msdn® training

Module 5: Using Trace in Microsoft ASP.NET Pages

Contents

Microsoft®

Overview

- Overview of Tracing
- Trace Information
- Page-Level Trace
- Application-Level Trace

In earlier versions of Active Server Pages (ASP), code tracing was accomplished by inserting **Response.Write** statements wherever required. A drawback of this technique was the removal of the excess code that was added when the application was deployed, and the resulting inability to use the output from the **Response.Write** statements in a production environment. Microsoft® ASP.NET overcomes such drawbacks by introducing an automatic tracing mechanism. This tracing mechanism can be enabled and disabled either on a page-by-page basis or for an entire application, and it can be configured in the web.config file.

After completing this module, you will be able to:

- Describe page-level and application-level tracing.
- Enable and disable tracing for an ASP.NET page.
- Add custom trace information.

Overview of Tracing

- **With Tracing You Can:**
  - Output variables or structures
  - Assert whether a condition is met
  - Trace through the execution path of the application
- **Can Be Enabled and Disabled, Either on a Page-By-Page Basis or for an Entire Application**

The tracing feature in ASP.NET enables you to insert debugging print statements into your code. These statements allow you to output variables or structures, assert whether a condition is met, or trace through the execution path of the application.

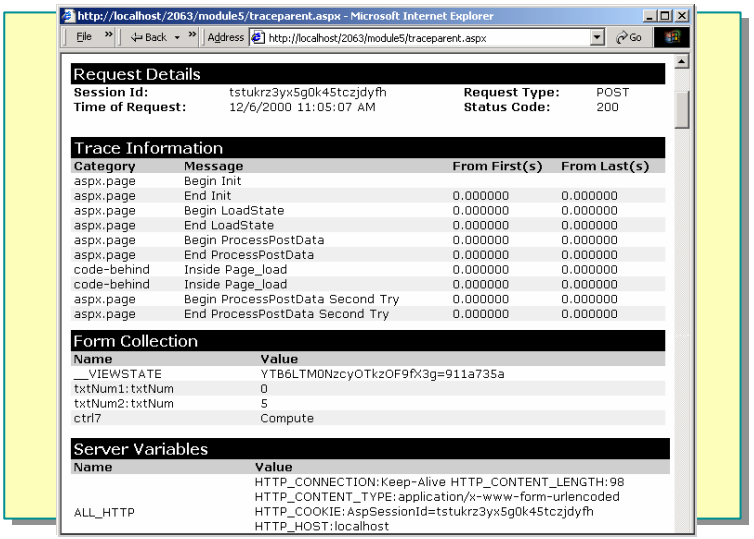
All of this information is output to an .aspx page or saved to a log, depending on settings in the web.config file.

You can also use the **Debug** object to output debugging information. By using methods in the **Debug** object to print debugging information and check your logic with assertions, you can make your code more robust without impacting your shipping product's performance and code size.

While you are debugging an application during development, both your trace and debug output statements go to the **Output** window in Microsoft Visual Studio® .NET. However, with trace statements, you can debug a production Web site simply by changing a setting in the web.config file and having the trace statements output on the page.

In this course, we will focus only on the trace feature of ASP.NET.

Trace Information



Tracing provides plenty of information to the developer of an ASP.NET application. The trace output screen contains the following categories of information.

Category	Description
Control Tree	List of all items on the page.
Cookies Collection	List of cookies being used.
Form Collection	List of controls and their values on the form being posted.
Headers Collection	List of items in the Hypertext Transfer Protocol (HTTP) header.
Request Details	Information about the request: session ID, time of request, type of request, and request status.
Server Variables	List of all server variables and their values.
Trace Information	Output from standard and custom trace statements.

