dll assembly, and then click OK.
- 2. Following the same procedure, set a reference to the following assemblies,
 - Microsoft.Practices.EnterpriseLibrary.Common.dll
 - Microsoft.Practices.ServiceLocation.dll
 - Microsoft.Practices.Unity.dll
 - Microsoft.Practices.Unity.Interception.dll
- 3. If you configure your application to use the Logging Exception Handler, set a reference to Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging.dll and the required Logging Application Block assemblies. For information about the Logging Application Block assemblies, see Adding Application Code in the documentation for The Logging Application Block.
- 4. (Optional) To use elements from the Exception Handling Application Block without fully qualifying the element reference, you can add the following **using** statement (C#) or **Imports** statement (Visual Basic®) to the top of your source code file.

C#

using Microsoft.Practices.EnterpriseLibrary.Exce

Visual Basic

Imports Microsoft.Practices.EnterpriseLibrary.Ex

✓ Note:

For Visual Basic projects, you can also use the **References** page of the Project Designer to manage references and imported namespaces. To access the **References** page, select a project node in Solution Explorer, and then click **Properties** on the **Project** menu. When the Project Designer appears, click the **References** tab.

Next, add the application code. Generally, code that uses the Exception Handling Application Block must complete the following steps:

- Obtain an instance of the **ExceptionManager** class. For more details, see <u>Creating Application Block Objects</u>.
- Catch an exception.
- Process an exception policy.
- Re-throw the original exception where appropriate.

The following topics explain how to incorporate these steps into an application:

- <u>Determining Appropriate Exception Policies and Actions</u>
- Specifying Different Handling Actions Based on Exception Type and Policy
- Sending an Exception to the Exception Handling Application Block
- Handling and Throwing Exceptions

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> <u>Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Determining Appropriate Exception Policies and Actions



This information provides an overview of managing exceptions in your applications. For complete design and implementation guidelines for creating exception management systems that use .NET technologies, see Design Guidelines for Exceptions on MSDN®.

To build successful and flexible applications that can be maintained and supported easily, you must use an appropriate strategy for exception management. You must design your system to make sure that it can do the following:

- Detect exceptions
- Log and report information

The following topics provide useful advice about how you should develop your exception management strategies:

- Generate Events That Can Be Monitored Externally to Help System Operation
- Exception Handling Process and the Exception Handling Application Block
- When to Catch Exceptions
- Exception Propagation
- <u>Hiding Exception Information</u>
- Exception Notification
- Planning for Exception Handling

■ Generate Events That Can Be Monitored Externally to Help System Operation

■ Exception Handling Process and the Exception Handling Application Block

■ When to Catch Exceptions

■ Exception Propagation

■ Hiding Exception Information

■ Exception Notification

■ Planning for Exception Handling

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Specifying Different Handling Actions Based on Exception Type and Policy



The Exception Handling Application Block separates the definition of how an exception should be processed (which is the exception policy) from the application code that uses the block to handle exceptions. You use the configuration tools to create and name policies.

By using exception policies, the application behavior that occurs in response to an exception can be modified without changing the application code.

■ Configuring Exception Policies

■ Configuring Exception Types

• Understanding Exception Handlers

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Sending an Exception to the Exception Handling Application Block



Interaction between application code and the Exception Handling Application Block occurs when the application code catches an exception and sends it to the block to be handled. Application developers do not have to know how exceptions will be handled because they have to specify only the name of the relevant exception policy.

There are three main scenarios in which you may need to handle exceptions. The most common is the first of these, but this topic explains how you can use the block to implement all three scenarios. The scenarios are:

- Handling All Exceptions in a Catch Section
- Handling Specific Exceptions in a Catch Section
- Executing Code Before or After Handling an Exception

■ Handling All Exceptions in a Catch Section

	Handling	Specific	Exceptions	in a	Catch	Section
--	----------	-----------------	-------------------	------	-------	---------

■ Executing Code Before or After Handling an Exception

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Handling and Throwing Exceptions



When an exception occurs, it passes up the stack and each catch block can potentially handle it. The order of catch statements is important. Put catch blocks that target specific exceptions before a general catch block. Otherwise, the compiler might issue an error. The common language runtime (CLR) determines the proper catch block by matching the type of the exception to the name of the exception specified in the catch block. If there is no specific catch block, a general catch block, if it exists, handles the exception.

■ Debugging and Exception Propagation

■ Handling Specific Exceptions

■ User-Filtered Exceptions

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Key Scenarios



This topic describes the most common situations that developers must address when handling exceptions. Each scenario explains the task, describes a real-world situation where such a task might occur, and includes code that demonstrates how to use the Exception Handling Application Block to complete the task. The scenarios are the following:

- <u>Logging an Exception</u>. This scenario demonstrates how to use the logging handler to collect exception information, format it, and send it to the Logging Application Block.
- Replacing an Exception. This scenario demonstrates how to use the replace handler to create a new exception of a defined type that replaces the original exception.
- Wrapping an Exception. This scenario demonstrates how to use the wrap handler to create a new exception of a defined type that wraps the original exception with another exception that is more relevant.
- <u>Propagating an Exception</u>. This scenario demonstrates how to propagate an exception in its original state after running a chain of exception handlers.
- <u>Displaying User-Friendly Messages</u>. This scenario demonstrates how to either replace or wrap an exception with one that provides support or instructional information for the user.
- Notifying the User. This scenario demonstrates methods for letting a user know that an error has occurred.
- <u>Assisting Support Staff</u>. This scenario demonstrates how support staff can match a user's error message with the detailed information that is stored in the exception log.

In addition, this section contains the topic <u>Shielding Exceptions at WCF Service</u> <u>Boundaries</u>, which describes how unknown exceptions occurring in Windows Communication Foundation (WCF) services should not be sent to the client application, in order to prevent details of the service implementation from escaping the secure boundary of the service.

■ Exceptions That Occur During Exception Handling

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Logging an Exception



A frequently required exception-handling task is to log the information associated with the exception. The Exception Handling Application Block, in conjunction with the Logging Application Block, lets you log formatted exception information in locations specified in the configuration file. For example, client applications typically log application information in the event log, while a server component of an e-commerce application may log exceptions in a database.

■ Typical Goals

■ Solution

■ Using the Logging Handler

■ Modify Your Application

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Replacing an Exception



A frequently required exception-handling task is to replace the original exception with another exception. For example, if an exception is going to cross a trust boundary, you may not want to send the original exception because it contains sensitive information.

■ Typical Goals

■ Solution

■ Using the Replace Handler

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Wrapping an Exception



A frequently required exception-handling task is wrapping one exception with a different exception. Wrapping an exception creates a new exception of a defined type and sets the original exception as the **InnerException** object of the new exception. Use the wrapping capability in situations where the original exception type must be mapped to a new exception type for use by other tiers within the architecture of the application. You can encapsulate and interpret details of the underlying layer's original exception without losing any details about that exception. You can wrap the original exception either in an existing exception type or in a custom exception type that you create. The following explains a situation when you would want to wrap an exception:

- 1. A business service named Update Customer calls a data layer service.
- 2. The data layer service fails and throws an exception. This could be any one of many exceptions that indicate that the update failed. Some sets of these exceptions indicate that recovery may be possible with a retry (for example, if a record is locked) while others are non-recoverable (for example, if there is a concurrency violation or a dirty record).
- 3. The exception handler maps and wraps these sets of exceptions into two custom exception types, **RecoverableUpdateException** and **FatalUpdateException**.
- 4. The business service handles the exception based on the wrapping type and takes the appropriate action, even though it is insulated from detailed knowledge of the underlying failure.

■ Typical Goals

■ Solution

■ Using the Wrap Handler

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Propagating an Exception



A frequently required exception-handling task is to allow the original exception to propagate up the call stack unchanged. You may want to do this because the handlers only perform actions such as logging that leave the exception unchanged or because other actions, such as wrapping and replacing, have been turned off. For example, a routine within a business logic component may log exceptions at the point where they are detected. It then propagates that exception to the caller for additional handling.

■ Typical Goals

■ Solution

■ Propagating an Exception

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Displaying User-Friendly Messages**



You may want to replace the message in the original exception with a more appropriate, user-friendly message. To do this, you must replace the original exception with another exception that has a more appropriate message associated with it. For example, exceptions that occur in the data access layer of an application can be replaced with an exception of type **System.ApplicationException**. This uses the message, "The application is unable to process your request at this time." This message is then displayed to the user.

■ Typical Goals

■ Solution

■ Displaying User-Friendly Messages

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Notifying the User**



When you use the Exception Handling Application Block, a frequently required task is to notify the user when an exception occurs. (Generally, this should be completed after the message has been changed to one that is suitable for the particular user. For more information, see <u>Displaying User-Friendly Messages</u>.) Depending on the application type, you can do this using a Windows Forms dialog box or a Web page. When an exception cannot be handled within the application, the user must receive a notification that an error occurred, along with some guidance of what he or she should do next.

■ Typical Goals

■ Solution

■ Notifying the User

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Assisting Support Staff



When you use the Exception Handling Application Block, a frequently required task is to allow support staff to access detailed information to assist users when exceptions occur. When an exception occurs that cannot be handled, users are generally shown a friendly error message. Users may have to call support staff, and support staff may need more than the user error message to determine what went wrong and how to fix the problem. This scenario demonstrates how to match the user's error message with the detailed exception log that can be accessed by support staff.

■ Typical Goals

■ Solution

■ Assisting Support Staff

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Shielding Exceptions at WCF Service Boundaries



In Windows Communication Foundation (WCF), to prevent details of the service implementation from escaping the secure boundary of the service, unknown exceptions should not be sent to the client application. This is controlled through the **includeExceptionDetailInFaults** attribute in the **<serviceDebug>** element in the WCF configuration. To enable exception shielding, this attribute must be set to **false**. If not specified in the configuration file, this property is set to **false**.

✓ Note:

The **includeExceptionDetailInFaults** configuration setting is used only for unknown or unhandled exceptions. It does not have any effect on known exceptions, where the operation has a **FaultContract** with the known fault type and the operation throws a **FaultException**<*knownFault*> where *knownFault* is in the fault contract.

Exception shielding helps prevent a Web service from disclosing information about the internal implementation of the service when an exception occurs. The following forces explain why you should use exception shielding:

- Exception details may contain clues that an attacker can use to exploit resources used by the system.
- Information related to anticipated exceptions needs to be returned to the client application.
- Exceptions that occur within a Web service should be logged to support troubleshooting.

Only exceptions that have been sanitized or are safe by design should be returned to the client application. Exceptions that are safe by design do not contain sensitive information in the exception message and they do not contain a detailed stack trace, either of which might reveal sensitive information about the Web service's inner workings. You should use the Exception Shielding pattern to sanitize unsafe exceptions by replacing them with exceptions that are safe by design.

The Exception Handling Application Block includes support for exception shielding at WCF service boundaries. This support consists of the following:

- The **Exception Shielding Attribute**, which is used to associate a named exception handing policy configured in the Exception Handling Application Block with a WCF service operation. For more information, see the following section <u>Using the Exception Shielding Attribute</u>.
- The **Fault Contract Exception Handler**, which converts an exception to a specific type of Fault Contract and maps the desired properties of the exception to the Fault Contract. For more information, see the following section <u>Using the Fault Contract Exception Handler</u>.

■ Using the Exception Shielding Attribute

■ Using the Fault Contract Exception Handler

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Design of the Exception Handling Application Block



The Exception Handling Application Block is designed to achieve the following goals:

- Encapsulate the logic used to perform the most common exception handling tasks into minimal application code.
- Relieve developers of the requirement to write duplicate code and custom code for common exception handling tasks.
- Allow exception handling policies to be changed after they have been deployed and to ensure that changes happen simultaneously and consistently.
- Incorporate best practices for exception handling, as described in the <u>Design Guidelines for Exceptions</u>.

■ Design Highlights

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Designing for Simplified Catch Blocks



Common exception handling behavior, such as logging and replacing exceptions to hide sensitive information, typically requires multiple lines of code. Updating exception-handling behavior to accommodate a change in an exception handling policy usually involves updating multiple files and lines of code. This process can be error-prone, and it is difficult to ensure that policies are updated consistently across all layers of an application.

The Exception Handling Application Block simplifies both the exception handling code and the process of updating that code. It does this by associating exception-handling behavior with policy names such as **Data Access Layer Policy** and **Trust Boundary Policy**. The behaviors represented by policy names are controlled externally, in the configuration for the application. This means that a developer needs to use only two elements to write code in the catch block of an application:

- A call to the **HandleException** or **Process** method that passes the policy name and the exception.
- A check of the return code from the **HandleException** method; if it returns **true**, the original exception should be re-thrown.

■ Design Implications

■ API Support for Policy Names

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Designing for Encapsulation of Behavior in Reusable Handlers



The Exception Handling Application Block helps developers to create exception handlers that represent common exception handling tasks. These handlers, as well as combinations of handlers, can be used by different policies.

■ Design Implications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending and Modifying the Exception Handling Application Block



In its original state, the Exception Handling Application Block works well for typical exception handling scenarios, such as logging an exception message, wrapping one exception with another, or replacing an exception with a different exception. However, there may be times when you have to customize certain behaviors of the block to better suit your application's particular requirements. There are two ways to do this. You can extend the Exception Handling Application Block using the built-in extension points. In addition, you may choose modify the block by making changes to its source code.

You extend the Exception Handling Application Block through designated extension points. Typically, these are custom classes written by you that implement a particular interface or derive from an abstract class. Because these custom classes exist in your application space, you do not have to modify or rebuild the Exception Handling Application Block; instead, you can designate your extensions through configuration settings.

You can extend the block by adding a new type of exception handler or exception formatter. The following table lists the interfaces and base classes that you can use to extend the block.

Custom Provider or Extension	Interface or Base Class
Exception Handler	IExceptionHandler
Exception Formatter	ExceptionFormatter

For more details of how to extend and modify the block and Enterprise Library, see the following topics:

- Adding a New Exception Handler
- Adding a New Exception Formatter
- Creating Custom Providers for Enterprise Library
- Extending and Modifying Enterprise Library

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Adding a New Exception Handler



To add a new exception handler, you must first create a new class that implements the **IExceptionHandler** interface. After you compile the new class into an assembly, you can use the Enterprise Library Configuration Console to add it to the Exception Handling Application Block configuration for your application.

■ Creating a New Exception Handler Class

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Adding a New Exception Formatter
```

To add a new exception formatter, you must create a new exception class that derives from the **ExceptionFormatter** class. Your new class can derive directly from **ExceptionFormatter** or from one of the exception formatters that ship with the Exception Handling Application Block. The block includes the classes **TextExceptionFormatter** and **XmlExceptionFormatter**, both of which derive from the **ExceptionFormatter** class.

Exception formatters must include the handling instance ID, **HandlingInstanceId.** The handling instance ID value is generated on each exception handling request. Each individual exception formatter handles the handling instance ID as appropriate for that formatter. The **XmlExceptionFormatter** adds it as an attribute of the top-level **Exception** element, while the **TextFormatter** adds it as the first line of text. A **HandlingInstanceId** equal to **Guid.Empty** can be ignored.

The logging exception handler does not add the exception handling ID to the message to log. The formatter handles the ID. Exception Formatter types used with the Logging exception handler must implement a constructor with parameters of type **TextWriter**, **Exception**, and **Guid**, as shown in the following code.

```
public MyExceptionFormatter(TextWriter writer, Excepti
  : base(ex, handlingInstanceId)
{
   ...
}
```

Visual Basic

```
Public Sub New(writer As TextWriter, ex As Exception,
   MyBase.New(ex, handlingInstanceId)
   ...
End Sub
```

✓ Note:

If you have custom formatters designed for use with versions of Enterprise Library prior to version 4.1, you must update them so that the handling instance ID is not lost.

■ Creating a New Exception Formatter Class

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Deployment and Operations



Two of an administrator's main tasks will be to make sure that the initial deployment of the Exception Handling Application Block is planned and managed and to make sure that subsequent updates are deployed with minimal impact to existing applications that use the block. For details of deploying and updating Enterprise Library and the blocks, see <u>Deploying Enterprise Library</u>.

In addition, administrators must decide if they wish to use the instrumentation exposed by the block. For details of how to enable and disable instrumentation, see Enabling Instrumentation. For information about the instrumentation contained within the Exception Handling Application Block, see the following topics:

- Exception Handling Application Block Performance Counters
- Exception Handling Application Block Event Log Entries

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Exception Handling Application Block Performance Counters



The following table describes the Exception Handling Application Block performance counters.

Performance Counter Name	Description
Exception Handlers Executed/sec	The rate at which exception handlers were executed.
Exceptions Handled/sec	The rate at which exceptions were handled.
Total Exceptions Handled	The total number of exceptions handled.
Total Exceptions Handlers Executed	The total number of exception handlers that were executed.

A rate counter samples an increasing count of events over time and divides the values by the change in time to display a rate of activity. For more information about performance counters, see Overview of Performance Monitoring in the .NET Framework Class Library.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Exception Handling Application Block Event Log Entries



The Exception Handling Application Block is instrumented to log entries to the application event log for a variety of events. This topic lists the Exception Handling Application Block event log entries. The listener is the class that raised the event.

■ Exception Handling Error Event

■ Configuration Error Event

■ Internal Error Event

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Logging Application Block



Developers frequently write applications that require logging functionality. Typically, these applications format and log information in response to application events. For example, they may be required to log information in response to unexpected conditions, such as an application exception, or failure to connect to a database. Developers also write code to trace application flow through components during the execution of an application use case or scenario. In addition, applications often need to write information locally and over a network. In some cases, you may need to collate events from multiple sources into a single location.

The Enterprise Library Logging Application Block simplifies the implementation of common logging functions. You can use the Logging Application Block to write information to a variety of locations:

- The event log
- An e-mail message
- A database
- A message queue
- A text file
- A Windows® Management Instrumentation (WMI) event
- Custom locations using application block extension points

This section includes the following topics that will help you to understand and use the Logging Application Block:

- What Does the Logging Block Do? This topic provides a brief overview that will help you to understand what the block can do, and explains some of the concepts and features it incorporates. It also provides a simple example of how to write code to use the block.
- When Should I Use the Logging Block? This topic will help you to
 decide if the block is suitable for your requirements. It explains the
 benefits of using the block, and any alternative techniques you may
 consider. It also provides details of any limitations of the block that may
 affect your decision to use it.
- <u>Developing Applications Using the Logging Application Block</u>. This topic explains how to use the Logging Application Block in your

- applications. It shows how to configure the application block to perform common tasks and how to add application code to the application block where required.
- <u>Key Scenarios</u>. This topic demonstrates how to use the application block to perform the most common logging operations.
- Design of the Logging Application Block. This topic explains the decisions that went into designing the application block and the rationale behind those decisions.
- Extending and Modifying the Logging Application Block. This topic
 explains how to extend the application block by creating your own
 custom trace listeners, log formatters, and log filters; and explains how to
 modify the source code.
- <u>Deployment and Operations</u>. This topic explains how to deploy and update the application block's assemblies and also contains information about the instrumentation in the block.

More Information

For more information about logging and managing other crosscutting concerns, see the following patterns & practices guides:

- Microsoft Application Architecture Guide, 2nd Edition
- Design Guidelines for Exceptions

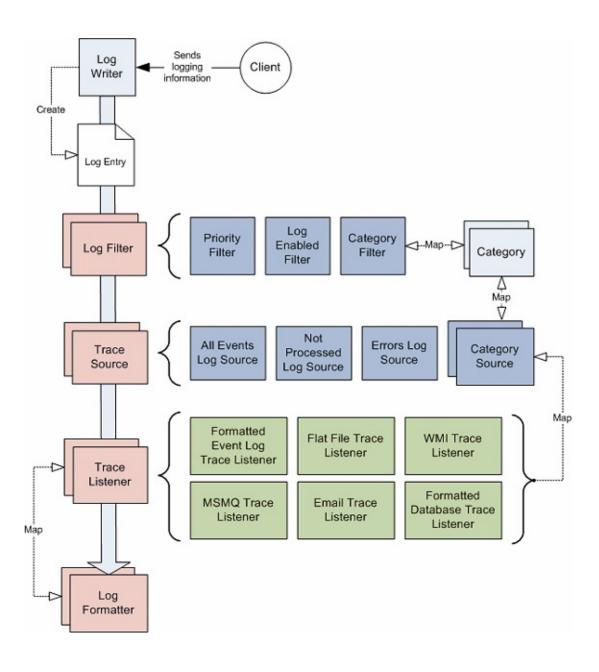
To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

What Does the Logging Block Do?



Although the process of creating and writing log entries is relatively simple, the number of options available (such as the many logging targets and the ability to filter entries) means that the underlying structure of the block and the options available for using it can seem complex. The following schematic shows how the main types of object in the block work together to provide flexibility when creating and writing log entries.



The five main types of objects are:

- **Log Writer**. The log writer is the main entry point for creating log entries and writing them to your chosen logging targets. It creates an instance of a log entry containing the information to be logged, and interacts with the other objects that filter the log entry, assign it to one or more categories, format it, and dispatch it to the appropriate targets.
- **Log Filters**. Log filters can block or allow a log entry based on a number of features. Each log entry is assigned to one or more categories (the default is the General category), and the category log filter can use these categories to pass or block a log entry. In addition, two special log filters can block all logging, or block log entries with a priority lower than a specified value. You define the categories, priorities, and the settings for the log filters in the configuration for the block.
- **Trace Sources**. Trace sources are effectively a set of buckets into which the block places all log entries that have not been blocked by a log filter. You use these buckets to define where log entries will be dispatched to—you can think of them as being the source of the log entries that will actually be dispatched to the target destinations. There are two basic types of trace sources:
 - There is a trace source for each category you define in the configuration of the block. These are called **Category Sources**.
 - There are three built-in trace sources: one that receives all log entries, one that receives log entries when an error occurs during processing or dispatching of the log entry, and one that receives all log entries that do not match any configured category. These are called **Special Sources**.
- Trace Listeners. Trace listeners represent the targets for your log entries, and you configure one for each type of target (such as the Windows® Event Log, a disk file, and an e-mail message) to which you want to send the log entries. Trace listeners listen for log entries arriving in the trace source buckets, format each log entry as required, and dispatch it to the target configured for that trace source. Your configuration maps each trace source (each category source you define plus the three special sources) to one or more trace listeners. The important point to note here is that this allows you to dispatch each log

- entry to zero, one, or more targets (such as sending it as e-mail as well as writing it to the Windows Event Log).
- Log Formatters. Each trace listener you add to your configuration can use a log formatter to convert the data in the log entry from a series of properties into format suitable for sending to the log target. The block contains a text formatter that you can configure with trace listeners that dispatch log entries to targets such as disk files, e-mail, or Windows Event Log; and a binary formatter that serializes the log entry data into a format suitable for transmission to targets such as Windows Message Queuing (MSMQ). The text formatter is configurable so that you can modify the format and content of the text message, including using placeholders for the values of the properties of the log entry.

■ The Logging Process Sequence

■ Example Application Code

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

When Should I Use the Logging Block?



If your applications have a requirement to log information to Windows Event Log, e-mail, a database, a message queue, Windows Management Instrumentation (WMI), or a file, you should consider using the Logging Application Block to provide this functionality. In particular, the Logging Application Block is useful if you need to filter logging messages based on category or priority, if you need to format the messages, or if you need to change the destination of the message without changing the application code. The Logging Application Block is also designed to be extensible and includes the facility to create custom formatters and trace listeners, which you can adapt to meet your application's logging requirements.

■ Scenarios for the Logging Application Block

■ Benefits of the Logging Application Block

■ Limitations of the Logging Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Developing Applications Using the Logging Application Block



This topic describes how to develop applications using the Logging Application Block. It explains how to configure the application block to perform particular tasks and how to use the application block according to particular application scenarios such as populating and raising events from code. It includes the following topics:

- Entering Configuration Information
- <u>Using the Distributor Service</u>
- Adding Application Code

If you want to deliver log entries at a central location for processing, you can use the Logging Application Block with Message Queuing (also known as MSMQ) to allow you to do this. By configuring multiple applications to use the same message queue, you can deliver all the log entries to one place. For details of how to install and configure the Logging Application Block in this scenario, see <u>Using the Distributor Service</u>.

If you are using the Logging Application Block with a Windows Communication Foundation (WCF) application, you must configure integration with WCF. For details, see <u>Configuring WCF Integration Trace Listeners</u>.

All application blocks ship as binary assemblies and as source code. If you want to use the source code, you must compile it. To learn how to compile the Enterprise Library source code, see <u>Building Enterprise Library from the Source Code</u>.

This topic assumes you are using the application block in its original state, without extending it. (To learn how to add functionality, see Extending and Modifying the Logging Application Block.)

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Entering Configuration Information



This section explains how to configure the Logging Application Block. For an overview and example of configuring the block, see <u>Configuration Overview</u>. For details of the schema for the Logging Application Block configuration, see <u>Source Schema for the Logging Application Block</u>. You can also configure the block in code by using an alternate configuration source. For more information, see <u>Advanced Configuration Scenarios</u> and <u>Using the Fluent Configuration API</u>.

✓ Note:

Run time changes to the configuration of the Logging Application Block are automatically detected after a short period, and the logging stack is updated. However, you cannot modify the logging stack at run time through code. For details of using configuration mechanisms that you can update at run time, see <u>Updating Configuration Settings at Run Time</u>.

To add the Logging Application Block

- 1. Open the configuration file. For more information, see <u>Configuring</u> <u>Enterprise Library</u>.
- 2. Open the **Blocks** menu and then click **Add Logging Settings**.
- 3. This adds a default general category, the three standard special categories, a default event log listener, and a default text formatter to the configuration.
- 4. Click the chevron expander arrow in the **Logging Settings** section to view all settings for this section.
- 5. Click the expander arrow at the left of the general category, the special categories, the event log listener, or the text formatter to view and set their properties as described in the following topics.

After you add the Logging Application Block to the application configuration, you can configure some or all of the following features. You should perform these tasks in the following order:

- Configuring Trace Listeners
- Configuring Formatters

- Configuring Trace Source Categories
- Configuring Logging Filters
- Configuring the Application Block

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

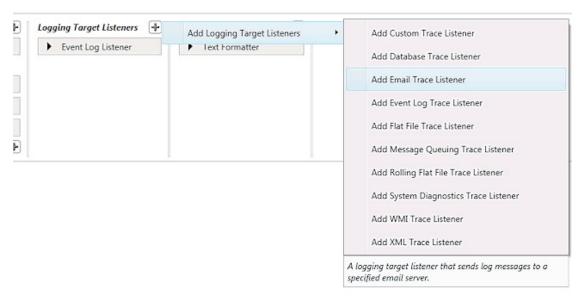
To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Configuration Overview

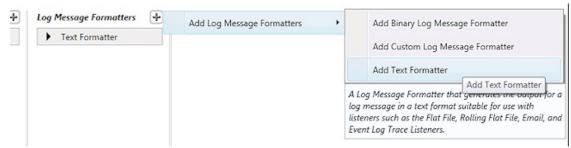
To understand how you configure and use the Logging Application Block, you must be familiar with the way the block processes log entries. The topic What Does the Logging Block Do? provides a simplified overview of the process and will help you when you come to configure the block.

The following steps illustrate the general process for configuring the Logging Application Block. It assumes you want to use two categories named Dev and Operations.

Add the trace listeners you want to use to output the logging information by clicking on the plus sign icon in the Logging Target Listeners section of the Logging Settings configuration pane and clicking Add Logging Target Listeners to display the list of listeners. When you select a listener, it is added to the Logging Target Listeners section where you can then set its properties.



2. Add the formatters you want the trace listeners to use to format the output by clicking on the plus sign icon in the Log Message Formatters section of the Logging Settings configuration pane and clicking Add Log Message Formatters to display the list of formatters. When you select a formatter, it is added to the Log Message Formatters section where you can then set its properties.

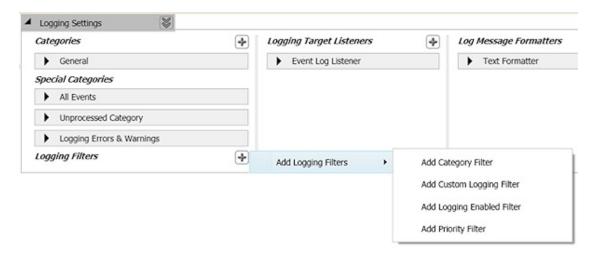


- 3. Set the **Formatter Name** property of each trace listener in the **Logging Target Listeners** section of the **Logging Settings** configuration pane to specify the appropriate formatter.
- 4. Add the category filters you want to use to the **Categories** section of the **Logging Settings** configuration pane by clicking on the plus sign icon in the **Categories** section of the **Logging Settings** configuration pane and clicking **Add Category**. A new category item is added to the **Categories** section where you can then set its properties. Click on its property expander arrow if the properties are not visible.
- 5. Add a reference to each of the trace listeners you want to use to output the logging information to each of the categories you created in the **Categories** section of the **Logging Settings** configuration by clicking the **Listeners** property plus sign icon and selecting a configured logging target listener. Each entry here is a reference to one of the trace listeners you previously configured in the **Trace Listeners** section. You can add more than one trace listener reference to each category filter you create. For example, you could specify the following:
 - A log entry containing the category name Dev will go to an event log trace listener (so that it appears in Windows Event Log)
 - A log entry containing the category name Operations will go to an e-mail trace listener and to an XML trace listener.
- 6. Add a reference to each of the trace listeners you want to use to output the logging information to the appropriate subsections of the **Special Categories** section of the **Logging Settings** configuration pane. For example, you may want to do the following:
 - Send the log entry to an e-mail trace listener when an error occurs within the logging system by adding a reference to this trace listener to the **Logging Errors & Warnings** section.
 - Send the log entry to an event log trace listener when the log entry does not match any configured category by adding a

- reference to this trace listener to the **Unprocessed Category** section.
- Send all log entries, irrespective of category or content, to a Flat File trace listener by adding a reference to this trace listener to the **All Events** section.
- 7. Set the remaining properties for each category or special filter. Specify the logging level, such as **Critical**, **Error**, **Warning**, or **All**, by setting the **Minimum Severity** property for each category filter.



8. Finally, add any log filters you want to use to the **Logging Filters** section of the **Logging Settings** configuration pane by clicking on the plus sign icon in the **Logging Filters** section of the **Logging Settings** configuration pane and clicking **Add Logging Filters**. The filter type you select is added to the **Logging Filters** section where you can then set its properties. Click on its property expander arrow if the properties are not visible. For example, you could use a category filter to block all logging operations except those that specify one of the two categories named Dev or Operations.



For specific details of how to configure each section of the Logging Application Block configuration, see <u>Entering Configuration Information</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Configuring Trace Listeners



Trace listeners receive log entries and write them to the appropriate destinations. The following procedure describes how to configure the trace listeners supplied with Enterprise Library and custom trace listeners that you create. There is also a specific procedure for configuring the two trace listeners that are required for integration with WCF. For more details of this procedure, see Configuring WCF Integration Trace Listeners.

✓ Note:

Trace listener logs have vulnerabilities and should be protected as appropriate, depending on whether the trace listener writes log entries to a file, database or MSMQ message queue.

For trace listeners that write log entries to files you should protect the logs by using access control lists. This includes Flat File, Rolling flat file, WMI and XML trace listener logs.

For trace listeners that write log entries to databases you should protect access to logs by using usernames and passwords. This includes the database trace listener.

For trace listeners that write log entries to MSMQ message queues you should allow only authorized users to read information from MSMQ. Protect the queue with relevant access control lists and follow any security considerations specific to MSMQ. This includes the message queuing trace listener.

■ Configuring the Built-in Trace Listeners

■ Configuring Custom Trace Listeners

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Trace Listener Properties



These tables explain the properties for the different trace listeners. There are tables for the following trace listeners:

- <u>Database Trace Listener</u>. This trace listener writes formatted log entries to a database.
- <u>E-mail Trace Listener</u>. This trace listener sends log entries as e-mail messages.
- Event Log Trace Listener. This trace listener formats log entries and writes them to Windows Event Log.
- Flat File Trace Listener. This trace listener writes log entries to a text file.
- <u>Message Queuing Trace Listener</u>. This trace listener writes log entries to a message queue.
- Rolling Flat File Trace Listener. This trace listener creates a new log file depending on the current log file age and/or size.
- System Diagnostics Trace Listener. This is one of the .NET Framework trace listeners such as the Console Trace Listener.
- <u>WMI Trace Listener</u>. This trace listener raises a WMI management event for each log entry received.
- XML Trace Listener. This trace listener is used to output log messages to an XML formatted file.

Database Trace Listener

The following table lists the properties that you can set when you add a **Database Trace Listener**.

Property	Description
Add Category Procedure	The name of the stored procedure that adds a category. The default is AddCategory . This is required.
Database Instance	The name of the database instance to use as configured in the Data Settings section of the configuration. This is required.
Severity Filter	Applies a filter that selects the level of message that it will detect. The valid values are All (the default), Off , Critical , Error , Warning , Information , Verbose , and ActivityTracing . The setting effectively means "the specified level and everything more important." For example, the Warning setting will detect warnings, errors, and critical events.
Formatter	The formatter to use with this trace listener. Select one from the drop-down list. The default is none. This is optional.
Name	The name of the trace listener. The default is Database Trace Listener . This is required.
Trace Output Options	A property used by trace listeners that do not output to a text formatter to determine which options, or elements, should be included in the trace output. Possible values are CallStack , DateTime , LogicalOperationStack , None , ProcessId , ThreadId , and Timestamp . The default is None . For an explanation of these values, see <u>TraceOutputOptions Values</u> . This is optional.
Write To Log Procedure	The name of the stored procedure that writes the log entries. The default is WriteLog . This is required.

Email Trace Listener

The following table lists the properties that you can set when you add an e-mail trace listener.

Property	Description
Authentication Mode	A value from the EmailAuthenticationMode enumeration that specifies how the listener will authenticate the user. Valid values are None , WindowsCredentials , and UserNameAndPassword .
Severity Filter	Applies a filter that selects the level of message that it will detect. The valid values are All (the default), Off , Critical , Error , Warning , Information , Verbose , and ActivityTracing . The setting effectively means "the specified level and everything more important." For example, the Warning setting will detect warnings, errors, and critical events.
Formatter Name	The formatter to use with this trace listener. Select one from the drop-down list. The default is none. This is optional.
From Address	The address where the log entry originated. The default is from@example.com. This is required.
Name	The name of the trace listener. The default is Email Trace Listener . This is required.
Authentication Password	Password when authenticating with user name and password.
Smtp Port	The SMTP port that receives e-mail messages. The default is 25 . This is optional.
Smtp Server	The SMTP server used to send e-mail messages. The default is 127.0.0.1 . This is optional.
Subject Line Suffix	The subject line suffix. This is optional.
Subject Line	The subject line prefix. This is optional.

Prefix	
To Address	The address where the log entry is sent. The default is to@example.com. This is required.
Trace Output Options	Trace listeners that do not output to a text formatter use this property to determine which options, or elements, should be included in the trace output. Possible values are: CallStack, DateTime, LogicalOperationStack, None, ProcessId, ThreadId, and Timestamp. The default is None. For an explanation of these values, see TraceOutputOptions Values. This is optional.
Authentication User Name	User name when authenticating with user name and password.
Use SSL	Specifies if the e-mail trace listener should use SSL when connecting to the mail server. Set to True to use SSL to connect, or False to use an unencrypted connection. The default is False

✓ Note:

The configuration files are not encrypted by default. A configuration file may contain sensitive information about connection strings, user IDs, passwords, database servers, and catalogs. You should protect this information against unauthorized read/write operations by using encryption techniques. For information about how to encrypt configuration files, see Encrypting Configuration Data and Configuration Enterprise Library.

In addition to this problem, e-mail messages exchanged with a SMTP server could be intercepted while in transit by a malicious user running a network sniffer or monitoring application. You can mitigate this problem by supporting Transport Layer Security or S/Mime with encryption of the e-mail messages.

Flat File Trace Listener

The following table lists the properties that you can set when you add a flat file trace listener.

Property	Description
File Name	The name of the file where entries are written. The default name is trace.log . This is a required value. It can include environment variables such as %WINDIR%, %TEMP%, and %USERPROFILE%.
Severity Filter	Applies a filter that selects the level of message that it will detect. The valid values are All (the default), Off , Critical , Error , Warning , Information , Verbose , and ActivityTracing . The setting effectively means "the specified level and everything more important." For example, the Warning setting will detect warnings, errors, and critical events.
Message Footer	Additional information contained in the file footer. The default is "" This is optional.
Formatter Name	The formatter to use with this trace listener. Select one from the drop-down list. The default is none. This is optional.
Message Header	Additional information contained in the file header. The default is "" This is optional.
Name	The name of the trace listener. The default is FlatFile Trace Listener . This is required.
Trace Output Options	Trace listeners that do not output to a text formatter use this property to determine which options, or elements, should be included in the trace output. Possible values are: CallStack, DateTime, LogicalOperationStack, None, ProcessId, ThreadId, and Timestamp. The default is None. For an explanation of these values, see TraceOutputOptions Values . This is optional.

✓ Note:			

If the file you specify for the **Flat FileTraceListener** is read-only, the trace listener does not write the data to the file and no exception occurs. Make sure that the file attributes are set to **read/write**.

When you use the **FlatFileTraceListener** class to write log information to a file, the block locks the file until the application closes. It is possible to open and read the file, but you cannot move or delete the log file until you close the application.

Event Log Trace Listener

The following table lists the properties that you can set when you add an **event log trace listener**.

Property	Description
Severity Filter	Applies a filter that selects the level of message that it will detect. The valid values are All (the default), Off , Critical , Error , Warning , Information , Verbose , and ActivityTracing . The setting effectively means "the specified level and everything more important." For example, the Warning setting will detect warnings, errors, and critical events.
Formatter Name	The formatter to use with this trace listener. Select one from the drop-down list. The default is none. This is optional.
Log Name	The name of the event log where entries are written. The default is Application . This is optional.
Machine Name	The name of the computer on which to write log entries. This is optional.
Name	The name of the trace listener. The default is Event Log Trace Listener . This is required.
Source Name	The source name to use when writing to the log. The default is Enterprise Library Logging. This is required.
Trace Output Options	Trace listeners that do not output to a text formatter use this property to determine which options, or elements, should be included in the trace output. Possible values are: CallStack, DateTime, LogicalOperationStack, None, ProcessId, ThreadId, and Timestamp. The default is None. For an explanation of these values, see TraceOutputOptions Values . This is optional.

Message Queuing Trace Listener

The following table lists the properties that you can set when you add a **Message Queuing (MSMQ) Trace Listener**.

Property	Description
Severity Filter	Applies a filter that selects the level of message that it will detect. The valid values are All (the default), Off , Critical , Error , Warning , Information , Verbose , and ActivityTracing . The setting effectively means "the specified level and everything more important." For example, the Warning setting will detect warnings, errors, and critical events.
Formatter Name	The formatter to use with this trace listener. Select one from the drop-down list. This must be the binary formatter when you use this listener with the message queuing distributor service. This is optional.
Message Priority	Sets the priority of a log entry. This determines its priority while the log entry is in transit and when it is inserted into its destination queue. Possible values are AboveNormal , High, Highest, Low, Lowest, Normal, VeryHigh, and VeryLow . It applies to the MsmqTraceListener class. The default is Normal . This is optional.
Name	The name of the trace listener. The default is Message Queuing Trace Listener .
Queue Path	The path to the queue that the Msmq TraceListener instance uses. This attribute is a message queuing path, and it applies to the MsmqTraceListener class. The default is .\Private\$\myQueue. This is required.
Recoverable	Specifies whether the log entry is guaranteed to be delivered if there is a computer failure or network problem. The default is False . This is optional.
Time To Be Received	The total time for a log entry to be received by the destination queue. The default is 49710.06:28:15 . This is optional.

Time To Reach Queue	The maximum time for the log entry to reach the queue. The default is 49710.06:28:15 . This is optional.
Trace Output Options	Attaches additional information to platform-provided trace listener output for listeners that do not output to a text formatter. Possible values are CallStack , DateTime , LogicalOperationStack , None , ProcessId , ThreadId , and Timestamp . The default is None . For an explanation of these values, see <u>TraceOutputOptions Values</u> . This is optional.
Transaction Type	The type of a Message Queuing transaction. Possible values are Automatic , None , and Single . It applies to the MsmqTraceListener class. The default is None . This is optional.
Use Authentication	Specifies whether the message was (or must be) authenticated before being sent. The default is False . This is optional.
Use Dead Letter Queue	Specifies whether a copy of a message that could not be delivered should be sent to a dead-letter queue. The default is False . This is optional.
Use Encryption	Specifies whether to make the message private. The default is False . This is optional.

Rolling Flat File Trace Listener

The following table lists the properties that you can set when you add a rolling flat file trace listener. This trace listener allows you to control the size and age of a log file.

Property	Description
File Name	This is the name of the rolling flat file. This is a required value. It can include environment variables such as %WINDIR%, %TEMP%, and %USERPROFILE%. If you also set the Max Archived Files property, See the advice on choosing a file name in the following Remarks section.
Severity Filter	Applies a filter that selects the level of message that it will detect. The valid values are All (the default), Off , Critical , Error , Warning , Information , Verbose , and ActivityTracing . The setting effectively means "the specified level and everything more important." For example, the Warning setting will detect warnings, errors, and critical events.
Message Footer	Additional information contained in the file footer. The default is "" This is optional.
Formatter Name	The formatter to use with this trace listener. Select one from the drop-down list. The default is none. This is optional.
Message Header	Additional information contained in the file header. The default is "" This is optional.
Max Archived Files	The maximum number of log files to retain. When set to an integer value, the trace listener will purge old files based on the file creation date when the number exceeds the specified value. See the note in the following Remarks section if you set this property.
Name	This is the name of the trace listener. The default is Rolling Flat File Trace Listener . This is required.
File Exists Behavior	This property determines what occurs to an existing file when it rolls over. If you select Increment , the application block

	creates a new file and names it by incrementing the timestamp. If you select Overwrite and do not provide a value for the Timestamp Pattern property, the existing file is overwritten.
Roll Interval	This property determines when the log file rolls over. You can select None (the default), Midnight (in which case the log will roll over at midnight), Minute , Hour , Day , Month , Week , or Year . This is optional.
Roll Size KB	This is the maximum size the file can reach, in kilobytes, before it rolls over. This is optional.
Timestamp Pattern	This is the date/time format that is appended to the new file name (see the Remarks section that follows this table).
Trace Output Options	Trace listeners that do not output to a text formatter use this property to determine which options, or elements, should be included in the trace output. Possible values are: CallStack, DateTime, LogicalOperationStack, None, ProcessId, ThreadId, and Timestamp. The default is None. For an explanation of these values, see TraceOutputOptions Values . This is optional.

Remarks

- If you set the **Max Archived Files** property, this trace listener will delete (purge) files using the file name pattern [file-name] *.file-extension. Therefore, it will delete all log files for any trace listener that matches this pattern when it purges archived log files. To prevent this, use a value for the **File Name** property that included an additional period. For example, use [file-name].[additional-name].file-extension.
- You can control either the size of the file, its age, or both. For example, if you specify in configuration a Roll Size KB value of 5 KB and a Roll Interval value of Day, the file rolls when its size exceeds 5 KB and it also rolls at the end of the day.
- If you select **Increment** for the **File Exists Behavior**, the application block creates a new file when the existing file rolls over. The file name includes the current timestamp. If a file with this name already exists, the application block adds an integer to the end of the timestamp and increments it until it cannot find a file with that name. For example, assume there is a file named **mylog2007-01-10.log** and the file rolls over while that timestamp is still valid. The Logging Application Block will then look for a file named **mylog2007-01-10.1.log**. If no such file exists, it will use that file name for the new file. If that file also exists, it will then attempt to locate the log with the next sequence number **mylog2007-01-10.2.log**.
- If you select **Overwrite** for the **File Exists Behavior**, the application block replaces the existing file with a new file when the current file rolls over. However, if you also set the **Timestamp Pattern** property, the application block will create a new file with the current time stamp instead of replacing the existing file. If, for some reason, it cannot overwrite the file, it will generate a name using the same process that is used with the **Increment** value.
- Relative path names resolve to a location that is relative to the *AppDomain.CurrentDomain.BaseDirectory* directory.

System Diagnostics Trace Listener

The **TraceListener** class provides the **abstract** base class for trace listeners that monitor trace and debug output. The following table lists the properties that appear when you add a **System DiagnosticsTrace Listener**.

Property	Description
InitData	If supplied, this property is used when the application block constructs a trace listener. It is a string whose meaning depends on the type of trace listener being constructed. For the .NET Framework trace listeners, the string has the following values: TextWriterTraceListener : filename; XmlWriterListener : filename; DelimitedLIstTraceListener : filename; ConsoleTraceListener : not applicable; DefaultTraceListener : not applicable; EventLogTraceListener : event source name. If the InitData field is not specified, the block uses the default constructor. If the user specifies the InitData field for a trace listener that does not have a constructor overload that accepts a string, an error occurs. This is optional.
Name	The name of the trace listener. The default is System Diagnostics Trace Listener . This is required.
Severity Filter	Applies a filter that selects the level of message that it will detect. The valid values are All (the default), Off , Critical , Error , Warning , Information , Verbose , and ActivityTracing . The setting effectively means "the specified level and everything more important." For example, the Warning setting will detect warnings, errors, and critical events.
Trace Output Options	Trace listeners that do not output to a text formatter use this property to determine which options, or elements, should be included in the trace output. Possible values are: CallStack, DateTime, LogicalOperationStack, None, ProcessId, ThreadId, and Timestamp. The default is None. For an explanation of these values, see TraceOutputOptions Values . This is optional.
Type Name	The type of the trace listener. Select by clicking the ellipsis button (). This opens the Type Selector . This is required.

✓ Note:

If you specify a **System Diagnostics** trace listener that writes to a file and that file is read-only, the trace listener does not write the data to the file and no exception occurs. Make sure the file attributes are set to **read/write**.

WMI Trace Listener

The WMI trace listener is a trace listener that raises a WMI management event for each log entry it receives. The following table lists the properties that you can set when you add a **WMI Trace Listener**.

Property	Description
Severity Filter	Applies a filter that selects the level of message that it will detect. The valid values are All (the default), Off , Critical , Error , Warning , Information , Verbose , and ActivityTracing . The setting effectively means "the specified level and everything more important." For example, the Warning setting will detect warnings, errors, and critical events.
Name	The name of the trace listener. The default is WMI Trace Listener . This is required.
Trace Output Options	Trace listeners that do not output to a text formatter use this property to determine which options, or elements, should be included in the trace output. Possible values are: CallStack, DateTime, LogicalOperationStack, None, ProcessId, ThreadId, and Timestamp. The default is None. For an explanation of these values, see TraceOutputOptions Values . This is optional.

XML Trace Listener

The following table lists the properties that you can set when you add an **XML Trace Listener**.

Property	Description
File Name	This is the name of the file where the trace listener writes the data it extracts from an XmlLogEntry object. This is a required value. It can include environment variables such as %WINDIR%, %TEMP%, and %USERPROFILE%.
Severity Filter	Applies a filter that selects the level of message that it will detect. The valid values are All (the default), Off , Critical , Error , Warning , Information , Verbose , and ActivityTracing . The setting effectively means "the specified level and everything more important." For example, the Warning setting will detect warnings, errors, and critical events.
Name	This is the name of the trace listener. The default is XML Trace Listener . This is required.
Trace Output Options	Trace listeners that do not output to a text formatter use this property to determine which options, or elements, should be included in the trace output. Possible values are: CallStack, DateTime, LogicalOperationStack, None, ProcessId, ThreadId, and Timestamp. The default is None. For an explanation of these values, see TraceOutputOptions Values . This is optional.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0 TraceOutputOptions Values

The following table lists the possible **TraceOutputOptions** values.

Value	Description
Callstack	Write the call stack, which is represented by the return value of the Environment.StackTrace property.
DateTime	Write the date and time.
LogicalOperationStack	Write the logical operation stack, which is represented by the return value of the CorrelationManager.LogicalOperationStack property.
ProcessId	Write the process identity, which is represented by the return value of the Process.ID property.
ThreadId	Write the ThreadIdentity , which is represented by the return value of the Thread.ManagedThreadIdProperty for the current thread.
Timestamp	Write the timestamp, which is represented by the return value of the System.Diagnostics.Stopwatch.GetTimeStamp method.

For more information about these values, see <u>TraceOptions Fields</u> in the .NET Framework Class Library on MSDN®.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Configuring WCF Integration Trace Listeners

¥

Windows Communication Foundation (WCF) is able to log directly only to **System.Diagnostics** trace sources, so it is necessary to configure a special trace listener named the **EntLibLoggingProxyTraceListener** in the <system.diagnostics> configuration section to enable the Logging Application Block to process WCF log messages. This trace listener receives messages from the trace source, wraps them in a **LogEntry** object, and forwards them to the Logging Application Block, where they can be processed according to the Logging Application Block's configuration. If the original message is in XML format (as is the case when messages are generated from WCF), the **EntLibLoggingProxyTraceListener** creates an **XmlLogEntry** object instead of a **LogEntry**. The **XmlLogEntry** class is derived from the standard **LogEntry** class and adds support for an XML payload.

The **EntLibLoggingProxyTraceListener** will add the name of its containing trace source as a category to each **XmlLogEntry** it creates. In addition, it is possible to configure the **EntLibLoggingProxyTraceListener** to extract information from the XML data for use as additional categories. This can be specified using XPath queries in the definition of the trace listener in the configuration file. The **categoriesXPathQueries** attribute can be set to a semicolon-delimited list of XPath queries, and the **namespaces** attribute can be set to a space-delimited list of XML namespaces used in the XPath queries, as shown in the following example.

XML

To use the **EntLibLoggingProxyTraceListener** with WCF, you will need to define a trace source named **System.ServiceModel** in the <system.diagnostics> configuration section and turn on logging in WCF by

specifying appropriate values in the <**diagnostics**> section in the <**system.serviceModel**> configuration section. Note that the Enterprise Library configuration tools do not support editing either of these sections, so you must use a text editor or alternative editor. For more information, see System.ServiceModel Namespace on MSDN. For more information about using tracing with WCF, see Configuring Tracing on MSDN.

Although you can use any of the trace listeners supported by the Logging Application Block with WCF, the most common scenario is to log the messages in XML format. XML files logged from WCF can be analyzed in the Service Trace Viewer application that is included in the Windows SDK. To configure the Logging Application Block to log messages in this XML format, you should use the **XmlTraceListener**. This trace listener derives from the **XmlWriterTraceListener**, which is a part of the .NET Framework, and is able to extract the XML payload from an **XmlLogEntry** object and write this data to an XML text file. You can analyze the output of this trace listener with the WCF log file analysis tools.

A sample configuration file that demonstrates what the configuration should look like follows the next procedure. The **<system.serviceModel>** section of the file defines how the WCF service behaves and is not relevant to the actual logging process.

The following procedure describes how to integrate the Logging Application Block with applications that use WCF.

To configure the WCF-integration trace listeners

- 1. Create or open a configuration file in one of the Enterprise Library configuration tools, and ensure the Logging Application Block is added to the application's configuration. For more information, see Configuring Enterprise Library.
- Click the plus sign icon in the LoggingTarget Listeners pane, point to Add Logging Target Listener, and then click AddXML Trace Listener.
- 3. (Optional) In the properties pane, set the **Name**, the **File Name** for the trace file, the **Severity Filter** for the level of message to detect, and select a value for the **Trace Output Options** property to specify which options or elements should be included in the trace output. For more information, see <u>TraceOutputOptions Values</u>.

- 4. Click the plus sign icon in the **Categories** pane and click **Add Category**.
- 5. In the properties pane, set the **Name** to **System.ServiceModel**; set **Auto Flush** and **Minimum Severity** as required.
- 6. Click the **Listeners** property plus sign icon, then select the **XML Trace Listener** from the drop-down list.
- 7. Save the configuration file.
- 8. Open the configuration file either in Visual Studio® or in the text editor of your choice.
- 9. Define the EntLib Proxy trace listener in the **<system.diagnostics>** section and use **System.ServiceModel** as the source. (See the sample configuration file.)
- 10. Modify the WCF configuration to specify the desired level of logging, as shown in the following sample configuration file.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="loggingConfiguration"</pre>
             type="Microsoft.Practices.EnterpriseLibra
                   .Logging.Configuration.LoggingSettir
                  Microsoft.Practices.EnterpriseLibrar
  </configSections>
  <loggingConfiguration name="Logging Application Bloc</li>
                         tracingEnabled="true"
                         defaultCategory="System.Service"
                         logWarningsWhenNoCategoriesMat
    steners>
      <add fileName="c:\\trace-xml.log"</pre>
           listenerDataType="Microsoft.Practices.Enter
                             .Logging.Configuration.Xml
                             Microsoft.Practices.Enterg
           traceOutputOptions="None"
           type="Microsoft.Practices.EnterpriseLibrary
                  .TraceListeners.XmlTraceListener,
                 Microsoft.Practices.EnterpriseLibrary
```

```
name="XML Trace Listener" />
  </listeners>
  <formatters>
  </formatters>
  <categorySources>
    <add switchValue="All" name="System.ServiceMode]</pre>
      steners>
        <add name="XML Trace Listener" />
      </listeners>
    </add>
  </categorySources>
  <specialSources>
    <allEvents switchValue="All" name="All Events" /
    <notProcessed switchValue="All" name="Unprocesse</pre>
    <errors switchValue="All" name="Logging Errors &</pre>
  </specialSources>
</le>
<system.serviceModel>
   <diagnostics>
     <messageLogging logEntireMessage="true"</pre>
                      logMalformedMessages="true"
                      logMessagesAtTransportLevel="tr
   </diagnostics>
  <services>
  <service name="WCFServiceLibrary1.service1"</pre>
           behaviorConfiguration="MyServiceTypeBehav
    <endpoint contract="WCFServiceLibrary1.IService1</pre>
              binding="wsHttpBinding"/>
    <endpoint contract="IMetadataExchange" binding='</pre>
              address="mex" />
  </service>
</services>
<behaviors>
  <serviceBehaviors>
    <behavior name="MyServiceTypeBehaviors" >
      <serviceMetadata httpGetEnabled="true" />
    </behavior>
  </serviceBehaviors>
</behaviors>
```

