

Lesson Objectives

- After completing this module you will understand:
 - Debugging techniques in ASP.NET



5.1: Debugging in Visual Studio



Overview

- Debugging is about running your code line by line to ensure that the execution path and data are both correct
- Debugging is trickier and difficult as compared to writing a code
- Around 30 percent of the developer's time is spent in debugging
- Visual Studio offers many debugging features

A developer normally has to spend 30 percent of their time debugging a code. Also we know that debugging is more tricky and difficult as compared to writing the program code. In order to successfully debug a code, you should be familiar with the debugging tools available to you and should be able to effectively use them.

Visual Studio offers us many debugging features. We will be exploring through them in this section. Visual Studio IDE always has an admirable support for warning of potential errors at design time. Any syntax errors that is likely to cause problems during compilation process are underlined(squiggled) or highlighted. An error notification pops up when an exception occurs during debugging session and recommends a course of action that would prevents the exception. These all features has been part of the Visual Studio IDE.

Having all these features incorporated into the IDE itself helps in writing an almost bug free code. Why almost?? Because it is near to impossible to write a Bug free code.

The editor that we user to write C# code shows squiggles and tooltips for many errors before compilation. Also Visual Studio now has greatly improved the XML editor with enhancements like full XML1.0 syntax checking, Support for DTD and XSD validation.

Not only the XML files but even the HTML or ASPX code is checked by the editor to show any potential compilation errors.

5.2: Debugging Tools in Visual Studio



Debugging Tools

- Debugging Tools
 - Integrated Debugger
 - Visual Debugger

Visual Studio.NET provides you with debugging tools like:

Integrated debugger: Integrated with the Visual Studio itself. There is no need to run a separate tool to debug your application.

Visual debugger: This is a separate tool that you can find at "Microsoft Visual Studio 5\SDK\v2.0\GuiDebug\DbgCLR.exe" (locate this at your own Visual Studio folder). Using this tool you can debug complex bugs that you can not fix using the aforementioned integrated debugger.

5.3: What does Debugging Offer?



Features

➤ Debugging Offers:

- Code execution examination. Step line by line
- Viewing variable values at each step
- Changing the value of a certain variable
- Moving the execution point and running application to another point

➤ These features can be categorized as:

- Breakpoints
- Stepping
- Data Viewing

ASP.NET provides you with two classes: 'Debug' and 'Trace'. These are diagnostic classes available in System.Diagnostics namespace, to help you generate logs during development and debugging.

During debugging, you can see how exactly your code is executed by stepping line by line, viewing the values of your variables during each step and showing how these values are changed by each step. You can change the value of a certain variable, You can move the execution point and run application to another point.

Visual Studio integrated debugger has many features you can use to debug your applications.

These features includes the following: Breakpoints, Stepping, and Data Viewing.

5.4: Breakpoints



Overview

- Places in code, where the debugger stops application execution
- To set the breakpoint, click the gray margin to the left of the line
- Point is visible as a red circle in the left gray margin
 - To clear the breakpoint, click the red circle or press F9.

Breakpoints are places in the code, where the debugger will stop the execution of the application. After stopping, you can watch the state of the application's variables, and then you can step through your code.

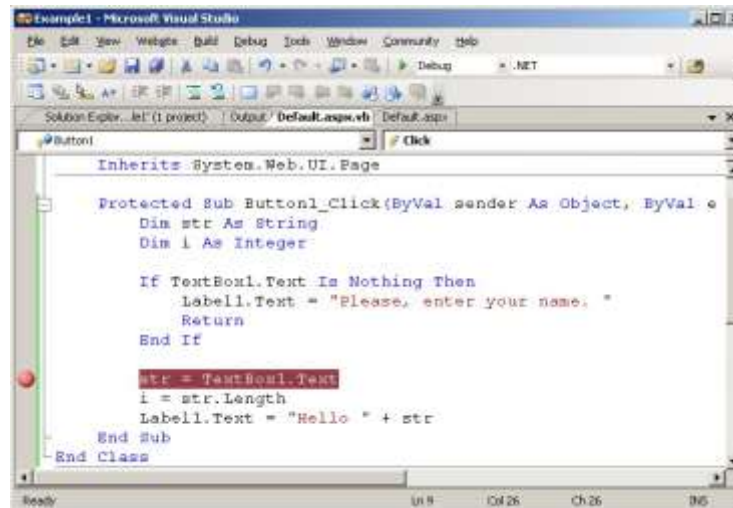
To set a breakpoint, simply click once at the gray margin next (left) to the line where you want to set the breakpoint. A red circle in the left gray margin, and a red shading over this line of code will appear, indicating that, this line is a breakpoint line.