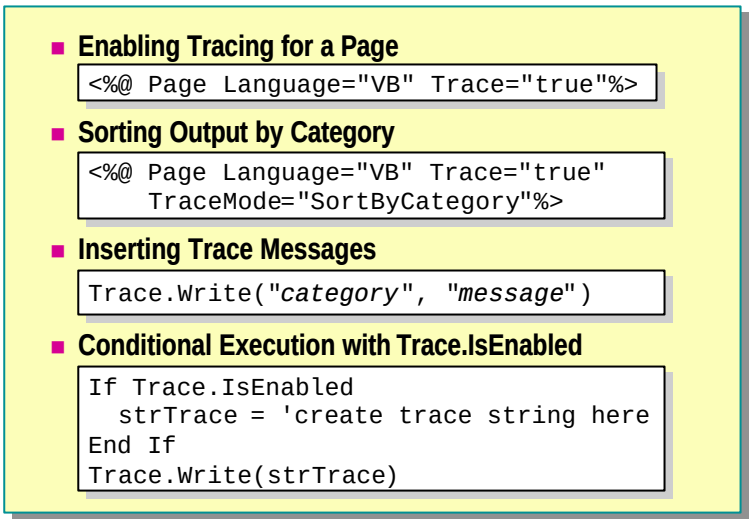
Page-Level Trace

- How Does Page-Level Trace Work?
- Demonstration: Adding Page-Level Trace Statements
- Tracing into a Component

You can use page-level tracing to write debugging statements directly to the output of a page and to conditionally execute debugging code when tracing is enabled. The performance data and the developer-specified messages (if any) are injected into the Hypertext Markup Language (HTML) output stream to the client to help clarify precisely what occurs when the framework processes a page request.

In this section, you will learn how the page-level trace mechanism works.

How Does Page-Level Trace Work?



The page-level trace mechanism performs two functions when it is enabled:

- It automatically appends a table of performance data to the end of the page.
- It allows developers to insert custom diagnostic messages throughout their code.

Enabling Tracing for a Page

The trace capability must be enabled prior to use. To enable tracing for a page, include the following directive at the top of the page code:

```
<%@ Page Language="VB" Trace="true" %>
```

The page exposes the **Trace** property of the type **TraceContext**, which can be used to output debugging statements to the page output. The default value of the **Trace** attribute is set to **False**.

Sorting Trace Messages

The **TraceContext** class contains the **TraceMode** property that defines how trace messages are sorted in the page output.

```
<%@ Page Trace="true" TraceMode="SortByCategory" %>
```

TraceMode is of the type **TraceModeEnum**, and has two possible values:

- SortByTime

  Trace messages are output in order of chronological occurrence. This is the default setting.
- SortByCategory

  Trace messages are sorted alphabetically by the category argument to the **Trace.Write** and **Trace.Warn** methods.

Inserting and Organizing Trace Messages

You can use the **Trace** object to write debugging statements. You can use two methods to insert custom messages: **Trace.Write** and **Trace.Warn**. **Trace.Write** is used to display messages. **Trace.Warn** is used to display warnings. **Trace.Write** and **Trace.Warn** are easy to distinguish because **Trace.Warn** uses a red font color for its output, whereas **Trace.Write** uses the default font color.

In their simplest forms, each method takes two string arguments:

```
Trace.Write("category", "message")
Trace.Warn("category", "message")
```

The category argument is used to organize messages. For example:

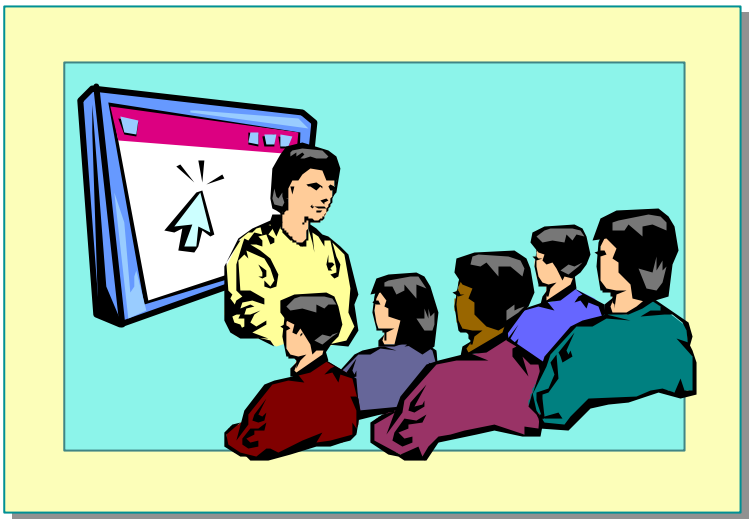
```
Trace.Write("Custom Trace","Beginning User Code...")
Trace.Warn("Custom Trace","Array count is null!")
```