For information about the WCF-integration trace listener properties, see <u>Trace</u> <u>Listener Properties</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Configuring Formatters



The first procedure describes how to configure the formatters. The text formatter converts a log entry into a text string. The contents of the string are determined by replacing tokens in the text formatter's **Template** property. The **BinaryLogFormatter** uses the .NET Framework **BinaryFormatter** to serialize and describing the log entry in binary format. The **BinaryLogFormatter** is required when using the Message Queuing (MSMQ) trace listener with the Message Queuing distributor service.

To configure formatters

- Click on the plus sign icon in the Log Message Formatters section of the Logging Settings configuration pane and click Add Log Message Formatters to display the list of formatters. When you select a formatter, it is added to the Log Message Formatters section where you can then set its properties.
- 2. Click the expander arrow in the formatter item you wish to configure if the properties are not visible.
- 3. (Optional) Change the **Name** property. The default name is the name of the formatter.
- 4. If you are adding a **text formatter** and want to change the template that contains the value placeholders, click the ellipsis button (...) for the **Template** property. Make the required changes in the **Template Editor** dialog. You can use the **Insert Token** button to add tokens with the appropriate syntax. After you finish making the changes, click **OK**.
- 5. If you are adding a **Custom Formatter** and want to add or edit the settings, change the list of keys and values in the **Attributes** property section.
- 6. Repeat the procedure for each formatter you want to use.

✓ Note:

The only property you can change for the binary formatter is the name.

■ Text Formatter Template Tokens

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Configuring Trace Source Categories



Trace source categories associate log entries with their target listener(s). You can configure the following two category types:

- **Categories** that specify the logging categories you want to use. When you create a log entry, you can assign it to one or more categories. Each category can filter the log entry based on its severity (such as **Critical**, **Error**, **Warning**, or **Information**), and route it to one or more trace listeners (logging targets).
- **Special Categories** that automatically receive all log entries, unprocessed log entries, or log entries where an error occurred during logging. Each of these special categories can filter the log entry based on its severity (such as **Critical**, **Error**, **Warning**, or **Information**), and route it to one or more trace listeners (logging targets).

■ Configuring Trace Source Categories

■ Configuring Trace Sources Special Category

■ Source Category Properties

■ Severity (SourceLevels) Values

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Configuring Logging Filters



You can filter log entries based on their categories and their priorities. You can also entirely enable or disable logging or add a custom logging filter. The following procedure describes how to configure the logging filters.

To configure logging filters

- 1. Select **Add Logging Filters** as described in **Configuration Overview**.
- 2. If the filter's property pane is not displayed, click the property expander arrow or right-click the filters node you wish to configure.
- 3. The **Category Filter** allows or denies log entries based on their categories:
 - If you want to use a Category Filter, you can select the specific category filter by editing the **Categories** property. Click the plus sign icon to add a category filter.
 - Type or select the category name for the category that will be filtered. You can either select a name from the drop-down list or type a category name. All Category Filters that have been added to the Category Filters section are available for selection in the drop-down list. The selected category name appears in the lower pane.
 - Repeat to add any other categories you require.
 - To remove a category, click the delete (x) icon beside the category you wish to remove.
 - Repeat to remove any other categories you no longer require.
 - Select a filter mode, either AllowAllExceptDenied or DenyAllExceptDenied.
 - (Optional) Change the name of the category filter. For information about the **CategoryFilter** properties, see <u>Category Filter Properties</u>.
- 4. The **Logging Enabled Filter** provides a global switch that you can use to turn logging on and off:
 - If you want to log events, set **All Logging Enabled** to **True**. To prevent all logging, set **All Logging Enabled** to **False**.
 - (Optional) Change the name of the **Logging Enabled Filter**.

For information about the **Logging Enabled Filter** properties, see <u>Logging Enabled Filter Properties</u>.

- 5. The **Priority Filter** allows or denies log entries based on their priority:
 - (Optional) Set the **Maximum Priority** property. This is the maximum priority value a log entry can have in order to be logged. If you do not set this property, the value is 2147483647 (this is the largest possible value of a 32-bit signed integer).
 - (Optional) Set the **Minimum Priority** property. This is the minimum value a log entry must have to be logged. If you do not set this property, the value is **-1**.
 - (Optional) Change the name of the priority filter. For information about the priority filter properties, see <u>Priority</u> <u>Filter Properties</u>.
- 6. A **Custom Logging Filter** is a class that you have created that derives from the **LogFilter** class. Its configuration information consists of a collection of name/value string pairs:
 - Set the **Type** property by clicking the ellipsis button (...) to
 display the Type Selector. Navigate to and click the filter type
 name in the **Type Selector** dialog box. You must select a class
 that derives from the **LogFilter** class. It must also have the type **CustomLogFilterData** specified as the value of the **ConfigurationElementType** attribute placed on the class.
 - Enter a key/value pair in the **Key** and **Value** text boxes. When you enter a key/value pair, a new pair of blank text boxes are displayed.
 - (Optional) Change the name of the **Custom Logging Filter**. For information about the **Custom Logging Filter** properties, see <u>Custom Logging Filter Properties</u>.
- 7. Repeat the preceding steps for each filter you require.

■ Category Filter Properties

■ Logging Enabled Filter Properties

■ Priority Filter Properties

■ Custom Logging Filter Properties

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Configuring the Application Block



The following procedure describes how to configure the Logging Application Block properties.

To configure the Logging Application Block

- 1. In the **Logging Settings** section, click each property you want to change. For information about the Logging Application Block properties, see the table that follows this procedure.
- 2. Set the properties if you need to. If you want to use a default category, click the drop-down arrow and select one of the category names. Log entries that are not assigned to a category belong to the default category.
- 3. The **Warn If No Category Match** property sends log entries that are assigned to a category that is not specified in configuration to the **Logging Errors & Warnings** special source. The default is **True**. If you do not want this to occur, click **False** in the drop-down list.
- 4. The **Activity Tracing Enabled** property specifies whether activity tracing is enabled. The default is **True**. If you do not want this to occur, click **False** in the drop-down list.

■ Logging Application Block Properties

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

This topic lists the elements and attributes used to configure the Logging Application Block. The configuration file has the following section-handler declaration.

The section-handler declaration contains the name of the configuration settings section and the name of the section-handler class that processes configuration data in that section. The name of the configuration settings section is loggingConfiguration. The name of the section-handler class is Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettir

■ loggingConfiguration Element

■ logFilters Child Element

■ categoryFilters Child Element

■ categorySources Child Element

■ listeners Child Element (categorySources)

• specialSources Child Element

■ listeners Child Element (errors)

■ notProcessed Child Element

■ listeners Child Element (notProcessed)

■ allEvents Child Element

• listeners Child Element (allEvents)

■ listeners Child Element (loggingConfiguration)

■ formatters Child Element

■ msmqDistributorSettings Element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Using the Distributor Service



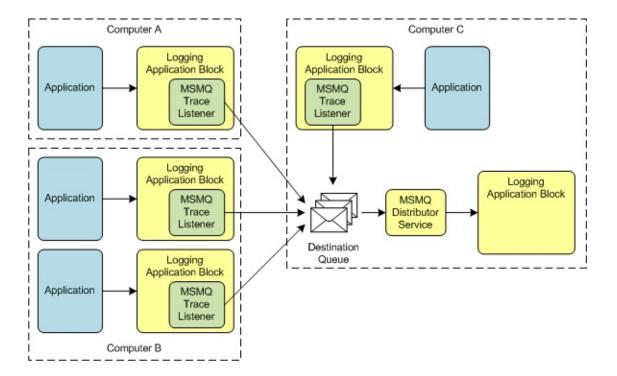
Applications must often send log entries from multiple sources to a common destination. The Logging Application Block takes advantage of Message Queuing (also known as MSMQ) to allow you to do this. By configuring multiple applications to use the same message queue, you can process log entries at a central location.

To distribute log entries to a central destination, configure your application to write log entries to the message queuing trace listener. When the application sends a log entry to the Logging Application Block, it places the log entry on a Message Queuing queue. The distributor service runs as a Windows service on either the same computer as the application or on a remote computer. It polls the queue to see if there are any log entries on it. The polling interval is determined by configuration.

If there are log entries on the queue, the distributor service uses an instance of the Logging Application Block to forward the messages to the trace listener(s). The trace listener(s) write the log entries to the destinations, such as an event log or a flat file.

The distributor service requires that all log entries be formatted using the **BinaryLogFormatter** class. If the service cannot interpret the entry, it will log an error to the Application Event Log and shut down.

The following schematic illustrates how multiple applications use the distributor service to send log entries to a central location.



Each instance of the Logging Application Block uses an instance of the message queuing trace listener (the **MsmqTraceListener** class) to send the log entries to a single destination queue. The distributor service polls the queue and uses another instance of the Logging Application Block to direct the log entries to the proper trace listeners. Note that the distributor service can run on a remote computer. The following sections describe installing and using the distributor service:

- Installing the Distributor Service
- Starting the Distributor Service
- <u>Understanding the serviceName Attribute</u>

■ Installing the Distributor Service

■ Starting the Distributor Service

■ Understanding the serviceName Attribute

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Adding Application Code



The Logging Application Block is designed to support the most common scenarios for logging information. When adding your application code, refer to the scenarios in the Key Scenarios sections and select the ones that best suit your situation. Use the code that accompanies the scenario either as it is or adapt it as necessary.

First, prepare your application to use the Logging Application Block. The following procedure describes how to include the necessary Enterprise Library assemblies and elements in your code.

To prepare your application

- 1. Set a reference to the Logging Application Block assembly:
 - If you are using C#, in Visual Studio, right-click **References** in Solution Explorer, and then click **Add References** to add references to the following assemblies:
 - Microsoft.Practices.Unity.dll
 - Microsoft.Practices.Unity.Interception.dll
 - Microsoft.Practices.ServiceLocation.dll
 - Microsoft.Practices.EnterpriseLibrary.Common.dll
 - Microsoft.Practices.EnterpriseLibrary.Logging.dll
 - If you are using Visual Basic®, double-click My Project in Visual Studio, click the References tab, and then click Add Reference to select the assembly. In the list of imported namespaces at the bottom of the tab, select the following check boxes:
 - Microsoft.Practices.Unity.dll
 - Microsoft.Practices.Unity.Interception.dll
 - Microsoft.Practices.ServiceLocation.dll
 - Microsoft.Practices.EnterpriseLibrary.Common.dll
 - Microsoft.Practices.EnterpriseLibrary.Logging.dll
- 2. (Optional) To use elements from the Logging Application Block without fully qualifying the element reference, you can add the following **using** statements (C#) or **Imports** statements (Visual Basic) to the top of your source code file.

```
using Microsoft.Practices.EnterpriseLibrary.Logg using Microsoft.Practices.EnterpriseLibrary.Logg using Microsoft.Practices.EnterpriseLibrary.Logg
```

Visual Basic

```
Imports Microsoft.Practices.EnterpriseLibrary.Lo Imports Microsoft.Practices.EnterpriseLibrary.Lo Imports Microsoft.Practices.EnterpriseLibrary.Lo
```

✓ Note:

For Visual Basic projects, you can also use the **References** page of the Project Designer to manage references and imported namespaces. To access the **References** page, select a project node in Solution Explorer, and then click **Properties** on the **Project** menu. When the Project Designer appears, click the **References** tab.

The **ExtraInformation** providers gather context information that is useful but not always necessary because it is expensive to collect. Examples are stack trace information and COM+ information. The **ExtraInformation** providers add the information to a dictionary. You can choose which providers to use (if any) and add the resulting dictionary to the **LogEntry.ExtendedProperties** property. Filters are optional. You only need to import the **Microsoft.Practices.EnterpriseLibrary.Logging.Filters** namespace if you are going to refer to specific filters in your application code.

- 3. If you are using the **DatabaseTraceListener** class, you must also do the following:
 - Configure the application to use the Data Access Application Block. For more information, see <u>The Data Access Application</u> Block.
 - Execute the script named **CreateLoggingDb.cmd** (located in the Source\Blocks\Logging\Src\DatabaseTraceListener\Scripts folder) to create the **Logging** database.

✓ Note:

It is also a good idea to add a reference to the **Microsoft.Practices.EnterpriseLibrary.Logging.Database.dll** to your project so that the assembly required at run time is copied to the output folder.

4. Add the application code.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Key Scenarios



This series of topics describes common situations developers must address when providing logging functionality in their applications. Each scenario explains the task, describes a real-world situation where such a task might arise, and includes code demonstrating how to use the Logging Application Block to complete the task. The scenarios are as follows:

- <u>Logging to a Database</u>. This topic describes the process you should follow for using the Logging Application Block to implement the common requirement of logging to a database.
- <u>Logging to Windows Event Log</u>. This topic describes the process you should follow for using the Logging Application Block to implement the common requirement of logging to Windows Event Log.
- <u>Logging to a Disk File</u>. This topic describes the process you should follow for using the Logging Application Block to implement the common requirement of logging to a disk file.
- <u>Logging to Windows Message Queuing</u>. This topic describes the process you should follow for using the Logging Application Block to implement the common requirement of logging to Windows Message Queuing.
- Logging to WMI. This topic describes the process you should follow for using the Logging Application Block to implement the common requirement of logging to the Windows Management Instrumentation repository.
- <u>Logging as E-mail Messages</u>. This topic describes the process you should follow for using the Logging Application Block to implement the common requirement of sending logging information in e-mail messages.
- <u>Populating and Raising Events from Code</u>. This scenario illustrates how to write code to specify the data to be logged, along with a category and priority, and pass it to the application block.
- <u>Populating a Log Message with Additional Context Information</u>. This scenario illustrates how to populate a dictionary of custom information to be added to a log entry.
- <u>Tracing Activities and Propagating Context Information</u>. This scenario illustrates how to log the start and end of an activity and trace key activity points in between.

• <u>Checking Filter Status before Constructing Log Messages</u>. This scenario illustrates how to avoid collecting log information for messages that will not be logged according to the current configuration information.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Logging to a Database



This scenario is one of several that describe the typical requirements when using the Logging Application Block in your applications. It describes the process for setting up the block to send log events to a database. The process involves configuring the block and performing other tasks to prepare your application. This topic acts as a reference to help you quickly set up the Logging Application Block to perform the required logging action.

■ Typical Goals

■ Solution

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Logging to Windows Event Log



This scenario is one of several that describe the typical requirements when using the Logging Application Block in your applications. It describes the process for setting up the block to send log events to Windows Event Log. The process involves configuring the block and performing other tasks to prepare your application. This topic acts as a quick reference to help you rapidly set up the Logging Application Block to perform the required logging action.

■ Typical Goals

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Logging to a Disk File



This scenario is one of several that describe the typical requirements when using the Logging Application Block in your applications. It describes the process for setting up the block to send log events to a disk file. The process involves configuring the block and performing other tasks to prepare your application. This topic acts as a quick reference to help you quickly set up the Logging Application Block to perform the required logging action.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Logging to Windows Message Queuing



This scenario is one of several that describe the typical requirements when using the Logging Application Block in your applications. It describes the process for setting up the block to send log events as messages through Windows Message Queuing. The process involves configuring the block and performing other tasks to prepare your application. This topic acts as a quick reference to help you quickly set up the Logging Application Block to perform the required logging action.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Logging to WMI



This scenario is one of several that describe the typical requirements when using the Logging Application Block in your applications. It describes the process for setting up the block to send log events to the Windows Management Instrumentation database. The process just involves configuring the block—there are no other tasks required to prepare your application. This topic acts as a quick reference to help you quickly set up the Logging Application Block to perform the required logging action.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Logging as E-mail Messages



This scenario is one of several that describe the typical requirements when using the Logging Application Block in your applications. It describes the process for setting up the block to send log events as e-mail messages. The process involves configuring the block and performing other tasks to prepare your application. This topic acts as a quick reference to help you rapidly set up the Logging Application Block to perform the required logging action.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Populating and Raising Events from Code



The ability to populate and raise events from code is fundamental to the functionality of the Logging Application Block.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Populating a Log Message with Additional Context Information



The **LogEntry** class defines properties to hold information common to typical logging scenarios. Developers often need to add context information to log entries. The same type of context information can be required for multiple log entries in the same application or in multiple applications. Because context information can be expensive to gather, certain types of information are not automatically collected. The **ExtraInformation** providers gather context information that is useful but not always necessary because it is expensive to collect. Examples are stack trace information and COM+ information.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Tracing Activities and Propagating Context Information



In some cases, you will need to log information at the start and end of a particular activity, including timing information. In addition, you can trace the progress of the activity at selected points in the application. Tracing allows you to associate all events between the start and end of an activity with an **ActivityID** property and a category. The **ActivityID** property allows you to correlate log entries that are written during the execution of an activity. You can use filters and categories to direct and control the information produced for an event that can occur in the context of any activity.

