From MSDN

[**Debug**](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.aspx)

Make your code more robust without impacting the performance and code size of your shipping product.

// Specify /d:DEBUG when compiling.

using System;

using System.Diagnostics;

class Test

{

static void Main()

{

Debug.Listeners.Add(new TextWriterTraceListener(Console.Out));

Debug.AutoFlush = true;

Debug.Indent();

Debug.WriteLine("Entering Main");

Console.WriteLine("Hello World.");

Debug.WriteLine("Exiting Main");

Debug.Unindent();

}

}

[**Tracing**](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.aspx)

Tracing helps you isolate problems and fix them without disturbing a running system.

// Specify /d:TRACE when compiling.

using System;

using System.Diagnostics;

class Test

{

static void Main()

{

Trace.Listeners.Add(new TextWriterTraceListener(Console.Out));

Trace.AutoFlush = true;

Trace.Indent();

Trace.WriteLine("Entering Main");

Console.WriteLine("Hello World.");

Trace.WriteLine("Exiting Main");

Trace.Unindent();

}

}

# Debug Class

**.NET Framework 4.5**

[Other Versions](javascript:;)

Description: http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [.NET Framework 4](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(d=printer,v=vs.100).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(d=printer,v=vs.85).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(d=printer,v=vs.80).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(d=printer,v=vs.71).aspx)
* [Silverlight](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(d=printer,v=vs.95).aspx)

Provides a set of methods and properties that help debug your code. This class cannot be inherited.

[Remarks](javascript:void(0)" \o "Click to collapse. Double-click to collapse all.)

If you use methods in the Debug class to print debugging information and check your logic with assertions, you can make your code more robust without impacting the performance and code size of your shipping product.

This class provides methods to display an [Assert](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.assert.aspx) dialog box, and to emit an assertion that will always fail. This class provides write methods in the following variations: [Write](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.write.aspx), [WriteLine](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.writeline.aspx), [WriteIf](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.writeif.aspx) and [WriteLineIf](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.writelineif.aspx).

The [BooleanSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.booleanswitch.aspx) and [TraceSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.traceswitch.aspx) classes provide means to dynamically control the tracing output. You can modify the values of these switches without recompiling your application. For information on using the configuration file to set a switch, see the [Switch](http://msdn.microsoft.com/en-us/library/system.diagnostics.switch.aspx) class and the [Trace Switches](http://msdn.microsoft.com/en-us/library/3at424ac.aspx) topic.

You can customize the tracing output's target by adding [TraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.tracelistener.aspx) instances to or removing instances from the [Listeners](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.listeners.aspx) collection. The [Listeners](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.listeners.aspx) collection is shared by both the Debug and the [Trace](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.aspx) classes; adding a trace listener to either class adds the listener to both. By default, the [DefaultTraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.defaulttracelistener.aspx) class emits trace output.

|  |
| --- |
| **Description: NoteNote** |
| Adding a trace listener to the [Listeners](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.listeners.aspx) collection can cause an exception to be thrown while tracing, if a resource used by the trace listener is not available. The conditions and the exception thrown depend on the trace listener and cannot be enumerated in this topic. It may be useful to place calls to the Debug methods in try/catch blocks to detect and handle any exceptions from trace listeners. |

You can modify the level of indentation using the [Indent](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.indent.aspx) method or the [IndentLevel](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.indentlevel.aspx) property. To modify the indent spacing, use the [IndentSize](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.indentsize.aspx) property. You can specify whether to automatically flush the output buffer after each write by setting the [AutoFlush](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.autoflush.aspx) property to true.

To set the [AutoFlush](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.autoflush.aspx) and [IndentSize](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.indentsize.aspx) for Debug, you can edit the configuration file corresponding to the name of your application. The configuration file should be formatted as shown in the following example.