Conditional Execution with Trace.IsEnabled

Often, you will need to execute additional code to construct the statements to pass to the **Trace.Write** or **Trace.Warn** methods. The code should execute only if tracing is enabled for the page. To support this, the page exposes a Boolean property, **Trace.IsEnabled**, that returns **True** only if tracing is enabled for the page. You should first check the **Trace.IsEnabled** property to guarantee that your debugging code will execute only when tracing is enabled.

```
If Trace.IsEnabled
  strTrace = "create a trace string here"
End If
Trace.Write(strTrace)
```

Demonstration: Adding Page-Level Trace Statements



In this demonstration, you will learn how to use trace at the page level.

↙ To run the demonstration

1. Edit the file add.aspx in the folder <*install folder*>\Democode\Mod05.
2. Enable tracing by adding the **Trace="True"** attribute to the **@Page** directive.

```
<%@ Page Language="vb" Trace="True"%>
```

3. View the page in Microsoft Internet Explorer.

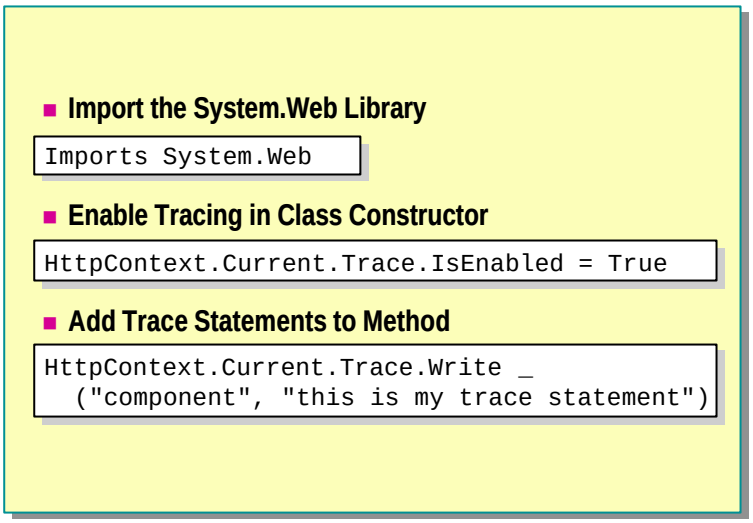
   Trace messages are shown on the page.
4. Add a custom trace message to the **Page_Load** event.

```
Trace.Write ("add Page_Load", "txtNum1 = " &
Cstr(txtNum1.pNum))
```

5. View the page again.
6. Disable tracing, and then view the page again.

   No trace messages are shown on the page.
7. Add a trace message to the **numberbox.ascx** user control.
8. Enable tracing in add.aspx, and view the page in Internet Explorer.

Tracing into a Component



If you have a component that is called from an ASP.NET page, you can also add trace statements to the component.

↙ To add trace to a component

1. At the top of the component, import the System.Web library.

   ```
   Imports System.Web
   ```

2. In the constructor of the class to which you want to add trace statements, enable tracing with the following statement:

   ```
   HttpContext.Current.Trace.IsEnabled = True
   ```

   In the preceding code, **HttpContext.Current** gets the **Context** object for the current request.

3. Next, in the method, use the following:

   ```
   HttpContext.Current.Trace.Write _
         ("component", "this is my trace statement")
   ```

The following table lists some useful information about the results of enabling trace on a page or component.

Case	Result
If you enable trace in the constructor of the component.	All pages that access the component will have trace enabled even if the page does not have it enabled.
If you enable trace in a component method.	Only the page that accesses the method of the component will have trace enabled, even if the page does not have trace enabled.
If you disable trace in the constructor of the component.	All pages that access the component will have trace disabled even if the page has trace enabled.
If you do not explicitly enable or disable trace for a component, but the page that calls the component has trace enabled.	Trace in the component will be enabled for that specific page.