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Checking Filter Status before Constructing Log Messages



By using the Logging Application Block, you can query the filter status to determine whether a log message should be logged according to the filter configuration.

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design of the Logging Application Block



The Logging Application Block includes the following features:

- A simple and consistent way of logging event information
- Distribution of information to multiple sources
- Activity tracing to mark the start and end of an activity such as a use case
- Simplified application block configuration using the configuration tools
- Extensibility through custom trace listeners and formatters

■ Design Goals

■ Design Highlights

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Designing for Simple and Consistent Logging Functionality



Developers face many implementation choices and requirements when they add logging functionality to their applications. Different applications may log information to different destinations. For example, one application may use an event log and another application may use a flat file. Even a single application may log information to multiple destinations. As a result, developers must often write duplicate code for common tasks such as writing to an event log or to a flat file.

Uniform implementations make the code easier to understand, more predictable, and easier to maintain. However, different development teams routinely implement different logging strategies. The Logging Application Block encapsulates the logic that performs logging and tracing operations into a few classes that have small numbers of methods. These methods are the same for all log message destinations. This means that applications that use the Logging Application Block are consistent in the ways that they log information. By using the Logging Application Block, this consistency remains across single projects, multiple projects, or enterprise-scale solutions.

■ Design Implications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Extending and Modifying the Logging Application Block



In its original state, the Logging Application Block works well for typical logging scenarios. However, there may be times when you need to customize some of the application block's behavior to better suit your application's particular requirements. There are two ways to do this: you can extend the Logging Application Block using the built-in extension points or you can modify the application block by making changes to its source code. For more details, see the following topics:

- Extending the Logging Application Block
- Extending and Modifying Enterprise Library

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending the Logging Application Block



The Logging Application Block is designed to suit a variety of applications and to provide the most commonly used logging functions. You can extend the application block through designated extension points. Typically, these are custom classes, written by you, that implement a particular interface or derive from an abstract class. Because these custom classes exist in your application space, you do not need to modify or rebuild the application block. Instead, you designate your extensions using configuration settings. Additionally, with extension points, you can adapt the application block to suit the needs of any particular application.

You can extend the capabilities of the block by adding custom formatters, trace listeners, and log filters.

Custom Provider or Extension	Interface or Base Class
Log Entry Formatter	ILogFormatter
Trace Listener	CustomTraceListener
Log Filter	ILogFilter LogFilter

For detailed information about how to integrate custom providers with the Enterprise Library configuration system and configuration tools see <u>Creating Custom Providers for Enterprise Library</u>.

The following procedure describes the general approach to extend the Logging Application Block.

To extend the Logging Application Block

- 1. Create a new custom class and add it to your project.
- 2. Make sure that the class implements the required interfaces, constructors, and methods.
- 3. Add the custom object to the configuration of the Logging Application Block using the Enterprise Library configuration tools:
 - Specify your custom class as the type name.

• Specify any custom configuration properties by modifying the attributes of the object.

■ Creating a Custom Log Entry Formatter

■ Creating a Custom Trace Listener

■ Creating a Custom Log Filter

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Deployment and Operations



Two of an administrator's main tasks will be to see that the initial deployment of the Logging Application Block is planned and managed and that subsequent updates are deployed with minimal impact to existing applications that use the application block. For details of deploying and updating Enterprise Library and the application blocks, see <u>Deploying Enterprise Library</u>.

In addition, administrators must decide whether they want to use the instrumentation exposed by the application block. For details of how to enable and disable instrumentation, see Enabling Instrumentation. For information about the instrumentation contained in the Logging Application Block, see the following topics:

- Logging Application Block Performance Counters
- Logging Application Block Event Log Entries

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Logging Application Block Performance Counters



The following table describes the Logging Application Block performance counters.

Performance Counter Name	Description
Avg. Trace Execution Time	The average execution time for traced operations.
Logging Events Raised/sec	The rate at which logging events were raised.
Total Logging Events Raised	The total number of logging events raised.
Total Trace Listener Entries Written	The total number of entries that were traced by individual trace listeners.
Total Trace Operations Started	The total number of tracing operations started.
Trace Listener Entries Written/sec	The rate at which log entries were traced by individual trace listeners.
Trace Operations Started/sec	The rate at which tracing operations were started.

An average counter measures a value over time and displays the average of the last two measurements. A rate counter samples an increasing count of events over time and divides the values by the change in time to display a rate of activity. For more information about performance counters, see Overview of Performance Monitoring in the .NET Framework Class Library on MSDN.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Logging Application Block Event Log Entries



This topic lists the Logging Application Block event log entries. The listener is the class that raised the event.

■ FailureLoggingError Event

■ LockAcquisitionError Event

■ ConfigurationFailure Event

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Policy Injection Application Block



Policy injection can be used to change the behavior of your custom objects, and almost any other .NET class, in order to better manage crosscutting concerns for these objects. In the current version of Microsoft® Enterprise Library, policy injection is implemented through the Unity interception mechanism. For more information see Interception with Unity.

While the Policy Injection Application Block is still included in this release of Enterprise Library, it is (with the exception of one call handler) a set of legacy components such as the **PolicyInjection** facade that supports backwards compatibility with applications that use versions of Enterprise Library prior to version 5.0.

For information about these legacy components, see the guidance for previous versions of Enterprise Library. This is available on the Microsoft Enterprise Library site on MSDN®, and includes specific information for Creating an Instance of an Interceptable Target Class, details on using the create method, details on using the wrap method, and Specifying a Configuration Instance When Creating and Wrapping Objects. You can, in addition, use the configuration tools provided with this release of Enterprise Library to configure policy injection if you decide to use the legacy approach.

Also be aware that, even if you decide to continue to use the backwards compatibility techniques included in the Policy Injection Application Block, you must still make some changes to your existing application. The location of all of the call handlers (with the exception of the Performance Counter Handler) has changed, so you must ensure that you reference the appropriate assemblies and namespaces in your code.

The five call handlers you can use, and their assemblies and namespaces are:

- Authorization handler
 - Class name: **AuthorizationCallHandler**
 - Assembly: Microsoft.Practices.EnterpriseLibrary.Security.dll
 - Namespace: Microsoft.Practices.EnterpriseLibrary.Security.PolicyInjection
- Exception handling handler

- Class name: **ExceptionCallHandler**
- Assembly: Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.dll
- Namespace: Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.PolicyIn

• Logging handler

- Class name: LogCallHandler
- Assembly: Microsoft.Practices.EnterpriseLibrary.Logging.dll
- Namespace: Microsoft.Practices.EnterpriseLibrary.Logging.PolicyInjection

Validation handler

- Class name: ValidationCallHandler
- Assembly: Microsoft.Practices.EnterpriseLibrary.Validation.dll
- Namespace: Microsoft.Practices.EnterpriseLibrary.Validation.PolicyInjection

• Performance counter handler

- Class name: PerformanceCounterCallHandler
- Assembly: Microsoft.Practices.EnterpriseLibrary.PolicyInjection.dll
- Namespace: Microsoft.Practices.EnterpriseLibrary.PolicyInjection.CallHandlers

The **Caching handler** is no longer included in Enterprise Library due to concerns around the issues of cache contamination and other limitations previously documented. If you require the Caching handler, you can download the previous version from the Enterprise Library community Web site at http://www.codeplex.com/entlib/ and integrate it with Enterprise Library.

More information about the changes to the Policy Injection Application Block can be found in the section <u>Changes in This Release</u> in the introduction to this guidance. Information about migrating existing applications to use the current version of Enterprise Library can be found in the Migration Guide available from the Enterprise Library community Web site at http://www.codeplex.com/entlib/.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to

pagdoc@microsoft.com.

Copyright $\ensuremath{\text{@}}$ 2010 by Microsoft Corporation. All rights reserved.

The Security Application Block



Developers frequently write applications that must authorize users using one or more security providers such as Microsoft® Active Directory® directory service, Authorization Manager, Active Directory Lightweight Directory Services (AD LDS), and custom authorization providers. These applications may also need to cache authentication or authorization data for the duration of a logon session.

The Security Application Block simplifies these tasks by handling them in a consistent manner, abstracting the application code from the specific security providers. You can even change underlying providers through configuration without changing the underlying application code.

The Security Application Block provides code that will help you with the following scenarios:

- Authorization
- Caching security-related credentials

This section includes the following topics that will help you to understand and use the Security Application Block:

- What Does the Security Application Block Do? This topic provides a
 brief overview that will help you to understand what the block can do,
 and explains some of the concepts and features it incorporates. It also
 provides a simple example of the way that you can write code to use the
 block.
- When Should I Use the Security Application Block? This topic will help you to decide if the block is suitable for your requirements. It explains the benefits of using the block, and any alternative techniques you may consider. It also provides details of any limitations of the block that may affect your decision to use it.
- <u>Developing Applications Using the Security Application Block</u>. This topic explains how to configure the Security Application Block to perform common tasks and how to use the block in your applications.
- <u>Key Scenarios</u>. This topic demonstrates how to use the Security Application Block to perform the most typical security operations.
- Design of the Security Application Block. This topic explains the

- decisions that went into designing the Security Application Block and the rationale behind those decisions.
- Extending and Modifying the Security Application Block. This topic explains how to extend the block by creating your own providers and how to modify the source code.
- <u>Deployment and Operations</u>. This topic explains how to deploy and update the Security Application Block's assemblies and also contains information about configuration.

More Information

For more information, see the following patterns & practices guides:

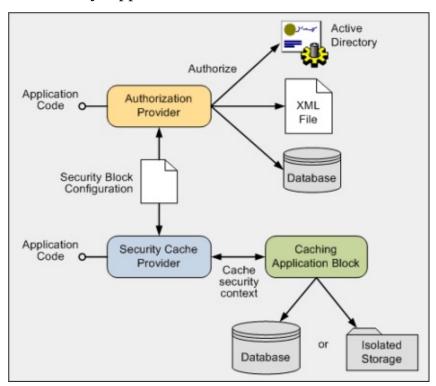
- Application Architecture for .NET: Designing Applications and Services
- .NET Data Access Architecture Guide
- Design Guidelines for Exceptions

To provide feedback, get assistance, or download additional content, please visit the Enterprise Library Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.



The Security Application Block allows you to perform two separate but related tasks. You can authorize users against a range of authorization providers, and/or cache a user's identity and security context for use throughout your application. The code required to use these features is simple, and the actual mechanics of accessing authorization systems and caching identities and security contexts are abstracted within the block. Often, you only need to write one line of code to perform common tasks. The following schematic shows the basic elements of the Security Application Block.



The Security Application Block exposes two interfaces that you can access in your code:

• An **Authorization Provider** interface, which exposes the single method named **Authorize** that takes an instance of an **IPrincipal** object containing details of the user's identity and roles. Depending on the way that you configure the block, the authorization can take place either through Windows® Authorization Manager (AzMan) against Active

- Directory, an XML file, or a database; or by using custom rules that you define and are stored as XML in the application configuration file.
- A **Security Cache Provider** interface, which exposes methods that allow you to save and retrieve a user's identity or security context as an **IIdentity** instance, **IPrincipal** instance, or ASP.NET **Profile** instance. Each cached identity or security context is identified by a token (by default a GUID, though you can create and use your own implementation of the **IToken** interface). The block stores this information in either a database or in Isolated Storage using the Caching Application Block. You can alternatively create a custom provider for the Caching Application Block and use it to cache the information in the location and using the techniques you implement in your provider.

Your application can use these interfaces to quickly and easily cache user identities and security contexts, obtain tokens that represent users, expire these users, and check if users are authorized to perform specific tasks or operations. However, to get the most from the block, you must understand the differences between the Windows **IIdentity** and **IPrincipal** interfaces and commonly used concrete implementations of these types.

An identity is represented by a concrete implementation of the **IIdentity** interface, usually a **WindowsIdentity**, **GenericIdentity**, **PassportIdentity**, or **FormsIdentity** depending on the authentication technique used. The **IPrincipal** interface provides the link between an identity and the roles for that identity. In ASP.NET, the current **IPrincipal** instance for a user is available from the **HttpContext.User** property. The methods that cache and retrieve user identity and security context accept an instance of a class that implements either the **IIdentity** or **IPrincipal** interface, or an ASP.NET **Profile** instance. For more information about Windows identities and security contexts, see <u>Principal and Identity Objects</u> and <u>Role-Based Security</u> on MSDN®.

■ Typical Usage of the Security Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

When Should I Use the Security Application Block?



The Security Application Block includes implementations of functionality that makes it easy to perform authorization, security-related caching, and session management. If your applications require the provided implementations, you can use the Security Application Block to provide this functionality. However, the block is also designed to be extensible and includes generic providers for each function. You can adapt the providers to meet your own security requirements.

■ Scenarios for the Security Application Block

Benefits of the Security	Application	Block
--------------------------------------------	--------------------	-------

■ Limitations of the Security Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Developing Applications Using the Security Application Block



This topic describes how you can use the Security Application Block to develop applications. It explains how to configure the block and incorporate it into your applications, and how to use the block for specific scenarios such as authorizing a user for a particular task. This topic assumes that you are using the Security Application Block in its original state, without extending it. (To learn how to add functionality, see Extending the Security Application Block.) This section includes the following topics:

- Entering Configuration Information
- Adding Application Code

All application blocks ship as binary assemblies and as source code. If you want to use the source code, you must compile. To learn how to compile the Enterprise Library source code, see <u>Building Enterprise Library from the Source Code</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\text{@}}$ 2010 by Microsoft Corporation. All rights reserved.

Entering Configuration Information



These procedures explain how to configure the Security Application Block. For details of the schema for the Security Application Block configuration, see Source Schema for the Security Application Block. You can also configure the block in code by using an alternate configuration source. For more information, see Advanced Configuration Scenarios and Using the Fluent Configuration API.

To add the Security Application Block

- 1. Open the configuration file. For more information, see <u>Configuring Enterprise Library</u>.
- 2. Click Add Security Settings on the Blocks menu.
- 3. The configuration tool automatically adds the **Security Settings** section with default settings. Click the chevron arrow at the right of the section heading to view **Security Settings** properties.
- 4. Click the **Authorization Providers** plus sign icon, point to **Add Authorization Providers**, then click the type of authorization rule provider you require and configure the provider as shown in the following procedures.

✓ Note:

The Windows Authentication Manager (AzMan) provider is available only if you have installed the required prerequisites, including the assembly **Microsoft.Interop.Security.AzRoles.dll**, and compiled the Security Application Block to include this provider. For more information, see <u>About the AzMan Provider</u> later in this topic.

- 5. (Optional) Click the **Security Caches** plus sign icon, point to **Add Security Caches**, then click the type of security cache provider you require and configure the provider as shown in the following procedures.
- 6. (Optional) In the properties pane of the **Security Settings** section, set the **Protection Provider** property. The text box drop-down provides three choices **(no protection),RsaProtectedConfigurationProvider**, and **DataProtectionConfigurationProvider**. The default is no protection. See Encrypting Configuration Data for information about

- the restrictions on using the **RsaProtectedConfigurationProvider**.
- 7. (Optional) In the properties pane, set the **Require Permission** property to **True** or **False**. The default is **True**.
- 8. (Optional) In the properties pane, set the **Default Authorization Provider** property. This is the authorization provider instance to use if one is not specified in the code. The default is none.
- 9. (Optional) In the properties pane, set the **Default Security Cache Provider** property. This is the security cache provider instance to use if one is not specified in the code. The default is none.

After you add the Security Application Block to the application configuration, you need to configure some or all of the following elements:

- Authorization Rule Provider
- AzMan Provider
- Custom Authorization Provider
- Security Cache
- Custom Security Cache Provider

■ Authorization Rule Provider

■ AzMan Provider

■ Custom Authorization Provider

■ Security Cache

■ Custom Security Cache Provider

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Source Schema for the Security Application Block
```

This topic lists the XML elements and attributes used to configure the Security Application Block. You can manually edit the XML data, but the Enterprise Library configuration tools greatly simplify this task. If you choose to manually edit the XML, use the schema information contained in this topic.

The configuration file has the following section-handler declaration.

The section-handler declaration contains the name of the configuration settings section and the name of the section-handler class that processes configuration data in that section. The name of the configuration settings section is **securityConfiguration**. The name of the section-handler class is **Microsoft.Practices.EnterpriseLibrary.Security.Configuration.SecuritySettir**

■ securityConfiguration Element

■ authorizationProviders Child Element

■ rules Child Element

• securityCacheProviders Child element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Adding Application Code



The Security Application Block is designed to support the most common scenarios for authorization and caching of security information. When you add your application code, refer to the scenarios in the <u>Key Scenarios</u> section, and select the ones that best match your situation. Use the code that accompanies the scenario as-is or adapt it as necessary.

First, you must prepare your application to use the Security Application Block.

To prepare your application

- 1. Add a reference to the Security Application Block assembly. In Visual Studio, right-click your project node in Solution Explorer, and then click **Add References**. Click the **Browse** tab, and then navigate to the location of the **Microsoft.Practices.EnterpriseLibrary.Security.dll** assembly. Select the assembly, and then click **OK** to add the reference.
- 2. Follow the same procedure to set a reference to the following assemblies:
 - Microsoft.Practices.EnterpriseLibrary.Common.dll
 - Microsoft.Practices.ServiceLocation.dll
 - Microsoft.Practices.Unity.dll
 - Microsoft.Practices.Unity.Interception.dll
- 3. If you intend to use security caching, set a reference to the assembly **Microsoft.Practices.EnterpriseLibrary.Security.Cache.CachingStore** You also need to add a reference to the Data Access Application Block assembly, **Microsoft.Practices.EnterpriseLibrary.Data.dll**, if you are using the **Database Cache Storage** store in the Caching Application Block.
- 4. (Optional) To use elements from the Security Application Block without fully qualifying the element reference, you can add the **using** statement (C#) or **Imports** statement (Visual Basic) to the top of your source code file. The following code shows how to add these statements for the **Microsoft.Practices.EnterpriseLibrary.Security** namespace.

C#

using Microsoft.Practices.EnterpriseLibrary.Secu

Visual Basic

Imports Microsoft.Practices.EnterpriseLibrary.Se

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Key Scenarios



This topic describes the most common situations developers must address when providing security functionality in their applications. Each scenario explains the task, describes a real-world situation where such a task might occur, and includes code that demonstrates how to use the Security Application Block to complete the task. The scenarios are the following:

- Obtaining a Temporary Token for an Authenticated User. This scenario illustrates how to use the **SaveIdentity** method to cache an authenticated identity and return a temporary token that serves as an alternative to user credentials for the duration of the user session. You can also use this technique to save a user principal or a user identity.
- Authenticating a User Using a Token. This scenario illustrates how to use the **GetIdentity** method to return an identity that has already been cached, when provided with a valid token. The same technique can be used to retrieve a user principal or user profile.
- Terminating a User Session (Expiring a Token). This scenario illustrates how to use the **ExpireIdentity** method to expire a token corresponding to an identity, when the user session ends. You can also use this technique to expire a user principal or a user profile.
- <u>Determining Whether a User Is Authorized to Perform a Task</u>. This scenario illustrates how to use the **Authorize** method of an authorization provider to perform authorization.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Obtaining a Temporary Token for an Authenticated User



An example of when you might want to obtain a temporary token for an authenticated user is when you want to improve the performance of your application by passing the token instead of frequently authenticating the same user during a single session. You can use the approach described here to save a user principal or a user identity in the security cache and obtain a token that represents the user's authenticated identity.

■ Typical Goals

■ Solution

■ Using SaveIdentity

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Authenticating a User Using a Token**



An example of when you might want to use a temporary token for authentication is when you want to improve the performance of your application by passing the token instead of frequently authenticating a user during a single session. You can use the approach described here to retrieve a saved user principal or a user identity from the security cache using a token you previously obtained that represents the user's authenticated identity.

■ Typical Goals

■ Solution

■ Using GetIdentity

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Terminating a User Session (Expiring a Token)**



An example of when you may want to expire a token is when you want to make sure that the token cannot be used by an attacker after the user logs out. You can use the approach described here to expire a user principal or a user identity.

■ Typical Goals

■ Solution

■ Using ExpireIdentity

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Determining Whether a User Is Authorized to Perform a Task



A common security requirement is the need to authorize users to perform tasks. The Security Application Block helps by standardizing access to authorization providers such as the **AzManAuthorizationProvider** or the **AuthorizationRuleProvider**, or to authorization rules stored within the application configuration.

■ Typical Goals

■ Solution

■ Using Authorize

■ Usage Notes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Design of the Security Application Block



The Security Application Block addresses the following areas:

- Authorization
- Security-related caching

■ Design Goals

■ Design Highlights

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Designing for Simplified Authorization



Uniform implementations make code easier to understand, more predictable, and easier to maintain. However, developers can implement authorization in applications in many different ways. For example, they may have to use an approach that conforms to the security policies of their organizations. Alternatively, they may use approaches that suit the needs of particular departments or of the applications themselves.

The Security Application Block encapsulates the logic that performs authorization operations into a single interface that specifies only a small number of methods. These methods can be used by different authorization providers. This means that applications that use the Security Application Block are consistent in the ways that they authorize users to perform tasks. By using the Security Application Block, this consistency remains across single projects, multiple projects, or enterprise-scale solutions.

■ Design Implications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Extending and Modifying the Security Application Block



In its original state, the Security Application Block works well for typical security scenarios. However, there may be times when you have to customize some of the block's behavior to better suit your application's particular requirements. There are two ways to do this. You can extend the Caching Application Block using the built-in extension points. You can also modify the block by making changes to its source code. For more details, see the following topics:

- Extending the Security Application Block
- Extending and Modifying Enterprise Library

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Extending the Security Application Block



You extend the Security Application Block through designated extension points. Typically, these are custom classes, written by you, that implement a particular interface or derive from an abstract class. Because these custom classes exist in your application space, you do not have to modify or rebuild the block. Instead, you designate your extensions using configuration settings.

You can extend the block by adding a new type of Authorization Provider or by adding a new security cache provider that integrates with your chosen caching mechanism. The following table lists the interfaces and base classes that you can use to extend the block.

Custom Provider or Extension	Interface or Base Class
Authorization Provider	AuthorizationProvider
Security Cache Provider	ISecurityCacheProvider

For detailed information about how to integrate custom providers with the Enterprise Library configuration system and configuration tools see <u>Creating Custom Providers for Enterprise Library</u>.

■ Creating an Authorization Provider

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Deployment and Operations



Two of an administrator's main tasks will be to make sure that the initial deployment of the Security Application Block is planned and managed, and that subsequent updates are deployed with minimal impact to existing applications that use the block. For details of deploying and updating Enterprise Library and the application blocks, see <u>Deploying Enterprise Library</u>.

In addition, administrators must decide whether they want to use the instrumentation exposed by the block. For details of how to enable and disable instrumentation, see Enabling Instrumentation. For information about the instrumentation contained within the Security Application Block, see the following topics:

- Security Application Block Performance Counters
- Security Application Block Event Log Entries

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Security Application Block Performance Counters



The following table describes the Security Application Block performance counters.

Performance counter name	Description
Authorization Requests Denied/sec	The rate at which authorization requests were denied.
Authorization Requests/sec	The rate at which authorization requests were received.
Security Cache Reads/sec	The rate at which security cache reads were requested.
Total Authorization Requests	The total number of authorization requests received.
Total Authorization Requests Denied	The total number of authorization requests denied.
Total Security Cache Reads	The total number of security cache reads requested.

A rate counter samples an increasing count of events over time and divides the values by the change in time to display a rate of activity. For more information about performance counters, see Overview of Performance Monitoring in the .NET Framework Class Library on MSDN.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Security Application Block Event Log Entries



This topic lists the Security Application Block event log entries. The listener is the class that raised the event.

■ Configuration Failure Event

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Validation Application Block



Any application that accepts input either from users or from other systems must ensure that the information is valid in terms of some set of rules that you specify. For example, when processing an order, you may need to check that a customer's phone number has the correct number of digits or that a date falls within a particular range. In addition, if the validation fails, you may need to send an error message that explains what is wrong.

The Enterprise Library Validation Application Block provides useful features that allow developers to implement structured and easy-to-maintain validation scenarios in their applications. In addition, the Validation Application Block includes adapters that allow you to use the application block with the following technologies:

- ASP.NET
- Windows® Communication Foundation (WCF)
- Windows Presentation Foundation (WPF)
- Windows Forms

This section includes the following topics that will help you to understand and use the Validation Application Block:

- What Does the Validation Application Block Do? This topic provides a brief overview that will help you to understand what the block can do, and explains some of the concepts and features it incorporates. It also provides a simple example of how you can write code to use the block.
- When Should I Use the Validation Application Block? This topic will help you to decide if the block is suitable for your requirements. It explains the benefits of using the block, and alternative techniques you may consider.
- Developing Applications Using the Validation Application Block. This topic explains how to include the Validation Application Block in your applications and how to configure it. It also contains more detailed information, such as how to create custom message templates and information on how validation works with inheritance.
- <u>Key Scenarios</u>. This topic shows different ways to use the Validation Application Block in your own applications.

- <u>Design of the Validation Application Block</u>. This topic includes a class diagram of the Validation Application Block.
- Extending and Modifying the Validation Application Block. This topic explains how to extend the application block by adding custom validators and attributes. It also contains advice about how to modify the source code.
- <u>Deployment and Operations</u>. This topic explains how to deploy and update the application block assemblies. It also explains the application block's instrumentation.

More Information

For related information, see the following patterns & practices guides:

- How To: Protect From Injection Attacks in ASP.NET
- Security Practices: ASP.NET Security Practices at a Glance
- Microsoft Application Architecture Guide, 2nd Edition

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

What Does the Validation Application Block Do?



Unlike many other validation mechanisms and libraries, which use separate validation controls to validate individual UI elements such as text boxes, the Validation Application Block is designed to allow you to easily validate instances of objects. This means that you can execute validation at any point in your application, and repeat it when required. For example, you can validate an instance of a class populated with values from the UI, validate an instance of the class populated with values received from a Web service, and validate an instance of the class as it passes between layers of your application.

The block provides a library of classes named validators, which implement functionality for validating .NET Framework data types. You can also group validators together in a rule set. A rule set allows you to validate a complex object or graph by composing different validators of different types and applying them to elements in the object graph. Examples of these elements include fields, properties, and nested objects.

You configure validation requirements for specific classes by defining the set of validators and the rules to apply when validating parameter and property values for instances of that class. Then, in many situations, you can validate an instance of the class with a single line of code.

You can define validation rules and carry out validation in the following ways:

- By using configuration to define rule sets for specific classes. These rules sets are stored in your application configuration file, and can be created using the graphical configuration tools.
- By adding attributes to members of your classes to define individual rules for public, readable parameters and properties that specify rule sets or individual validation rules. These may be attributes defined within the Validation Application Block that directly target the validators provided with the block, or .NET Data Annotation attributes. The Validation Application Block works with both of these types of attributes.
- By adding code to your classes that perform self validation of the object parameters or properties. This is a useful way to implement very complex validation rules that depend on the environment or external factors.

• By using code to create instances of validators and then execute validation on demand.

✓ Note:

The Validation Application Block only applies rules specified for the return values for public methods that have no parameters. To validate parameters when a method is invoked, or to validate return values for methods that accept parameters, you can use <u>The Validation Handler</u>. This handler validates parameter values based on the rules for each parameter's type and any validation attributes on the parameters themselves.

The Validation Application Block contains a wide variety of validators, and you can easily create custom validators yourself for your own specific scenarios. As examples of the validators, the block includes a validator that checks for null strings and another validator that checks whether a number falls within a specified range. There are also special validators named the And Composite Validator and the Or Composite Validator. If you create an And Composite Validator, which aggregates other validators, all validators in the composite validator must return true for successful validators in the composite validator must return true for successful validation.

The Validation Application Block also includes adaptors that you can use in your application UI that support the same type of implementation as ASP.NET, Windows Presentation Foundation (WPF), and Windows Forms validation controls to provide feedback and information to users when validation errors occur. In addition, the block includes an adapter that makes it easy to use in WCF applications.

■ Example Application Code

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

When Should I Use the Validation Application Block?



You should consider using the Validation Application Block if you want to encapsulate validation good practice into easily maintainable code that you can reuse. Encapsulation also allows you to separate the application code from the validation logic. In some situations, you may be able to update the validation logic without redeploying the application.

In addition, consider using the block when your validation code must work across multiple layers of the application's architecture. By defining rules in the configuration for each segment, you can reuse the same rule sets in multiple locations within your code.

Scenarios for	the	Validation	Application	Block

■ Benefits of the	Validation A	pplication	Block
--------------------------	--------------	------------	--------------

■ Alternatives to the Validation Application Block

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Developing Applications Using the Validation Application Block



This section describes how to use the Validation Application Block in your applications. It explains how to configure the application block, and how to incorporate the application block into your application. This section includes the following topics:

- Entering Configuration Information
- Adding Application Code
- Using the Validation Block Validators
- <u>Understanding Common Validator Properties</u>
- Understanding Validation Results
- How Validators Are Created
- Validation and Inheritance

All application blocks ship as binary assemblies and as source code. If you want to use the source code, you must compile it. To learn how to compile the Enterprise Library source code, see <u>Building Enterprise Library from the Source Code</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community <u>Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Entering Configuration Information



These procedures explain how to configure the Validation Application Block with the configuration tools. The Validation Application Block also allows you to use attributes and code to perform many of the tasks described here. For information about how to use attributes and code, see <u>Using the Validation Block Validators</u>. For information about properties that are associated with validators such as **Tag** and **Message Template**, see <u>Understanding Common Validator Properties</u>.

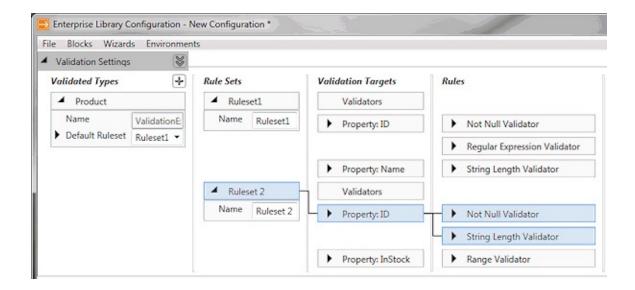
This procedure explains how to add the Validation Application Block to the configuration file. For details of the schema for the Validation Application Block configuration, see <u>Source Schema for the Validation Application Block</u>.

To add the Validation Application Block by using the configuration tool

- 1. Open the configuration file. For more information, see <u>Configuring Enterprise Library</u>.
- 2. Open the **Blocks** menu and then click **Add Validation Settings**.
- 3. (Optional) If you want to encrypt the configuration file, click the chevron expander arrow to show the properties, and then select a protection provider from the drop-down list in the **Protection Provider** field.

The next procedure shows how to define a rule set for the members of a type. It assumes that you have already added the Validation Application Block to your configuration. Members of a type that you will validate can be fields, methods, or properties. Note that it is possible for a type to have multiple rule sets associated with each type you configure. For example, in the following screenshot, the **Product** type has two rule sets, **Ruleset1** and **Ruleset2**.

- **Ruleset1** specifies a not null validator and a regular expression validator for the **ID** property, and a string length validator for the **Name** property.
- **Ruleset2** specifies a not null validator and a string length validator for the **ID** property, and a range validator for the **InStock** property.



When you click on an item in the configuration tool, related items are highlighted and links appear between them, as shown for the **Property: ID** item in the screenshot.

To define a rule set for members of a type by using the configuration tool

- 1. If the settings for the **Validation Settings** section are not visible, click the child expander arrow to the left of **Validation Settings**.
- 2. Click the plus sign icon in **Validated Types** and click **Add Type to Validate**.
- 3. In the type selector dialog box, expand the assembly you want to use. To filter the list, type part of the name in the **Type name** edit box; for example, type "string" to filter for all classes containing the word "string". If the assembly is not shown in the list, click **Add from GAC** (the global assembly cache) or **Add from File** and navigate to it. Select the type you want to validate and click **OK**.
- 4. To define a rule set, right-click on the type in the **Validated Types** column and then click **Add Validation Ruleset**. This adds a rule set with the default name **Validation Ruleset** and a **Validators** item that you can use to validate the type itself.
- 5. Edit the **Name** property of the rule set as required.
- 6. To add a validator that applies to an instance of a class as a whole, rather than to individual members of that class, right-click the **Validators** item in the **Validation Targets** column, point to **Add Validators**, and then click the validator you want to apply. Repeat this

- step to add additional validators that will be applied at the type level if required. Typically you will use this feature for only a not null validator, object collection validator, composite validator, or a custom validator.
- 7. To select the individual members of a type to be validated, you can use either of the following methods:
 - Right-click on the heading of the rule set item, and then click
 Add Field to Validate, Add Method to Validate, or Add
 Property to Validate. Then enter the name of the field, method, or property in the Validation Targets pane.
 - Alternatively, you can select several members of a type simultaneously. Right-click on the rule set and click **Select Members**. In the **Member Selector** dialog box, select the
 Properties, **Methods**, and/or **Fields** that you want to validate, and then click **OK**.
- 8. To add a validator for a type member, right-click the heading of a field, method, or property of a member for the type in the **Validation Targets** column, point to **Add Validators**, and then click the validator you want to apply. Repeat this step to add additional validators that will be applied to individual type members as required.
- 9. Edit the properties of each validator you added to the configuration:
 - (Optional) Edit the default **Name** property.
 - Specify the validation error message. Enter either a Message
 Template (which may include the validation message tokens
 described in <u>Understanding Common Validator Properties</u>), or
 set the Template Resource Name and Template Resource
 Type properties if you want to load the message template from
 a resources file. To use a resources file, enter the name of the
 resource for the Template Resource Name then click on the
 ellipsis button (...) in the Template Resource Type property
 and use the type selector to locate and select the resources file.
 - (Optional) If you want the validator to operate in reverse, so that the validator will return false (failed validation) when the validation rule is satisfied, and true (no error) when the validation test fails, set the **Negated** property to **True**. The default is **False**.
 - (Optional) If you want to pass an additional text value to the

- application when validation fails, enter this text as the **Tag** property. You can filter validation results on the values you specify for this property.
- Enter values for the remaining validator properties. The
 properties available differ for each type of validator. For a list of
 properties for each type of validator, see <u>Using the Validation</u>
 <u>Block Validators</u>.

The next procedure explains how to define an **AndCompositeValidator** or an **OrCompositeValidator**. Composite validators contain individual validators that are combined with a Boolean **AND** or **OR** operation. For example, the following screenshot shows an Or composite validator applied to the **ID** property of a type in **Ruleset1**. The validation specifies that either the rule defined by a property comparison validator or the rule defined by a range validator must be satisfied for validation of the **ID** property value to succeed.



You can also nest composite validators to create complex logic for a member, such as (A OR (B AND C)).

To define composite validators

- Right-click the member of the type you want to validate, or on the Validator item to add a validator for the type itself, and click Add Validators. Then click either Add And Composite Validator or Add Or Composite Validator.
- 2. Right-click the And Composite Validator or the Or Composite Validator you added, point to **Add Validators** and then click one of the validators that will be a part of the composite validator. Repeat this step to add additional validators as required.
- 3. Edit the properties of the And Composite Validator or the Or Composite Validator, and edit the properties of each validator you add to the composite validator:
 - (Optional) Edit the default **Name** property.

- Specify the validation error message. Enter either a Message
 Template (which may include the validation message tokens
 described in <u>Understanding Common Validator Properties</u>), or
 set the Template Resource Name and Template Resource
 Type properties if you want to load the message template from
 a resources file. To use a resources file, enter the name of the
 resource for the Template Resource Name then click on the
 ellipsis button (...) in the Template Resource Type property
 and use the type selector to locate and select the resources file.
- (Optional) If you want the validator to operate in reverse, so that the validator will return false (failed validation) when the validation rule is satisfied, and true (no error) when the validation test fails, set the **Negated** property to **True**. The default is **False**.
- (Optional) If you want to pass an additional text value to the application when validation fails, enter this text as the **Tag** property.
- Enter values for the remaining validator properties. The properties available differ for each type of validator. For a list of properties for each type of validator, see <u>Using the Validation</u> <u>Block Validators</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

This topic lists the elements and attributes used to configure the Validation Application Block. The configuration file has the following section-handler declaration:

XML

The section-handler declaration contains the name of the configuration settings section and the name of the section-handler class that processes configuration data in that section. The name of the configuration settings section is **validation**. The name of the section-handler class is **Microsoft.Practices.EnterpriseLibrary.Validation.Configuration.ValidationS**

■ validation Element

■ EncryptedData Element

• type Element

■ ruleset Element

• fields Element

■ methods Element

■ properties Element

■ validator Element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Adding Application Code



The following procedure explains how to incorporate the Validation Application Block into your application.

To prepare your application

- 1. Add a reference to the Validation Application Block assembly. In Visual Studio, right-click your project node in Solution Explorer, and then click **Add Reference**. Click the **Browse** tab and find the location of the **Microsoft.Practices.EnterpriseLibrary.Validation.dll** assembly. Select the assembly, and then click **OK** to add the reference.
- 2. Use the same procedure to set a reference to the following assemblies:
 - Microsoft.Practices.EnterpriseLibrary.Common.dll
 - Microsoft.Practices.ServiceLocation.dll
 - Microsoft.Practices.Unity.dll
 - Microsoft.Practices.Unity.Interception.dll
- 3. If you are using the ASP.NET, WPF, Windows Forms, or WCF integration assemblies, add one of the following references as appropriate.
 - Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WinI
 - Microsoft.Practices.EnterpriseLibrary.Validation.Integration.Aspl
 - Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WPF
 - Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCI
- 4. (Optional) To use elements from the Validation Application Block without fully qualifying the type with the namespace, add the following using statements (C#) or Imports statements (Visual Basic) to the top of your source code file.

C#

using Microsoft.Practices.EnterpriseLibrary.Valiusing Microsoft.Practices.EnterpriseLibrary.Vali

Visual Basic

Imports Microsoft.Practices.EnterpriseLibrary.Va

Imports Microsoft.Practices.EnterpriseLibrary.Va

✓ Note:

For Visual Basic projects, you can use the References page of the Project Designer to manage references and imported namespaces. To access the References page, select a project node in Solution Explorer. On the **Project** menu, click **Properties**. When the Project Designer appears, click the **References** tab.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Using the Validation Block Validators



The Validation Application Block includes classes named validators, which derive from the **Validator** class. There is a generic version of this class named **Validator**<T>.

Every validator is associated with a specific type. For example, the **StringLengthValidator** class checks to see if a **System.String** value has a length within a predefined range.

There are four ways that you can associate validators with your types:

- You can use configuration. For more information, see <u>Entering Configuration Information</u>.
- You can use attributes. For more information, see <u>Using Validation Block Attributes</u> and <u>Using Data Annotation Attributes</u>.
- You can use a combination of configuration and attributes.
- You can use self validation, which means that you include validation logic within the object you want to validate. For more information, see <u>Using Self Validation</u>.

You can also instantiate validators within your code without associating them with a specific type. For more information, see <u>Creating Validators</u> <u>Programmatically</u>.

The following sections describe the validator types that are included with the Validation Application Block. These validators are the following:

- And Composite Validator
- Contains Characters Validator
- Date Time Range Validator
- <u>Domain Validator</u>
- Enum Conversion Validator
- Not Null Validator
- Object Collection Validator
- Object Validator
- Or Composite Validator
- Property Comparison Validator
- Range Validator

- Regular Expression Validator
- Relative Date Time Validator
- <u>String Length Validator</u>
- Type Conversion Validator
- Single Member Validators

Each entry contains examples for how to use the validator with attributes and with code.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **And Composite Validator**



Class Name: AndCompositeValidator

Attribute Name: ValidatorCompositionAttribute

Configuration tool name: And Composite Validator

Description

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Contains Characters Validator



Class Name: ContainsCharactersValidator

Attribute Name: ContainsCharactersValidatorAttribute

Configuration tool name: Contains Characters Validator

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Date Time Range Validator**



Class Name: **DateTimeRangeValidator**

Attribute Name: **DateTimeRangeValidatorAttribute**

Configuration tool name: Date Time Range Validator

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Domain Validator



Class Name: **DomainValidator**<T>

Attribute Name: **DomainValidatorAttribute**

Configuration tool name: **Domain Validator**

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Enum Conversion Validator**



Class Name: EnumConversionValidator

Attribute Name: EnumConversionValidatorAttribute

Configuration tool name: **Enum Conversion Validator**

■ Examples

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Not Null Validator**



Class Name: NotNullValidator

Attribute Name: NotNullValidatorAttribute

Configuration tool name: Not Null Validator

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Object Collection Validator**



 $Class\ Name: \textbf{ObjectCollectionValidator}$

Attribute Name: ObjectCollectionValidatorAttribute

Configuration tool name: Object Collection Validator

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Object Validator**



Class Name: ObjectValidator

Attribute Name: **ObjectValidatorAttribute**

Configuration tool name: **Object Validator**

■ Example

■ Differences between the Object Validator and the Factory-Created Validators

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Or Composite Validator**



Class Name: OrCompositeValidator

Attribute Name: ValidatorCompositionAttribute

Configuration tool name: Or Composite Validator

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Property Comparison Validator**



Class Name: **PropertyComparisonValidator**

Attribute Name: **PropertyComparisonAttribute**

Configuration tool name: Property Comparison Validator

Description

• Properties

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Range Validator



Class Name: RangeValidator<T>

Attribute Name: RangeValidatorAttribute

Configuration tool name: Range Validator

Description

• Properties

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Regular Expression Validator**



Class Name: RegexValidator

Attribute Name: **RegexValidatorAttribute**

Configuration tool name: Regular Expression Validator

Description

• Properties

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Relative Date Time Validator**



Class Name: RelativeDateTimeValidator

Attribute Name: RelativeDateTimeValidatorAttribute

Configuration tool name: Relative Date Time Validator

Description

• Properties

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **String Length Validator**



Class Name: StringLengthValidator

Attribute Name: **StringLengthValidatorAttribute**

Configuration tool name: String Length Validator

Description

• Properties

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Type Conversion Validator**



Class Name: TypeConversionValidator

Attribute Name: TypeConversionValidatorAttribute

Configuration tool name: Type Conversion Validator

Description

• Properties

Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Single Member Validators
```

