**Debug and Trace**

Debug is used while you are developing the product. It depends heavily upon  
debug/symbol information that is additionally added to your Debug-mode builds.  
When you switch the build options to Release mode, this information, and  
any Debug code will not be compiled. The assumption is that you'll no longer  
need it because you're done debugging the app and are compiling a release  
version. Debug information and code can substantially decrease performance  
and efficiency of an executable.  
  
On the other hand, Trace is meant to instrument release builds. This helps  
you monitor running code. Think of this like turning on Tracing for ODBC  
as an example. Trace is enabled by default, so it's available in Debug and  
Release modes, but you can optionally turn Trace compiling off.

**Trace and Debug Classes**

Fortunately, you do not have to step through an application line by line to figure out what is happening. The **Systems.Diagnostics** namespace includes **Trace** and **Debug** classes. These two classes (which are essentially identical) include a number of static methods that can be used to cause your code to gather information about code-execution paths, code coverage, and even performance profiling. Both classes also provide an **Assert** method that checks for a condition and displays a message if the condition is **false**.

To use the **Trace** and **Debug** classes, perform the following three steps.

1. Define the **TRACE** or **DEBUG** symbol.
2. Create a new instance of the corresponding **Listener**.
3. Add calls to the **Trace** or **Debug** class to your code.

**Define —** To enable debugging or tracing, you must first define the corresponding symbol. This can be done at compile time using the **/d(efine)** conditional compilation switch for Visual C# (and the Managed Extensions for C++) or for Visual Basic .NET, as shown here:

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl01');)

csc /target:library /debug+ /d:TRACE math.cs

vbc /target:library /debug+ /d:TRACE=True math.vb

The symbol can also be defined in the source code itself, as shown here:

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl02');)

#define TRACE // for C# or the Managed Extensions for C++

#Const TRACE=1 ' for Visual Basic .NET

**Create —** Both **Debug** and **Trace** output are captured by a **TraceListener**. By default, output is automatically sent to the **DefaultTraceListener**, which sends output to the Windows system debugger using the **OutputDebugString** Windows API and to the debugger using the **Debugger.Log** method. Messages sent using **Debugger.Log** have no effect if there is no debugger attached.

If you are operating inside the Windows-based debugger, the **Output** window will show **Debug** and **Trace** messages. However, if you are running outside the debugger — for example, on a tester's computer — or if you want to save debugging information to a text-based log file, you must create a new **Listener** and add it to the **Listeners** collection. The **System.Diagnostic** namespace provides an **EventLogTraceListener** and a **TextWriterTraceListener**. Creating a text file to handle debugging messages requires only the following single line of code.

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl03');)

Trace.Listeners.Add(new

TextWriterTraceListener(File.Create("output.txt")));

**Add —** Lastly, you can add calls to the various **Trace.Write** methods to generate debugging information at run time, as shown here:

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl04');)

Trace.WriteLine("");

Trace.WriteLine("Starting tracing...");

Because the first few bytes of the output file contain information about the text encoding of the file and are not readable, you might want to start logging on a new line, by first executing an empty **WriteLine** method.

The next thing to notice is the call to **Trace.Indent**, which by default indents the output by four spaces:

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl05');)

Trace.Indent();

Subsequent calls to **Trace.Unindent** will justify the output back towards the left.

This sample program also uses the **WriteLineIf** method of **Trace** to generate a logging message, but only if a certain condition is not met. In this particular case, the code anticipates that the first numeric entry will begin with a digit other than zero. If that test fails, a message is written, as shown in the following line.

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl06');)

Trace.WriteLineIf(test == "0", "Oops: Leading zero"); // C#

Trace.WriteLineIf(test = "0", "Oops: Leading zero") ' VB

Putting all this together, executing the sample program creates the following Output.txt file.

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl07');)

Starting Tracing...

InitializeComponent()

Creating controls

Setting up Label and TextBox

Setting up numeric buttons

Setting up operations buttons

Adding Clear and Calculate

Adding to Controls collection

9 Clicked

- Clicked

6 Clicked

Calculate Clicked

Dispose()

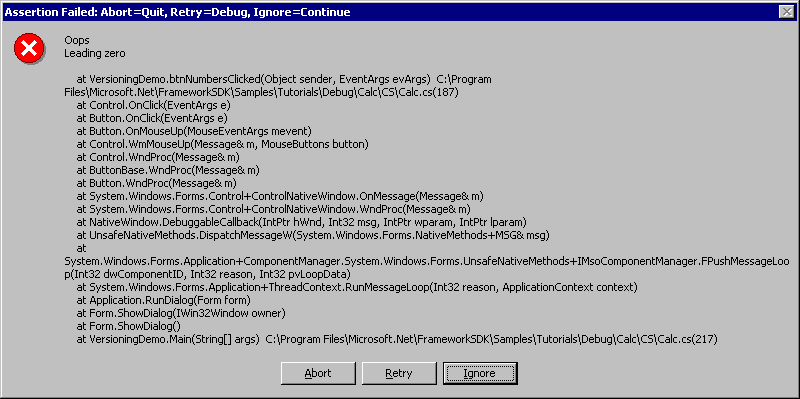
The **Assert** method operates in a similar fashion, except that the output is in the form of a detailed error dialog, and the test fails. The corresponding **Assert** method looks like the following code.

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl08');)

Trace.Assert(test != "0", "Oops", "Leading zero"); // C#

Trace.Assert(test <> "0", "Oops", "Leading zero") ' Visual Basic .NET

When Calc.exe is run and the user enters a leading zero, the following dialog box is produced.



**Note**If your program fails to run to completion, a 0-byte Output.txt file will be produced.

**Compiling Conditionally with Trace and Debug**

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.
3. In the **Property Page** dialog box that appears, click the **Configuration Properties** tab in the left pane and choose the **Build** option.
4. Choose the desired compiler settings.
   * In Visual Basic, select the check boxes for the compiler settings you want to enable. Clear the check boxes for settings you want to disable.
   * In C#, in the **Conditional Compilation Constants** field, type the name of the setting you want to enable.

**Visual C# Note**   To enable both Debug and Trace, type **DEBUG;TRACE** (or **TRACE;DEBUG**). Delete the name of any setting you want to disable. C# is case-sensitive, and the setting names must be uppercase.

**To compile instrumented code using the command line**

* Set a conditional compiler switch on the command line. The compiler will include traceor 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**

**Tip**   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 |

**Note**   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/cc483141(VS.71).aspx) (C#) or [Invoking the Command-Line Compiler](http://msdn.microsoft.com/en-us/library/cc437483(VS.71).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 |
|  | **#undefine TRACE** | Disables tracing |
|  | **#define DEBUG** | Enables debugging |
|  | **#undefine DEBUG** | Disables debugging |

**To disable tracing or debugging**

* Delete the compiler directive from your source code.

- or -

* Comment out the compiler directive.

**Note**   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.