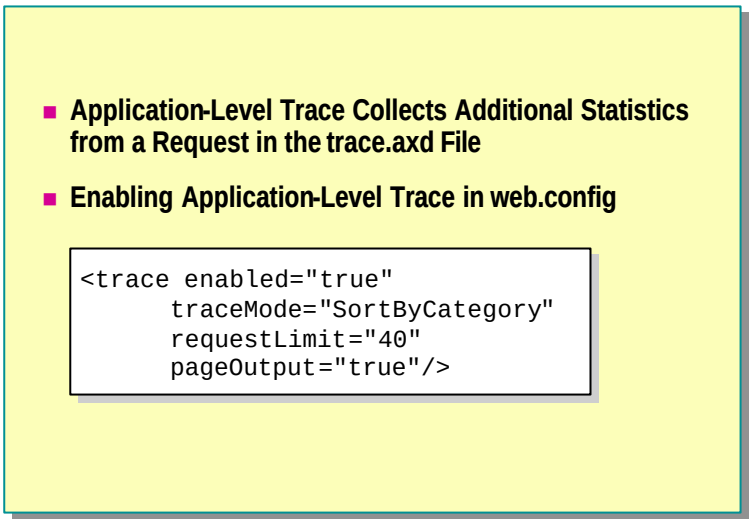
Application-Level Trace

- How Does Application-Level Trace Work?
- Demonstration: Adding Application-Level Trace Statements

ASP.NET also provides a way to enable tracing for the entire application, rather than just a single page of the application. An ASP.NET application is the collection of files and folders in a virtual root of Internet Information Services (IIS).

In this section, you will learn how an application-level trace works.

How Does Application-Level Trace Work?



In addition to the page-level trace functionality, ASP.NET provides a way to enable trace output for an entire application. Enabling trace at the application level has the effect of enabling page-level trace for every page within that application, provided that there is no page-level directive to explicitly disable trace. When application-level tracing is enabled, the ASP.NET runtime also collects several additional statistics, such as the state of the control hierarchy, the contents of session and application state, the form and query string input values, and so on. These statistics are collected for a specified number of requests as determined by the application's configuration file.

Enabling Tracing

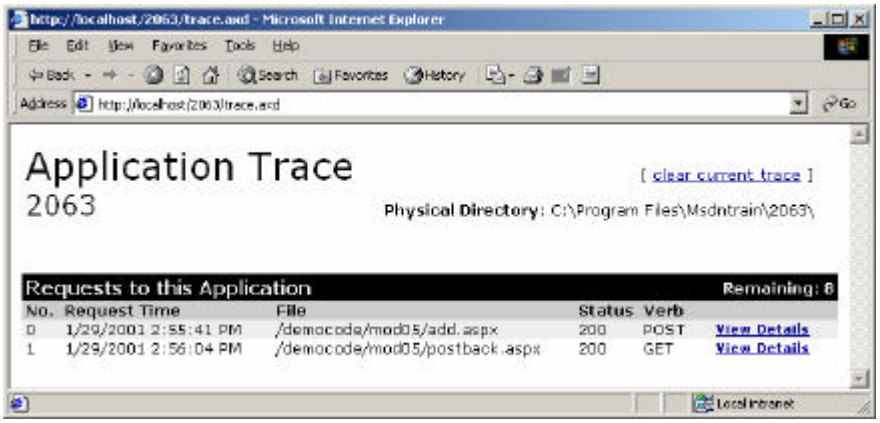
With ASP.NET, there is a configuration file that you can use to configure certain application attributes. This file is named web.config and is located in the root folder of the application. You will learn more about the web.config file in Module 7, "Creating a Microsoft ASP.NET Web Application," in Course 2063B, *Introduction to Microsoft ASP.NET.*

The web.config file is in Extensible Markup Language (XML) format; therefore, all section and attribute names are case-sensitive.