The Validation Application Block contains three validators that you can use to validate individual members of types, instead of validating the entire type using attributes or rule sets. While not a common scenario, this technique may be useful when integrating with other frameworks such as WPF and Windows Forms. The three validators are:

- **FieldValueValidator**. Use this validator to validate a field of a type.
- **MethodReturnValueValidator**. Use this validator to validate the return value of a method of a type.
- **PropertyValueValidator**. Use this validator to validate the value of a property of a type.

For example, you can programmatically create a validator for an instance of a class named **MyClass** that validates the value of a property named **MyProperty** using a regular expression validator as shown here.

```
Validator propValidator = new PropertyValueValidator
New RegexValidator("some
MyClass myInstance = new MyClass();
myInstance.MyProperty = "Some value";
ValidationResults results = propValidator.Validate(myI)

Visual Basic

Dim propValidator As New PropertyValueValidator(Of MyClass)
New RegexValidator("some-regulation myInstance As New MyClass()
myInstance.MyProperty = "Some value"
Dim results As ValidationResults = propValidator.Valid
```

That second parameter to the constructor is the validator to use for the property value. You can also create a composite validator from a combination of validators, and specify this composite validator in the code above. A similar

technique can be used with the **FieldValueValidator** and **MethodReturnValueValidator**.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Understanding Common Validator Properties



There are a number of properties defined for the **Validator** class. These properties are also exposed by the **ValidatorAttribute** class, which means that they can be used with any attribute that derives from this class.

These properties are the following:

- **MessageTemplate**. For more details, see <u>Using the Message Template</u> <u>Property</u>.
- MessageTemplateResourceName and MessageTemplateResourceType . For more details, see <u>Using the Message Template Resources</u>.
- **Negate**. For more details, see <u>Using the Negate Property</u>.
- **Tag**. For more details, see <u>Using the Tag Property</u>.
- **Ruleset**. For more details, see <u>Using the Ruleset Property</u>.

■ Using the Message Template Property

■ Using the Message Template Resources

■ Understanding	Message	Template	Tokens

■ Using the Negate Property

■ Using the Tag Property

■ Using the Ruleset Property

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Understanding Validation Results

The **ValidationResults** type is a collection of **ValidationResult** objects. Each **ValidationResult** object contains a report that includes references to the **Validator** instance, the object that was validated, and error messages.

The following table lists the properties and methods of the **ValidationResults** collection class.

Member	Description
AddAllResults	Takes a collection of ValidationResult instances and adds them to an existing ValidationResults instance.
AddResult	Creates a new ValidationResult instance and adds it to an existing ValidationResults instance.
Count	Returns the number of ValidationResult instances in the collection.
FindAll	Filters a ValidationResults instance and eliminates results that do not match the value of the Tag property that was specified for the validator. This method returns a new ValidationResults object that contains the required set of ValidationResult instances and leaves the original ValidationResults instance unchanged.
IsValid	This property returns true if validation succeeded or false if validation failed.

The following table lists the properties of the **ValidationResult** class.

Property	Description
Key	This is a name that describes the location of the validation result. It contains the name of the member that is associated with the validator. It is null if the validator is defined at the type level.
NestedVlidationResults	The nested validation results for a composite

	failed validation.
Message	This is a message that describes the validation failure.
Tag	This is a value supplied by the user as the Tag property of the validator. Typically, it is used for categorization or filtering. The tag can be supplied through the constructor but it is typically set either by using a property in the validation attributes or with configuration.
Target	This is the object to which the validation rule was applied.
Validator	This is the validator that performed the validation.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

How Validators Are Created



When the application block creates a validator, it uses a set of requirements to determine which types and members of those types can be associated with that validator. (Members can be methods, properties, and fields.)

In general, when the application block creates a validator, it evaluates the associated type and its members. Only members that have the following characteristics can be associated with a validator:

- They must be public members.
- Methods must be non-void and take no parameters.
- Properties must be readable.

Self validation has a different set of requirements. They are the following:

- Self validation applies only to methods.
- A self validation method can be non-public.
- The method signature must be **void** [method](ValidationResults).
- Self validation can apply to inherited methods but not to methods that are private and inherited.

■ Creating Validators with Configuration	

■ Creating Validators with Attributes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Validation and Inheritance
```

If you use inheritance, you need to know how the validation rules are applied throughout the class hierarchy. Here is an example of a simple hierarchy, where class **PreferredCustomer** inherits from class **Customer**. (The validator attributes used in the example, such as **CustomerNameValidator** refer to custom validators and are not validators included with the Validation Application Block.)

```
C#
                                                     Copy Code
public class Customer
  [CustomerNameValidator]
  public string Name
    get { ... } set { ... }
  }
  [DiscountValidator]
  public virtual double Discount
    get { ... } set { ... }
}
public class PreferredCustomer : Customer
{
  [PreferredDiscountValidator]
  public override double Discount
    get { ... } set { ... }
```

```
Visual Basic
```

```
Public Class Customer
  <CustomerNameValidator()> _
  Public Property Name(ByVal _name As String)
    Get
    End Get
    Set(ByVal value)
      ٠...
    End Set
  End Property
  <DiscountValidator()> _
  Public Overridable Property Discount(ByVal _discount
    Get
    End Get
    Set(ByVal value)
      ٠...
    End Set
  End Property
End Class
Public Class PreferredCustomer
  Inherits Customer
  <PreferredDiscountValidator()> _
  Overrides Public Property Discount(ByVal _discount /
    Get
    End Get
    Set(ByVal value)
      . . . .
    End Set
  End Property
```

End Class

In this example, the **PreferredCustomer** class derives from the **Customer** class, and it also overrides the **Discount** property.

There are two rules for how validators work within a class hierarchy:

- If a derived class inherits a member and does not override it, the member's validators from the base class apply to the derived class.
- If a derived class inherits a member but overrides it, the member's attributes from the base class do not apply to the derived class.

In this example, the **CustomerNameValidator** attribute applies to the **PreferredCustomer** class, but the **DiscountValidator** attribute does not. Instead, the **PreferredDiscountValidator** attribute applies.

If this is not the desired behavior, you can use validators of base classes to check instances of derived classes. The following code example shows how to do this. It assumes that you have resolved an instance of the Validation Application Block **ValidatorFactory** class and stored it in a variable named **valFactory**.

C#

Validator<Customer> customerValidator = valFactory.Cre PreferredCustomer myPreferredCustomer = new PreferredC // Set properties of PreferredCustomer here ValidationResults r = customerValidator.Validate(myPre

Visual Basic

Dim customerValidator As Validator(Of Customer) = valF Dim myPreferredCustomer As PreferredCustomer = New Pre 'Set properties of PreferredCustomer here Dim r As ValidationResults = customerValidator.Validat

This example validates a **PreferredCustomer** object. However, the validation is based on the attributes of the **Customer** base class. The validation rules defined

on the **PreferredCustomer** class are not applied.

You can use the **CreateValidator**(*Type*) overload of the **ValidatorFactory** class to create a validator that is specific to a class that you provide at run time.

```
C#
public ValidationResults CheckObject(object obj)
  if (obj != null)
    Validator v = valFactory.CreateValidator(obj.GetTy
    return v.Validate(obj);
  }
  else
    return null;
}
Visual Basic
Public Function ValidationResults(ByVal obj As Object)
  If Not obj Is Nothing Then
    Dim v As Validator = valFactory.CreateValidator(ot
    Return v. Validate(obj)
  Else
    Return Nothing
  End If
End Function
```

This example creates a validator based on the run time type of the input argument to the **CheckObject** method.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Key Scenarios



This section describes the most common situations developers must address when using the Validation Application Block. The following scenarios are included:

- Validating Objects
- Creating Validators Programmatically
- <u>Using Validation Block Attributes</u>
- <u>Using Data Annotation Attributes</u>
- Defining Attributes in Metadata Classes
- <u>Using Self Validation</u>
- Integrating with ASP.NET, WPF, Windows Forms, and WCF

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Validating Objects



You can validate an object once you have obtained a reference to a validation factory and attached the validators and the rule sets. The easiest way to obtain a reference to a validation factory is through the Enterprise Library container, as described in Creating and Referencing Enterprise Library Objects. This topic demonstrates the following techniques:

- Creating Validator Instances
- Specifying the Location of Validation Rules
- Validating an Object
- Validating Objects Using Rule Sets

✓ Note:

In previous releases of Enterprise Library, you could use the static **Validation** facade to validate objects without first creating a validator. You could also use the static **ValidationFactory** class (as opposed to the replacement non-static **ValidatorFactory** class) to create validator instances. These approaches are supported in this release for backwards compatibility with existing application code. However, to take advantage of benefits available when using dependency injection, you should use the techniques described in this section. For information about using the static facades, see the **Enterprise Library documentation** on MSDN.

■ Creating Validator Instances

■ Specifying the Location of Validation Rules

■ Validating an Object

■ Validating Objects Using Rule Sets

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Creating Validators Programmatically
```

Using attributes to create rule sets is only possible if you have the source code for the class that you want to validate. In some cases, this may not be the case. Another team, or a utility such as the Web Services Description Language Tool (Wsdl.exe), may create the class—and you have only the binary assembly.

In addition, you may want to validate individual values, rather than entire objects. For both of these scenarios, you can create validators programmatically. This is shown in the following code example.

In this example, the first line uses **new** to create an instance of the **RegExValidator** class with the parameter value set to the @"\w+([-.]\w+)*\.\w+([-.]\w+)*" regular expression string. The variable **emailAddressValidator** contains the reference to this instance. The second line uses **new** to create an instance of the **AndCompositeValidator**

class. The variable **shortStringValidator** contains the reference to this instance. You can use the **AndCompositeValidator** class to create a composite validator. In this example, the composite validator contains a **NotNullValidator** instance and a **StringLengthValidator** instance.

After you programmatically create a **Validator** object, you can use the **Validate** method to validate an object as described in the topic <u>Validating Objects</u>. Notice how the second call to the **Validate** method uses the overload that accepts an existing instance of the **ValidationResults** class and adds any validation errors it finds to the list.

As an alternative to creating validators by executing the constructor, you can resolve individual validators through the Enterprise Library Container. If you specify a name when you resolve the instance, this is interpreted as the name of the rule set for that validator to use when validating objects. You must add the Validation Block Extension to the container to use this approach.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\mathbb{C}}$ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0 **Using Validation Block Attributes**



Validation attributes (both the built-in Validation Application Block attributes and those described in the topic <u>Using Data Annotation Attributes</u>) can be used with various targets that include classes, fields, properties, methods, and (in limited cases) parameters. For information, see <u>Validation Attribute Targets</u>. There is also a set of attributes that allow you to change the behavior of other attributes. These are discussed in <u>Validation Modifier Attributes</u>. You can also specify the attributes you want to use in a separate metadata class. For more details of this, see <u>Defining Attributes</u> in <u>Metadata Classes</u>.

■ Using Validation	Block Attribut	tes to Define Va	alidation Rule Sets

■ Validation Attribute Targets

■ Validation Modifier Attributes

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Using Data Annotation Attributes
```

The System.ComponentModel.DataAnnotations namespace in the .NET Framework contains a series of attributes that you can add to your classes and class members to signify metadata for these classes and members. They include a range of validation attributes that you can use to apply validation rules to your classes, in much the same way as you can with the Validation block attributes. For example, the following shows how you can use the Range attribute to specify that the value of the property named **OnOrder** must be between 0 and 50.

```
C#
[Range(0, 50, ErrorMessage = "Quantity on order must be public int OnOrder { get; set; }

Visual Basic

<Range(0, 50, ErrorMessage:="Quantity on order must be public Property OnOrder() As Integer
...
End Property</pre>
```

Compared to the validation attributes provided with the Validation block, there are some limitations when using the validation attributes from the DataAnnotations namespace:

- The range of supported validation operations is less comprehensive, though there are some new validation types available in version 4.0 of the .NET Framework that extend the range. However, some validation operations such as property value comparison, enumeration membership checking, and relative date and time comparison are not available when using data annotation validation attributes.
- There is no capability to use **Or** composition, as there is with the Or composite validator in the Validation Application Block. The only composition available with data annotation validation attributes is the

And operation.

- You cannot specify rule sets names, and so all rules implemented with data annotation validation attributes belong to the default rule set. You will see more details about rule sets later in this chapter.
- There is no simple built-in support self-validation, as there is in the Validation block.

☑ Note:

Data Annotation validators can only be applied to properties.

You can, of course, include both data annotation and Validation block attributes in the same class if you wish, and implement Self validation using the Validation block mechanism in a class that contains data annotation validation attributes. The validation methods in the Validation block will process both types of attributes.

For more information about data annotations, see System.ComponentModel.DataAnnotations Namespace (.NET Framework version 3.5) and System.ComponentModel.DataAnnotations Namespace (.NET Framework version 4.0).

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{^{\odot}}$ 2010 by Microsoft Corporation. All rights reserved.

```
Microsoft Enterprise Library 5.0

Defining Attributes in Metadata Classes

•
```

In some cases, you may want to locate your validation attributes (both Validation block attributes and .NET Data Annotation validation attributes) in a separate file to the one that defines the class that you will validate. This is a common scenario when you are using tools that generate the class files, and would therefore overwrite your validation attributes. To avoid this you can locate your validation attributes in a separate file that forms a partial class along with the main class file. This approach makes use of the **MetadataType** attribute from the System.ComponentModel.DataAnnotations namespace.

You apply the **MetadataType** attribute to your main class file, specifying the type of the class that stores the validation attributes you want to apply to your main class members. You must define this as partial class, as shown here. The only change to the content of this class compared to the attributed versions you saw in the previous sections of this chapter is that it contains no validation attributes.

```
[MetadataType(typeof(ProductMetadata))]
public partial class Product
{
    ... Existing members defined here, but without attri
}

Visual Basic

<MetadataType(GetType(ProductMetadata))> _
Partial Public Class Product
    ... Existing members defined here, but without attri
End Class
```

You then define the metadata type as a normal class, except that you declare simple properties for each of the members to which you want to apply validation attributes. The actual type of these properties is not important, and is

ignored by the compiler. The accepted approach is to declare them all as of type **Object**. As an example, if your **Product** class contains the **ID** and **Description** properties, you can define the metadata class for it as shown here.

```
C#
public class ProductMetadata
  [Required(ErrorMessage = "ID is required.")]
  [RegularExpression("[A-Z]{2}[0-9]{4}",
          ErrorMessage = "Product ID must be 2 capital
  public object ID;
  [StringLength(100, ErrorMessage = "Description must
  public object Description;
}
Visual Basic
Public Class ProductMetadata
  <Required(ErrorMessage:="ID is required.")> _
  <RegularExpression("[A-Z]{2}[0-9]{4}",</pre>
          ErrorMessage:="Product ID must be 2 capital
  Public ID As Object
  <StringLength(100, ErrorMessage:="Description must k</pre>
  Public Description As Object
End Class
```

For more information, see MetadataTypeAttribute Class on MSDN.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> <u>Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright @ 2010 by Microsoft Corporation. All rights reserved.

```
Microsoft Enterprise Library 5.0

Using Self Validation
```

Self validation allows you to implement validation logic within the class you want to validate. Use the **HasSelfValidation** attribute to mark the class that contains the validation logic. Use the **SelfValidation** attribute to mark each Self validation method within that class. The following code example shows how to do this.

```
C#
using Microsoft.Practices.EnterpriseLibrary.Common.Cor
using Microsoft.Practices.EnterpriseLibrary.Validation
using Microsoft.Practices.EnterpriseLibrary.Validation
[HasSelfValidation]
public class TemperatureRange
  private int min;
  private int max;
  // ...
  [SelfValidation]
  public void CheckTemperature(ValidationResults resul
  {
    if (max < min)
      results.AddResult(new ValidationResult("Max less
  }
}
```

Visual Basic

```
Imports Microsoft.Practices.EnterpriseLibrary.Common.(Imports Microsoft.Practices.EnterpriseLibrary.ValidatiImports Microsoft.Practices.EnterpriseLibrary.Validati
```

In this example, the **CheckTemperature** method provides self validation. When the **Validation.Validate** method is called on an instance of **TemperatureRange**, the **CheckTemperature** method is invoked.

If you do not specify a rule set name in the **SelfValidation** attribute, the self validation routine is part of the default rule set. To specify a rule set name, include the **Ruleset** parameter in the **SelfValidation** attribute as shown below. You can include more than one **SelfValidation** method in a class and differentiate them using rule set names.

```
C#

[SelfValidation(Ruleset="SimpleRuleset")]
public void Validate(ValidationResults results)
{
    ...
}

Visual Basic

<SelfValidation(Ruleset:="SimpleRuleset")> _
```

```
Public Sub Validate(results As ValidationResults)
...
End Sub
```

Each Self validation method must have a void return value and take a **ValidationResults** instance as its only parameter. The Self validation method should update the **ValidationResults** instance after performing the validation if the validation fails. For more information about the **ValidationResults** class, see <u>Understanding Validation Results</u>.

If you have a derived class and you want it to inherit the Self validation behavior of its base class, you must mark both the base class and the derived class with the **HasSelfValidation** attribute. The Self validation methods in the base class must be public in order for them to be included in the self validation of the derived class.

Self validation works in combination with any validators that are assigned to a class. Therefore, the **ValidationResults** for an object instance will include both the results from the self validation as well as the results from validators within the class. In the following code example, the **Address** class uses self-validation, and the string **ZipCode** has the **StringLengthValidator** assigned.

```
using Microsoft.Practices.EnterpriseLibrary.Common.Cor
using Microsoft.Practices.EnterpriseLibrary.Validatior
using Microsoft.Practices.EnterpriseLibrary.Validatior

[HasSelfValidation]
public class Address
{
    private string _zipCode;

    [StringLengthValidator(1,10, MessageTemplate="ZipCoc
    public string ZipCode
    {
        get { return _zipCode; }
        set { _zipCode = value; }
    }
}
```

```
[SelfValidation]
public void DoValidate(ValidationResults results)
{
   ValidationResult result = new ValidationResult("Er
   ValidationResult result2 = new ValidationResult("E
   results.AddResult(result);
   results.AddResult(result2);
}
```

Visual Basic

```
Imports Microsoft.Practices.EnterpriseLibrary.Common.(
Imports Microsoft.Practices.EnterpriseLibrary.Validati
Imports Microsoft.Practices.EnterpriseLibrary.Validati
<HasSelfValidation()> _
Public Class Address
 Private _zipCode As String
 <StringLengthValidator(1, 10, MessageTemplate:="Zip(</pre>
 Public Property ZipCode() As String
    Get
      Return _zipCode
    End Get
   Set(ByVal value As String)
      _zipCode = value
    End Set
 End Property
 <SelfValidation()> _
 Sub DoValidate(ByVal results As ValidationResults)
    Dim result As New ValidationResult("Error Message"
    Dim result2 As New ValidationResult("Error Message
    results.AddResult(result)
```

results.AddResult(result2) End Sub

End Class

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.



Integration allows you reuse the validators that are associated with your application classes when you perform validation at the user-interface level (for ASP.NET, WPF, and Windows Forms applications) or at the message-sending level of a multi-tiered application (for WCF applications). By integrating the Validation Application Block with your applications, you can reuse your validation rules across several levels of your system architecture.

The integration provided by the Validation Application Block at the user-interface level does the following:

- It provides a way to associate properties of validated application objects with user-interface controls.
- It provides a way to convert values from an input data type to an application-specific data type. For example, you can convert a string input to a **System.DateTime** value.
- It helps avoid coding errors in type names and property names. For example, it throws exceptions when type names or property names cannot be found.
- It does not require instances of application objects to be created in order for validators to be invoked at the user-interface level. In other words, you do not need to instantiate a **Customer** object just to check whether an input string entered by the user meets the validation requirements of the **Customer.Name** property.

The integration provided by the Validation Application Block for the messagepassing level allows you to validate WCF messages, data contracts, and parameters.

The Validation Application Block is designed to integrate with a range of presentation technologies. For details, see the flowing topics:

- Integrating with ASP.NET
- Integrating with WPF
- Integrating with Windows Forms
- Integrating with WCF

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

You can integrate the Validation Application Block with ASP.NET applications. For example, you can use the application block to validate information a user enters into a Web form. ASP.NET is integrated with the Validation Application Block through the types defined in the

Microsoft.Practices.EnterpriseLibrary.Validation.Integration.AspNet assembly.

The **PropertyProxyValidator** class defined in this assembly provides the main integration point. Its purpose is to check an ASP.NET control's value using the validators that are included in a user-provided application class. The **PropertyProxyValidator** class acts as a wrapper that links a control to a validator in an application-level class.

The **PropertyProxyValidator** class raises the **ValueConvert** event when it needs to convert the type of a value entered into the associated edit control. You can handle this event to provide a custom type converter for converting values entered by the user to values required by the validators. If you do not specify a handler, a default conversion is performed by the ASP.NET **TypeConverter** services.

To use the **PropertyProxyValidator** control, you must manually add the code to an .aspx file, as shown in the following example.

```
<cc1:propertyproxyvalidator
   id="firstNameValidator"
    runat="server"
   ControlToValidate="firstNameTextBox"
   PropertyName="FirstName"
   RulesetName="RuleSetA"
   SourceTypeName="BusinessEntities.Customer">
</cc1:propertyproxyvalidator>
```

The **SourceTypeName** attribute shown in the designer-generated source

references the **Customer** class defined elsewhere in the application. This is the class whose validators will be used.

When using ASP.NET with the Validation Application Block, you may need to convert data types such as dates before you can validate the data. The integration library provides a way for you to insert code that performs this conversion. To do this, you need to provide an event handler for the conversion and then reference this handler in the .aspx file. The following code excerpt shows a handler that converts a date of birth string into a date.

```
using Microsoft.Practices.EnterpriseLibrary.Common.Cor
using Microsoft.Practices.EnterpriseLibrary.Validatior
using Microsoft.Practices.EnterpriseLibrary.Validatior
using Microsoft.Practices.EnterpriseLibrary.Validatior
using Microsoft.Practices.EnterpriseLibrary.Validatior
using Microsoft.Practices.EnterpriseLibrary.Validatior

protected void dateOfBirthValidator_ValueConvert(object
{
    string value = e.ValueToConvert as string;
    try
    {
        e.ConvertedValue = DateTime.Parse(value, System.G]
    }
    catch
    {
        e.ConversionErrorMessage = "Date Of Birth is not if
        e.ConvertedValue = null;
    }
}
```

Visual Basic

```
Imports Microsoft.Practices.EnterpriseLibrary.Common.(Imports Microsoft.Practices.EnterpriseLibrary.Validati Imports Microsoft.Practices.EnterpriseLibrary.Validati Imports Microsoft.Practices.EnterpriseLibrary.Validati Imports Microsoft.Practices.EnterpriseLibrary.Validati
```

```
Protected Sub dateOfBirthValidator_ValueConvert(ByVal Handles dateOfBirthValidator.ValueConvert Dim stringValue As String = CStr(e.ValueToConvert) Dim dateValue As DateTime

Dim success As Boolean = DateTime.TryParse(stringVal If success Then e.ConvertedValue = dateValue Else e.ConversionErrorMessage = "Date Of Birth is not i e.ConvertedValue = Nothing End If

End Sub
```

The following code shows the **dateOfBirthTextBox** control and the **dateOfBirthValidator** in an ASP.NET page. The **OnValueConvert** attribute of the **PropertyProxyValidator** specifies the name of the event handler shown in the previous code, and is located in the associated ASP.NET code-behind file.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Integrating with WPF



You can integrate the Validation Application Block with Windows Presentation Foundation (WPF) applications. For example, you can use the application block to validate the values in UI controls in accordance with rules defined for bound properties of an object. Validation is performed as part of the general Binding mechanism. For information about the WPF binding mechanism, see DataBinding Overview on MSDN. The validation rules you specify can be configured to validate at different stages in the binding processing pipeline. For information about applying validation rules, see How to: Implement Binding Validation and Validation Step Enumeration on MSDN.

✓ Note:

The Validation Application Block cannot be used in XML Browser Applications (XBAP) due to issues with the partial trust environment that XBAP mandates.

To use the WPF integration feature, you must add a reference to the assembly **Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WPF**. The integration is performed through a custom **ValidationRule** named **ValidatorRule** defined in this assembly, which performs its validation by invoking the Validation Application Block validator for a specific property on a given type. Validation only occurs if a binding exists, and that binding is connected (if the source of the binding exists).

You can add a validation rule directly to a binding by specifying the desired type and property names, as shown here.

XAML

```
SourceType="{x:Type test:ValidatedObject}
SourcePropertyName="ValidatedStringProper
</Binding.ValidationRules>
</Binding>
</TextBox.Text>
</TextBox>
```

You can also set the **RulesetName** and **ValidationSpecificationSource** properties to refine how the validator for the specified property is created.

If the value of the validated control that carries the **Required** validation attribute is empty to begin with, and remains empty during validation, the source is not updated and validation does not occur. In this particular case the **ValidateOnTargetUpdate** property will not work either because the null default value of the target will not change. Instead, you can invoke **UpdateSource** on the binding to force validation to occur, as shown here.