<configuration>

<system.diagnostics>

<trace autoflush="true" indentsize="7" />

</system.diagnostics>

</configuration>

The [ConditionalAttribute](http://msdn.microsoft.com/en-us/library/system.diagnostics.conditionalattribute.aspx) attribute is applied to the methods of Debug. Compilers that support [ConditionalAttribute](http://msdn.microsoft.com/en-us/library/system.diagnostics.conditionalattribute.aspx) ignore calls to these methods unless "DEBUG" is defined as a conditional compilation symbol. Refer to a compiler's documentation to determine whether [ConditionalAttribute](http://msdn.microsoft.com/en-us/library/system.diagnostics.conditionalattribute.aspx) is supported and the syntax for defining a conditional compilation symbol.

|  |
| --- |
| **Description: NoteNote** |
| In Visual Studio C# and Visual Basic projects, by default, the "DEBUG" conditional compilation symbol is defined for debug builds, and the "TRACE" symbol is defined for both debug and release builds. For information about how to disable this behavior, see the Visual Studio documentation. For information about conditional debugging in Visual C++, see [Debug Class (C++/CLI)](http://msdn.microsoft.com/en-us/library/ms235216.aspx). |

To define the "DEBUG" conditional compilation symbol in C#, add the /d:DEBUG option to the compiler command line when you compile your code using a command line, or add #define DEBUG to the top of your file. In Visual Basic, add the /d:DEBUG=True option to the compiler command line or add #Const DEBUG=True to the file.

[Examples](javascript:void(0))

The following example uses Debug to indicate the beginning and end of a program's execution. The example also uses [Indent](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.indent.aspx) and [Unindent](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug.unindent.aspx) to distinguish the tracing output.

C#

[C++](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(d=printer).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-3)

[VB](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

// Specify /d:DEBUG when compiling.

using System;

using System.Data;

using System.Diagnostics;

class Test

{

static void Main()

{

Debug.Listeners.Add(new TextWriterTraceListener(Console.Out));

Debug.AutoFlush = true;

Debug.Indent();

Debug.WriteLine("Entering Main");

Console.WriteLine("Hello World.");

Debug.WriteLine("Exiting Main");

Debug.Unindent();

}

}

# Trace Class

**.NET Framework 4.5**

[Other Versions](javascript:;)



* [.NET Framework 4](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace(d=printer,v=vs.100).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace(d=printer,v=vs.85).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace(d=printer,v=vs.80).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace(d=printer,v=vs.71).aspx)

Provides a set of methods and properties that help you trace the execution of your code. This class cannot be inherited.

[Remarks](javascript:void(0))

You can use the properties and methods in the Trace class to instrument release builds. Instrumentation allows you to monitor the health of your application running in real-life settings. Tracing helps you isolate problems and fix them without disturbing a running system.

This class provides methods to display an [Assert](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.assert(v=vs.110).aspx) dialog box, and to emit an assertion that will always [Fail](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.fail(v=vs.110).aspx). This class provides write methods in the following variations: [Write](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.write(v=vs.110).aspx), [WriteLine](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.writeline(v=vs.110).aspx), [WriteIf](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.writeif(v=vs.110).aspx), and [WriteLineIf](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.writelineif(v=vs.110).aspx).

The [BooleanSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.booleanswitch(v=vs.110).aspx) and [TraceSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.traceswitch(v=vs.110).aspx) classes provide means to dynamically control the tracing output. You can modify the values of these switches without recompiling your application. For information on using the configuration file to set a switch, see the [Switch](http://msdn.microsoft.com/en-us/library/system.diagnostics.switch(v=vs.110).aspx) class and the [How to: Configure Trace Switches](http://msdn.microsoft.com/en-us/library/t06xyy08(v=vs.110).aspx) topic.