To enable tracing for an application, place the following code in the web.config file in the application's root folder.

```
<trace enabled="true"/>
```

By using the above configuration, each page in the application will output page-level trace statements to the file **trace.axd** in the application's root folder, as shown in the following illustration.



If you want trace messages to be displayed on the individual pages of your application, set the **PageOutput** attribute of the trace element to **True**.

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

Note The trace section in the configuration file is contained in a <system.web> section inside the <configuration> section.

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

Setting Trace Attributes

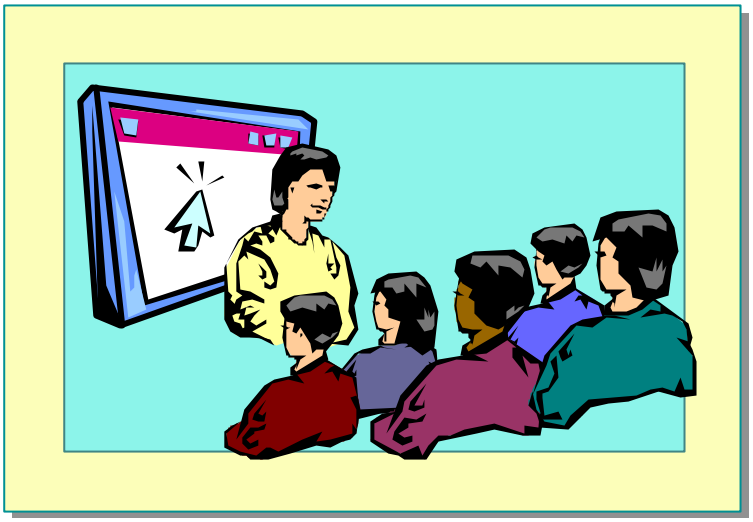
The trace section of the configuration file also supports an attribute for controlling whether trace statements are output to the client browser or are available only from trace.axd. The attributes supported in the trace configuration section are listed in the following table.

Value	Description
enabled	Set to **True** or **False**. Indicates whether tracing is enabled for the application or not. Default is **True**.
pageOutput	Indicates whether trace information should be rendered at the end of each page, or accessible only through the trace.axd utility. Default is **False** (trace information is only accessible through trace.axd).
requestLimit	Number of trace requests to store on the server. Default is 10.
traceMode	Set to **sortByTime** or **sortByCategory**. Indicates the display order for trace messages. Default is **sortByTime**.

The following example shows how you can set trace attributes. The following configuration collects trace information for up to 40 requests and prevents trace statements from being output to the requesting browser:

```
<trace enabled="true" traceMode="sortByCategory"
   requestLimit="40" pageOutput="false"/>
```

Demonstration: Adding Application-Level Trace Statements



In this demonstration, you will learn how to use trace at the application level.

↙ To run the demonstration

1. Copy the web.config file from the folder <*install folder*>\DemoCode\Mod05 to the <*install folder*>, which is the virtual root of the 2063 Web site.
2. Edit the web.config file and enable application-level tracing by adding the following line of code to the file:

   ```
   <trace enabled="true"/>
   ```

3. View the page http://localhost/2063/DemoCode/Mod05/postback.aspx in Internet Explorer.

   You will not see any trace output in the page.

4. View the page http://localhost/2063/trace.axd in Internet Explorer and look at the details for the postback.aspx request.

   Note You may need to click **Refresh** to see the latest entries in the trace.axd page.

   This is where the details are shown. Scroll down to show the values sent to the form.

5. Change web.config to display the trace messages on the page by adding the **pageOutput="true"** attribute to the trace element.
6. View the page http://localhost//2063/DemoCode/Mod05/postback.aspx in Internet Explorer again.

   Now the trace messages are displayed on the page.
7. When you are done with this demonstration, delete the web.config file in the root of the 2063 virtual directory so that tracing will be turned off for future demonstrations.

Lab 5: Adding Trace to an ASP.NET Page

Objectives

After completing this lab, you will be able to:

- Enable and disable tracing for an ASP.NET page.
- Add custom trace messages to an ASP.NET page and a middle-tier component.