To clear the breakpoint, just click the red circle again and it will disappear. You can also use F9 or select, on the Debug menu, click Toggle Breakpoint to set a breakpoint or to clear it.

You can think of a breakpoint as a signal to the debugger. It tells the integrated debugger to pause the execution of your application and puts it at the break mode. While in break mode, your application's data is still in memory, and you still have the ability to resume the execution at any time, but the difference here is that you can control the execution of your code step by step, line by line, watching every piece of code closely looking for violations or bugs.

5.4: Breakpoints

Overview (Contd...)



The above figure shows a red circle indicating it is a breakpoint line

5.4: Breakpoints



Advantages of Breakpoints

- No code required to be added to your program
- Set or disable breakpoints without changing source codes
- Pause the execution of a program at any point
- Breakpoint management is simple

Advantages of Breakpoints:

Breakpoints are not actual source code you have to add to your program. You do not type a breakpoint statement into your source window.

Because are not statements, breakpoints produce no extra code, and never changes the size of your code file.

You can set or disable a breakpoint without changing your source code.

A breakpoint gives you the ability to pause the execution of your program at a certain point you specify, and this is very important in large applications. The alternative is that you run the program line by line from the first line till the line you want to stop at, which is undesirable at all.

You can manage all your breakpoints from the 'Breakpoints' window. In this window you can enable, disable, or even delete a breakpoint without need to search for it in your source code.

To display the 'Breakpoints' window, on the Debug menu, on the Windows sub-menu, click Breakpoints.

5.4: Breakpoints



Important Features of Breakpoints

➤ Hit Count:

- Specify how many times you wish to hit your breakpoint before the application breaks.
- Useful when one has to deal with loops.

➤ Condition:

- Set a condition.
 - Breakpoint is hit when this condition evaluates to TRUE.

Breakpoints have many magnificent features and properties. Using these features, you can control and modify the behavior of a breakpoint. The two most used features are Hit Count and Condition.

Hit Count

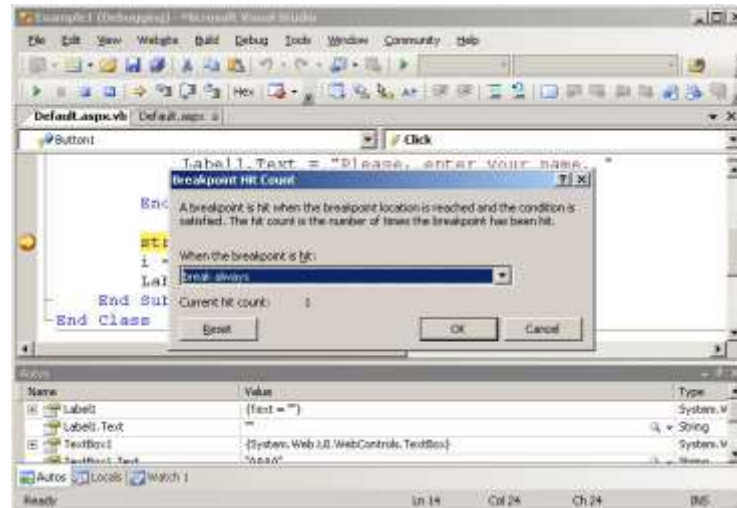
The Hit Count property gives you the ability to specify how many times you want to hit your breakpoint before breaking the application. You can set a hit counter to stop execution after hitting the breakpoint 2 times, may be 10 times, or any number you choose. This is very useful when you are dealing with loops. Some times a bug does not appear the first time you execute a loop, access a variable, or call a function.

Condition Property

The Condition property gives you the ability to set a condition, that the debugger will always evaluate. When the condition evaluates to true, the breakpoint will be hit, or otherwise it will be skipped as if it does not exist at all.