You can customize the tracing output's target by adding [TraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.tracelistener(v=vs.110).aspx) instances to or removing instances from the [Listeners](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.listeners(v=vs.110).aspx) collection. The [Listeners](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.listeners(v=vs.110).aspx) collection is shared by both the [Debug](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(v=vs.110).aspx) and the Trace classes; adding a trace listener to either class adds the listener to both. By default, trace output is emitted using the [DefaultTraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.defaulttracelistener(v=vs.110).aspx) class.

|  |
| --- |
| **NoteNote** |
| Adding a trace listener to the [Listeners](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.listeners(v=vs.110).aspx) collection can cause an exception to be thrown while tracing, if a resource used by the trace listener is not available. The conditions and the exception thrown depend on the trace listener and cannot be enumerated in this topic. It may be useful to place calls to the Trace methods in try/catch blocks to detect and handle any exceptions from trace listeners. |
| **NoteNote** |
| If you add trace listeners to partially trusted code, you will get a [SecurityException](http://msdn.microsoft.com/en-us/library/system.security.securityexception(v=vs.110).aspx) exception, because adding trace listeners requires [UnmanagedCode](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag(v=vs.110).aspx) permission. To trace partially trusted code that is running in a sandbox in Visual Studio, do not add trace listeners. Instead, view the Trace and [Debug](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(v=vs.110).aspx) messages in the Output window. |

The Trace class provides properties to get or set the level of [Indent](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.indent(v=vs.110).aspx), the [IndentSize](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.indentsize(v=vs.110).aspx), and whether to [AutoFlush](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.autoflush(v=vs.110).aspx) after each write.

To set the [AutoFlush](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.autoflush(v=vs.110).aspx) and [IndentSize](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.indentsize(v=vs.110).aspx) for Trace, you can edit the configuration file that corresponds to the name of your application. The configuration file should be formatted like the following example:

<configuration>

<system.diagnostics>

<trace autoflush="false" indentsize="3" />

</system.diagnostics>

</configuration>

The [ConditionalAttribute](http://msdn.microsoft.com/en-us/library/system.diagnostics.conditionalattribute(v=vs.110).aspx) attribute is applied to the methods of Trace. Compilers that support [ConditionalAttribute](http://msdn.microsoft.com/en-us/library/system.diagnostics.conditionalattribute(v=vs.110).aspx) ignore calls to these methods unless "TRACE" is defined as a conditional compilation symbol. Refer to a compiler's documentation to determine whether [ConditionalAttribute](http://msdn.microsoft.com/en-us/library/system.diagnostics.conditionalattribute(v=vs.110).aspx) is supported and the syntax for defining a conditional compilation symbol.

|  |
| --- |
| **NoteNote** |
| In Visual Studio projects, by default, the "DEBUG" conditional compilation symbol is defined for debug builds, and the "TRACE" symbol is defined for both debug and release builds. For information about how to disable this behavior, see the Visual Studio documentation. |

To define the "TRACE" conditional compilation symbol in C#, add the /d:TRACE option to the compiler command line when you compile your code using a command line, or add #define TRACE to the top of your file. In Visual Basic, add the /d:TRACE=True option to the compiler command line or add #Const TRACE=True to the file.

[ConditionalAttribute](http://msdn.microsoft.com/en-us/library/system.diagnostics.conditionalattribute(v=vs.110).aspx) is not supported by the C++ compiler. To provide equivalent functionality, you must enclose calls to the methods of Trace in an #if defined(TRACE) ... #endif block, and add the /DTRACE option to the compiler command line or add #define TRACE to the file.

[Examples](javascript:void(0))

The following example uses Trace to indicate the beginning and the end of a program's execution. The example also uses the [Trace.Indent](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.indent(v=vs.110).aspx) and [Trace.Unindent](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace.unindent(v=vs.110).aspx) methods to distinguish the tracing output. For a more complete example of the use of Trace, see [How to: Add Trace Statements to Application Code](http://msdn.microsoft.com/en-us/library/zd83saa2(v=vs.110).aspx).