Prerequisite

Before working on this lab, you must know how to compile a component.

Lab Setup

There are starter and solution files associated with this lab. The starter files are in the folder <*install folder*>\Labs\Lab05\Starter and the solution files for this lab are in the folder <*install folder*>\Labs\Lab05\Solution.

Estimated time to complete this lab: 30 minutes

Exercise 1
Enabling Tracing

In this exercise, you will enable and disable tracing and add custom messages to the trace stream.

↙ To enable tracing in a page

1. Open the file ProductsList.aspx in the folder InetPub\wwwRoot\ASPNET.
2. At the beginning of the code, modify the existing **@Page** directive, add the **trace** attribute, and set its value to **True**, as shown in the following code:

   ```
   <%@ Page Language="vb" trace="true" %>
   ```

3. Save your changes to the ProductsList.aspx page.
4. View the page in Internet Explorer.

   You should see the trace information at the bottom of the page.

↙ To add custom trace messages

1. Go to the beginning of the **Page_Load** event procedure of the ProductsList.aspx page and add a trace message that displays **Beginning of Page_Load** in a category named **Product List**.

   Your code should look like the following:

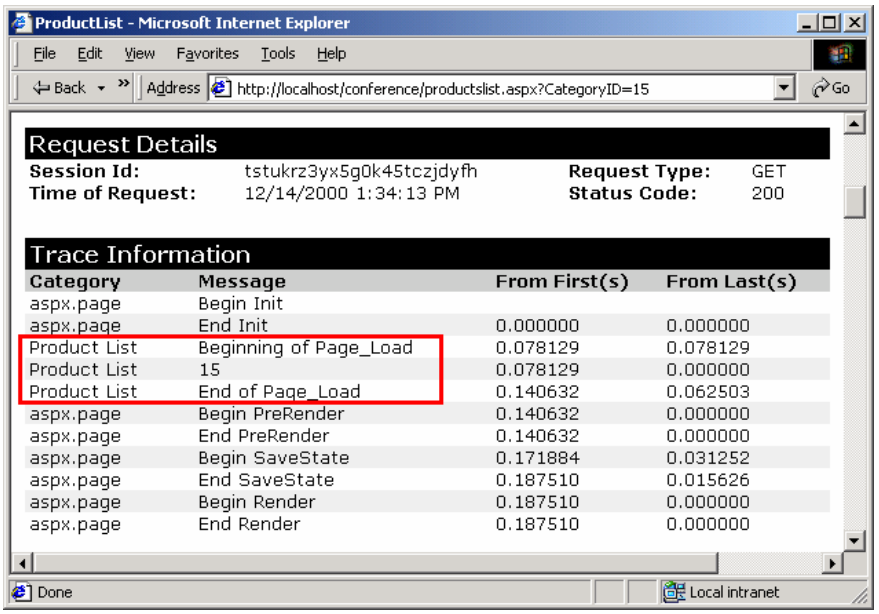
   ```
   Trace.Write("Product List", "Beginning of Page_Load")
   ```

2. Add another trace message to the end of the **Page_Load** event procedure that displays **End of Page_Load**.
3. Add a third trace message that displays the value of the variable **categoryID** after reading it from the **QueryString**.
4. Save your changes to ProductsList.aspx.

5. View ProductsList.aspx in Internet Explorer, passing the CategoryID=15:

   http://localhost/ASPNET/ProductsList.aspx?CategoryID=15

   You should see your custom messages in the **Trace Information** section, as shown in the following illustration.



Exercise 2
Tracing into a Component

In this exercise, you will add trace statements to the DB2063 component that is called from the ProductsList.aspx page.

↙ To enable tracing in the DB2063 component

1. Open the file GetProducts.vb in the folder InetPub\wwwRoot\ASPNET\Components.
2. Import the System.Web library.

   Your import statement should look like the following:

   ```
   Imports System.Web
   ```

3. In the **GetProducts** method, enable tracing by setting the **HttpContext.Current.Trace.IsEnbled**property to **True**.