```
C#
this.Required.GetBindingExpression(TextBox.TextPropert

Visual Basic

Me.Required.GetBindingExpression(TextBox.TextProperty)
```

The validation rule operates in the **ConvertedProposedValue** step, after the value has been converted but before it is set on the source. This means that value conversion errors must be detected through other mechanism. One approach is to enable the **ValidatesOnDataErrors** property on the validated binding.

■ Configuring Validation through Attached Properties

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Integrating with Windows Forms



You can integrate the Validation Application Block with Windows Forms applications. For example, you can use the application block to validate information a user enters into a Windows Form. Windows Forms is integrated with the Validation Application Block through the types defined in the **Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WinForms** assembly.

The **ValidationProvider** class defined in this assembly provides the main integration point. This class is an extender provider that adds properties to Windows Forms controls. This provider allows you to specify how to use the Validation Application Block to validate the controls' values.

You can perform validation automatically by using the **Control.Validating** event or you can initiate it in your code by invoking the **ValidationProvider.PerformValidation(Control)** method. In both cases, you must use the validation provider to configure the control.

You can also specify an instance of the **ErrorProvider** class for the control. If you do this, the validation errors that result if the validation fails are sent to the error provider as a properly formatted error message.

The **ValidationProvider.Enabled** property enables and disables the validation provider. When it is disabled no validation occurs, the validation provider is considered valid, and any error messages posted to the **ErrorProvider** are cleared. When it is re-enabled, the provider continues to be valid and does not post any error messages until validation is triggered again.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Integrating with WCF



Windows Communication Foundation (WCF) provides extension points that allow you to support custom scenarios such as checking security attributes and performing policy assertions. The Validation Application Block includes a WCF integration library that adds a custom behavior, and a parameter inspector. This extension allows the application block to validate WCF messages, data contracts, and parameters.

When the WCF service begins, it invokes the custom behavior. This custom behavior then adds the parameter inspector. WCF calls the parameter inspector twice for each message it processes. The first time is before it sends the call to the service implementation. The second time is after the service generates the return message. This section explains how you can use the Validation Application Block with your WCF applications. In general, you must do the following:

- <u>Include the Validation Application Block in your service contract</u>.
- <u>Define a behavior that uses the application block</u>.
- Define an endpoint that uses the behavior.
- Modify the service contract so that the client can receive relevant failure information.
- Add validators to parameters as required.

■ Including t	he Val	lidation	Appli	ication	Block

■ Defining a Behavior

■ Defining an Endpoint

■ Reporting Validation Faults

■ Validating Parameters

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design of the Validation Application Block



The Validation Application Block is designed to do the following:

- Encapsulate the logic used to perform the most common validation tasks into minimal application code.
- Relieve developers of the requirement to write duplicate code and custom code for common validation tasks.
- Allow validation rules to be changed after the application has been deployed, and ensure that changes happen simultaneously and consistently.

■ Design Highlights

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending and Modifying the Validation Application Block



In its original state, the Validation Application Block works well for typical validation scenarios, such as validating data for an ASP.NET application. However, there may be times when you have to customize certain behaviors of the application block to better suit your application's particular requirements. There are two ways to do this. You can extend the Validation Application Block using the built-in extension points. You can also modify the application block by making changes to its source code. For more details, see the following topics:

- Extending the Validation Application Block
- Extending and Modifying Enterprise Library

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Extending the Validation Application Block



You extend the Validation Application Block through designated extension points. Typically, these are custom classes written by you that implement a particular interface or derive from an abstract class. Because these custom classes exist in your application space, you do not have to modify or rebuild the Validation Application Block; instead, you can designate your extensions through configuration settings.

You can extend the Validation Application Block by implementing new validators. You can create your own **Validator** classes and your own attributes if the ones provided with the Validation Application Block do not fit your requirements. These classes could validate data types in new ways or they could deal with more complex data types, such as a **Customer** data type that includes many different data types.

The following table lists the base classes that you can use to extend the block.

Custom Provider or Extension	Base Class		
Validator	Validator <t> or Validator</t>		
Validator Attribute	ValueValidatorAttribute		

You can extend the **Validator**<**T**> base class to create strongly typed validators. You can use the **Validator** base class to create loosely typed validators.

For detailed information about how to integrate custom providers with the Enterprise Library configuration system and configuration tools see <u>Creating Custom Providers for Enterprise Library</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Deployment and Operations



Two of an administrator's main tasks will be to see that the initial deployment of the Validation Application Block is planned and managed and that subsequent updates are deployed with minimal impact to existing applications that use the application block. For details of deploying and updating Enterprise Library and the application blocks, see <u>Deploying Enterprise Library</u>.

In addition, administrators must decide whether they want to use the instrumentation exposed by the application block. For details of how to enable and disable instrumentation, see Enabling Instrumentation. For information about the instrumentation contained within the Validation Application Block, see the following topics:

- Validation Application Block Performance Counters
- Validation Application Block Event Log Entries

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Validation Application Block Performance Counters



The following table lists the Validation Application Block performance counters and their types.

Performance Counter	Counter Type
% Validation Successes	RawFraction
Number of Validation Calls	NumberOfItems32
Number of Validation Failures	NumberOfItems32
Number of Validation Successes	NumberOfItems32
Validation Calls/sec	RateOfCountsPerSecond32
Validation Failures/sec	RateOfCountsPerSecond32
Validation Successes Base	RawBase
Validation Successes/sec	RateOfCountsPerSecond32

These performance counters are in the **Enterprise Library Validation Counters** category. Each performance counter has two instances:

- **AppDomain Total**, where Total is all of the types that are validated within the application domain.
- **AppDomain TypeName**, where TypeName is the name of the type being validated.

For more information about performance counters, see <u>Overview of</u> <u>Performance Monitoring</u> in the .NET Framework Class Library on TechNet.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Validation Application Block Event Log Entries



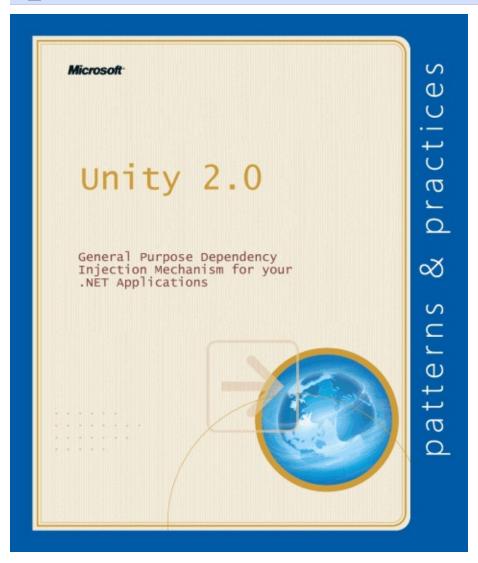
The Validation Application Block is instrumented to log entries to the application event log if a validation fails. The failure is attributed to **Enterprise Library Validation** and it is described as an **Error**.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Unity Dependency Injection and Interception





Welcome to Unity. The following sections of this guidance describe the ways that you can use Unity dependency injection and Unity interception in your applications. The sections are:

- What Is Unity?
- What Does Unity Do?
- When Should I Use Unity?
- About This Release of Unity
- Configuring Unity

- <u>Using Unity in Applications</u>
- Design of Unity
- Extending and Modifying Unity
- Deployment and Operations
- <u>Unity QuickStarts</u>. This topic walks through the QuickStart applications that demonstrate how to execute common operations in your applications.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **What Is Unity?**



Unity is a lightweight, extensible dependency injection container that supports interception, constructor injection, property injection, and method call injection. You can use Unity in a variety of different ways to help decouple the components of your applications, to maximize coherence in components, and to simplify design, implementation, testing, and administration of these applications.

Unity is a general-purpose container for use in any type of Microsoft® .NET Framework-based application. It provides all of the features commonly found in dependency injection mechanisms, including methods to register type mappings and object instances, resolve objects, manage object lifetimes, and inject dependent objects into the parameters of constructors and methods and as the value of properties of objects it resolves.

In addition, Unity is extensible. You can write container extensions that change the behavior of the container, or add new capabilities. For example, the interception feature provided by Unity, which you can use to add policies to objects, is implemented as a container extension.

The following sections of this guidance describe what Unity can do, when you should choose Unity, and the ways that you can use it in your applications:

- What Does Unity Do? This topic provides a brief overview that will help you to understand what Unity can do, and explains some of the concepts and features it incorporates. It also provides a simple example of the way that you can write code to use Unity.
- When Should I Use Unity? This topic will help you to decide if Unity is suitable for your requirements. It explains the benefits of using Unity, and any alternative techniques you may consider. It also provides details of any limitations of Unity that may affect your decision to use it.
- About This Release of Unity. This topic contains information about the changes in this release, the target audience and system requirements, migration and side-by-side execution, and links to other Microsoft patterns & practices resources.
- <u>Configuring Unity</u>. This topic describes how you can populate a Unity container with the type registrations, mappings, extensions, and other

information required by your application.

- <u>Using Unity in Applications</u>. This topic explains how to use Unity in your own applications. It explains how to add Unity to your application, how to resolve objects, and how to take advantage of the many other capabilities of Unity.
- <u>Design of Unity</u>. This topic explains the decisions that went into designing Unity and the rationale behind those decisions.
- Extending and Modifying Unity. This topic explains how to extend Unity and how to modify the source code.
- <u>Deployment and Operations</u>. This topic explains how to deploy and update the Unity assemblies and use the instrumentation exposed by Unity.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

What Does Unity Do?



By using dependency injection frameworks and inversion of control mechanisms, you can generate and assemble instances of custom classes and objects that can contain dependent object instances and settings. The following sections explain the ways that you can use Unity, and the features it provides:

- The Types of Objects Unity Can Create
- Registering Existing Types and Object Instances
- Managing the Lifetime of Objects
- Specifying Values for Injection
- Populating and Injecting Arrays, Including Generic Arrays
- Intercepting Calls to Objects

■ The Types of Objects Unity Can Create

■ Registering Existing Types and Object Instances

■ Managing the Lifetime of Objects

■ Configuring Types for Injection into Constructors, Methods, and Properties

Populating	and Injecting	Arrays, Including	Generic Arrays	

■ Intercepting Calls to Objects

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

When Should I Use Unity?



Dependency injection provides opportunities to simplify code, abstract dependencies between objects, and automatically generate dependent object instances. In general, you should use Unity when:

- You wish to build your application according to sound object oriented principles (following the five principles of class design, or SOLID), but doing so would result in large amounts of difficult-to-maintain code to connect objects together.
- Your objects and classes may have dependencies on other objects or classes.
- Your dependencies are complex or require abstraction.
- You want to take advantage of constructor, method, or property call injection features.
- You want to manage the lifetime of object instances.
- You want to be able to configure and change dependencies at run time.
- You want to intercept calls to methods or properties to generate a policy chain or pipeline containing handlers that implement crosscutting tasks.
- You want to be able to cache or persist the dependencies across post backs in a Web application.

The following sections provide more information to help you decide whether Unity is suitable for your requirements:

- Scenarios for Unity
- Benefits of Unity
- <u>Limitations of Unity</u>

✓ Note:

Enterprise Library, also from the Microsoft patterns & practices group, uses Unity as its primary mechanism for generating instances of Enterprise Library objects. For information about this and other features of Enterprise Library, see the Enterprise Library Community site on CodePlex, or the Microsoft Enterprise Library pages on MSDN®.

■ Scenarios for Unity

■ Benefits of Unity

■ Limitations of Unity

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

About This Release of Unity



This section contains the following topics that will help you to understand this release of Unity and help you understand how to use it alongside earlier versions or migrate your applications to this version. This section includes the following topics:

- Changes in This Release
- Target Audience and System Requirements
- Migration and Side-by-Side Execution
- Related patterns & practices Links
- Copyright and Terms of Use

■ How to Use This Guidance

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Changes in This Release



Unity 2.0 – April 2010 is a new release of the Microsoft patterns & practices Unity dependency injection and interception system. This release also includes additions in functionality, and has been adapted to work with Microsoft Visual Studio® 2010; and with the Microsoft .NET Framework versions 3.5 SP1 and 4.0.

Go to CodePlex for information on Known Issues.

The following sections discuss the changes to Unity:

Breaking Changes to Unity

Changes in Unity

■ Breaking Changes to Unity

■ Changes in Unity

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{\text{@}}$ 2010 by Microsoft Corporation. All rights reserved.

Target Audience and System Requirements



This guidance is intended for software architects and software developers. To get the greatest benefit from this guidance, you should have an understanding of the following technologies:

- Microsoft Visual C#® or Microsoft Visual Basic .NET
- Microsoft .NET Framework

■ System	Requi	irement	s and	Prerequ	uisites

■ System Requirements for Unity for Silverlight

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Migration and Side-by-Side Execution



In general, applications that use previous versions of Unity will function with this release without the need for any code changes. It will be necessary, however, to update the references to refer to the new assemblies and to update the configuration files to reference the correct version of the assemblies.

This version of Unity can also be installed side by side with earlier versions. You can deploy new applications written for this version of Unity along with applications written for earlier versions. In addition, you can also choose to migrate existing applications to the new version.

If you decide to use side-by-side execution, you must deploy the different Unity versions in different directories. In any specific directory, you cannot mix and match assemblies from different versions.

The shipped project files use data in the AssemblyInfo.cs file to build assemblies that have different version information. This allows you to use strong names and to add different versions to the global assembly cache (GAC) for side-by-side execution.

■ Partial Migration

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Reusing Configuration Files Based on a Previous Schema



Although this documentation is based on the Unity 2.0 configuration schema and all examples use Unity 2.0, partial backward compatibility is provided for the Unity 1.2 configuration schema. However, you cannot simply use a Unity 1.2 configuration file. In order to use the contents of a Unity 1.2 configuration file you must:

- 1. Create a new configuration file.
- 2. Edit the **<configSections>** to point to the correct assembly.
- 3. Add a **<sectionExtension>** section if you are using container extensions for Unity.
- 4. Cut and paste the portions of the Unity 1.2 configuration file you wish to reuse.

✓ Note:

Check the results as you still may get errors depending upon the specific portions you cut and paste.

For information on using the Unity 1.2 configuration schema see <u>Unity</u> Configuration Schematic on MSDN®.

■ Migrating Custom Extensions

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Related patterns & practices Links



For information related to Unity and other tools, and guidance for designing and building applications, see the patterns & practices website and guides:

- Microsoft patterns & practices Developer Center
- Microsoft Application Architecture Guide, 2nd Edition
- Solution Development Fundamentals
- Security Guidance for Applications Index
- .NET Data Access Architecture Guide
- Improving .NET Application Performance and Scalability
- Monitoring in .NET Distributed Application Design
- Deploying .NET Framework-based Applications

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright and Terms of Use



This document is provided "as-is". Information and views expressed in this document, including URL and other Internet website references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2010 Microsoft. All rights reserved.

Microsoft, Windows, Windows Server, Windows Vista, Visual C#, Visual Basic, and Visual Studio are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Configuring Unity



One of the key tasks you need to perform when using Unity is to configure the container with the required aliases, type registrations, mappings, and other information that it requires in order to resolve objects at run time and inject the appropriate objects and values into dependent objects. This section covers all of the configuration options for Unity, and describes how you can configure the container using a configuration file, or at run time using code.

Unity can read configuration information from an XML configuration file. By default, this is the App.config or Web.config file for your application. However, you can load configuration information from any other XML format file or from other sources if required.

In addition, Unity exposes a series of methods that you use in your application code to configure the container. This approach is useful when the registrations and mappings depend on the environment or run-time information, and you can change the configuration at run time using these methods. Run-time configuration is also a good choice if you want to be able to manipulate container hierarchies at run time to change the overall behavior of type resolution, injection, and interception.

You can, of course, use a combination of design-time (configuration files) and run-time configuration to achieve exactly the configuration you require at any point during application execution.

To help you understand how to configure Unity, this section divides the information into two separate subsections—one for design-time configuration and one for run-time configuration. Each section contains basically the same set of topics that describe specific configuration scenarios.

✓ Note:

The configuration files are not encrypted by default. A configuration file may contain sensitive information about connection strings, user IDs, passwords, database servers, and catalogs. You should protect this information against unauthorized read/write operations by using encryption techniques. If you wish to restrict access to the configuration file, it must be encrypted or

protected using <u>Access Control Lists</u>. It is recommended that the configuration store is in the same trust boundary and that decrypting the configuration is done in the same trust boundary after the configuration is read.

The complete set of topics in this section is as follows:

- Design-Time Configuration
 - Using the Configuration Tool
 - <u>Using Design-Time Configuration</u>
 - Specifying Types in the Configuration File
 - Specifying Values for Injection
 - Extending the Unity Configuration Schema
 - Configuration Files for Interception
 - Default Aliases and Assemblies
- Run-Time Configuration
 - Using Run Time Configuration
 - Registering Types and Type Mappings
 - Creating Instance Registrations
 - Registering Injected Parameter and Property Values
 - Registering Generic Parameters and Types
 - Registering Container Extensions
 - Registering Interception

For information about using container hierarchies, see <u>Using Container Hierarchies</u>. For information about using Unity, see <u>Using Unity in Applications</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design-Time Configuration



This topic describes the techniques you can use to configure Unity containers using a configuration file or a configuration you load into the container at run time.

✓ Note:

This documentation is based on the Unity 2.0 configuration schema and all examples use Unity 2.0. Partial backward compatibility is provided for the Unity 1.2 configuration schema.

The following topics describe Unity configuration:

- <u>Using the Configuration Tool</u>. Not available for Unity configuration.
- <u>Using the Unity XSD to Enable Visual Studio IntelliSense</u>. This topic explains how to use the Unity XSD to enable IntelliSense® in Visual Studio to assist in manually editing the Unity configuration.
- Reusing Configuration Files Based on a Previous Schema. This topic explains how to create a current configuration file by reusing a configuration file based on a previous Unity schema.
- <u>Using Design-Time Configuration</u>. This topic explains the overall structure of the Unity configuration file, how you load configuration information at run time, and how you can use alternative configuration sources with Unity.
- Specifying Types in the Configuration File. This topic explains how to configure mappings in the container between types. In general, you will create mappings between an interface and a type that implements the interface, or between a base class and a type that inherits that base class. You can also use this section to specify concrete types for which you want Unity to manage the lifetime.
- <u>The Unity Configuration Schema</u>. This topic describes the configuration schema elements for Unity.
- <u>Specifying Values for Injection</u>. This topic explains how to configure registrations for instance types such as string, date and time, or integer

- values that you can resolve in your application.
- Extending the Unity Configuration Schema. This topic explains how to configure Unity to load and use container extensions that add additional functionality to the container, and how you can specify configuration information for these extensions.
- <u>Configuration Files for Interception</u>. This topic explains how to configure interceptors, behaviors, policies, handlers, and matching rules that Unity will use when creating instances of types to which you want to add interception capabilities in order to change the behavior of that object or type.

For information about how to configure Unity using code that executes at run time and calls the registration methods of the Unity container, see Run-Time
Configuration. For information about resolving types at run time, see Resolving Objects.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Using the Configuration Tool



Not available for Unity configuration.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Using the Unity XSD to Enable Visual Studio IntelliSense



You can enable IntelliSense in Visual Studio to assist the manual editing of Unity configuration files.

✓ Note:

The XSD is not required. Configuration will work at run time without it. It is only required for IntelliSense in Visual Studio.

In order for the XSD to be used by the Visual Studio editor the **<unity>** element must have an XMLNS attribute with the correct namespace. The following are the two ways to get the correct namespace instead of manually entering it:

- You can force the editor to use the schema by clicking **Schemas** on the **XML** menu and selecting the entry for the unity configuration schema. After that, the **<unity>** element will appear as an alternative in the IntelliSense dropdown and by choosing it from IntelliSense the xmlns attribute will be populated. This is a per-user and per-project setting, so every user working on the project would be required to select this setting.
- You can enter **<unity xmlns=**" and then IntelliSense will show a list of namespaces which are targeted by a known schema. You can then choose the right namespace,
 - **http://schemas.microsoft.com/practices/2010/unity**, which will show up in the IntelliSense list when you click on the **xmlns** attribute and complete the entry. Visual Studio will then associate the URL with the actual physical file. This is the recommended option, since the setting persists when you pass the configuration file to another user.

Collection elements, such as aliases, containers, extensions, namespaces, and assemblies are not supported by the xsd, but they do work in the configuration file.

There are some pre-defined type names for some type attributes, such as for lifetime managers, but these are just suggestions and any type name is accepted.

The schema for the **register** element imposes a specific order for its children, an

order that is not required by the configuration runtime but makes the schema more robust. The order of children is as follows: one optional **lifetime**, one optional **constructor**, and then as many of **method**, **property**, **interceptor**, **interceptionBehavior**, **addInterface** and any custom element as desired, in any order.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Using Design-Time Configuration



Using Unity typically requires the configuration of a Dependency Injection (DI) container. You can configure a container by using the Unity API, a .NET configuration file, or to a limited degree by using attributes. This topic describes how to use an XML configuration file to supply the required configuration information..

Dependency injection is a very flexible pattern, and to be used successfully requires the developer to provide information to the container about his applications. The two most common configuration tasks are setting up type mappings and configuring injection of a type. Type mappings enable you to request a type from the container that results in the container returning an instance of a different type (typically a derived class or interface implementation). Configuring injection for a type entails specifying information such as which constructor gets called, which properties get injected, and what their values are. The Unity configuration schema encompasses these types of configuration and is also extensible to allow for additional kinds of configuration such as Unity interception configuration, see The Unity Configuration Schema. The following sections provide more details:

- Format of the Unity Configuration File
- Loading Configuration File Information into a Container
- Loading the Configuration from Alternative Files

■ Format of the Unity Configuration File

■ Loading	Configuration	File Informa	ation into a	Container

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Specifying Types in the Configuration File



This topic explains how to use types in Unity configuration files. At the core of Unity's functionality are types and how you specify and handle them. You will need to specify types many times in the typical configuration file. The configuration files have their own set of rules for writing type names—rules that differ from those for types written in C# or Visual Basic. These rules apply everywhere you can specify a type in the Unity configuration section.

This topic contains the following section that describe how you can specify types:

- CLR Type Names
- Type Aliases
- Automatic Type Lookup
- Default Aliases and Assemblies
- Generic Types

CLR Type Names

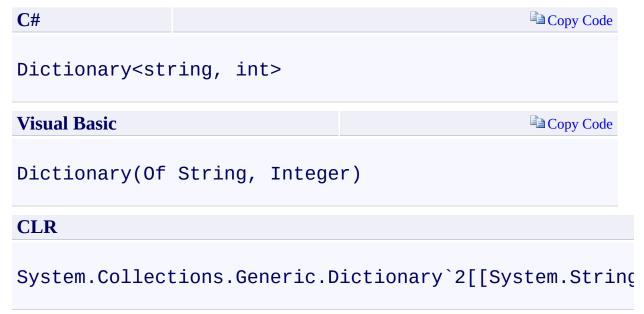
You can specify a type name by using the CLR standard type name syntax, as shown in the following example:

```
<namespace>.<typename>, <assembly>
```

You can use either partial assembly names or fully qualified assembly names which include the culture, version, and public key token. These names are straightforward for simple types.

In order to specify a name for a type that is in the global assembly cache, you must use the fully qualified assembly name for the type to be correctly loaded. For example, for System.String, a type in mscorlib, you cannot use **System.String, mscorlib**. You must use the fully qualified assembly name, **System.String, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089**.

CLR type names can be very verbose, especially when working with generic types. For example, compare the following simple dictionary in C# or Visual Basic with the CLR example:



In order to expedite the process and make type names less error prone, Unity configuration provides two options you can use in the configuration file: aliases and automatic type lookup.

Type Aliases

An alias is simply a shorthand name that will be replaced with the full type name when configuration is applied to the container. You specify an alias in the configuration file inside the section, but outside any **<container>** elements, as shown in the following example:

There are the following rules for using aliases:

- You can have an arbitrary number of <alias> elements in the configuration file.
- Anywhere you can give a type name you can use an alias instead.
- There are no recursive aliases, which means that you cannot use an alias to define the type for an alias.
- Alias names are case sensitive: <alias alias="int" /> and <alias alias="Int" /> are two different aliases and are not interchangeable.

✓ Note:

Aliases only exist at configuration time. They are not available at run time.

Automatic Type Lookup

In many cases, like for **ILogger** in the <u>Format of the Unity Configuration File</u> example, the name of a type is all that is required. But given Unity's dependence on types and the large number of types typically involved in a configuration, the ability to perform automatic type lookups further expedites the process. By incorporating automatic type lookups, Unity also eliminates the need to define an alias for every type in an assembly, which saves effort and serves to reduce the chance for error from repeatedly typing the namespace and assembly name.

The Unity configuration system can search for types. However, it will only look for types if the type name specified is not a full type name and it is not an alias. You can provide the configuration section with the namespaces and assemblies to look through by using the **namespace** and **assembly** elements, as shown in the following example.

With the configuration shown in the previous example, when the configuration system hits a name it does not recognize as a type name or alias, it will then search through the assemblies and namespaces for a match. So, to find **ILogger**, it will try to match the following names in order:

- 1. MyApp.Interfaces.ILogger, MyApp
- 2. System.ILogger, MyApp
- 3. MyApp.Interfaces.ILogger, mscorlib, 2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
- 4. System.ILogger, mscorlib, 2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

The search will stop at the first matching type.

The system uses simple string concatenation to create the type name it attempts to load. However, you cannot specify a namespace qualified name plus the type, **MyApp.Interfaces.ILogger, MyApp**, if you have any namespace elements in your configuration section, **<namespace name="System"** />.The namespace from the configuration section will be appended to the namespace, resulting in a search on the wrong name, **System.MyApp.Interfaces.ILogger**. You should put namespaces in the **<namespace**> elements instead of on the type names in the configuration file to avoid this possibility.

If you have a large number of assemblies and namespaces, then the system type search could take a significant amount of time to complete. Normally, containers are only configured at application startup, so this time hit will not be significant during the operation of your application. If you find it becomes a significant issue, then you should consider using an explicit alias for the types that take the greatest search times, since aliases are matched first.

When matching a name with a type, the configuration system performs the following steps in order. The first one to succeed stops the process:

- 1. Attempt to load a type using the name directly (treated as a full type name)
- 2. Attempt to match a name to an alias
- 3. Do automatic type search

Default Aliases and Assemblies

Some types and assemblies are used frequently in Unity configuration files. The Unity configuration system provides a set of predefined default aliases so you do not have to explicitly add aliases for these common types. Any user-defined entries will overwrite the default ones.

✓ Note:

Aliases are case sensitive.

In addition to the default aliases, the Unity configuration system also automatically adds System and mscorlib assemblies to the list of assemblies that are searched for types.

The following table has the complete list of pre-defined type aliases provided by Unity:

Default Alias	Type
sbyte	System.SByte
short	System.Int16
int	System.Int32
integer	System.Int32
long	System.Int64
byte	System.Byte
ushort	System.UInt16
uint	System.UInt32
ulong	System.UInt64
float	System.Single
single	System.Single
double	System.Double
double	System. Double

System.Decimal
System.Char
System.Boolean
System.Object
System.String
System.DateTime
System.DateTime
System.DateTime
Microsoft.Practices.Unity.ContainerCor
Microsoft.Practices.Unity.ContainerCon
Microsoft.Practices.Unity.TransientLife
Microsoft.Practices.Unity.TransientLife
Microsoft.Practices.Unity.PerThreadLif
Microsoft.Practices.Unity.PerThreadLif
Microsoft.Practices.Unity.ExternallyCo
Microsoft.Practices.Unity.ExternallyCo
Microsoft.Practices.Unity.HierarchicalL
Microsoft.Practices.Unity.HierarchicalI
Microsoft.Practices.Unity.PerResolveLi
Microsoft.Practices.Unity.PerResolveLi
Microsoft.Practices.Unity.PerResolveLi

Generic Types

The CLR type name syntax for generic types is extremely verbose, and it also does not allow for things like aliases. The Unity configuration system allows for a shorthand syntax for generic types that also allows for aliases and type searching.

To specify a closed generic type, you provide the type name followed by the type parameters in a comma-separated list in square brackets.

The Unity shorthand would look like the following example.

```
XML

<container>
     <register type="IDictionary[string,int]" </registe
</container>
```

If you wish to use an assembly name-qualified type as a type parameter, rather than an alias or an automatically found type, you must place that entire name in square brackets, as shown in the following example:

```
XML
<register type="IDictionary[string, [MyApp.Interfaces.</pre>
```

To specify an open generic type you simply leave out the type parameters. You have two options:

- Use the CLR notation of `N where N is the number of generic parameters.
- Use the square brackets, with commas, to indicate the number of generic parameters.

Generic Type	Configuration file XML using CLR notation	Configuration file XML using comma notation
IList <t></t>	IList`1	IList[]
IDictionary <k,v></k,v>	IDictionary`2	IDictionary[,]

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Unity Configuration Schema



Unity 2.0 uses a new streamlined configuration schema for configuring Unity.

The following sections describe the schema configuration elements, their child elements, and their attributes in more detail:

- The <unity> Configuration Section
- The <container> Element
- The <register> Element
- The Element
- The <constructor> Element
- The Flement
- The <method> Element
- The <param> Element
- The <dependency> Element
- The <value> Element
- The <optional> Element
- The <array> Element
- The <extension> Element
- The <instance> Element
- The <namespace > Element
- The <alias> Element
- The <sectionExtension> Element

■ The <unity> Configuration Section

■ The <register> Element





■ The <property> Element



■ The <param> Element

■ The <dependency> Element

■ The <value> Element

■ The <optional> Element

■ The <array> Element



■ The <instance> Element

■ The <namespace> Element

■ The <assembly> Element

■ The <alias> Element

■ The <sectionExtension> Element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright @ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0 **Specifying Values for Injection**



This topic explains how to configure the information required so that Unity will automatically populate constructor and method parameters and property values when it resolves instances of types. The **param></code> and property></code> elements both let you specify a value to be supplied for the parameter or property, respectively. There are many different kinds of values that can be specified, each with a separate element. In addition, the Unity configuration schema supports a shorthand notation for the most common cases.**

For more information about the format of the Unity configuration file, see <u>Using</u> <u>Design-Time Configuration</u>.

This topic contains the following sections describing values for injection:

Resolving Values from the Container

Giving Values in Configuration

Configuring Array Values

■ Resolving Values from the Container

■ Giving Values in Configuration

■ Configuring Array Values

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Extending the Unity Configuration Schema



The Unity container is highly customizable. No one fixed configuration format can cover everything that you might want to do with the container. As a result, the Unity configuration system itself is extensible, allowing you to add new valid elements to your configuration file. The **sectionExtension** element allows you to load the code that adds these new options to the configuration file. This lets you specify an implementation of the **SectionExtension** type.

Section extensions can do the following to the configuration:

- Add new predefined aliases
- Add new container-level elements
- Add new registration-level elements
- Add new value-level elements

You can create your own custom extensions, or use extensions created by third parties, with Unity. Unity also uses default extensions to implement its own functionality. For information about using extensions, see Creating and Using Container Extensions.

One example of a section extension is the

InterceptionConfigurationExtension section extension, which ships with the Unity package. This section extension adds the following elements and aliases to the schema:

- Aliases are defined for each of the types (like VirtualMethodInterceptor, TransparentProxyInterceptor, and various matching rules) that are used by the interception configuration.
- The **<interception>** element is added as a valid element child element for the **<container>** element.
- The **<interceptor>**, **<interceptionBehavior>**, **<addInterface>**, and **<policyInjection>** elements are added as valid child elements for the **<register>** element.

This extension mechanism allows for almost unlimited extensibility of the configuration file on an opt-in basis. Though the schema extension will modify the schema allowed at run time, it does not modify the XSD file used by Visual Studio IntelliSense. As a result, you will still get warnings in the Visual Studio

XML editor even though the file will work fine at run time. In order to resolve this problem, the section extension author must provide an updated XSD document for use with their extension.

✓ Note:

The **InterceptionConfigurationExtension** is supported by the schema shipped with Unity.

The **sectionExtension**> element also accepts a user-provided prefix attribute. This is useful in the case where two section extensions both provide extension elements with the same name. In the case of a collision, you can specify a prefix for one or both section extensions, and then use that prefix to disambiguate them.

Consider two schema extensions, both of which add a **<containerCustomization>** element. Using the prefix attribute, a configuration file that uses both would look like the following example.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web</u> site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Configuration Files for Interception



Unity 2.0 treats interception like any extension you add to Unity. As with any extension in Unity 2.0, the Unity interception mechanism can be configured through either the API or through a Unity configuration section.

✓ Note:

Unity provides partial backward compatibility for implementing interception through a container. Earlier versions used a container extension named **InterceptionExtension**, which resides in the assembly named **Microsoft.Practices.Unity.Interception.dll**. To configure interception, you specify this extension in the **extensions** element of your application configuration, and then define the behavior of interception in the **extensionConfig** section.

Using the **extension** and **register** elements in Unity 2.0 is comparable to **Interceptor** element use in the **extensionConfig** section in earlier versions. For more information on backward compatibility, see <u>Reusing Configuration Files Based on a Previous Schema</u>.

For more information about Unity 1.2 interception, see <u>Using Interception</u> <u>with Unity</u> on MSDN.

This topic contains the following sections to describe the interception configuration file:

- <u>Using the Configuration File to Enable Interception</u>
- Standard Interception Aliases
- Enabling Interception of a Type
- Configuring Policy Injection Policies
- <u>Legacy Interception Configuration</u>
- Interception Configuration Schema Elements
- Registering Interception at Run Time

+	Using	the	Config	guration	File to	Enable	Interce	ption

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Interception Configuration Schema Elements



When configuring files for interception, the following elements may appear as children of a **register** element. For more information see <u>The register</u> <u>Element</u>. These elements are used to describe the **interceptors** and **policies** elements, their child elements, and their attributes in more detail:

- The <interceptor> Element
- The <interceptionBehavior> Element
- The <addInterface> Element
- The <interception> Element
- The <policy> Element
- The <matchingRule> Element
- The <callHandler> Element
- The <interceptors> Element
- The interceptors <interceptor> Element
- The <default> Element
- The <key> Element

For more information about interception, and selecting the objects and their members to add a handler pipeline, see <u>Using Interception and Policy Injection</u>.

■ The <interceptor> Element

■ The <interceptionBehavior> Element



■ The <interception> Element

■ The <policy> Element

■ The <matchingRule> Element



■ The <interceptors> Element

■ The interceptors <interceptor> Element

The <default> Element

■ The <key> Element

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Run-Time Configuration



This topic explores the techniques you can use to configure Unity containers using code that executes at run time and calls the registration methods of the Unity container. It contains the following topics:

- <u>Using Run-Time Configuration</u>. This topic describes the fluent interface that Unity exposes, and other issues you should be aware of when configuring the container at run time using code.
- Registering Types and Type Mappings. This topic explains how to register mappings in the container between types. In general, you will create mappings between an interface and a type that implements the interface, or between a base class and a type that inherits that base class.
- <u>Creating Instance Registrations</u>. This topic explains how to register existing objects in the container that you can resolve in your application. This technique is useful if you want Unity to manage the lifetime of the objects you register.
- Registering Injected Parameter and Property Values. This topic explains
 how to register the information required so that Unity will automatically
 populate constructor and method parameters and property values when it
 resolves instances of types.
- Registering Generic Parameters and Types. This topic explains how you can register the information required for injection for generic types, including generic arrays.
- Registering Container Extensions. This topic explains how to register information that instructs Unity to load and use container extensions that add additional functionality to the container, and how you can register configuration information for these extensions.
- Registering Interception. This topic explains how to register behaviors, policies, handlers, and matching rules that Unity will use when creating instances of types to which you want to add interception capabilities to change the behavior of that object or type.

For information about how to configure Unity at design time, including the techniques for loading configuration from a file or other source, see <u>Design-Time Configuration</u>. For information about resolving types at run time, see

Resolving Objects.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Using Run Time Configuration



This topic discusses some of the factors you should keep in mind when using run-time configuration, and explains some features of the Unity run-time configuration mechanism. For details on how to specify configuration using a configuration file, see Design-Time Configuration.

■ Using the UnityContainer Fluent Interface

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0

Registering Types and Type Mappings



This topic explains how to register types in the container. Registering a type lets you configure how the container creates instances of the specified type. In general, you will create mappings between an interface and a type that implements the interface, or between a base class and a type that inherits that base class. However, you can register types in the container without creating a mapping.

The **RegisterType** method registers a type with the container. At the appropriate time, the container will build an instance of the type you specify. This could be in response to dependency injection through class attributes or when you call the **Resolve** method. The lifetime of the object it builds will correspond to the lifetime you specify in the parameters of the method. If you do not specify a value for the lifetime, the type is registered for a transient lifetime, which means that a new instance will be created on each call to **Resolve**.

This topic contains the following sections that explain use of the **RegisterType** method:

- Registering an Interface or Class Mapping to a Concrete Type
- Registering a Named Type
- Registering Type Mappings with the Container
- <u>Using a Lifetime Manager with the RegisterType Method</u>
- Summary of the RegisterType Method Overloads
- More Information

■ Registering an Interface or Class Mapping to a Concrete Type

■ Registering a Named Type

■ Registering Type Mappings with the Container

■ Using a Lifetime Manager with the RegisterType Method

■ Summary	of the	RegisterType	e Method	Overloads

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Creating Instance Registrations



This topic explains how to register existing objects in the container that you can resolve in your application. This technique is useful if you already have an instance of an object that you have previously created and want Unity to manage its lifetime, or if you want Unity to inject that object into other objects that it is resolving.

The **RegisterInstance** method registers an existing instance with the container. You specify the instance type and optional lifetime in the parameter list. The container will return the specified existing instance for the duration of the specified lifetime. The **RegisterInstance** method overloads are very similar to the **RegisterType** method overloads, but they accept an additional parameter, the object instance to register. The use of the registration type and an optional name are identical for the two methods.

When resolving types with dependencies, instances of objects added to the container with the **RegisterInstance** method act just like those registered through **RegisterType**. The **RegisterType** method with a

ContainerControlledLifetimeManager automatically generates this single instance the first time your code calls it, while the **RegisterInstance** method accepts an existing instance for which it will return references. If you do not specify a lifetime manager, the container will use a

ContainerControlledLifetimeManager and it will return a reference to the original object on each call to **Resolve**.

This topic contains the following sections, which explain the use of the **RegisterInstance** method:

- Registering an Existing Object Instance of an Interface or Type to a Container
- <u>Using a Lifetime Manager with the RegisterInstance Method</u>
- Summary of the RegisterInstances Method Overloads
- More Information on Using the RegisterInstance Method

■ Registering an Existing Object Instance of an Interface or Type to a Container

■ Using a Lifetime Manager with the RegisterInstance Method

■ Summary	of the	RegisterInstance	Overloads

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Registering Injected Parameter and Property Values



This topic explains how to configure a container to perform dependency injection at run time by using the **RegisterType** method overloads with the **InjectionMembers** parameter and avoid relying on annotating the classes to resolve with attributes. This topic includes information on configuring Unity to automatically populate constructor and method parameters and property values when it resolves instances of types.

This topic contains the following sections to explain the use of the **InjectionMembers** methods:

- Registering Injection for Parameters Properties and Methods Using InjectionMembers
- Injecting Arrays at Run Time
- Summary of the InjectionMember Methods and Overloads
- For More Information on InjectionMembers

Registering Injection for Parameters, Properties, and Methods using InjectionMembers

The **RegisterType** overloads allow for configuring injection by accepting **InjectionMembers**. Include the **InjectionConstructor**, **InjectionProperty**, and **InjectionMethod** classes as a RegisterType parameter to provide dependency injection configuration in a container for InjectionMember objects.

The following example shows the general syntax for using an **InjectionMember** subclass, **InjectionConstructor**, with the **RegisterType** method. In this example the default constructor is called.