C#

[C++](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-3)

[VB](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

// Specify /d:TRACE when compiling.

using System;

using System.Diagnostics;

class Test

{

static void Main()

{

Trace.Listeners.Add(new TextWriterTraceListener(Console.Out));

Trace.AutoFlush = true;

Trace.Indent();

Trace.WriteLine("Entering Main");

Console.WriteLine("Hello World.");

Trace.WriteLine("Exiting Main");

Trace.Unindent();

}

}

# How to: Compile Conditionally with Trace and Debug

**.NET Framework 4.5**

[Other Versions](javascript:;)



* [.NET Framework 4](http://msdn.microsoft.com/en-us/library/64yxa344(d=printer,v=vs.100).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/64yxa344(d=printer,v=vs.85).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/64yxa344(d=printer,v=vs.90).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/64yxa344(d=printer,v=vs.80).aspx)

While you are debugging an application during development, both your tracing and debugging output go to the Output window in Visual Studio. However, to include tracing features in a deployed application, you must compile your instrumented applications with the **TRACE** compiler directive enabled. This allows tracing code to be compiled into the release version of your application. If you do not enable the **TRACE** directive, all tracing code is ignored during compilation and is not included in the executable code that you will deploy.

Both the tracing and debugging methods have associated conditional attributes. For example, if the conditional attribute for tracing is **true**, all trace statements are included within an assembly (a compiled .exe file or .dll); if the **Trace** conditional attribute is **false**, the trace statements are not included.

You can have either the **Trace** or **Debug** conditional attribute turned on for a build, or both, or neither. Thus, there are four types of build: **Debug**, **Trace**, both, or neither. Some release builds for production deployment might contain neither; most debugging builds contain both.

You can specify the compiler settings for your application in several ways:

* The property pages
* The command line
* **#CONST** (for Visual Basic) and **#define** (for C#)

### To change compile settings from the property pages dialog box

1. Right-click the project node in **Solution Explorer**.
2. Choose **Properties** from the shortcut menu.
   * In Visual Basic, click the Compile tab in the left pane of the property page, then click the Advanced Compile Options button to display the Advanced Compiler Settings dialog box. Select the check boxes for the compiler settings you want to enable. Clear the check boxes for settings you want to disable.
   * In C#, click the Build tab in the left pane of the property page, then select the check boxes for the compiler settings you want to enable. Clear the check boxes for settings you want to disable.

### To compile instrumented code using the command line

* Set a conditional compiler switch on the command line. The compiler will include trace or debug code in the executable.

For example, the following compiler instruction entered on the command line would include your tracing code in a compiled executable:

For Visual Basic: **vbc /r:System.dll /d:TRACE=TRUE /d:DEBUG=FALSE MyApplication.vb**

For C#: **csc /r:System.dll /d:TRACE /d:DEBUG=FALSE MyApplication.cs**

|  |
| --- |
| **TipTip** |
| To compile more than one application file, leave a blank space between the file names, for example, **MyApplication1.vb MyApplication2.vb MyApplication3.vb** or **MyApplication1.cs MyApplication2.cs MyApplication3.cs**. |

The meaning of the conditional-compilation directives used in the above examples is as follows:

|  |  |
| --- | --- |
| **Directive** | **Meaning** |
| vbc | Visual Basic compiler |
| csc | C# compiler |
| /r: | References an external assembly (EXE or DLL) |
| /d: | Defines a conditional compilation symbol |

|  |
| --- |
| **NoteNote** |
| You must spell TRACE or DEBUG with uppercase letters. For more information about the conditional compilation commands, enter vbc /? (for Visual Basic) or csc /? (for C#) at the command prompt. For more information, see [Building from the Command Line](http://msdn.microsoft.com/en-us/library/1700bbwd(v=vs.110).aspx) (C#) or [Invoking the Command-Line Compiler](http://msdn.microsoft.com/en-us/library/380cz5k4(v=vs.110).aspx) (Visual Basic). |

### To perform conditional compilation using #CONST or #define

* Type the appropriate statement for your programming language at the top of the source code file.