   Your code should look like the following:

   ```
   HttpContext.Current.Trace.IsEnabled = true
   ```

Note You can also put this command in the constructor of the class if you need to use tracing in the whole component.

↙ To add custom trace messages

1. Go to the beginning of the **GetProducts** method and add a custom trace message that displays **Beginning of GetProducts** in a category named **DB2063.Products Component**.

   Your additional instruction should look like the following:

   ```
   HttpContext.Current.Trace.Write _
      ("DB2063.Products Component", _
          "Beginning of GetProducts")
   ```

2. Add another trace message at the end of the procedure, before the **Return** command that displays **End of GetProducts**.
3. Add a third trace message to the **GetProducts** method that displays the value of the **categoryID** parameter.

↙ To save and test

1. Save your changes to GetProducts.vb.
2. Compile the component.
   a. To compile the component by using an MS-DOS® command prompt, open an MS-DOS command prompt, navigate to the folder InetPub\wwwroot\ASPNET\components, and run the following command:

```
vbc /t:library /out:..\bin\db2063.dll /r:System.dll
/r:System.Web.dll /r:System.Xml.dll /r:System.Data.dll
GetProducts.vb
```

   b. To compile the component by using Visual Studio .NET, click **Build** on the **Build** menu.

Note If you compile the component by using Visual Studio .NET, you will need to change the **Page_Load** event procedure in the ProductsList.aspx page to instantiate the **Products** object from the newly built DB2063 component in the ASPNET project.

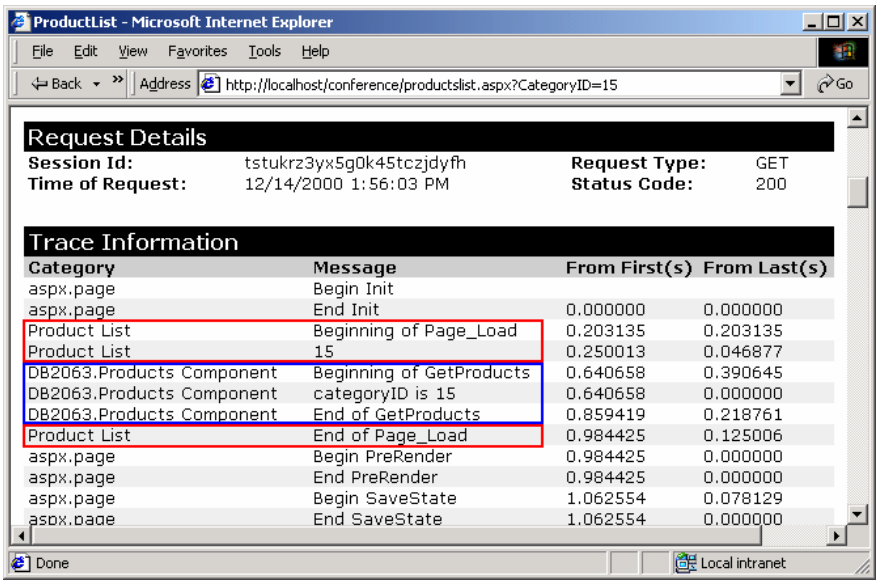
```
Dim productCatalog As New ASPNET.db2063.Products
```

---

3. View the ProductsList.aspx page in Internet Explorer, again passing the categoryID=15:

   http://localhost/ASPNET/ProductsList.aspx?CategoryID=15

   You should see your custom messages from the page and from the component in the **Trace Information** section, as shown in the following illustration.



Review

- Overview of Tracing
- Trace Information
- Page-Level Trace
- Application-Level Trace

---

1. How do you enable page-level tracing?

2. What is the difference between the **Trace.Write** and **Trace.Warn** methods?

3. What property is used to sort trace messages in a page?

4. How do you enable application-level trace?

5. If you have trace enabled in a component, but disabled in the page that calls the component, will the trace messages be displayed?

6. If you have trace enabled for a page, but neither enabled nor disabled in a component called by the page, will the trace messages from the component be displayed?