5.4: Breakpoints

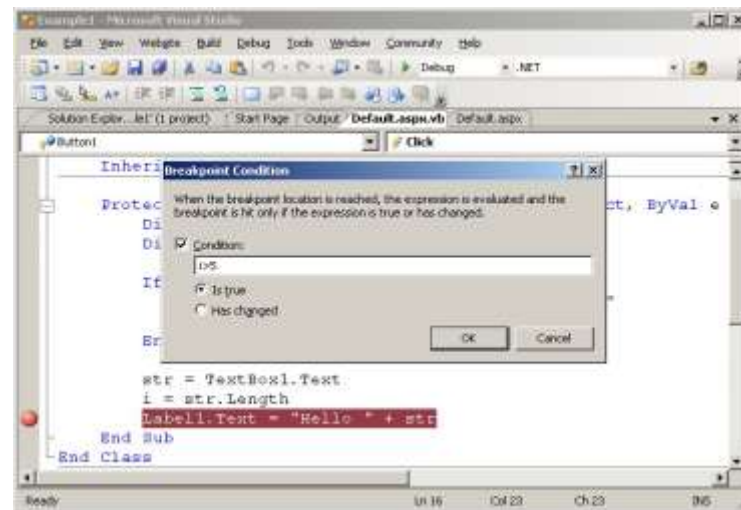
Setting a Hit Count



To set a hit counter, position your mouse pointer over a breakpoint line. Right-click the right-mouse button and click Breakpoint Hit Count as shown in the above figure. You need to select a proper option from the combo box to be able to set the Hit Count.

5.4: Breakpoints

Setting a Breakpoint Condition



To set a Breakpoint Condition, position the mouse pointer over the new breakpoint. Right-click and click Breakpoint Condition. Specify the condition in the textbox.

5.4: Breakpoints

Demo: BreakPoints

- Adding Breakpoints
- Setting Hit Count
- Setting Breakpoints Condition
- Viewing and Changing Data



5.5: Stepping



Overview

➤ Following are options to step through code:

- Step Into
- Step Over
- Step Out

Once you stopped your application at a given point, you can step through your code line by line, that is, execute your code line by line. The integrated debugger provides you with a number of features to help you step through your code.

There are three commands you can use for stepping through code. These commands are:

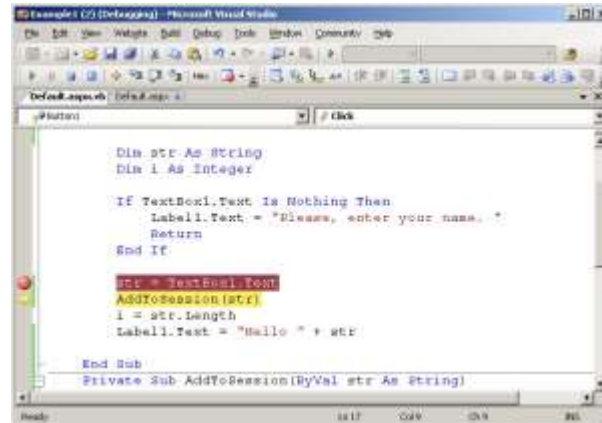
- Step Into
- Step Over
- Step Out

5.5: Stepping

Step Into

➤ Step through the code line by line

- Press F11 or on the Debug menu, click Step Into.



Step Into

This command is to provide the ability to walk through your code while you are in the break mode. After stopping at a breakpoint line, you certainly need to go to the next line, and the line next to the next line and so on. Press the F11 key (or on the Debug menu click Step Into) that executes the step into command one can do so.

5.5: Stepping



Step Over

- Step Over differs from Step Into in the way it handles a function call
- When we use Step Over, the function is executed as a whole. No need to step through it line by line
- Press F10 or on the Debug menu, click Step Over

Step Over

You can step over line by pressing F10, or on the Debug menu click Step Over. Step Over is just like step into, but it differs in only one aspect, which is the way it handles a function call.

When you have a function call, step into will move you inside the function code line by line, which means executing the function call line by line. When you use step over, the function is executed as a whole without the need to step through its lines line by line. The highlighting indicator then moves to the next line.