|  |  |  |
| --- | --- | --- |
| **Language** | **Statement** | **Result** |
| **Visual Basic** | **#CONST TRACE = true** | Enables tracing |
|  | **#CONST TRACE = false** | Disables tracing |
|  | **#CONST DEBUG = true** | Enables debugging |
|  | **#CONST DEBUG = false** | Disables debugging |
| **C#** | **#define TRACE** | Enables tracing |
|  | **#undef TRACE** | Disables tracing |
|  | **#define DEBUG** | Enables debugging |
|  | **#undef DEBUG** | Disables debugging |

### To disable tracing or debugging

1. Delete the compiler directive from your source code.

- or -

1. Comment out the compiler directive.

|  |
| --- |
| **NoteNote** |
| When you are ready to compile, you can either choose **Build** from the **Build** menu, or use the command line method but without typing the **d:** to define conditional compilation symbols. |

# Trace Listeners

**.NET Framework 4.5**

[Other Versions](javascript:;)



* [.NET Framework 4](http://msdn.microsoft.com/en-us/library/4y5y10s7(d=printer,v=vs.100).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/4y5y10s7(d=printer,v=vs.85).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/4y5y10s7(d=printer,v=vs.90).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/4y5y10s7(d=printer,v=vs.80).aspx)

When using **Trace**, **Debug** and [TraceSource](http://msdn.microsoft.com/en-us/library/system.diagnostics.tracesource(v=vs.110).aspx), you must have a mechanism for collecting and recording the messages that are sent. Trace messages are received by listeners. The purpose of a listener is to collect, store, and route tracing messages. Listeners direct the tracing output to an appropriate target, such as a log, window, or text file.

Listeners are available to the **Debug**, **Trace**, and [TraceSource](http://msdn.microsoft.com/en-us/library/system.diagnostics.tracesource(v=vs.110).aspx) classes, each of which can send its output to a variety of listener objects. The following are the commonly used predefined listeners:

* A [TextWriterTraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.textwritertracelistener(v=vs.110).aspx) redirects output to an instance of the [TextWriter](http://msdn.microsoft.com/en-us/library/system.io.textwriter(v=vs.110).aspx) class or to anything that is a [Stream](http://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.110).aspx) class. It can also write to the console or to a file, because these are [Stream](http://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.110).aspx) classes.
* An [EventLogTraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.eventlogtracelistener(v=vs.110).aspx) redirects output to an event log.
* A [DefaultTraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.defaulttracelistener(v=vs.110).aspx) emits **Write** and **WriteLine** messages to the **OutputDebugString** and to the **Debugger.Log** method. In Visual Studio, this causes the debugging messages to appear in the Output window. **Fail** and failed **Assert** messages also emit to the **OutputDebugString** Windows API and the **Debugger.Log** method, and also cause a message box to be displayed. This behavior is the default behavior for **Debug** and **Trace** messages, because **DefaultTraceListener** is automatically included in every Listeners collection and is the only listener automatically included.
* A [ConsoleTraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.consoletracelistener(v=vs.110).aspx) directs tracing or debugging output to either the standard output or the standard error stream.
* A [DelimitedListTraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.delimitedlisttracelistener(v=vs.110).aspx) directs tracing or debugging output to a text writer, such as a stream writer, or to a stream, such as a file stream. The trace output is in a delimited text format that uses the delimiter specified by the [Delimiter](http://msdn.microsoft.com/en-us/library/system.diagnostics.delimitedlisttracelistener.delimiter(v=vs.110).aspx) property.
* An [XmlWriterTraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.xmlwritertracelistener(v=vs.110).aspx) directs tracing or debugging output as XML-encoded data to a [TextWriter](http://msdn.microsoft.com/en-us/library/system.io.textwriter(v=vs.110).aspx) or to a [Stream](http://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.110).aspx), such as a [FileStream](http://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx).

