ASP.NET - ERROR HANDLING

Error handling in ASP.NET has three aspects:

- **Tracing** - tracing the program execution at page level or application level.
- **Error handling** - handling standard errors or custom errors at page level or application level.
- **Debugging** - stepping through the program, setting break points to analyze the code

In this chapter, we will discuss tracing and error handling and in this chapter, we will discuss debugging.

To understand the concepts, create the following sample application. It has a label control, a dropdown list, and a link. The dropdown list loads an array list of famous quotes and the selected quote is shown in the label below. It also has a hyperlink which has points to a nonexistent link.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="errorhandling._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

   <head runat="server">
      <title>
         Tracing, debugging and error handling
      </title>
   </head>

   <body>
      <form >

         <div>
            <asp:Label ID="lblheading" runat="server" Text="Tracing, Debuggin  and Error
Handling">
            </asp:Label>

            <br /> <br />

            <asp:DropDownList ID="ddlquotes" runat="server" AutoPostBack="True"
onselectedindexchanged="ddlquotes_SelectedIndexChanged">
            </asp:DropDownList>

            <br /> <br />

            <asp:Label ID="lblquotes" runat="server">
            </asp:Label>

            <br /> <br />

            <asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="mylink.htm">Link
to:</asp:HyperLink>
         </div>

      </form>
   </body>

</html>
```

The code behind file:

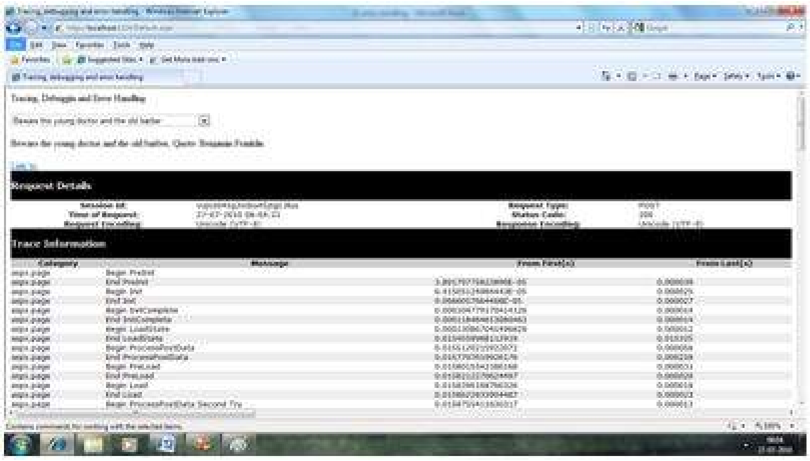
```
public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      if (!IsPostBack)
      {
         string[,] quotes =
         {
            {"Imagination is more important than Knowledge.", "Albert Einsten"},
            {"Assume a virtue, if you have it not" "Shakespeare"},
            {"A man cannot be comfortable without his own approval", "Mark Twain"},
            {"Beware the young doctor and the old barber", "Benjamin Franklin"},
            {"Whatever begun in anger ends in shame", "Benjamin Franklin"}
         };

         for (int i=0; i<quotes.GetLength(0); i++)
            ddlquotes.Items.Add(new ListItem(quotes[i,0], quotes[i,1]));
      }
   }

   protected void ddlquotes_SelectedIndexChanged(object sender, EventArgs e)
   {
      if (ddlquotes.SelectedIndex != -1)
      {
         lblquotes.Text = String.Format("{0}, Quote: {1}", ddlquotes.SelectedItem.Text,
ddlquotes.SelectedValue);
      }
   }
}
```

Tracing

To enable page level tracing, you need to modify the Page directive and add a Trace attribute as shown:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
   Inherits="errorhandling._Default" Trace ="true" %>
```

Now when you execute the file, you get the tracing information:



Tracing, Debuggin and Error Handling

Beware the young doctor and the old barber

Beware the young doctor and the old barber, Quote: Benjamin Franklin

Request Details

Session Id:	[illegible]	**Request Type:**	POST
Time of Request:	[illegible]	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	[illegible]	[illegible]
aspx.page	Begin Init	[illegible]	[illegible]
aspx.page	End Init	[illegible]	[illegible]
aspx.page	Begin InitComplete	[illegible]	[illegible]
aspx.page	End InitComplete	[illegible]	[illegible]
aspx.page	Begin LoadState	[illegible]	[illegible]
aspx.page	End LoadState	[illegible]	[illegible]
aspx.page	Begin ProcessPostData	[illegible]	[illegible]
aspx.page	End ProcessPostData	[illegible]	[illegible]
aspx.page	Begin PreLoad	[illegible]	[illegible]
aspx.page	End PreLoad	[illegible]	[illegible]
aspx.page	Begin Load	[illegible]	[illegible]
aspx.page	End Load	[illegible]	[illegible]
aspx.page	Begin ProcessPostData Second Try	[illegible]	[illegible]

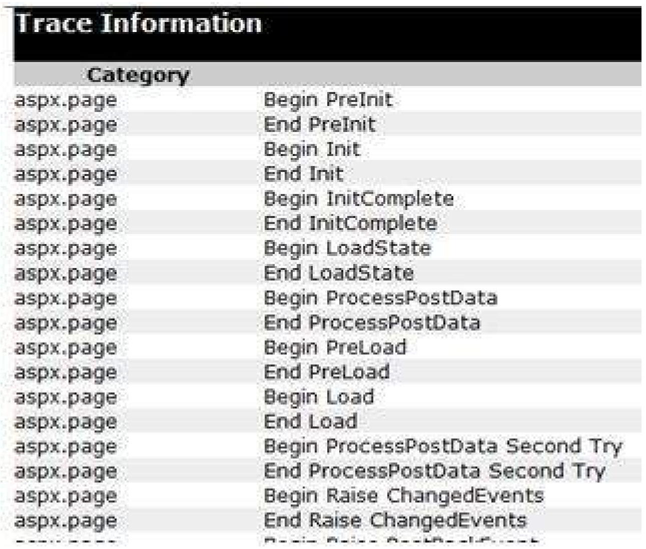
It provides the following information at the top:

- Session ID
- Status Code
- Time of Request
- Type of Request
- Request and Response Encoding

The status code sent from the server, each time the page is requested shows the name and time of error if any. The following table shows the common HTTP status codes:

Number	Description
Informational 100 − 199	
100	Continue
101	Switching protocols
Successful 200 − 299	
200	OK
204	No content
Redirection 300 − 399	
301	Moved permanently
305	Use proxy
307	Temporary redirect
Client Errors 400 − 499	
400	Bad request
402	Payment required
404	Not found
408	Request timeout
417	Expectation failed
Server Errors 500 − 599	
500	Internal server error
503	Service unavailable
505	HTTP version not supported

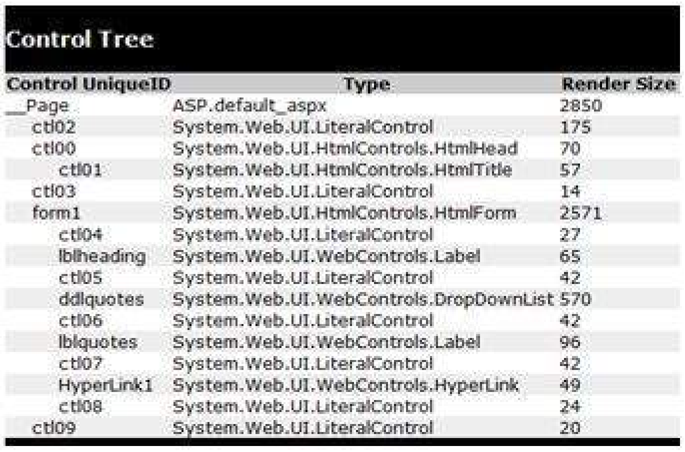
Under the top level information, there is Trace log, which provides details of page life cycle. It provides elapsed time in seconds since the page was initialized.