```
C#

IUnityContainer container = new UnityContainer()
    .RegisterType<AType>(new InjectionConstructor());

AType aType = container.Resolve<AType>();

Visual Basic

Dim container As IUnityContainer = New UnityContainer(
    .RegisterType(Of AType)(New InjectionConstructor())

Dim aType As AType = container.Resolve(Of AType)()
```

You can also use attributes applied to target class members to instruct Unity to inject dependent objects. For more information, see <u>Using Injection Attributes</u>.

You can use the **RegisterType** overloads to do the following:

- Register Constructors and Parameters
- Specify a Property for Injection
- Specify a Method for Injection

■ Register Constructors and Parameters

■ Specify a Property for Injection

■ Specify a Method for Injection

■ Injecting Specific Array Instances

■ Injecting All Array Named Instances

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Registering Generic Parameters and Types



This topic explains how you can register the information required for injection for generic types, including generic arrays. You can specify a generic type when you register a type in the Unity container in almost exactly the same way as you register non-generic types. Unity provides two classes specifically for registering generics, **GenericParameter** for specifying that an instance of a generic type parameter should be resolved, and

GenericResolvedArrayParameter for specifying that an array containing the registered instances of a generic type parameter should be resolved.

See the "Specifying Types in the Configuration File" section in the <u>Specifying Types in the Configuration File</u> topic for more details on generics, including a discussion of unbounded, closed, and open generic types.

This topic contains the following sections that explain registering generics:

- Registering Generic Interfaces and Classes
- Registering Type Mappings For Generics
- Registering Generic Arrays
- Support for Generic Decorator Chains
- Methods for Registering Generic Parameters and Types
- More Information

■ Registering Generic Interfaces and Classes	

■ Registering Type Mappings for Generics

■ Registering Generic Arrays

Methods for Registering Generic Parameters and Types	3

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Registering Container Extensions



This topic explains how to register information that instructs Unity to load and use container extensions that add additional functionality to the container, and how you can register configuration information for these extensions.

This topic contains the following sections that explain container extensions:

- Adding and Removing Extensions
- Accessing Configuration Information for Extensions
- Methods for Registering and Configuring Container Extensions
- More Information

■ Adding and Removing Extensions

• Accessing	Configuration	Information	for	Extensions

■ Methods for Registerin	ng and Configurii	ng Container Exte	nsions

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Registering Interception



This topic explains run-time registration of the various interception elements, including interceptors, behaviors, policies, handlers, and matching rules that Unity uses to configure a container for interception. The configuration information is used when creating instances of types for which you want to add interception capabilities to change the behavior of that object or type. In order to provide backward compatibility, Unity 2.0 supports calling the older API **SetInterceptorFor** and **SetDefaultInterceptorFor** methods on the **Interception** container extension in addition to supporting the Unity 2.0 approach using the **RegisterType** API to explicitly configure interceptors, behaviors, and additional interfaces.

- Registering Interceptors and Interceptor Behaviors Explicitly Using RegisterType
- Default Interceptor for a Type
- Registering Additional Interface
- Registering Policy Injection Components

For information on using a configuration file to configure a container for interception, see <u>Configuration Files for Interception</u>.

For information on the design of Unity interception see <u>Interception with Unity</u>.

For information on using interception without a dependency injection (DI) container, see the "Stand Alone Unity Interception" section in <u>Using Interception in Applications</u>.

Registering Interceptors and Interceptor Behaviors Explicitly Using RegisterType

Unity 2.0 enables interception like any other container extension by using **container.AddNewExtension**. Then you can configure a type for interception using an interceptor of your choosing, with behaviors of your choosing. In Unity 2.0 you explicitly configure which object is to be intercepted by which interception mechanism and specify the behavior by using **InterceptionBehavior**; Unity 1.2 implicitly set up policy injection when you configured an interceptor. The following example shows how to configure interception for a type and turn on a custom behavior. This example first adds the **Interception** extension by calling **AddNewExtension**, and then uses **RegisterType** to register a **VirtualMethodInterceptor** and an interception behavior. The behavior must be defined elsewhere.

Using this overload of the **Interceptor** constructor actually tells the container to resolve the interceptor through the container. You can pass an optional string, which becomes the name to resolve with. In most applications you would simply leave this blank, but if you have implemented custom interceptors, you might want to provide additional configuration. There is another overload of the **Interceptor** constructor you can use to specify the interceptors and behaviors

by creating instances and passing the actual instances into the container, as done in the following example:

```
C#
                                                 Copy Cc
// Add the interception extension to the container
IUnityContainer container = new UnityContainer();
container.AddNewExtension<Interception>();
// Configure interception
container.RegisterType<IInterface, BaseClass>(
    "myInterceptor",
    new Interceptor(new InterfaceInterceptor()),
    new InterceptionBehavior(new CustomBehavior()),
    new InterceptionBehavior(new SomeOtherBehavior()))
Visual Basic
                                                    Cc Cc
' Add the interception extension to the container
Dim container As IUnityContainer = New UnityContainer(
container.AddNewExtension(Of Interception)()
  Configure interception
container.RegisterType(Of IInterface, BaseClass) _
        ("myInterceptor", _
        New Interceptor(New InterfaceInterceptor()),
        New InterceptionBehavior(New CustomBehavior())
        New InterceptionBehavior(New SomeOtherBehavior
```

■ Default Interceptor for a Type

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Registering Policy Injection Components



This topic explains how to register the various elements, including interceptors, behaviors, policies, call handlers, and matching rules that Unity uses to configure a container for interception and for a policy injection behavior.

When you configure policy injection you must specify which objects will be intercepted with the policy injection behavior and which policies in the container are to be used. Then when building the object, the policy injection behavior is set up using the policies already defined in the container.

This topic contains the following sections:

- Policy Injection Run-Time Configuration
- Defining Policies by Using the API

Policy Injection Run-Time Configuration

There are two steps to configuring a type for policy injection. First, you must register the type in the container. In that registration, you must configure an interceptor and enable the **PolicyInjectionBehavior**. Second, you must configure the policy injection policies that determine which call handlers execute on which methods.

Visual Basic

■ Defining Policies by Using the API

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.



This topic describes how to develop applications using Unity, and how to create and build instances of objects. It assumes that you understand how to configure the Unity container. This section includes the following topics:

- Application Design Concepts with Unity. This topic explains how Unity can help you to implement common design patterns and achieve decoupling and coherence in your designs.
- Adding Unity to Your Application. This topic describes how to add Unity to your project, and how to reference the appropriate assemblies in your code.
- Resolving Objects. This topic contains a series of sections that describe how you can resolve objects through the Unity container so that it creates the appropriate type and optionally populates any dependencies specified for these types.
- <u>Understanding Lifetime Managers</u>. This topic describes the way that Unity manages the lifetime of objects it creates, and how you can use the lifetime managers included with Unity.
- <u>Using Container Hierarchies</u>. This topic explains how you can use a hierarchy of nested Unity containers to achieve finely grained control over the configuration of Unity and manage this configuration at run time.

For information on how to configure Unity, see **Configuring Unity**.

Unity ships as both source code and signed binary assemblies. You can use the signed assemblies directly. If you intend to compile the source code, see <u>Target Audience and System Requirements</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Application Design Concepts with Unity



Features such as inversion of control, dependency injection, interception, factory, and lifetime (some of which are described in the "Scenarios for Unity" section of the topic When Should I Use Unity?) provide several major advantages when building applications that consist of many individual classes and components. Designing applications that conform to these patterns can provide the following:

- The capability to substitute one component for another using a pluggable architecture.
- The capability to centralize and abstract common features and to manage crosscutting concerns such as logging, authentication, caching, and validation.
- Increased configuration flexibility.
- The capability to locate and instantiate services and components, including singleton instances of these services and components.
- Simplified testability for individual components and sections of the application.
- Simplified overall design, with faster and less error-prone development.
- Ease of reuse for common components within other applications.

Of course, implementing these patterns can initially make the design and development process more complex, but the advantages easily justify this extra complexity. In addition, the use of a comprehensive dependency injection mechanism can actually make the task of designing and developing applications much easier.

Fundamentally, there are two approaches to using a dependency injection mechanism:

- You can arrange to have dependent objects automatically injected, using techniques such as constructor injection, property (setter) injection, and method call injection that inject dependent objects immediately when you instantiate the parent object. This approach is generally most appropriate for applications that require a pluggable architecture or where you want to manage crosscutting concerns.
- You can have objects injected only on demand, by calling the **Resolve**

- method of the container only when you need to retrieve a reference to a specific object.
- This approach is known as service locator. It is more intrusive into your application, but can be simpler if your architecture does not lend itself to having a central container.

In addition to dependency injection, developers may wish to implement patterns such as Interception, Decorator, Chain Of Responsibility, and Intercepting Filter, where a call from a client or process passes through a graph of objects, with each one able to access and act upon details of the call, such as the method or property name, the parameter types and values, the returned type and value, and other information. Unity achieves this through interception of method calls, providing opportunities to apply policies to objects using a technique often referred to as policy injection.

Unity provides a comprehensive dependency injection and interception mechanism, and is easy to incorporate into your applications. However, it does change the way that you design these applications. The following sections of this topic describe areas where dependency injection is useful:

- Pluggable Architectures
- Managing Crosscutting Concerns
- Service and Component Location
- Policy Injection through Interception

■ Pluggable Architectures

■ Managing Crosscutting Concerns

■ Service and Component Location

■ Policy Injection through Interception

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.



Unity is designed to support a range of common scenarios for resolving instances of objects that, themselves, depend on other objects or services. However, you must first prepare your application to use Unity. The following procedure describes how to include the necessary assemblies and elements in your code.

To prepare your application

- Add a reference to the Unity assembly. In Visual Studio, right-click your project node in Solution Explorer, and then click **Add Reference**. Click the **Browse** tab and find the location of the **Microsoft.Practices.Unity.dll** assembly. Select the assembly, and then click **OK** to add the reference.
- 2. (Optional) If you intend to use the configuration types when you create extensions for Unity, use the same procedure to set a reference to the Unity configuration assembly, named

Microsoft.Practices.Unity.Configuration.dll.

3. (Optional) If you intend to use the interception and policy injection features of Unity, use the same procedure to set a reference to the Unity interception assembly, named

Microsoft.Practices.Unity.Interception.dll.

4. (Optional) If you intend to use the configuration types for the interception and policy injection features of Unity, use the same procedure to set a reference to the Unity interception configuration assembly, named

Microsoft.Practices.Unity.Interception.Configuration.dll.

5. (Optional) To use elements from Unity without fully qualifying the element reference, add the following **using** statements (C#) or **Imports** statements (Visual Basic) to the top of your source code file as required.

```
using Microsoft.Practices.Unity;
using Microsoft.Practices.Unity.Configuration;
using Microsoft.Practices.Unity.InterceptionExte
```

Visual Basic

```
Imports Microsoft.Practices.Unity
Imports Microsoft.Practices.Unity.Configuration
Imports Microsoft.Practices.Unity.InterceptionEx
```

- 6. (Optional) If you are using the IServiceLocator interface, add a reference to the service location binary Microsoft.Practices.ServiceLocation.dll. Visual Studio may automatically copy this file to your bin directory when it compiles, but you do not need to include it unless you are explicitly using the UnityServiceLocatorAdapter class.
- 7. Add your application code. For more information about how you can use Unity in your own applications, see What Does Unity Do?

For Visual Basic projects, you can also use the **References** page of the Project Designer to manage references and imported namespaces. To access the **References** page, select a project node in Solution Explorer, and then click **Properties** on the **Project** menu. When the Project Designer appears, click the **References** tab.

✓ Note:

There are limitations when using Unity in a partial trust environment. For more information, see <u>Using Unity in Partial Trust Environments</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Resolving Objects



You can use the Unity container to generate instances of any object that has a public constructor (in other words, objects that you can create using the **new** operator), without registering a mapping for that type with the container. When you call the **Resolve** method and specify the default instance of a type that is not registered, the container simply generates and returns an instance of that type. However, the only time that this is realistically practical is when the object you are generating contains dependency attributes that the container will use to inject dependent objects into the requested object.

The Unity container identifies type registrations and type mappings in the container using a type and, optionally, a name. The type is an interface or a class (usually an interface or base class) that the desired concrete object type implements or inherits. This identifies the mapping so that the container can retrieve the correct object type in response to a call to the **Resolve** or **ResolveAll** method. Where there is more than one mapping for the same type, the optional name differentiates these mappings and allows code to specify which of the mappings for that type to use.

The provision of both generic and non-generic overloads of many of the Unity container methods ensures that Unity can be used in languages that do not support generics. You can use either approach (the generic or the non-generic overloads) in your code and mix them as required. For example, you can register mappings using the generic overloads and then retrieve object instances using the non-generic overloads, and vice versa.

✓ Note:

When you attempt to resolve an abstract base class or interface where there is no matching type mapping in the container, Unity will attempt to create a new instance of the class you specified. As it cannot construct and populate an instance of an abstract class or an interface, Unity will raise an exception.

When you attempt to resolve a non-mapped concrete class that does not have a matching registration in the container, Unity will create an instance of that class and populate any dependencies.

The following topics describe how you can resolve objects using the **Resolve** or **ResolveAll** methods:

- Resolving an Object by Type.
- Resolving an Object by Type and Registration Name.
- Resolving All Objects of a Particular Type.
- Resolving Objects by Using Overrides
- Retrieving Container Registration Information

For more information about how you can configure Unity with type registrations and mappings, see <u>Configuring Unity</u>.

For more information about how you can perform dependency injection on existing object instances, see <u>Using BuildUp to Wire Up Objects Not Created by</u> the Container.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

$Microsoft\ Enterprise\ Library\ 5.0$

Resolving an Object by Type



Unity provides a method named **Resolve** that you can use to resolve an object by type, and optionally by providing a registration name. Registrations that do not specify a name are referred to as default registrations. This topic describes how to use the **Resolve** method to resolve types and mappings registered as default registrations. For information about resolving named registrations, see <u>Resolving an Object by Type and Registration Name</u>.

■ Using the Resolve Method with Default Registrations

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Resolving an Object by Type and Registration Name



Unity provides a method named **Resolve** that you can use to resolve an object by type, and optionally by providing a registration name. Registrations that specify a name are referred to as named registrations. This topic describes how to use the **Resolve** method to resolve types and mappings registered as named registrations. For information about resolving default registrations, see <u>Resolving an Object by Type</u>.

■ The	Resolve	Method	Overlo	ads f	or I	Named	Registra	ations
■ The	Resolve	Method	Overlo	ads f	or I	Named	Registra	ations

■ Using	the	Resolve	e Method	with	Named	Registra	tions
0						0	

■ Resolving Generic Types by Name

■ More Information

■ Resolving Generic Types

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Resolving All Objects of a Particular Type



When you want to obtain a list of all the registered objects of a specific type, you can use the **ResolveAll** method. The two overloads of this method accept either an interface or a type name, and they return an instance of **IEnumerable** that contains references to all registered objects of that type that are **not** default mappings. The list returned by the **ResolveAll** method contains only named instance registrations. The **ResolveAll** method is useful if you have registered multiple object or interface types using the same type but different names. You can also use the **params** to provide constructor overrides for the **ResolveAll** calls.

■ The ResolveAll Method Overloads

■ Using the ResolveAll Method

■ Resolving All Generic Types by Name

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Resolving Objects by Using Overrides



The parameter and dependency overrides, **ParameterOverride** and **DependencyOverride**, are **ResolverOverride** implementations that provide support for overriding the registration information for resolving instances of types. When you call the **Resolve** method, these classes enable you to override values specified when the type was registered, such as by a **RegisterType** or **RegisterInstance** statement. In effect, **RegisterType** supplied values are overridden by **Resolve** supplied values.

Use **ParameterOverride** to override the specified constructor parameter or parameters. The override applies everywhere the parameter appears unless you use **OnType** to constrain the override to a specified type. Since the purpose of overrides is to affect the resolution of dependencies for all relevant created objects, not just the object requested in the call to **Resolve**, unconstrained overrides can produce errors if there are unconstrained **ParameterOverride** parameters that match parameters with the same name but different types on the selected constructors for objects created in a given resolve operation.

Use **PropertyOverride** to override the value of the specified property or properties. The override applies everywhere the property appears unless you use **OnType** to constrain the override to a specified type.

Use **DependencyOverride** to override the value injected whenever there is a dependency of the given type. **DependencyOverride** overrides all instances where the type matches. Both parameter overrides and dependency overrides support generic types and multiple overrides.

✓ Note:

If the overridden object was previously created and is a Singleton, the override is ignored. The lifetime manager takes precedence and Singletons always return the same instance.

The container does not store a reference for the overridden object.

Overrides work with the constructor that is selected for the type, by attribute or configuration. If the constructor to be used is not identified with an attribute or

explicit container configuration, then the default behavior is that the constructor with the most parameters will be used.

A parameter and property override never affects what element gets selected. They only control the value of the specified parameter or property. You do not change which constructor is called with an override, and you do not change which properties get set with an override.

✓ Note:

If the property is not set as a dependency through attribute, container API, or configuration file, then the override does nothing.

This topic contains the following sections to explain overrides in more detail:

- <u>Using Parameter Overrides</u>
- <u>Using Property Overrides</u>
- <u>Using Dependency Overrides</u>
- More Information

■ Using Parameter Overrides

■ Using Property Overrides

■ Using Dependency Overrides

■ More information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Deferring the Resolution of Objects

Unity provides a technique to facilitate holding a reference to an object you need, but do not want to construct right away. You wish to defer resolution of the object. Instead of creating a factory for the type and injecting the factory into your class, then using it to create the type you want you can use the .NET standard type **Func**<**T**> (C#) or **Func**(**Of T**) (Visual Basic) with the **Resolve** method. This returns a delegate that, when invoked, calls into the container and returns an instance of the specified type (in this case, **T**).

You can even create a delegate in this way without creating a registration or mapping for the specified type in the container if you wish. Because the resolve action only takes place when you invoke the delegate, subsequent registrations added to the container are available when the target object is resolved. This means that you can manipulate the registrations and mappings in the container at any point before you resolve the target object (although you can obviously register the type before you create the delegate if you prefer).

For example, you can create a delegate for a component named MyClass, and then register a mapping for it and perform deferred resolution when required using the following code.

```
// Create a Unity container
IUnityContainer myContainer = new UnityContainer();

// Create a delegate for the IMyClass interface type
var resolver = myContainer.Resolve<Func<IMyClass>>();

// ... other code here...

// Register a mapping for the IMyClass interface to the myContainer.RegisterType<IMyClass, MyClass>();

// Resolve the mapped target object
IMyClass myClassInstance = resolver();
```

```
' Create a Unity container
Dim myContainer As IUnityContainer = New UnityContaine
' Create a delegate for the IMyClass interface type
Dim resolver = myContainer.Resolve(Of Func(Of IMyClass
' ... other code here...
' Register a mapping for the IMyClass interface to the
myContainer.RegisterType(Of IMyClass, MyClass)()
' Resolve the mapped target object
Dim myClassInstance As IMyClass = resolver()
```

You can use this approach when you resolve the type using the **Resolve** method, or you can specify the delegate when you configure constructor, property setter, or method call injection. You can also use named (non-default) registrations by including the registration name in the call to the **Resolve** method and the **RegisterType** method, just as you would when using these methods for non-deferred resolution.

In addition, you can use this feature to perform deferred resolution of multiple named registrations, as an alternative to using the ResolveAll method. For example, if you have multiple named registrations for the **IMyClass** interface to suitable concrete types, you can obtain a collection of the resolved types. The following code illustrates this.

```
// Create a Unity container
IUnityContainer myContainer = new UnityContainer();
// Create an IEnumerable resolver for the IMyClass int
var resolver = myContainer.Resolve<Func<IEnumerable<IN
// ... other code here...</pre>
```

```
// Register mappings for the IMyClass interface to app
myContainer.RegisterType<IMyClass, FirstClass>("First"
myContainer.RegisterType<IMyClass, SecondClass>("Secon
myContainer.RegisterType<IMyClass, ThidClass>("Third")

// Resolve a collection of the mapped target objects
IEnumerable<IMyClass> myClassInstances = resolver();
```

- ' Create a Unity container

 Dim myContainer As IUnityContainer = New UnityContaine
- 'Create an IEnumerable resolver for the IMyClass inte Dim resolver = myContainer.Resolve(Of Func(Of IEnumera
- ' ... other code here...
- 'Register mappings for the IMyClass interface to appr myContainer.RegisterType(Of IMyClass, FirstClass)("Fir myContainer.RegisterType(Of IMyClass, SecondClass)("Se myContainer.RegisterType(Of IMyClass, ThidClass)("Thir
- ' Resolve a collection of the mapped target objects
 Dim myClassInstances As IEnumerable(Of IMyClass) = res

You can also use the deferred resolver to resolve instance registrations. For example, the following code shows how you can resolve an IEnumerable collection of string values.

```
C#

// Create a Unity container
IUnityContainer myContainer = new UnityContainer();

// Create an IEnumerable resolver for string instance
```

```
var resolver = myContainer.Resolve<Func<IEnumerable<st
// ... other code here...

// Register mappings for the IMyClass interface to app
myContainer.RegisterInstance("one", "FirstString");
myContainer.RegisterInstance("two", "SecondString");
myContainer.RegisterInstance("three", "ThirdString");
// Resolve a collection of the strings
IEnumerable<string> myStringInstances = resolver();
```

- ' Create a Unity container Dim myContainer As IUnityContainer = New UnityContaine
- 'Create an IEnumerable resolver for string instance r Dim resolver = myContainer.Resolve(Of Func(Of IEnumera
- ' ... other code here...
- 'Register mappings for the IMyClass interface to appr myContainer.RegisterInstance("one", "FirstString") myContainer.RegisterInstance("two", "SecondString") myContainer.RegisterInstance("three", "ThirdString")
- ' Resolve a collection of the strings
 Dim myStringInstances As IEnumerable(Of String) = reso

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Retrieving Container Registration Information



You can retrieve a list of registrations from a container, and check if a specific registration is in the container.

This topic contains the following sections:

- Viewing the Container Registrations and Mappings
- Checking for the Existence of a Specific Registration

■ Viewing the Container Registrations and Mappings

■ Checking for the Existence of a Specific Registration

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Using Container Hierarchies



Unity supports nested containers, allowing you to build container hierarchies. Nesting containers enables you to control the scope and lifetime of singleton objects, and register different mappings for specific types. This topic contains the following sections that describe how you can create container hierarchies and use them in your applications:

- Constructing and Disposing Unity Containers
- Controlling Object Scope and Lifetime
- Registering Different Mappings for Specific Types

■ Constructing and	d Disposing U	Unity (Containers	

■ Controlling Object Scope and Lifetime

■ Registering Different Mappings for Specific Types

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Understanding Lifetime Managers



The Unity container manages the creation and resolution of objects based on a lifetime you specify when you register the type of an existing object, and uses the default lifetime if you do not specify a lifetime manager for your type registration.

When you register a type in configuration, or by using the **RegisterType** method, the default behavior is for the container to use a transient lifetime manager. It creates a new instance of the registered, mapped, or requested type each time you call the **Resolve** or **ResolveAll** method or when the dependency mechanism injects instances into other classes. The container does not store a reference to the object. However, when you want nontransient behavior (such as a singleton) for objects Unity creates, the container must store a reference to these objects. It must also take over management of the lifetime of these objects.

Unity uses specific types that inherit from the **LifetimeManager** base class (collectively referred to as lifetime managers) to control how it stores references to object instances and how the container disposes of these instances.

When you register an existing object using the **RegisterInstance** method, the default behavior is for the container to take over management of the lifetime of the object you pass to this method using the

ContainerControlledLifetimeManager. This means that at the end of the container lifetime, the existing object is disposed. You can also use this lifetime manager when defining registrations in configuration, or when using the **RegisterType** method, to specify that Unity should manage the object as a singleton instance.

Using a non-default lifetime manager with **RegisterInstance** will result in different behaviors, depending on the context of the requests.

- **Resolve** requests in the same context where the **RegisterInstance** call was made, such as the same thread if using a per-thread manager, or the same parent container when using the hierarchical one, will return the registered instances.
- **Resolve** requests in other contexts, such as a different thread if using a per-thread manager, or a child container when using the hierarchical

lifetime manager, will result in a new instance being created by the container and it will be made the singleton for that context. The creation of an instance under these circumstances could fail if the container cannot resolve the instance, for example if you registered an instance for an interface with no mappings to a matching class.

For information about using lifetime managers with the **RegisterType** and **RegisterInstance** methods, see <u>Registering Types and Type Mappings</u> and <u>Creating Instance Registrations</u> in the <u>Run-Time Configuration</u> section of this documentation. For information about specifying the lifetime of objects at design time, see <u>Specifying Types in the Configuration File</u>, and <u>The <instance></u> Element in the <u>Design-Time Configuration</u> section of this documentation.

■ Unity Built-In Lifetime Managers

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Dependency Injection with Unity



If you are using dependency injection (DI) through a DI container approach to your application development, you can use any available DI container including the container provided by Unity. Using the Unity dependency injection container provides opportunities for you to more easily decouple components, business objects, and services you use in applications, and can simplify how you organize and architect these applications.

You can create instances of objects using the DI container provided by Unity. Unity is available as a stand-alone dependency injection mechanism.

The following sections of this topic will help you to understand the overall process, and use Unity dependency injection in your applications:

- <u>Using BuildUp to Wire Up Objects Not Created by the Container</u>. This topic explains how to use BuildUp to pass existing object instances through the container in order to apply dependency injection to that object. This is an alternative to resolving the object using any of the other techniques available with Unity.
- <u>Using Injection Attributes</u>. This topic contains a series of sections that describe how you can use attributes applied to members of target classes to instruct Unity to inject dependent objects for constructor and method parameters, and as the values of properties.
- <u>Circular References with Dependency Injection</u>. This topic describes how you should be aware of the possibility of circular references arising when using dependency injection techniques.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Using BuildUp to Wire Up Objects Not Created by the Container



Unity exposes a method named **BuildUp** that you can use to pass existing object instances through the container in order to apply dependency injection to that object. This is an alternative to resolving the object using any of the other techniques available with Unity. However, remember that the **BuildUp** method cannot inject dependent objects into constructor parameters, because the object has already been created; it is not created by Unity.

The **BuildUp** method is useful when you do not have control of the construction of an instance, but you still want property or method call injection performed. For example, ASP.NET pages, Windows Communication Foundation (WCF) applications, and XAML code often create instances of objects and pass a reference to your code. The **BuildUp** method will usually return the original object after passing it through the container, although container extensions may add other features that cause the method to return a different object that is typecompatible with the existing object. For example, an injection strategy may create and return a proxy for an object or a derived object instead of the actual object.

If you have created or added extensions to the Unity container, these extensions can access and use a name that you specify when you execute the **BuildUp** method. This allows the extensions to change their behavior, depending on the value you specify. For example, they may use the name to control how dependencies are resolved or to control features such as event wiring or interception. The actual behavior depends on the individual extension.

■ The BuildUp Method Overloads

■ Using the BuildUp Method

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Using Injection Attributes**



One of the most useful and powerful techniques when using Unity is to take advantage of dependency injection for the parameters of class constructors and methods, and for the values of properties. This approach allows you to resolve and populate the entire hierarchy of objects used in your application based on type registrations and mappings defined in the container, with the subsequent advantages this offers.

You can specify constructor, property, and method call injection information in configuration or by adding registrations to the container at run time. You can also apply attributes to members of your classes. When you resolve these classes through the container, Unity will generate instances of the dependent objects and wire up the target class with these instances.

Unity performs constructor injection automatically on resolved classes, choosing the most complex constructor and populating any parameters for which you do not provide values when it constructs the object. You can also specify which constructor Unity should use to construct the object. For more information, see <u>Annotating Objects for Constructor Injection</u>.

Property and method call injection do not occur automatically unless you have registered injection types in the container at design time or run time. If you have not registered injection types in the container, you can add attributes to the members of your resolved class to force injection of dependent objects when the target class is resolved. For more information, see <u>Annotating Objects for Property (Setter) Injection</u> and <u>Annotating Objects for Method Call Injection</u>.

For information about registering injection types in the configuration at design time or run time, see <u>Configuring Unity</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Annotating Objects for Constructor Injection



Unity supports automatic dependency injection for class constructors. You can use the Unity container to generate instances of dependent objects and wire up the target class with these instances. This topic explains how to use both the automatic constructor injection mechanism and an attribute applied to the constructor of a class to define the dependency injection requirements of that class. The attribute can also specify parameters that the constructor will pass to the dependent object that the container generates.

To perform injection of dependent classes into objects you create through the Unity container, you can use the following techniques:

- <u>Single Constructor Automatic Injection</u>. With this technique, you allow the Unity container to satisfy any constructor dependencies defined in parameters of the constructor automatically. You use this technique when there is a single constructor in the target class.
- <u>Specifying Named Type Mappings</u>. With this technique, you specify named mappings for dependencies in the parameters of a class constructor. Named mappings allow you to specify more than one mapping for an interface or base class, or for a type registration.
- Multiple Constructor Injection Using an Attribute. With this technique, you apply attributes to the class constructor(s) that specify the dependencies. You use this technique when there is more than one constructor in the target class.

Constructor injection is a form of mandatory injection of dependent objects, as long as developers use the Unity container to generate the target object. The dependent object instance is generated when the Unity container creates an instance of the target class using the constructor. For more information, see Notes on Using Constructor Injection.

Single Constructor Automatic Injection

For automatic constructor injection, you simply specify as parameters of the constructor the dependent object types. You can specify the concrete type, or specify an interface or base class for which the Unity container contains a registered mapping.

To use automatic single-constructor injection to create dependent objects

1. Define a constructor in the target class that takes as a parameter the concrete type of the dependent class. For example, the following code shows a target class named **MyObject** containing a constructor that has a dependency on a class named **MyDependentClass**.

```
public class MyObject
{
  public MyObject(MyDependentClass myInstance)
  {
    // work with the dependent instance
    myInstance.SomeProperty = "SomeValue";
    // or assign it to a class-level variable
  }
}
```

```
Public Class MyObject
Public Sub New(myInstance As MyDependentClass)
' work with the dependent instance
myInstance.SomeProperty = "SomeValue"
' or assign it to a class-level variable
End Sub
End Class
```

2. In your run-time code, use the **Resolve** method of the container to create an instance of the target class. The Unity container will instantiate the dependent concrete class and inject it into the target

class. For example, the following code shows how you can instantiate the example target class named **MyObject** containing a constructor that has a dependency on a class named **MyDependentClass**.

```
IUnityContainer uContainer = new UnityContainer(
MyObject myInstance = uContainer.Resolve<MyObjec

Visual Basic

Dim uContainer As IUnityContainer = New UnityCon
Dim myInstance As MyObject = uContainer.Resolve(</pre>
```

3. Alternatively, you can define a target class that contains more than one dependency defined in constructor parameters. The Unity container will instantiate and inject an instance of each one. For example, the following code shows a target class named **MyObject** containing a constructor that has dependencies on two classes named **DependentClassA** and **DependentClassB**.

```
public class MyObject
{
  public MyObject(DependentClassA depA, Dependen
  {
     // work with the dependent instances
     depA.SomeClassAProperty = "SomeValue";
     depB.SomeClassBProperty = "AnotherValue";
     // or assign them to class-level variables
  }
}
```

Visual Basic

Public Class MyObject

```
Public Sub New(depA As DependentClassA, depB A
  ' work with the dependent instance
  depA.SomeClassAProperty = "SomeValue"
  depB.SomeClassBProperty = "AnotherValue"
  ' or assign them to class-level variables
  End Sub
End Class
```

4. In your run-time code, use the **Resolve** method of the container to create an instance of the target class. The Unity container will create an instance of each of the dependent concrete classes and inject them into the target class. For example, the following code shows how you can instantiate the example target class named **MyObject** containing a constructor that has constructor dependencies.

IUnityContainer uContainer = new UnityContainer(
MyObject myInstance = uContainer.Resolve<MyObjec

Visual Basic

Dim uContainer As IUnityContainer = New UnityCon
Dim myInstance As MyObject = uContainer.Resolve(</pre>

5. In addition to using concrete types as parameters of the target object constructor, you can use interfaces or base class types and then register mappings in the Unity container to translate these types into the correct concrete types. Define a constructor in the target class that takes as parameters the interface or base types of the dependent class. For example, the following code shows a target class named **MyObject** containing a constructor that has a dependency on a class that implements the interface named **IMyInterface** and a class that inherits from **MyBaseClass**.

C #	

```
public class MyObject
{
   public MyObject(IMyInterface interfaceObj, MyB
   {
      // work with the concrete dependent instance
      // or assign them to class-level variables
   }
}
```

```
Public Class MyObject
Public Sub New(interfaceObj As IMyInterface, b
' work with the dependent instance
' or assign them to class-level variables
End Sub
End Class
```

6. In your run-time code, register the mappings you require for the interface and base class types, and then use the **Resolve** method of the container to create an instance of the target class. The Unity container will instantiate an instance of each of the mapped concrete types for the dependent classes and inject them into the target class. For example, the following code shows how you can instantiate the example target class named **MyObject** containing a constructor that has a dependency on the two objects of type **IMyInterface** and **MyBaseClass**.

```
IUnityContainer uContainer = new UnityContainer(
    .RegisterType<IMyInterface, FirstObject>()
    .RegisterType<MyBaseClass, SecondObject>();
MyObject myInstance = uContainer.Resolve<MyObjec</pre>
Visual Basic

Dim uContainer As IUnityContainer = New UnityCon
```

```
.RegisterType(Of IMyInterface, FirstObject)()
.RegisterType(Of MyBaseClass, SecondObject)()
Dim myInstance As MyObject = uContainer.Resolve(
```

Specifying Named Type Mappings

The preceding example shows how you can resolve types for constructor parameters using the default (unnamed) mappings in the container. If you register more than one mapping for a type, you must differentiate them by using a name. In this case, you can specify which named mapping the container will use to resolve each constructor parameter type.

To use attributed constructor injection with named container type mappings

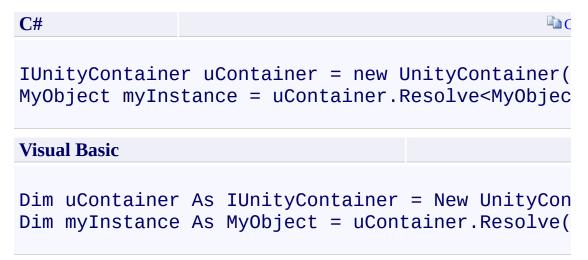
1. Define a constructor in the target class that takes as a parameter the concrete type of the dependent class, and apply a **Dependency** attribute to the parameter that specifies the name of the registered mapping to use. For example, the following code shows a target class named **MyObject** containing a constructor that has a dependency on a service registered with the name **myDataService**, and which implements the **IMyService** interface. It assumes that the container contains a mapping defined with the name **DataService** between the **IMyService** interface and a concrete implementation of this interface.