If you want any listener besides the [DefaultTraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.defaulttracelistener(v=vs.110).aspx) to receive **Debug**, **Trace** and [TraceSource](http://msdn.microsoft.com/en-us/library/system.diagnostics.tracesource(v=vs.110).aspx) output, you must add it to the Listeners collection. For more information, see [How to: Create and Initialize Trace Listeners](http://msdn.microsoft.com/en-us/library/sk36c28t(v=vs.110).aspx) and [How to: Use TraceSource and Filters with Trace Listeners](http://msdn.microsoft.com/en-us/library/ms228993(v=vs.110).aspx). Any listener in the **Listeners** collection gets the same messages from the trace output methods. For example, suppose you set up two listeners: a **TextWriterTraceListener** and an **EventLogTraceListener**. Each listener receives the same message. The **TextWriterTraceListener** would direct its output to a stream, and the **EventLogTraceListener** would direct its output to an event log.

The following example shows how to send output to the **Listeners** collection.

C#

[VB](http://msdn.microsoft.com/en-us/library/4y5y10s7(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

// Use this example when debugging.

System.Diagnostics.Debug.WriteLine("Error in Widget 42");

// Use this example when tracing.

System.Diagnostics.Trace.WriteLine("Error in Widget 42");

Debug and trace share the same **Listeners** collection, so if you add a listener object to a **Debug.Listeners** collection in your application, it gets added to the **Trace.Listeners** collection as well.

The following example shows how to use a listener to send tracing information to a console:

C#

[VB](http://msdn.microsoft.com/en-us/library/4y5y10s7(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

System.Diagnostics.Trace.Listeners.Clear();

System.Diagnostics.Trace.Listeners.Add(

new System.Diagnostics.TextWriterTraceListener(Console.Out));

[Developer-Defined Listeners](javascript:void(0))

You can define your own listeners by inheriting from the **TraceListener** base class and overriding its methods with your customized methods. For more information on creating developer-defined listeners, see [TraceListener](http://msdn.microsoft.com/en-us/library/system.diagnostics.tracelistener(v=vs.110).aspx) in the .NET Framework reference.

# Trace Switches

**.NET Framework 4.5**

[Other Versions](javascript:;)



* [.NET Framework 4](http://msdn.microsoft.com/en-us/library/3at424ac(d=printer,v=vs.100).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/3at424ac(d=printer,v=vs.85).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/3at424ac(d=printer,v=vs.90).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/3at424ac(d=printer,v=vs.80).aspx)

Trace switches enable you to enable, disable, and filter tracing output. They are objects that exist in your code and can be configured externally through the .config file. There are three types of trace switches provided in the .NET Framework: the [BooleanSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.booleanswitch(v=vs.110).aspx) class, the [TraceSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.traceswitch(v=vs.110).aspx) class, and the [SourceSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.sourceswitch(v=vs.110).aspx) class. The [BooleanSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.booleanswitch(v=vs.110).aspx) class acts as a toggle switch, either enabling or disabling a variety of trace statements. The [TraceSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.traceswitch(v=vs.110).aspx) and [SourceSwitch](http://msdn.microsoft.com/en-us/library/system.diagnostics.sourceswitch(v=vs.110).aspx) classes allow you to enable a trace switch for a particular tracing level so that the [Trace](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace(v=vs.110).aspx) or [TraceSource](http://msdn.microsoft.com/en-us/library/system.diagnostics.tracesource(v=vs.110).aspx) messages specified for that level and all levels below it appear. If you disable the switch, the trace messages will not appear. All these classes derive from the abstract (**MustInherit**) class **Switch**, as should any user-developed switches.

Trace switches can be useful for filtering information. For example, you might want to see every tracing message in a data access module, but only error messages in the rest of the application. In that case, you would use one trace switch for the data access module and one switch for the rest of the application. By using the .config file to configure the switches to the appropriate settings, you could control what types of trace message you received. For more information, see [How to: Create and Initialize Trace Switches](http://msdn.microsoft.com/en-us/library/t20ke01d(v=vs.110).aspx) and [How to: Configure Trace Switches](http://msdn.microsoft.com/en-us/library/t06xyy08(v=vs.110).aspx).

Typically, a deployed application is executed with its switches disabled, so that users need not observe a lot of irrelevant trace messages appearing on a screen or filling up a log file as the application runs. If a problem arises during application execution, you can stop the application, enable the switches, and restart the application. Then the tracing messages will be displayed.