5.5: Stepping



Step Out

- Use Step Out when inside a function and you wish to go directly to the point from where the function is called without stepping to the end of the function
- If the control is at the top level of a program, choosing the Step Out option causes the program to resume running as normal.
- Press shifted+F11 or on the Debug menu, click Step Out

5.5: Stepping

Demo: Stepping

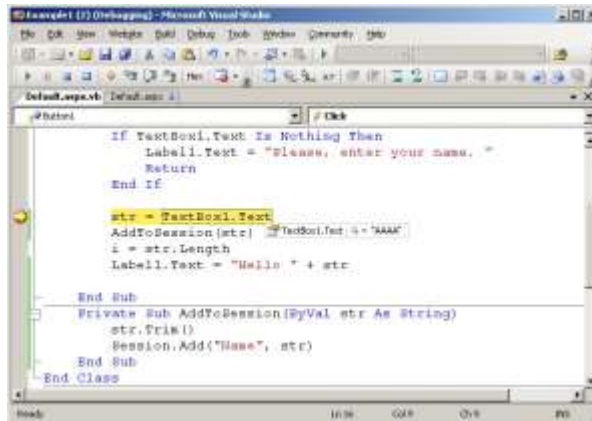
➤ Demonstrating Step Into, Step Over and Step Out



5.6: Data Viewing

Overview

- View and Track data while debugging
- Edit values in break mode



The integrated debugger provides you with many different tools that you can use to view and track data while debugging. It also gives you the ability to edit your data values while you are in break mode. You can then complete the debugging process depending on the new modified values. Most of these tools function properly only in break mode.

You can watch the current value of a variable by simply placing your mouse pointer over it. An example for this behavior can be seen in the above figure. You can also change its value by right clicking the data tip. You need to choose Edit Value from the shortcut menu to achieve this. Type the new value then press enter.

5.7: Variables Window



Types of Windows

➤ Integrated Developer provides following Windows to display and edit application's variables and expressions:

- Locals Window:
 - Watch variables inside your current scope
 - On the Debug menu, navigate click Windows from the Locals menu to view this window
- Autos Window:
 - View variables in the current line of code and above it.
 - On the Debug menu, navigate click Windows from the Autos menu to view this window
- Watch Window:
 - Watch certain variables or expressions

The integrated debugger also provides a number of windows to display and edit your application's variables and expressions. These windows are:

Locals Window
Autos Window
Watch Window

Each window has three columns: 'Name', 'Value', and 'Type'.

In the 'Locals' window you can watch the variables inside your current scope. For example, when you are inside a function or a sub. The entries inside this window are created automatically by the debugger. To display this window on the Debug menu, on the Windows sub-menu, click Locals.

The entries inside the 'Autos' window is also created automatically by the debugger. Use this window to view the variables inside the current line of code, and the line above it. To display this window on the Debug menu, on the Windows sub-menu, click Autos.

The 'Watch' window is the window you can use to watch certain variables or expressions. You can add the variable or expressions you want to watch by positioning your mouse pointer over it, then click the right mouse button and choosing 'Add Watch' from the pop up menu. You can add more than one variable. To display this window on the Debug menu, on the Windows sub-menu, click Watch 1.

5.7: Variables Window

Demo: Variables Window

➤ Trying to view values in Variables Window



5.8: Tracing



Overview

- Technique to monitor the execution of ASP.Net applications
- ASP.Net provides a rich support to trace

Tracing is a technique to monitor the execution of your ASP.Net applications. The exception details and program flow can be recorded in away that the output of the program is not affected.

Earlier developers used lot of Response.Write statement to trace if a particular piece of code is being reached. Obviously this resulted in a cluttered code and this kind of intrusive tracing was very inconvenient to clean up.

In ASP.Net there is rich support for tracing. ASP.Net 2.0 include a number of small improvements to tracing over ASP.Net1.1 including trace forwarding between the ASP.Net page-specific Trace class and standard Base Class Library System.Diagnostics.Trace which is more often used by non-web developers.

5.8: Tracing



Page Level Tracing

- ASP.Net tracing can be enabled on a page level basis by adding Trace="true" to the Page Level directive

```
<% Page Language="C#" Trace="True" %>
```

- Use TraceMode attribute to sort the methods in the trace output by Category or Time
- Tracing can be enabled programmatically by using
 - Trace.IsEnabled property

ASP.NET tracing can be enabled on a page-by-page basis by adding Trace=true" to the Page directive in any ASP.NET page.

Additionally, one can add the TraceMode attribute that sets SortByCategory or the default, SortByTime.

You can use SortByTime to see the methods that take up the most CPU time for your application. One can enable tracing programmatically using the Trace.IsEnabled property.

5.8: Tracing



Application Tracing

- Add trace settings to web.config file to enable application-level tracing

```
<configuration>
  <system.web>
    <trace enabled="true" pageOutput="false"
      requestLimit="20" traceMode="SortByTime" localOnly="true"/>
  </system.web>
</configuration>
```

Tracing can be enabled at application level by modifying the web.config file. You can see the snippet on the Slide.