```
public class MyObject
{
  public MyObject([Dependency("DataService")] IM
  {
    // work with the service here
  }
}
```

Visual Basic

```
Public Class MyObject
   Public Sub New(<Dependency("DataService")> myD
   ' work with the service here
   End Sub
End Class
```

2. In your run-time code, use the **Resolve** method of the container to create an instance of the target class. The Unity container will instantiate the dependent concrete class defined in the mapping named **DataService** and inject it into the target class. For example, the following code shows how you can instantiate the example target class shown above.



You can use the **Dependency** attribute on more than one constructor parameter. You can also use it when the constructor defines more than one parameter of the same type to differentiate the mappings and ensure that the appropriate concrete type is returned for each parameter.

✓ Note:

If you specify a named mapping and there is no mapping registered for that type and name, the container will raise an exception.

Multiple Constructor Injection Using an Attribute

When a target class contains more than one constructor with the same number of parameters, you must apply the **InjectionConstructor** attribute to the constructor that the Unity container will use to indicate which constructor the container should use. As with automatic constructor injection, you can specify the constructor parameters as a concrete type, or you can specify an interface or base class for which the Unity container contains a registered mapping.

To use attributed constructor injection when there is more than one constructor

1. Apply the **InjectionConstructor** attribute to the constructor in the target class that you want the container to use. In the simplest case, the target constructor takes as a parameter the concrete type of the dependent class. For example, the following code shows a target class named **MyObject** containing two constructors, one of which has a dependency on a class named **MyDependentClass** and has the **InjectionConstructor** attribute applied.

```
Public Class MyObject

Public Sub New(myObjA As SomeOtherClass)
...
End Sub

<InjectionConstructor()> _
Public Sub New(myObjB As MyDependentClass)
...
End Sub

End Class
```

2. In your run-time code, use the **Resolve** method of the container to create an instance of the target class. The Unity container will instantiate the dependent concrete class defined in the attributed constructor and inject it into the target class. For example, the following code shows how you can instantiate the example target class named **MyObject** containing an attributed constructor that has a dependency on a class named **MyDependentClass**.

```
IUnityContainer uContainer = new UnityContainer(
MyObject myInstance = uContainer.Resolve<MyObjec

Visual Basic

Dim uContainer As IUnityContainer = New UnityCon
Dim myInstance As MyObject = uContainer.Resolve(</pre>
```

3. Alternatively, you can define a multiple-constructor target class that contains more than one dependency defined in the target constructor parameters. The Unity container will instantiate and inject an instance of each one. For example, the following code shows a target class named **MyObject** containing an attributed constructor that has dependencies on two classes, **DependentClassA** and

DependentClassB.

```
C#
public class MyObject
  public MyObject(SomeClassA objA, SomeClassB ob
  [InjectionConstructor]
  public MyObject(DependentClassA depA, Dependen
  }
}
Visual Basic
Public Class MyObject
  Public Sub New(objA As SomeClassA, objB As Som
  End Sub
  <InjectionConstructor()>
  Public Sub New(depA As DependentClassA, depB A
  End Sub
End Class
```

4. In your run-time code, use the **Resolve** method of the container to create an instance of the target class. The Unity container will create an instance of each of the dependent concrete classes defined in the

attributed constructor and inject them into the target class. For example, the following code shows how you can instantiate the example target class named **MyObject** containing a constructor that has constructor dependencies

IUnityContainer uContainer = new UnityContainer(
MyObject myInstance = uContainer.Resolve<MyObjec

Visual Basic

Dim uContainer As IUnityContainer = New UnityCon
Dim myInstance As MyObject = uContainer.Resolve(</pre>

5. In addition to using concrete types as parameters of the target object constructor, you can use interfaces or base class types, and then register mappings in the Unity container to translate these types into the correct concrete types. For details, see steps 5 and 6 of the procedure Single Constructor Automatic Injection.

■ Notes on Using Constructor Injection

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Annotating Objects for Property (Setter) Injection



Unity supports dependency injection to set the values or properties through attributes applied to members of the target class. You can use the Unity container to generate instances of dependent objects and wire up the target class properties with these instances. This topic explains how to use an attribute that is applied to one or more property declarations of a class to define the dependency injection requirements of that class. The attribute can specify parameters for the attribute to control its behavior, such as the name of a registered mapping.

To perform property injection of dependent classes into objects you create through the Unity container, you apply the **Dependency** attribute to the property declarations of a class. The Unity container will create an instance of the dependent class within the scope of the target object (the object you specify in a **Resolve** method call) and assign this dependent object to the attributed property of the target object.

Property injection is a form of optional injection of dependent objects, as long as developers use the Unity container to generate the target object. The dependent object instance is generated before the container returns the target object. In addition, unlike constructor injection, you must apply the appropriate attribute in the target class to initiate property injection. You can also perform property injection of optional dependent classes by applying the **OptionalDependency** attribute. This simply marks a dependency as optional, which means that the container will try to resolve it, and return **null** if the resolution fails rather than throw an exception. For more information, see Notes

To use property (setter) injection to create dependent objects for a class

1. Define a property in the target class and apply the **Dependency** attribute to it to indicate that the type defined and exposed by the property is a dependency of the class. The following code demonstrates property injection for a class named **MyObject** that exposes as a property a reference to an instance of another class named **SomeOtherObject** (not defined in this code).

C#

Copy Code

```
public class MyObject
{
   private SomeOtherObject _dependentObject;

   [Dependency]
   public SomeOtherObject DependentObject
   {
     get { return _dependentObject; }
     set { _dependentObject = value; }
   }
}
```

```
Public Class MyObject
Private _dependentObject As SomeOtherObject

<Dependency()> _
Public Property DependentObject() As SomeOther
Get
Return _dependentObject
End Get
Set(ByVal value As SomeOtherObject)
_dependentObject = value
End Set
End Property

End Class
```

2. In your run-time code, use the **Resolve** method of the container to create an instance of the target class, and then reference the property containing the dependent object. The Unity container will instantiate the dependent concrete class defined in the attributed property and inject it into the target class. For example, the following code shows how you can instantiate the example target class named **MyObject** containing an attributed property that has a dependency on a class

named **SomeOtherObject** and then retrieve the dependent object from the **DependentObject** property.

```
IUnityContainer uContainer = new UnityContainer(
MyObject myInstance = uContainer.Resolve<MyObjec

// now access the property containing the depend
SomeOtherObject depObj = myInstance.DependentObj</pre>
```

Visual Basic

Dim uContainer As IUnityContainer = New UnityCon Dim myInstance As MyObject = uContainer.Resolve(

' now access the property containing the depende Dim depObj As SomeOtherObject = myInstance.Depen

3. In addition to using concrete types for the dependencies in target object properties, you can use interfaces or base class types, and then register mappings in the Unity container to translate these types into the correct concrete types. Define a property in the target class as an interface or base type. For example, the following code shows a target class named MyObject containing properties named InterfaceObject and BaseObject that have dependencies on a class that implements the interface named IMyInterface and on a class that inherits from MyBaseClass.

```
public class MyObject
{
  private IMyInterface _interfaceObj;
  private MyBaseClass _baseObj;
  [Dependency]
```

```
public IMyInterface InterfaceObject
{
    get { return _interfaceObj; }
    set { _interfaceObj = value; }
}

[Dependency]
public MyBaseClass BaseObject
{
    get { return _baseObj; }
    set { _baseObj = value; }
}
```

Visual Basic Copy

```
Public Class MyObject
  Private _interfaceObj As IMyInterface
  Private _baseObj As MyBaseClass
  <Dependency()> _
  Public Property InterfaceObject() As IMyInterf
    Get
      Return _interfaceObj
    End Get
    Set(ByVal value As IMyInterface)
      _interfaceObj = value
    End Set
  End Property
  <Dependency()> _
  Public Property BaseObject() As MyBaseClass
    Get
      Return _baseObj
    End Get
```

```
Set(ByVal value As MyBaseClass)
_baseObj = value
End Set
End Property

End Class
```

4. In your run-time code, register the mappings you require for the interface and base class types, and then use the **Resolve** method of the container to create an instance of the target class. The Unity container will create an instance of each of the mapped concrete types for the dependent classes and inject them into the target class. For example, the following code shows how you can instantiate the example target class named **MyObject** containing two properties that have dependencies on the two classes named **FirstObject** and **SecondObject**.

C#

```
IUnityContainer uContainer = new UnityContainer(
    .RegisterType<IMyInterface, FirstObject>()
    .RegisterType<MyBaseClass, SecondObject>();
MyObject myInstance = uContainer.Resolve<MyObjec</pre>
```

// now access the properties containing the depe
IMyInterface depObjA = myInstance.InterfaceObjec
MyBaseClass depObjB = myInstance.BaseObject;

Visual Basic

```
Dim uContainer As IUnityContainer = New UnityCon
    .RegisterType(Of IMyInterface, FirstObject)()
    .RegisterType(Of MyBaseClass, SecondObject)()
Dim myInstance As MyObject = uContainer.Resolve()
```

' now access the properties containing the depen Dim depObjA As IMyInterface = myInstance.Interfa Dim depObjB As MyBaseClass = myInstance.BaseObje 5. You can register multiple named mappings with the container for each dependency type, if required, and then use a parameter of the **Dependency** attribute to specify the mapping you want to use to resolve the dependent object type. For example, the following code specifies the mapping names for the **Key** property of the **Dependency** attribute for two properties of the same type (in this case, an interface) in the class **MyObject**.

```
C#
                                         Copy Code
public class MyObject
{
  private IMyInterface _objA, _objB;
  [Dependency("MapTypeA")]
  public IMyInterface ObjectA
    get { return _objA; }
    set { _objA = value; }
  }
  [Dependency("MapTypeB")]
  public IMyInterface ObjectB
    get { return _objB; }
    set { _objB = value; }
  }
}
Visual Basic
                                         Copy Code
Public Class MyObject
  Private _objA, _objB As IMyInterface
  <Dependency("MapTypeA")> _
```

```
Public Property ObjectA() As IMyInterface
    Get
      Return _objA
    End Get
    Set(ByVal value As IMyInterface)
      obiA = value
    End Set
  End Property
  <Dependency("MapTypeB")>
  Public Property ObjectB() As IMyInterface
    Get
      Return _objB
    End Get
    Set(ByVal value As IMyInterface)
      _objB = value
    End Set
  End Property
End Class
```

6. In your run-time code, register the named (non-default) mappings you require for the two concrete types that the properties will depend on, and then use the **Resolve** method of the container to create an instance of the target class. The Unity container will instantiate an instance of each of the mapped concrete types for the dependent classes and inject them into the target class. For example, the following code shows how you can instantiate the example target class named **MyObject** containing two properties that have dependencies on the two classes named **FirstObject** and **SecondObject**.

```
IUnityContainer uContainer = new UnityContainer(
    .RegisterType<IMyInterface, FirstObject>("Map
    .RegisterType<IMyInterface, SecondObject>("Map
MyObject myInstance = uContainer.Resolve<MyObject</pre>
```

```
// now access the properties containing the depe
IMyInterface depObjA = myInstance.ObjectA;
IMyInterface depObjB = myInstance.ObjectB;
```

Visual Basic

```
Dim uContainer As IUnityContainer = New UnityCon
    .RegisterType(Of IMyInterface, FirstObject)("
    .RegisterType(Of IMyInterface, SecondObject)(
Dim myInstance As MyObject = uContainer.Resolve()
```

' now access the properties containing the depen Dim depObjA As IMyInterface = myInstance.ObjectA Dim depObjB As IMyInterface = myInstance.ObjectB

■ Using Optional Dependencies

■ Notes on Using Property (Setter) Injection

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Annotating Objects for Method Call Injection



Unity supports dependency injection to set the values of parameters of methods specified through attributes applied to members of the target class. You can use the Unity container to generate instances of dependent objects and wire up the target class method parameters with these instances. This topic explains how to use an attribute that is applied to one or more method declarations of a class to define the dependency injection requirements of that class.

To perform injection of dependent classes into objects you create through the Unity container, you apply the **InjectionMethod** attribute to the method declarations of a class. The Unity container will force the target object (the object you specify in a **Resolve** method call) to create an instance of the dependent class and then call the target method. If required, your code in the method can save this instance by assigning it to a class-level variable.

Method call injection is a form of optional injection of dependent objects that you can use if you use the Unity container to generate the target object. Unity instantiates dependent objects defined in parameters of methods that carry the **InjectionMethod** attribute within the scope of the target object. Then it calls the attributed method of the target object before returning the object to the caller. You must apply the **InjectionMethod** attribute in the target class to initiate method call injection. For more information, see <u>Notes on Using Method Call Injection</u>.

To use method call injection to create dependent objects for a class

1. Define a method in the target class and apply the **InjectionMethod** attribute to it to indicate that any types defined in parameters of the method are dependencies of the class. The following code demonstrates the most common scenario—saving the dependent object instance in a class-level variable—for a class named **MyObject** that exposes a method named **Initialize** that takes as a parameter a reference to an instance of another class named **SomeOtherObject** (not defined in this code).

C#		

```
public class MyObject
{
   private SomeOtherObject dependentObject;
   [InjectionMethod]
   public void Initialize(SomeOtherObject dep)
   {
      // assign the dependent object to a class-le dependentObject = dep;
   }
}

Visual Basic

Public Class MyObject
   Private dependentObject As SomeOtherObject
```

Public Sub Initialize(dep As SomeOtherObject)

assign the dependent object to a class-lev

End Class

End Sub

<InjectionMethod()> _

dependentObject = dep

2. In your run-time code, use the **Resolve** method of the container to create an instance of the target class. The Unity container will instantiate the dependent concrete class defined in the attributed method, inject it into the target class, and execute the method. For example, the following code shows how you can instantiate the example target class named **MyObject** containing an attributed method that has a dependency on a class named **SomeOtherObject**.

C#

IUnityContainer uContainer = new UnityContainer(
MyObject myInstance = uContainer.Resolve<MyObjec</pre>

Visual Basic

```
Dim uContainer As IUnityContainer = New UnityCon
Dim myInstance As MyObject = uContainer.Resolve(
```

3. In addition to using concrete types for the dependencies in target object methods, you can use interfaces or base class types and then register mappings in the Unity container to translate these types into the appropriate concrete types. Define a method in the target class that takes as parameters interfaces or base types. For example, the following code shows a target class named **MyObject** containing a method named **Initialize** that takes as parameters an object named **interfaceObj** that implements the interface named **IMyInterface** and an object named **baseObj** that inherits from the class **MyBaseClass**.

```
public class MyObject
{
  private IMyInterface depObjectA;
  private MyBaseClass depObjectB;

  [InjectionMethod]
  public void Initialize(IMyInterface interfaceO)
  {
    depObjectA = interfaceObj;
    depObjectB = baseObj;
  }
}
```

Visual Basic

Public Class MyObject

```
Private depObjectA As IMyInterface
Private depObjectB As MyBaseClass

<InjectionMethod()> _
Public Sub Initialize(interfaceObj As IMyInter
    depObjectA = interfaceObj
    depObjectB = baseObj
End Sub
End Class
```

4. In your run-time code, register the mappings you require for the interface and base class types, and then use the **Resolve** method of the container to create an instance of the target class. The Unity container will instantiate an instance of each of the mapped concrete types for the dependent classes, and inject them into the target class. For example, the following code shows how you can instantiate the example target class named **MyObject** containing an attributed method that has dependencies on the two classes, **FirstObject** and **SecondObject**.

```
IUnityContainer uContainer = new UnityContainer(
    .RegisterType<IMyInterface, FirstObject>()
    .RegisterType<MyBaseClass, SecondObject>();
MyObject myInstance = uContainer.Resolve<MyObjec

Visual Basic

Dim uContainer As IUnityContainer = New UnityCon
    .RegisterType(Of IMyInterface, FirstObject)()
    .RegisterType(Of MyBaseClass, SecondObject)()
Dim myInstance As MyObject = uContainer.Resolve()</pre>
```

Specifying Named Type Mappings

The preceding example shows how you can resolve types for method parameters using the default (unnamed) mappings in the container. If you register more than one mapping for a type, you must differentiate them by using a name. In this case, you can specify which named mapping the container will use to resolve the method parameter types.

To use attributed method call injection with named container type mappings

1. Define a method in the target class that takes as a parameter the concrete type of the dependent class, and apply a **Dependency** attribute to the parameter that specifies the name of the registered mapping to use. For example, the following code shows a target class named **MyObject** containing a method named **Initialize** that has a dependency on a service that implements the **IMyService** interface. The code assumes that the container contains a mapping defined with the name **DataService** between the **IMyService** interface and a concrete implementation of that interface.

```
public class MyObject
{
  private IMyService myDataService;

  [InjectionMethod]
  public void Initialize([Dependency("DataServic {
      // assign the dependent object to a class-le myDataService = theService;
  }
}

Visual Basic

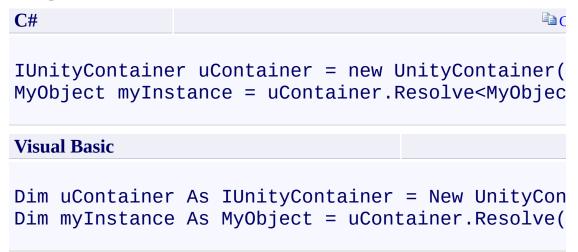
Public Class MyObject
```

```
Private myDataService As IMyService

<InjectionMethod()> _
  Public Sub Initialize(<Dependency("DataService
    ' assign the dependent object to a class-lev
    myDataService = theService
  End Sub

End Class</pre>
```

2. In your run-time code, use the **Resolve** method of the container to create an instance of the target class. The Unity container will instantiate the dependent concrete class defined in the attributed method, inject it into the target class, and execute the method. For example, the following code shows how you can instantiate the example class shown above.



✓ Note:

If you specify a named mapping and there is no mapping registered for that type and name, the container will raise an exception.

■ Notes on Using Method Call Injection

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0 Circular References with Dependency Injection

Dependency injection mechanisms carry the risk of unintentional circular references, which are not easy to detect or prevent. This topic describes the situations where you may inadvertently cause circular references to occur, resulting in a stack overflow and application error. The most common causes of circular references with dependency injection are the following:

- Objects generated through constructor injection that reference each other in their constructor parameters
- Objects generated through constructor injection where an instance of a class is passed as a parameter to its own constructor
- Objects generated through method call injection that reference each other
- Objects generated through property (setter) injection that reference each other

For example, the following code shows two classes that reference each other in their constructors.

```
### Copy Code

public class Class1

{
    public Class1(Class2 test2)
    { . . . . }
}

public class Class2

{
    public Class2(Class1 test1)
    { . . . . }
}

Visual Basic

Public Class Class1
    Public Sub New(test2 As Class2)
```

```
End Sub
End Class

Public Class Class2
Public Sub New (test 1 As Class1)
...
End Sub
End Class
```

It is the responsibility of the developer to prevent this type of error by ensuring that the members of classes they use with dependency injection do not contain circular references.

✓ Note:

You could use constructor injection to specify any of a series of constructors or method overloads; however, you could inadvertently cause endless recursion. To avoid the endless recursion, specify which constructor to call in the **RegisterType** call.

Unity's default behavior is to resolve the constructor with the most parameters. This would cause endless recursion in the following example.

```
C#

container.RegisterType<IServiceProvider, ServiceContai
var sp = container.Resolve<IServiceProvider>();

Visual Basic

container.RegisterType(Of IServiceProvider, ServiceCor
Dim sp = container.Resolve(Of IServiceProvider)()
```

To avoid the endless recursion, specify which constructor to call in the **RegisterType** call, as in the following example:

C#

container.RegisterType<IServiceProvider, ServiceContai

Visual Basic

In this case, when creating the service container, the zero argument constructor is explicitly requested.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Interception with Unity



Unity interception enables you to effectively capture calls to objects and add additional functionality to the target object. Interception is useful when you want to modify the behavior for individual objects but not the entire class, very much as you would do when using the <u>Decorator pattern</u>. It provides a flexible approach for adding new behaviors to an object at run time.

This section contains the following topics that will help you to understand interception:

- <u>About Unity Interception</u>. This section of this topic describes the basic principles of interception in Unity.
- <u>Scenarios for Interception</u>. This topic describes common scenarios addressed by Unity interception.
- <u>Behaviors for Interception</u>. This topic describes behaviors you might implement with the **IInterceptionBehavior** interface to configure a container through the **RegisterType** method.
- <u>Configuring a Container for Interception</u>. This topic describes how to configure the Unity container for interception.
- <u>Unity Interception Techniques</u>. This topic describes in detail the design of Unity interception.
- <u>Using Interception in Applications</u>. This topic describes how you use Unity interception in your applications.
- <u>Using Interception and Policy Injection</u>. This topic explains how policy injection through interception works in Unity, how you can use matching rules to select target classes and class members for policy injection, and how you can use the Enterprise Library call handlers with Unity.

■ About Unity Interception

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright @ 2010 by Microsoft Corporation. All rights reserved.

Scenarios for Interception



Unity interception addresses the following scenarios:

- Adding responsibilities to individual objects and not the entire class and avoiding a static solution, much as in a decorator pattern. In a manner similar to the way a decorator forwards requests to the object and enables you to perform additional actions before or after forwarding the request, interception intercepts the call to the target object and dynamically adds behaviors to individual objects without affecting any other objects. This can be useful in managing crosscutting concerns that access common features such as logging or validation.
- To augment or modify the behavior from existing classes that you cannot modify, provided that they are interceptable by the available interception mechanisms.
- Enabling the developer and administrator to configure the behavior of objects in an application through configuration when used in conjunction with a dependency injection (DI) container, by adding or removing behaviors that execute common tasks or add custom features.
- Enabling the developer and administrator to capture calls to objects and add or remove behaviors that execute common tasks or add custom features at run time, but in this case independent of a DI container.
- Minimizing the work required and the code that the developer must write to perform common tasks within an application, such as logging, validation, authorization, and instrumentation.
- Reducing development time and cost, and minimizing bugs in complex applications that use common and shared tasks and services.

■ Benefits of Using Unity Interception

■ Limitations of Unity Interception

■ Alternatives to Using Unity Interception

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright © 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0 **Behaviors for Interception**



Interception is based on a behavior or series of behaviors in the behaviors pipeline that describe what to do when an object is intercepted. Unity provides a built-in default policy injection behavior to facilitate the implementation of policy injection. The policy injection behavior attaches or injects some functionality to specific methods by using call handlers and matching rules on a per-method basis. For more information on policy injection see Using Interception and Policy Injection.

You can also create your own custom behaviors by implementing the **IInterceptionBehavior** interface. The interception behaviors are added to a pipeline and are called for each invocation of that pipeline. You have wide latitude in what functionality you design for your behavior. Some practical uses of behaviors include implementing custom tasks and business rules, implementing **INotifyPropertyChanged** to support a property change event, to provide support for the **ErrorProvider/IDataErrorInfo** approach to validation in Windows® Presentation Foundation (WPF), and to implement a **Mocking** framework.

Custom tasks and business rules you might chose to implement with individual custom behaviors would include tasks such as validating parameters or authorizing users.

When implementing a WPF view model, in order to get the property change event action you must implement **INotifyPropertyChanged**. Rather than implementing this every time, you can create a behavior that adds and implements the interface.

There is no built-in support for the **ErrorProvider/IDataErrorInfo** approach to validation in WPF; hence there is no **ErrorProvider** component in WPF either. You must create an **ErrorProvider** for use in WPF applications. The **DataGrid** (1.1) and **DataGridView** (2.0) in Windows Forms both automatically detected the presence of this interface on objects they were bound to, and showed any errors without any work. The Windows Forms **ErrorProvider** could be used to automatically display errors on any control that came from the objects they (and the **ErrorProvider**) were bound to, all without any extra code being written. You can use **IDataErrorInfo** to take advantage of the validation work in the

.NET Framework. You can implement **IDataErrorInfo** in a class and bind the class concrete object to the **DataGridView** control through the **Bindingsource** property. Then use the Validation Application Block to validate the class object and store the results for each property.

Implementing a mocking framework through an interception behavior could be useful in cases where the code requires an interface that has no implementation. The behavior can give you a mock up interface. You could use a mocking framework to implement a mock database, mock logger, or mock builder context.

The following topics explain or demonstrate interception behaviors in more detail:

- Custom Interception Behaviors
- Implement IDataErrorInfo Example

■ Custom Interception Behaviors

■ Implement INotifyPropertyChanged Example

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright @ 2010 by Microsoft Corporation. All rights reserved.

Configuring a Container for Interception



You can use interception with or without a dependency injection (DI) container such as Unity. Using a DI container relieves you of the need to manually create all the dependencies and pass them into the correct objects. If you choose to use the Unity DI container, you can configure the container by using a configuration file or at run time by using the API.

In Unity, interception is just another extension point instead of a self-contained part of the configuration file, as in previous versions. Prior to Unity 2.0 when you specified interception, policy injection was implicitly configured by the underlying code. Where the behavior of interception was implicit before, it is explicit now; you specify that interception is to happen and you specify what is to happen upon interception. Interception becomes just one more thing that you can describe about how an object is resolved. Unity version 2.0 enables you to modify how interception happens and how the object is created.

There are two approaches for setting up Unity interception:

- The approach introduced in Unity 2.0 in which interception is configured as just another extension point element in the entry for a container type. Configure an extension point by using the <sectionExtension> and <extension> tags in the configuration file or by using the RegisterType method at run time.
- The pre-Unity 2.0 approach used in earlier versions of Unity in which interception is a self-contained part of the container configured by using the interception extension elements, <extensionConfig> and <interceptors> in the configuration file or by using the SetInterceptorFor or SetDefaultInterceptorFor methods on the interception container extension. This earlier style is still available primarily for backward compatibility.

You can use a configuration file to specify a container-created behavior with any logical configuration. For detailed interception schema information, see Configuration Files for Interception.

This topic contains the following sections to describe how to configure a container for interception:

- Adding the Interception Extension to the Container
- Configuring Interception of a Type
- More Information

■ Configuring Interception of a Type

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Unity Interception Techniques



In order to perform interception, Unity must be able to capture the original call and pass it through a behaviors pipeline to the target object, then pass the result back through the behaviors pipeline to the original caller. The two common approaches to the interception process are instance interception and type interception, and Unity provides techniques for both. Instance interceptors work by creating a proxy to the intercepted instance. Type interceptors work by deriving a new type that implements interception. The instance interception target is the original, non-intercepted object, but when performing type interception the target is intercepted and a derived object used.

✓ Note:

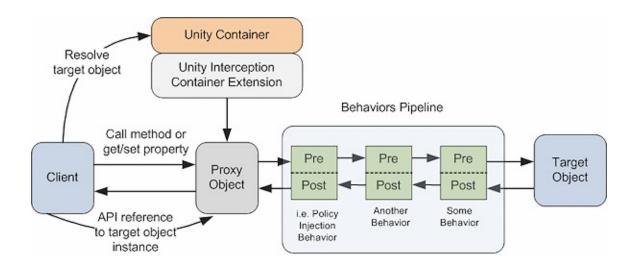
Instance interception works only on instance methods; it does not work for static methods or constructors since the constructor has already executed by the time the client code gets back an interception-ready object. Instance interception can only intercept public instance methods. Type interception can intercept public and protected methods.

This topic contains the following sections that will help you to understand interception:

- Instance Interception
- Type Interception
- Comparison of Interception Techniques
- Summary of Interception Approaches

Instance Interception

Instance interception works with both existing instances of objects, and with new instances created by Unity. The following schematic shows the basic process of instance interception.



When the application resolves the object through the Unity container, the Unity interception container extension manages the process. It obtains a new or existing instance of the object from the container, and creates a proxy to the object. Then it creates the handler pipeline and connects it to the target object before returning a reference to the proxy. The client then calls methods and sets properties on the proxy as though it were the target object.

✓ Note:

Unity interception can be used without a Unity DI container by using the stand-alone API through the static **Intercept** class. For more information, see <u>Using Interception in Applications</u>.

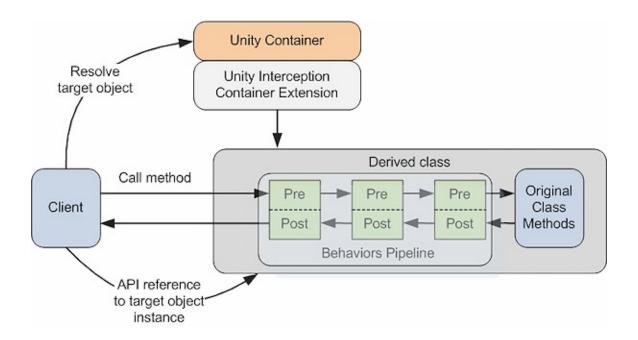
These calls pass through the interception behaviors, executing the preprocessing stage of each one, with the final behavior in the chain passing the call to the target object. The return value from the target object passes back through the behaviors in the reverse order, executing the post-processing stage of each one. The first behavior in the pipeline then passes the result back to the caller.

Instance interception is the most common and widely used technique. It can be used with objects that either implement the **MarshalByRefObject** abstract class, or implement a public interface that defines all of the methods to be intercepted. Unity provides the two interceptors,

TransparentProxyInterceptor and **InterfaceInterceptor**, that support these two scenarios. For more details, see the tables in <u>Comparison of Interception Techniques</u> later in this topic.

Type Interception

Type interception uses a derived class instead of a proxy. As described in the previous section, instance interception works by creating a proxy to the target object. Type interception, on the other hand, more closely resembles aspect-oriented programming (AOP) techniques common in Java-based systems. Type interception avoids the possible performance penalties of using a proxy object by dynamically deriving a new class from the original class, and inserting calls to the behaviors that make up the pipeline. The following schematic shows the basic process of type interception.



When the application resolves the required type through the Unity container, the Unity interception container extension creates the new derived type and passes it, rather than the resolved type, back to the caller. Because the type passed to the caller derives from the original class, it can be used in the same way as the original class. The caller simply calls the object, and the derived class will pass the call through the behaviors in the pipeline just as is done when using instance interception.

✓ Note:

Unity interception can be used without a Unity DI container by using the stand-alone API through the static **Intercept** class. For more information, see <u>Using Interception in Applications</u>.

However, due to the nature of the dynamic type generation, there are some limitations with this approach. It can only be used to intercept public and protected virtual methods, and cannot be used with existing object instances. In general, type interception is most suited to scenarios where you create objects especially to support interception and allow for the flexibility and decoupling provided by policy injection, or when you have mappings in your container for base classes that expose virtual methods. For more details, see the tables of comparisons in the following topic Comparison of Interception Techniques.

Comparison of Interception Techniques

The previous sections demonstrated how it is important to choose the appropriate interception technique based on your requirements and the type of object you want to intercept. The following table lists the three interceptor classes included in Unity, and describes when you should use each type.

Type	Description	Use
Transparent Proxy Interceptor	An instance interceptor. The proxy is created by using the .NET TransparentProxy/RealProxy infrastructure.	When the type to intercept is a MarshalByRefObject or when only methods from the type's implemented interfaces need to be intercepted.
Interface Interceptor	An instance interceptor. It can proxy only one interface on the object. It uses dynamic code generation to create the proxy class.	When resolving an interface mapped to a type.
Virtual Method Interceptor	A type interceptor. It uses dynamic code generation to create a derived class that is instantiated instead of the original intercepted class, and to hook up the behaviors.	When only virtual methods need to be intercepted.

Selection of a specific interceptor depends on your specific needs, because each one has various tradeoffs. The following table summarizes the three interceptors and their advantages and disadvantages.

Type	Advantages	Disadvantages
Proxy	Can intercept all methods of the target object (virtual, non-	The object must either implement an interface or
Interceptor	virtual, or interface).	inherit from System.MarshalByRefObjec If the marshal by reference

		object is not a base class, you can only proxy interface methods. The Transparent Proxy process is much slower than a regular method call.
Interface Interceptor	Allows interception on any object that implements the target interface. It is much faster than the TransparentProxyInterceptor .	It only intercepts methods on a single interface. It cannot cast proxy back to the target object class or to other interfaces on the target object.
Virtual Method Interceptor	Calls are much faster than the Transparent Proxy Interceptor.	Interception only happens on virtual methods. You must set up interception at object creation time and cannot intercept an existing object.

Summary of Interception Approaches

Unity provides instance and type interception with intercepted objects for which you have obtained a reference either through the container, or by using the stand-alone API to explicitly intercept a known instance. Instance interceptors use a separate proxy object between your code and your target object. Using interception, you make a call on the proxy object instead of directly calling the target object. The proxy invokes the various interception behaviors, and then it forwards the call to the target object. Different implementations of instance interceptors can have different constraints. Instance interceptors have the following characteristics:

- They can intercept objects created by the container.
- They can intercept objects not created by the container.
- The **TransparentProxyInterceptor** can intercept more than one interface, and marshal-by-reference objects.
- The **InterfaceInterceptor** can only be used to intercept a single interface.

Type interceptors create a new type that inherits from the target type. This new type is then instantiated instead of the original type you requested. The new type overrides all the virtual methods on the original target type. Type instance interceptors have the following characteristics:

- Only one object is created; there is no proxy object between the caller and the new object.
- The new object has full type compatibility because it is derived from the target type.
- They are able to intercept objects only at creation, and cannot intercept existing instances.
- They can only intercept virtual public and protected methods.

The system that Unity uses to automatically create a derived target object, or a proxy and behaviors pipeline, is similar to the aspect-oriented programming (AOP) approach. However, Unity is not an AOP framework implementation for the following reasons:

- It uses interception to enable only preprocessing behaviors and postprocessing behaviors.
- It does not insert code into methods, although it can create derived

classes containing policy pipelines.

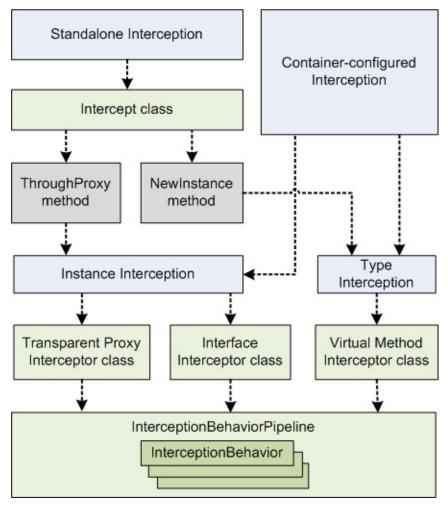
• It does not provide interception for class constructors.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Using Interception in Applications

This topic describes how to use Unity interception in your applications. You can use Unity interception with a dependency injection (DI) container or as a standalone feature with no DI container.



When you use a DI container, the container provides a reference to the intercepted object. When the container resolves an object that is configured for interception, the object is intercepted as defined in the configuration. When performing stand-alone interception, you must already have a reference to the object you want intercepted. You invoke the stand-alone interception API and explicitly intercept the object. For more information on using containers, see Dependency Injection with Unity.

Unity interception requires an interceptor, and a collection of interception

behaviors that comprise a pipeline. Interceptors are used when performing interception through the container and through the stand-alone API. Using the container results in adding interception objects that are being resolved, while using the stand-alone API enables you to perform just interception.

In both cases you can implement additional interfaces on intercepted objects. You must provide the information for the additional interfaces that will be added for interception. Using additional interfaces has some limitations. You cannot add open generic interfaces and you cannot reimplement an interface that has already been implemented with nonvirtual methods. Additional interfaces are supported by all three types of interceptors. When you are performing interface interception, you can only invoke methods on the particular interface you used to intercept and any additional interfaces that you have specified. Casts to other interfaces implemented by the intercepted object will fail.

Though behaviors normally can have both pre- and post-processing functionality, with any additional interfaces the behaviors must handle the processing in its entirety. There cannot be any post-processing because the original object has no implementation for the additional interfaces' methods. Any calls to the additional interface methods handled by the behaviors will result in a **NotImplementedException** being thrown.

You can also intercept abstract classes with abstract methods, allowing the behaviors pipeline to provide the implementation for these abstract methods. This is similar to adding interfaces in that in both cases there are undefined methods for which an implementation must be provided, but they are different mechanisms. Abstract class interception only works with the

VirtualMethodInterceptor. You can implement abstract classes with abstract methods. If you intercept a type that already implements some interfaces, both virtual method and transparent proxy interception allow for casting to these interfaces and invoke methods on them, which will be intercepted. There cannot be any post processing because the original object has no implementation for the additional interfaces' methods

This topic contains the following sections:

- <u>Stand-alone Unity Interception</u> describes how to use Unity interception as a stand-alone feature.
- <u>Interception Behavior Pipeline</u> describes how to add a behavior and interceptor to a new interception behaviors pipeline.

• <u>Interception with a Container</u> describes interception with a container.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Stand-alone Unity Interception



You can use Unity interception as a stand-alone feature with no dependency injection container by using the **Intercept** class. As with a container, interception as a stand-alone feature enables you to perform instance or type interception. The **Intercept** class contains the

NewInstance,NewInstanceWithAdditionalInterfaces,ThroughProxy, and ThroughProxyWithAdditionalInterfaces methods, enabling you to perform either proxy or instance interception. And both methods include the AdditionalInterfaces parameter, enabling you to implement additional interfaces on the target object. This corresponds to the AdditionalInterface feature when using interception with a container.

✓ Note:

The first parameter on **Intercept.ThroughProxy**, **Intercept.ThroughProxyWithAdditionalInterfaces** and **Intercept. NewInstance** is the corresponding interceptor when setting up interception through the stand-alone API.

The **AdditionalInterfaces** parameter on the object enables you to receive more messages and to augment the set of methods the object can respond to.

This section contains the following sections describing stand-alone interception:

- <u>Stand-Alone Interception with a Proxy</u> describes creating a proxy to the intercepted instance.
- <u>Stand-Alone Interception with a Derived Type</u> describes interception by using a derived type.

■ Stand-Alone Interception with a Proxy

■ Stand-alone Interception with a Derived Type

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

You must add behaviors to the behavior pipeline to use them. The pipeline maintains a list of interception behaviors and manages them, calling them in the proper order with the correct inputs.

The following example adds a behavior and interceptor to a new interception behaviors pipeline.

First create an interception behaviors pipeline.

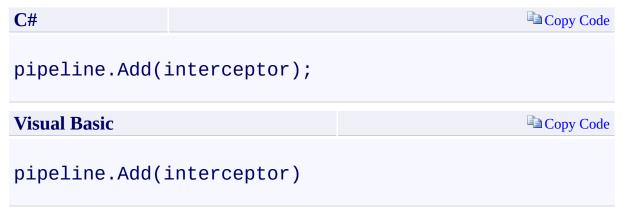
C#

private InterceptionBehaviorPipeline pipeline = new Ir

Visual Basic

Private pipeline As New InterceptionBehaviorPipeline()

Then add an interception behavior to the pipeline.



To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

$Microsoft\ Enterprise\ Library\ 5.0$

Interception with a Container



The Unity container extension enables you to configure a container for interception. You can use the container configuration to determine whether an object should be intercepted, which mechanism should be used to perform the interception, and what to do when the object is intercepted. It also provides a convenient set of methods for configuring injection for

Microsoft.Practices.Unity.InterceptionExtension.RuleDrivenPolicy instances.

For more information about configuring a container see <u>Configuring a Container for Interception</u> and <u>Configuring Unity</u>.

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Using Interception and Policy Injection



Policy injection by a combination of Unity and the <u>patterns & practices</u> <u>Enterprise Library</u> uses a set of call handlers and the equivalent call handler attributes in conjunction with the underlying Unity interception mechanism. Interception enables you to effectively capture calls to objects and provide additional functionality to the target object by using behaviors and call handlers in the pipeline to define and manage the results of the interception. In Enterprise Library, policy injection is just one implementation of a Unity interception behavior. The **PolicyInjectionBehavior** captures calls to objects you resolve through the container, and applies a policy that uses call handlers and matching rules inherited from Unity to define its policy injection behavior on a permethod basis.

Typically, you will use this technique to change the behavior of existing objects, or to implement the management of crosscutting concerns through reusable handlers. You can specify how to match the target object using a wide range of matching rules, and construct a behavior which is effectively a policy pipeline that contains one or more call handlers.

Calls to the intercepted methods or properties of the target object are passed through the call handlers in the order in which you add them to the pipeline, and returned through them in the reverse order. Your call handlers can access the values in the call, change these values, and control execution of the call. For example, the call handlers might authorize users, validate parameter values, cache the return value, and, if the logic so dictates, shortcut execution so that the target method does not actually execute.

Unity enables you to specify and customize any interception behavior and also enables you to use interception with or without a container. The earlier approach to policy injection is still supported, but you can also provide policy injection by using interception behaviors.

For information on using behaviors see **Behaviors for Interception**.

For information on using interception without a container see the "Stand-Alone Unity Interception" section in the <u>Using Interception in Applications</u> topic.

This topic contains the following sections that describe using policy injection

and containers with interception:

- Process Flow for Interception Using Policy Injection
- Using the Built-In Policy Injection Behavior
- Interception Policies
- Matching Rules
- Call Handlers and Policy Injection
- More Information

■ Process Flow for Interception Using Policy Injection

■ Using the Built-In Policy Injection Behavior

■ Interception Policies

■ Matching Rules

■ Call Handlers and Policy Injection

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Policy Injection Matching Rules**



Unity includes matching rule implementations that provide a wide range of capabilities for selecting the objects and their members to which Unity will add a handler pipeline. Interception policies use the matching rules to define which methods will be intercepted.

A matching rule is essentially a predicate that Unity checks each time it intercepts object creation. If all of the specified matching rules evaluate to **True** for any particular invocation, the application block will create and add the handler pipeline for that policy. If any one of the matching rules does not evaluate to true, Unity generates an instance of the original object or a derived object, and does not create a proxy or a handler pipeline.

The following table lists the matching rules provided with Unity, and summarizes their use and parameters.

Matching rule	Description
Assembly Matching Rule	Selects classes in a specified assembly.
Custom Attribute Matching Rule	Selects classes or members that have an arbitrary attribute applied.
Member Name Matching Rule	Selects class members based on the member name.
Method Signature Matching Rule	Selects class methods that have a specific signature.
Namespace Matching Rule	Selects classes based on the namespace name.
Parameter Type Matching Rule	Selects class members based on the type name of a parameter for a member of the target object.
Property Matching Rule	Selects class properties by name and accessor type.
Return Type	Selects class members that return an object of the

Matching Rule	specified type.
Tag Attribute Matching Rule	Selects class members that carry a Tag attribute with the specified name.
Type Matching Rule	Selects classes that are of a specified type.

The following sections describe the built-in matching rules in detail:

- The Assembly Matching Rule
- The Custom Attribute Matching Rule
- The Member Name Matching Rule
- The Method Signature Matching Rule
- The Namespace Matching Rule
- The Parameter Type Matching Rule
- The Property Matching Rule
- The Return Type Matching Rule
- The Tag Attribute Matching Rule
- The Type Matching Rule

You can also create and use custom matching rules. For more information, see <u>Creating Policy Injection Matching Rules</u>. For information about using interception, see <u>Using Interception and Policy Injection</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

The Assembly Matching Rule



The assembly matching rule allows developers, operators, and administrators to select target classes based on the assembly name or by specifying a reference to an assembly.

■ Behavior of the Assembly Matching Rule

■ Creating an Assembly Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

The Custom Attribute Matching Rule



The custom attribute matching rule allows developers, operators, and administrators to select target classes based on a custom attribute type that is applied to class members.

■ Behavior of the Custom Attribute Matching Rule

■ Creating a Custom Attribute Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

The Member Name Matching Rule



The member name matching rule allows developers, operators, and administrators to select target classes based on the name of the class members (methods or properties), and allows you to use wildcard characters for the member name.

■ Behavior of the Member Name Matching Rule

■ Creating a Member Name Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Method Signature Matching Rule



The method signature matching rule allows developers, operators, and administrators to select target classes based on the name and signature (the list of parameter types) of its members. This rule allows the use of wildcard characters for the member names.

■ Behavior of the Method Signature Matching Rule

■ Creating a Method Signature Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Namespace Matching Rule



The namespace matching rule allows developers, operators, and administrators to select target classes based on their namespace, using wildcard characters for the child namespace names but not for the root namespace name.

■ Behavior of the Namespace Matching Rule

■ Creating a Namespace Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Parameter Type Matching Rule



The parameter type matching rule allows developers, operators, and administrators to select target classes based on the type name of a parameter for a member of the target object.

■ Behavior of the Parameter Type Matching Rule

■ Creating a Parameter Type Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Property Matching Rule



The property matching rule allows developers, operators, and administrators to select individual properties of the target classes based on their name, including using wildcard characters, and the combination of accessors they implement.

■ Behavior of the Property Matching Rule

■ Creating a Property Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Return Type Matching Rule



The return type matching rule allows developers, operators, and administrators to select target classes based on the type or the type name of the return value, using wildcard characters if required.

■ Behavior of the Return Type Matching Rule

■ Creating a Return Type Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

The Tag Attribute Matching Rule
```