To use a switch you must first create a switch object from a **BooleanSwitch** class, a **TraceSwitch** class, or a developer-defined switch class. For more information about creating developer-defined switches, see [Switch Class](http://msdn.microsoft.com/en-us/library/system.diagnostics.switch(v=vs.110).aspx) in the .NET Framework reference. Then you set a configuration value that specifies when the switch object is to be used. You then test the setting of the switch object in various **Trace** (or **Debug**) tracing methods.

[Trace Levels](javascript:void(0))

When you use **TraceSwitch**, there are additional considerations. A **TraceSwitch** object has four properties that return **Boolean** values indicating whether the switch is set to at least a particular level:

* [TraceSwitch.TraceError Property](http://msdn.microsoft.com/en-us/library/system.diagnostics.traceswitch.traceerror(v=vs.110).aspx)
* [TraceSwitch.TraceWarning Property](http://msdn.microsoft.com/en-us/library/system.diagnostics.traceswitch.tracewarning(v=vs.110).aspx)
* [TraceSwitch.TraceInfo Property](http://msdn.microsoft.com/en-us/library/system.diagnostics.traceswitch.traceinfo(v=vs.110).aspx)
* [TraceSwitch.TraceVerbose Property](http://msdn.microsoft.com/en-us/library/system.diagnostics.traceswitch.traceverbose(v=vs.110).aspx)

Levels allow you to limit the amount of tracing information you receive to only that information needed to solve a problem. You specify the level of detail you want in your tracing output by setting and configuring trace switches to the appropriate trace level. You can receive error messages, warning messages, informational messages, verbose tracing messages, or no message at all.

It is entirely up to you to decide what kind of message to associate with each level. Typically, the content of tracing messages depends on what you associate with each level, but you determine the differences between levels. You might want to provide detailed descriptions of a problem at level 3 (**Info**), for example, but provide only an error reference number at level 1 (**Error**). It is entirely up to you to decide what scheme works best in your application.

These properties correspond to the values 1 through 4 of the **TraceLevel** enumeration. The following table lists the levels of the **TraceLevel** enumeration and their values.

|  |  |  |
| --- | --- | --- |
| **Enumerated value** | **Integer value** | **Type of message displayed (or written to a specified output target)** |
| Off | 0 | None |
| Error | 1 | Only error messages |
| Warning | 2 | Warning messages and error messages |
| Info | 3 | Informational messages, warning messages, and error messages |
| Verbose | 4 | Verbose messages, informational messages, warning messages, and error messages |

The **TraceSwitch** properties indicate the maximum trace level for the switch. That is, tracing information is written for the level specified as well as for all lower levels. For example, if **TraceInfo** is **true**, then **TraceError** and **TraceWarning** are also **true** but **TraceVerbose** might well be **false**.

These properties are read-only. The **TraceSwitch** object automatically sets them when the **TraceLevel** property is set. For example:

C#

[VB](http://msdn.microsoft.com/en-us/library/3at424ac(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

System.Diagnostics.TraceSwitch myTraceSwitch =

new System.Diagnostics.TraceSwitch("SwitchOne", "The first switch");

myTraceSwitch.Level = System.Diagnostics.TraceLevel.Info;

// This message box displays true, because setting the level to

// TraceLevel.Info sets all lower levels to true as well.

MessageBox.Show(myTraceSwitch.TraceWarning.ToString());

// This message box displays false.

MessageBox.Show(myTraceSwitch.TraceVerbose.ToString());

[Developer-Defined Switches](javascript:void(0))

In addition to providing **BooleanSwitch** and **TraceSwitch**, you can define your own switches by inheriting from the **Switch** class and overriding the base class methods with customized methods. For more information about creating developer-defined switches, see [Switch Class](http://msdn.microsoft.com/en-us/library/system.diagnostics.switch(v=vs.110).aspx) in the .NET Framework reference.