In the example `pageOutput="false"` and `requestLimit="20"` are used which indicates that trace information is stored for 20 requests but not displayed by the page.

The page level settings take precedence over the application level settings so if `enabled="false"` is set in Web.config but `trace="true"` is set on the page, tracing occurs.

5.8: Tracing



Viewing Trace Data

- Request a page "trace.axd" to view tracing for multiple page requests at the application level
- Set pageOutput to "True" to view Trace data

```
<configuration>  
  <system.web>  
    <trace enabled="true" pageOutput="true">  
  </system.web>  
</configuration>
```

Tracing can be viewed for multiple page requests at the application level by requesting a special page called trace.axd. The trace.axd does not actually exist but is provided by System.Web.Handlers.TraceHandler, a special IHttpHandler to which trace is bound.

When ASP.Net detects an HTTP request for trace.axd, that request is handled by the TraceHandler rather than by a page.

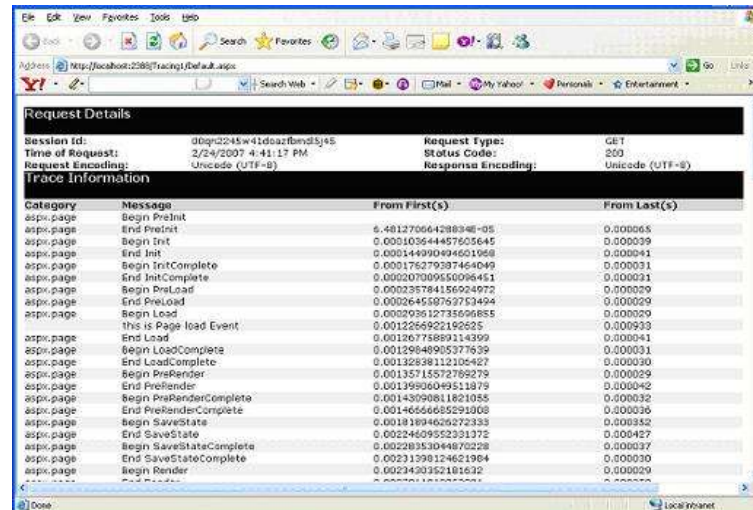
Create a Web site and a page, in the Page_Load event, call Trace.Write(). Enable tracing in the web.config as shown on the slide and write the code.

```
protected void Page_Load(object sender, EventArgs e)  
{  
  Trace.Write("This message is from the Start of the Page_Load method");  
}
```

Hit the page in the browser a few times and see the Trace output being generated. The message from Trace.Write appears after the Begin Load and before End Load.

5.8: Tracing

Viewing Trace Data (Contd...)



Category	Message	From First(s)	From Last(s)
asp:page	Begin PreInit	6.48127066428834E-05	0.000065
asp:page	End PreInit	0.000103644457605645	0.000039
asp:page	Begin Init	0.000144990494601068	0.000041
asp:page	End Init	0.000175279397464049	0.000031
asp:page	Begin InitComplete	0.000207009650096451	0.000021
asp:page	End InitComplete	0.000235784156924072	0.000029
asp:page	Begin PreLoad	0.000264558762753494	0.000029
asp:page	End PreLoad	0.000293612735696855	0.000029
asp:page	Begin Load	0.0012269922192625	0.000933
asp:page	End Load	0.00126775889114399	0.000041
asp:page	Begin LoadComplete	0.00129848905377639	0.000031
asp:page	End LoadComplete	0.00132838112105427	0.000030
asp:page	Begin PreRender	0.00135718372789279	0.000029
asp:page	End PreRender	0.00139906049511879	0.000042
asp:page	Begin PreRenderComplete	0.00143090811821055	0.000032
asp:page	End PreRenderComplete	0.0014606685291008	0.000036
asp:page	Begin SaveState	0.00181894626272335	0.000352
asp:page	End SaveState	0.00224609552331372	0.000427
asp:page	Begin SaveStateComplete	0.00228353044870228	0.000037
asp:page	End SaveStateComplete	0.00231398124621984	0.000030
asp:page	Begin Render	0.0023420352181632	0.000029

If you have set trace enabled="true" and pageoutput="true", the browser will show the trace information as shown above.