The tag attribute matching rule allows developers, operators, and administrators to select target classes based on the name of an attribute of type **Tag** that is applied to a class, or to members (methods or properties) within a class. For example, the following code shows a class with two tagged members.

```
C#
                                              Copy Code
public class AnnotatedWithTags
  [Tag("MyTagName")]
  public void TaggedMethod(string parameter1)
    ... method implementation here ...
  [Tag("AnotherTagName")]
  public string TaggedProperty
     ... property implementation here ...
}
Visual Basic
                                              Copy Code
Public Class AnnotatedWithTags
  <Tag("MyTagName")>
  Public Sub TaggedMethod(parameter1 As String)
    ... method implementation here ...
  End Sub
  <Tag("AnotherTagName")>
  Public Property TaggedProperty As String
```

```
... property implementation here ...
End Property

End Class

The following code shows a tagged class.

C# □ Copy Code

[Tag("MyClassTagName")]
public class AnnotatedWithTagOnClass
{
... class implementation here ...
}

Visual Basic □ Copy Code

<Tag("MyClassTagName")> _
Public Class AnnotatedWithTagOnClass
... class implementation here ...
End Class
```

■ Behavior of the Tag Attribute Matching Rule

■ Creating a Tag Attribute Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Type Matching Rule



The type matching rule allows developers, operators, and administrators to specify the target class using the namespace and class name of the target type.

■ Behavior of the Type Matching Rule

■ Creating a Type Matching Rule at Run Time

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Attribute-Driven Policies**



A common scenario when using any policy injection framework is the requirement to specify policies for classes and their members using attributes directly applied to the appropriate classes and members. Unity interception supports this technique—it actively discovers classes and members with attributes that define call handlers and applies the appropriate policies.

Developers specify handlers for classes and their members (methods and properties) using call handler attributes. Each attribute automatically instantiates the appropriate call handler, and applies the values of the attribute parameters to the properties of the call handler. Using directly applied attributes has the following advantages:

- Developers can ensure that Unity adds handlers that are specifically required in all circumstances and which should never be removed from the handler pipeline.
- Developers can fix the settings or values of specific parameters on classes and class members—for example, by defining that specific parameter values must always be greater than zero or that logging will always occur for specific methods.
- Developers can prevent the application of a handler pipeline to specific methods and properties, or to whole classes, using the ApplyNoPoliciesAttribute attribute.

However, applying policies through attributes applied directly to members of the target classes means that developers, administrators, and operators can no longer control the behavior of interception without changing the source code and recompiling the solution. In addition, using the **ApplyNoPoliciesAttribute** attribute may cause unexpected behavior for developers, administrators, and operators, who may attempt to add policies to an application without being aware of the applied attributes.

■ Policy and Handler Precedence with Attributes

■ Example of an Attribute-Driven Policy

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Enterprise Library Call Handlers



The patterns & practices Enterprise Library includes a set of call handlers and the equivalent call handler attributes designed for use with the Unity **PolicyInjectionBehavior**. This section discusses some of the common scenarios for using the Policy Injection Application Block call handlers. It examines the following common scenarios where the block can simplify and accelerate application development:

- Logging Method Invocation and Property Access
- Handling Exceptions in a Structured Manner
- Validating Parameter Values
- Authorizing Method and Property Requests
- Measuring Target Method Performance

■ Logging Method Invocation and Property Acce	ess.
-----------------------------------------------	------

Handling	Except	tions in	a Str	uctured	Manner
----------------------------	--------	----------	-------	---------	--------

■ Validating Parameter Values

Authorizing Method and Pr	operty Requests

■ Measuring Target Method Performance

■ More Information

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Authorization Handler



The authorization handler provides the capability to check that the current user (the security principal for the current thread) has the requisite permission to access the selected object method or property. This handler uses the Security Application Block and takes advantage of the features that it exposes.

The authorization handler applies the security check **before** invocation of the selected method or setting of the selected property of the target object. If the current user does not have permission to access the method or property accessor, the authorization handler aborts execution during the preprocessing stage and does not invoke the target method or set the target property. It also generates an **UnauthorizedAccessException** and packages it into the message passed back to the previous handler in the pipeline.

✓ Note:

This call handler is implemented in the

Microsoft.Practices.EnterpriseLibrary.Security.PolicyInjection namespace of the Security Application Block in the

Microsoft.Practices.EnterpriseLibrary.Security.dll assembly.

Behavior o	f the A	Authorization	Handler
------------------------------	---------	---------------	---------

■ Creating Instances of the Authorization Handler

■ Using the Authorization Handler Attribute

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Exception Handling Handler



The exception handling handler provides the capability to manage and process exceptions in a standard way. This handler uses the Exception Handling Application Block, taking advantage of the wide range of options that it supports.

The exception handler applies **after** invocation of the selected method or access to the selected property of the target object. If the method or property accessor raises an exception, the exception handling handler will invoke a named exception handling policy defined within the Exception Handling Application Block. This policy may ignore the exception, return the original exception, or replace it with a new exception. The exception handling handler then packages the exception (if the Exception Handling Application Block returns one) into the message passed back to the previous handler in the chain.

Each instance of the exception handling handler maintains its own hierarchy of exception policies and any dependent objects. When using the logging handler with the Exception Handling Application Block, each exception handling handler instance will contain its own **LogWriter** instance and set of **TraceListeners**. If the Logging Application Block is configured to use a flat file trace listener or a rolling flat file trace listener, you may see multiple log files with GUIDs in their file names because multiple instances of the trace listeners are not able to write to the configured log file at the same time.

✓ Note:

This call handler is implemented in the Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.PolicyInjection namespace of the Exception Handling Application Block in the Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.dll assembly.

■ Behavior of the Exception Handling Handler

■ Creating Instances of the Exception Handling Handler

■ Using the Exception Handling Handler Attribute

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Logging Handler



The logging handler provides the capability to write log messages and trace messages as the client code invokes the selected method or accesses the selected property of the target object. This handler uses the Logging Application Block, taking advantage of the wide range of log types, formatting, and tracing features that it provides.

The logging handler applies both before and after the invocation of the selected method or accessing the selected property of the target object, depending on settings in the application configuration.

✓ Note:

This call handler is implemented in Microsoft.Practices.EnterpriseLibrary.Logging.PolicyInjection namespace of the Logging Application Block in the assembly Microsoft.Practices.EnterpriseLibrary.Logging.dll.

The logging handler will initialize the Logging Application Block using the same configuration source as used to create the logging handler. By default, this will be the default configuration source. It is possible to specify an alternative configuration source if you instantiate the logging handler yourself using code. If you do this, you should create the configuration source once and use the same instance each time you create a logging handler to prevent performance issues and memory leaks.

✓ Note:

The Enterprise Library 5.0 Configuration tool does not support **Environmental Overrides** for the logging handler **Categories**. This means you will not be able to use the configuration tools at design time to customize the run-time settings of your logging handler **Categories** configuration to suit a particular environment such as a test or instrumentation environment.

■ Behavior of the Logging Handler

■ Creating Instances of the Logging Handler

■ Using the Logging Handler Attribute

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Performance Counter Handler



The performance counter handler increments a specific counter each time it executes in response to invocation of the selected method or setting of the selected property. This handler uses the instrumentation features that are part of the Enterprise Library Core.

The performance counter handler applies both before and after invocation of the selected method or access to the selected property of the target object. The handler can increment different types of counters and increment more than one counter each time (such as a single instance and a total counter).

✓ Note:

This call handler is implemented in the Microsoft.Practices.EnterpriseLibrary.PolicyInjection.CallHandlers namespace of the Microsoft.Practices.EnterpriseLibrary.PolicyInjection.dll assembly.

+	Be	havior	of	the	Performance	Counter	Handl	er

■ Creating Instances of the Performance Counter Handler

■ Using the Performance Counter Handler Attribute

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

The Validation Handler



The validation handler provides the capability to test whether the value provided for the selected property, or the values specified for the parameters of the selected method, are valid against specific rules. This handler uses the Validation Application Block, taking advantage of the wide range of capabilities that it offers.

The validation handler applies the validation before invoking the method or setting the property of the target object. If validation fails, the validation handler aborts execution of the preprocessing handler pipeline, does not invoke the method or set the property, and raises an **ArgumentValidationException**.

✓ Note:

This call handler is implemented in Microsoft.Practices.EnterpriseLibrary.Validation.PolicyInjection namespace of the Validation Application Block, in the assembly Microsoft.Practices.EnterpriseLibrary.Validation.dll.

Behavior of the	Validation	Handler
-----------------	-------------------	---------

■ Creating Instances of the Validation Handler

■ Using the Validation Handler Attribute

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Design of Unity



This topic describes the design goals, the architecture, and the design highlights of Unity. You do not have to understand the design to use Unity; however, this topic will help you to understand how it works and how it interacts with the underlying ObjectBuilder subsystem.

■ Design Goals

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 **Extending and Modifying Unity**



If required, you can extend and modify Unity to better suit your own requirements. You can extend Unity by doing the following:

- <u>Creating Lifetime Managers</u> that control how and when the container will dispose of instances of objects it resolves.
- <u>Creating and Using Container Extensions</u> that can change the behavior of the container, the instance generation mechanism, and the dependency injection and interception features.
- <u>Creating Policy Injection Matching Rules</u> that provides alternative techniques for selecting classes and class members to which Unity will attach a handler pipeline.
- <u>Creating Interception Policy Injection Call Handlers</u> that perform the task-specific processing you require for method invocations and property accessors.
- <u>Creating Interception Handler Attributes</u> that cause Unity to add built-in or custom call handlers to the handler pipeline. If you create a custom handler, you may also want to create a custom attribute that developers can use to apply your handler by adding the attribute directly to classes or class members within the source code of an application.
- <u>Creating Interception Behaviors</u> that describe what to do when an object is intercepted.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Creating Lifetime Managers



Unity supports several approaches for registering classes, interfaces, and instances of existing objects. You can use the **RegisterType** and **RegisterInstance** methods to register object instances at run time, or specify the mappings and registrations at design time in configuration. You can also specify a lifetime manager with all of these approaches that will control how Unity resolves instances of the specified types and how it holds references to these instances. The built-in lifetime managers allow you to specify objects as singletons, with weak references, or as per-thread instances. For more details, see <u>Understanding Lifetime Managers</u>.

You can create custom **LifetimeManager** classes if you require additional functionality not available in the default lifetime managers. Documentation to help you do this is available from the <u>Unity Community Web site</u> on CodePlex.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> Community Web site.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Creating and Using Container Extensions



You can create your own custom Unity container extensions, or use container extensions created by third parties with Unity. Unity uses default container extensions to implement its own functionality. For example, the interception mechanism provided by Unity is implemented as a container extension.

Documentation to help you understand ObjectBuilder, and the steps required to create custom container extensions, is available from the <u>Unity Community</u> <u>Web site</u> on CodePlex.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Creating Policy Injection Matching Rules
```

Unity defines an interface named **IMatchingRule**, which all classes that implement matching rules must implement. This interface declares a single method named **Matches** that takes a **MethodBase** instance and returns a **Boolean** value.

```
C#

public interface IMatchingRule
{
   bool Matches(MethodBase member);
}

Visual Basic

Public Interface IMatchingRule
   Function Matches(ByVal member As MethodBase) As BoolEnd Interface
```

Inside a concrete implementation of this interface, a custom matching rule class can access details of the current member (method or property) of the target object and determine whether Unity should add a handler pipeline to this member. The **Matches** method should return **True** if the current member matches the requirements of this matching rule; if the current member does not match the requirements of this matching rule, it should return **False**.

✓ Note:

Remember that there may be several matching rules defined for a policy, and every one must return **True** when Unity calls their **Matches** method in order for Unity to add the handler pipeline to the target class member.

■ Example: The TagAttributeMatchingRule

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Creating Interception Policy Injection Call Handlers



To create a call handler for policy injection, you must understand the way that Unity passes calls through the policy pipeline. This topic explains how the pipeline executes call handlers, and how it can block or abort execution when an error occurs, or on demand (such as when a validation handler detects a validation error or an authorization handler detects an unauthorized user). This topic contains the following sections:

- The ICallHandler Interface and Pipeline Execution
- Outline Implementation of a Call Handler
- Exceptions and Aborted Pipeline Execution

■ The ICallHandler	Interface	and l	Pipeline	Execution

■ Outline Imp	lementation of	a Call Handler
----------------------	----------------	----------------

■ Exceptions and Aborted Pipeline Execution

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

```
Microsoft Enterprise Library 5.0

Creating Interception Handler Attributes
```

Handler attributes allow developers to apply handlers to classes and class members directly, without configuring them in the application configuration file. Developers creating custom handlers may want to provide an attribute for their handlers. To build a custom handler attribute, you create a class that derives from the **HandlerAttribute** base class shown here.

```
C#
public abstract class HandlerAttribute : Attribute
  /// Derived classes implement this method. When call
  /// new call handler as specified in the attribute c
  /// The parameter "container" specifies the IUnityCo
  /// to use when creating handlers, if necessary.
  /// returns a new call handler object.
  public abstract ICallHandler CreateHandler(IUnityCor
  private int executionorder;
  /// <summarv>
  /// Gets or sets the order in which the handler will
  /// </summary>
  public int Order
    get { return this.executionorder; }
    set { this.order = value; }
  }
}
```

Visual Basic

```
Public MustInherit Class HandlerAttribute : Inherits /
''' Derived classes implement this method. When call
''' new call handler as specified in the attribute of
```

```
''' The parameter container specifies the IUnityCont
  ''' to use when creating handlers, if necessary.
  ''' Returns a new call handler object.
 Public MustOverride Function CreateHandler(container
 Private executionorder As Integer
  ''' <summary>
  ''' Gets or sets the order in which the handler will
  ''' </summary>
 Public Property Order As Integer
      Return Me. executionorder
   End Get
    Set
      Me.order = value
    End Set
 End Property
End Class
```

In your custom attribute class, you must implement one or more constructors that accept values from the attribute, and/or implement named properties that the developer can use to set the properties of the class. Then you simply override the **CreateHandler** abstract method declared within the base class to create and return the required handler class as an **ICallHandler** instance.

Example Call Handler Attribute

As an example, you could create a call handler attribute for a call handler similar to that described in the topic <u>Creating Interception Policy Injection Call Handlers</u> that prevents invocation of business processes on weekend days. In this case, assume that the handler has a property named **SaturdayOK** that allows you to set it to allow calls to occur on a Saturday. The call handler has two constructors: one that takes a parameter that sets the value of the **SaturdayOK** property to the specified value (true or false), and one that takes no parameters and sets the default value (false) for the **SaturdayOK** property. The following code shows an implementation of the

WeekdayOnlyCallHandlerAttribute.

```
C#

[AttributeUsage(AttributeTargets.Class | AttributeTargets | AttributeTargets | AttributeTargets | Attribute | Attribu
```

Visual Basic

```
<AttributeUsage(AttributeTargets.Class Or AttributeTar
Public Class WeekdayOnlyCallHandlerAttribute : Inherit

Private allowSaturday As Boolean

Public Sub New()
    allowSaturday = False
End Sub

Public Sub New(SaturdayOK As Boolean)
    allowSaturday = SaturdayOK
End Sub

Public Overrides Function CreateHandler(ignored As Return New WeekdayOnlyCallHandler(allowSaturda)</pre>
```

End Class

End Function

Notice the **AttributeUsage** attribute that specifies where developers can apply the new custom attribute (on a class, a property, or a method), and—in this case —the provision of two constructors. The first (default) constructor uses the default value (false), while the second accepts a value for the **SaturdayOK** property. The **CreateHandler** method override instantiates the **WeekdayOnlyCallHandler** class with the appropriate values and returns this as an **ICallHandler** reference.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library</u> <u>Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Creating Interception Behaviors



Unity uses the **Interceptor** class to specify how interception happens, and the **InterceptionBehavior** class to describe what to do when an object is intercepted. Unity interception utilizes a behavior pipeline to for the behaviors. The <u>Interception Behavior Pipeline</u> maintains a list of interception behaviors and manages them, calling them in the proper order with the correct inputs.

For information on the details about interception behaviors see <u>Interception with Unity</u> and <u>Behaviors for Interception</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Deployment and Operations



When you use Unity in your applications, you must deploy the required assemblies with your application or install the assemblies on the target computer in the global assembly cache (GAC). You must include the assembly named **Microsoft.Practices.Unity.dll**. If you are using interception, you will also require the assembly named **Microsoft.Practices.Unity.Interception.dll**.

You can deploy an application that uses Unity in one of two configurations:

- As private assemblies in the application folder hierarchy
- As shared assemblies in any file system location or in the global assembly cache

For advice on using Unity with applications that run in partial trust modes, see <u>Using Unity in Partial Trust Environments</u>.

For advice on updating existing versions of Unity assemblies, see <u>Updating the Unity Assemblies</u>.

When you compile the installed version of Unity source code, the assemblies produced will not be strong named. As a result, they cannot be installed in the global assembly cache, nor will they have the other benefits associated with strong-named assemblies. To learn how to strong name Unity assemblies, see Strong Naming the Unity Assemblies.

✓ Note:

Unity throws (and handles internally) **LockSynchronization** exceptions. **LockSynchronization** exceptions may be observed in the debugger output but they are handled internally and no action is required.

■ Using XCopy

■ Using the Global Assembly Cache

Versioning

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Using Unity in Partial Trust Environments



Unity uses dynamically generated methods to perform injection, and the .NET Framework security model imposes some security limitations that you should be aware of if you want to use Unity in applications that will run in less than full trust environments. The limitation when using Unity in a partial trust environment is that you cannot register and use mappings using the **RegisterType** methods where the target class is **internal** (C#), **Friend** (Visual Basic .NET), **private** (C#), or **Private** (Visual Basic .NET).

For more information about security issues when using dynamically generated Microsoft intermediate language (MSIL) code, see <u>Security Issues in Reflection Emit for .NET 3.5</u> and <u>Security Issues in Reflection Emit for .NET 2.0</u>.

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Copyright $\ensuremath{^{\odot}}$ 2010 by Microsoft Corporation. All rights reserved.

Microsoft Enterprise Library 5.0 **Updating the Unity Assemblies**



If an upgraded version of Unity becomes available, you can install the new version and have all applications use the updated assembly. However, if the new version introduces compatibility problems for certain applications, you can install the new version in the global assembly cache and configure some applications to use the updated version, while others continue to use the earlier version.

■ Updating Private Assemblies

• Updating Shared Assemblies

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Strong Naming the Unity Assemblies



If you build Unity from the source code, you may decide to apply strong naming to the assemblies. A strong name consists of the assembly's identity—the simple text name, version number, and culture information (if provided)—plus a public key and a digital signature. The strong name is generated from an assembly file (the file that contains the assembly manifest, which in turn contains the names and hashes of all the files that make up the assembly), using the corresponding private key. Signing an assembly with a strong name ensures that its name is globally unique. Assemblies with the same strong name are expected to be identical.

For example, if you intend to share Unity assemblies among several applications, you can install them into the global assembly cache. Each assembly in the global assembly cache must have a globally unique name. You can use a strong name to ensure this. Even if you only use the assemblies within a single application, you can strong name the assemblies to ensure that your application uses the correct version of the assemblies.

Strong names satisfy the following requirements:

- Strong names guarantee name uniqueness by relying on unique key pairs. No one can generate the same assembly name that you can because an assembly generated with one private key has a different name than an assembly generated with another private key.
- Strong names protect the version lineage of an assembly. A strong name
 can ensure that no one can produce a subsequent version of your
 assembly. Users can be sure that a version of the assembly they are
 loading comes from the same publisher that created the version
 originally provided with the application.
- Strong names provide a strong integrity check. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built. However, note that strong names themselves do not imply a level of trust such as the level provided by, for example, a digital signature and supporting certificate.

For information about deploying assemblies into the global assembly cache, see Working with Assemblies and the Global Assembly Cache.

■ Using Visual Studio to Strong Name the Unity Assemblies

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0

Unity QuickStarts



The instructions in this QuickStart topic are directed at the Silverlight solution. Though there are many similarities, some instructions in this topic, such as references to Program.cs or Program.vb, do not apply to the Silverlight project. The following QuickStart applications demonstrate some of the key features of Unity:

- Walkthrough: The Unity StopLight QuickStart. This QuickStart demonstrates dependency injection techniques. This is the only QuickStart project in the Silverlight solution.
- Walkthrough: The Unity Event Broker Extension QuickStart. This QuickStart provides an example extension for the Unity container. The Silverlight solution does no include the Event Broker QuickStart.

✓ Note:

Unity QuickStarts are only available if you install the standalone Unity MSI. The Unity MSI is available at <u>patterns & practices - Unity</u> on CodePlex.

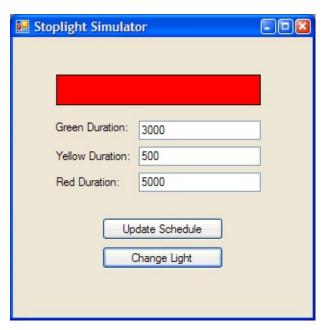
■ Building the QuickStarts

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Microsoft Enterprise Library 5.0 Walkthrough: The Unity StopLight QuickStart •

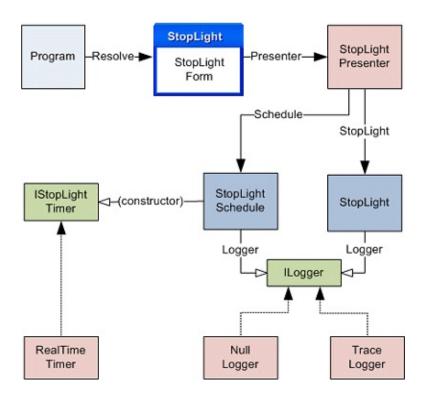
The StopLight QuickStart demonstrates the ways that you can use Unity and the Unity container in your applications. The user interface is a simple Windows Forms application that displays the three colors of a stop light or traffic light—it shows red, yellow, and green, in turn, for specified periods. You can configure the display periods for each color. The following illustration shows the user interface.



The StopLight QuickStart demonstrates the following features of Unity:

- Registering mappings for types with the container
- Implementing the Model View Presenter pattern by injecting a presenter into the user interface
- <u>Injecting a business component into objects using property (setter) injection</u>
- Implementing a configurable pluggable architecture

The following diagram shows the classes and architecture of the StopLight QuickStart application.



■ Registering Mappings for Types with the Container

■ Implementing	the Model-	View-Presenter	Pattern

■ Injecting a Business Component using Property (Setter) Injection

■ Implementing a Configurable Pluggable Architecture

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.

Walkthrough: The Unity Event Broker Extension QuickStart



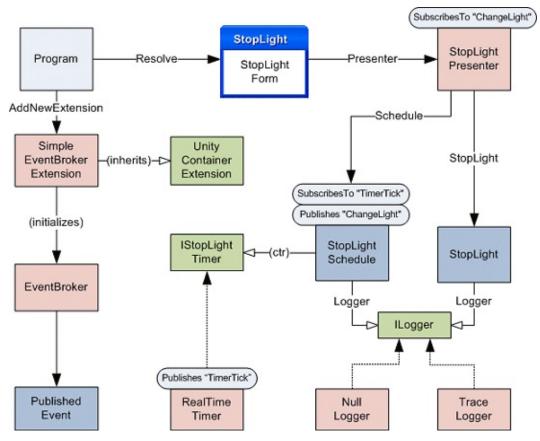
The Event Broker Extension QuickStart demonstrates how you can extend the Unity container by adding a custom extension. The QuickStart implements an event broker for the container as a container extension and demonstrates the new extension using the StopLight application discussed in Walkthrough: The Unity StopLight QuickStart.

The Event Broker Extension QuickStart contains five projects:

- **SimpleEventBroker**. This project implements a simple publish and subscribe mechanism that supports multiple event publishers and multiple subscribers.
- **EventBrokerExtension**. This project implements the custom container extension that allows applications to publish and subscribe to events using attributes or explicitly using code.
- **StopLight**. This project is basically the same as that described in the Unity StopLight QuickStart, but it uses the custom container extension to manage the publishing of, and subscription to, two events within the application.
- **Tests.EventBrokerExtension**. A test fixture for the **EventBrokerExtension**.
- Tests.SimpleEventBroker. A test fixture for the SimpleEventBroker.

For information about how you can create and use custom container extensions, see <u>Creating and Using Container Extensions</u>.

The following diagram shows the classes and architecture of the Event Broker Extension QuickStart.



If you compare this diagram to the structure of the StopLight QuickStart shown in Walkthrough: The Unity StopLight QuickStart, you can see that the EventBroker Extension QuickStart has the following additional features:

- The **Program** class, which registers the type mappings in the container and calls the **Resolve** method to instantiate the main **StopLight** form, also adds the **SimpleEventBrokerExtension** to the container.
- The SimpleEventBrokerExtension, which inherits from the UnityContainerExtension base class, creates an instance of the EventBroker class that implements the publish and subscribe pattern for distributed events.
- The EventBroker class creates an instance of the PublishedEvent class that provides the facilities for maintaining a list of event subscriptions and raising events to registered subscribers.
- The StopLightPresenter, StopLightSchedule, and RealTimeTimer classes include attributes that register event publications and subscriptions with the SimpleEventBrokerExtension class.

The Event Broker Extension QuickStart demonstrates the following features of Unity and the custom container extension mechanism:

- Creating the custom Unity container extension
- Adding an extension to the Unity container at run time
- Using the example Event Broker Extension

■ Creating a Custom Unity Container Extension

■ Adding an Extension to the Unity Container at Run Time

■ Using the Example Event Broker Extension

To provide feedback, get assistance, or download additional content, please visit the <u>Enterprise Library Community Web site</u>.

To report documentation errors or provide feedback on this documentation, please send email to pagdoc@microsoft.com.