5.8: Tracing



Information in Trace

- Information in Trace is categorized into eleven sections:
 - Request Details
 - Trace Information
 - Control Tree
 - Session State
 - Application State

There are eleven different sections in Trace providing great deal of information:

Request Details: This section includes the ASP.NET Session ID, the character encoding of the request and response, and the HTTP conversation's returned status code.

TraceInformation: This section includes all the `Trace.Write` methods called during the lifetime of the HTTP request and a great deal of information about timing. The timing information located here is valuable when profiling and searching for methods in your application that take too long to execute.

Control Tree: Control tree presents an HTML representation of the ASP.NET Control Tree. Shows each control's ID, run time type, the number of bytes it takes to be rendered, and the bytes it requires in View State and Control State.

Session State: Lists all the keys for a particular user's session, their types and their values.

Application State: Lists all the keys in the current application's Application object and their type and values.

5.8: Tracing



Information in Trace (Contd...)

- Request Cookies
- Response Cookies
- Headers Collection
- Form Collection
- QueryString Collection
- Server Variables

Request Cookies: Lists all the cookies passed in during the page is requested.

Response Cookies: Lists all the cookies that were passed back during the page's response.

Headers Collection: Shows all the headers that might be passed in during the request from the browser, including Accept-Encoding, indicating whether the browser supports the compressed HTTP responses and Accept languages.

Form Collection: Displays a complete dump of the Form Collection and all its keys and values.

QueryString Collection: Displays a dump of the Querystring collection and all its contained keys and values.

Server Variables: A complete dump of name-value pairs of everything that the web server knows about the application.

5.8: Tracing

Trace.axd



- Trace.axd:
 - Collects detailed information for all the requests
- Invoke for the application using:
 - <http://localhost/applicationname/trace.axd>

Trace.axd

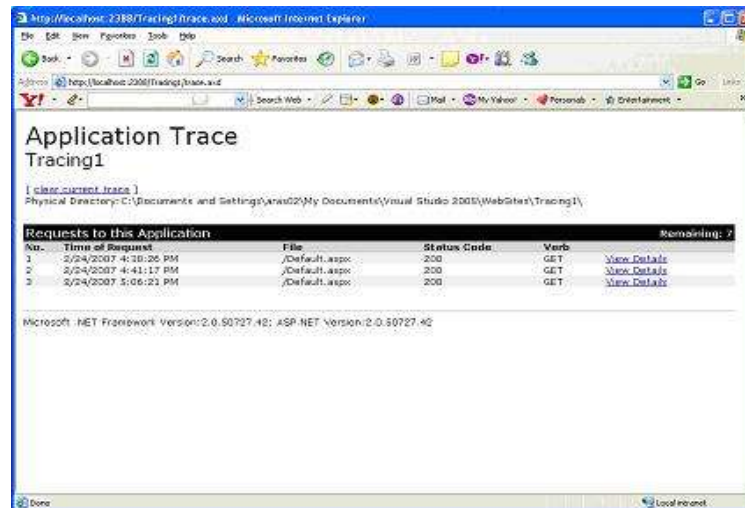
Page output of tracing shows only the data collected for the current page request. However, if you want to collect detailed information for all the requests then we need to use Trace.axd.

We can invoke Trace.axd tool for the application using the following URL <http://localhost/applicationname/trace.axd>.

Simply replace page name in URL with Trace.axd.

5.8: Tracing

Trace.axd (Contd...)



Trace.axd displays all the tracing information for all requests up to a present limit. Above figure shows that three requests have been made to this application and the right side of the header indicates "Remaining:7" which means that there are seven more requests remaining before tracing stops for this application.

After that final request, tracing data is not saved until an application recycle or until you click "Clear Current Trace" from the Trace.axd page.

The request limit can be raised in Web.config by setting requestLimit to a higher value as shown below

```
<trace enabled="true" requestLimit="20" pageOutput="true"/>
```

5.8: Tracing

Demo: Tracing

➤ Demonstrating Tracing with various options

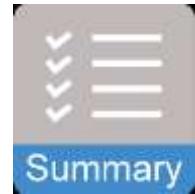


Summary



➤ In this lesson you have learnt:

- Debugging techniques in ASP.NET
- Tracing in ASP.NET



Review Question

➤ Question 1: _____ window shows the variables in the active statement and the previous code line

- Autos
- Locals
- Watch
- Immediate

➤ Question 2: Tracing can be viewed for multiple page requests at the application level by requesting a page called as _____

- Trace.axd
- Application.axd
- Global.axd
- Global.asax