Trace Information

Category	
aspx.page	Begin PreInit
aspx.page	End PreInit
aspx.page	Begin Init
aspx.page	End Init
aspx.page	Begin InitComplete
aspx.page	End InitComplete
aspx.page	Begin LoadState
aspx.page	End LoadState
aspx.page	Begin ProcessPostData
aspx.page	End ProcessPostData
aspx.page	Begin PreLoad
aspx.page	End PreLoad
aspx.page	Begin Load
aspx.page	End Load
aspx.page	Begin ProcessPostData Second Try
aspx.page	End ProcessPostData Second Try
aspx.page	Begin Raise ChangedEvents
aspx.page	End Raise ChangedEvents

The next section is control tree, which lists all controls on the page in a hierarchical manner:



Control Tree

Control UniqueID	Type	Render Size
__Page	ASP.default_aspx	2850
ctl02	System.Web.UI.LiteralControl	175
ctl00	System.Web.UI.HtmlControls.HtmlHead	70
ctl01	System.Web.UI.HtmlControls.HtmlTitle	57
ctl03	System.Web.UI.LiteralControl	14
form1	System.Web.UI.HtmlControls.HtmlForm	2571
ctl04	System.Web.UI.LiteralControl	27
lblheading	System.Web.UI.WebControls.Label	65
ctl05	System.Web.UI.LiteralControl	42
ddlquotes	System.Web.UI.WebControls.DropDownList	570
ctl06	System.Web.UI.LiteralControl	42
lblquotes	System.Web.UI.WebControls.Label	96
ctl07	System.Web.UI.LiteralControl	42
HyperLink1	System.Web.UI.WebControls.HyperLink	49
ctl08	System.Web.UI.LiteralControl	24
ctl09	System.Web.UI.LiteralControl	20

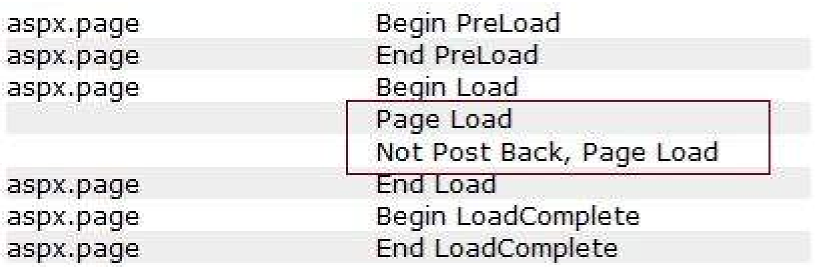
Last in the Session and Application state summaries, cookies, and headers collections followed by list of all server variables.

The Trace object allows you to add custom information to the trace output. It has two methods to accomplish this: the Write method and the Warn method.

Change the Page_Load event handler to check the Write method:

```
protected void Page_Load(object sender, EventArgs e)
{
   Trace.Write("Page Load");

   if (!IsPostBack)
   {
      Trace.Write("Not Post Back, Page Load");
      string[,] quotes =
      .......................
   }
}
```

Run to observe the effects:

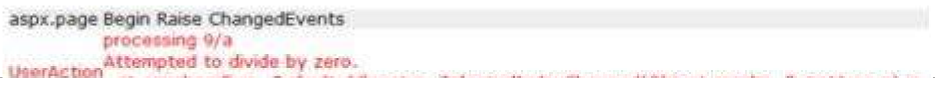


```
aspx.page                 Begin PreLoad
aspx.page                 End PreLoad
aspx.page                 Begin Load
                          Page Load
                          Not Post Back, Page Load
aspx.page                 End Load
aspx.page                 Begin LoadComplete
aspx.page                 End LoadComplete
```

To check the Warn method, let us forcibly enter some erroneous code in the selected index changed event handler:

```
try
{
   int a = 0;
   int b = 9 / a;
}catch (Exception e)
{
   Trace.Warn("UserAction", "processing 9/a", e);
}
```

Try-Catch is a C# programming construct. The try block holds any code that may or may not produce error and the catch block catches the error. When the program is run, it sends the warning in the trace log.



```
aspx.page Begin Raise ChangedEvents
          processing 9/a
UserAction Attempted to divide by zero.
```

at errorhandling._Default.ddlquotes_SelectedIndexChanged(Object sender, EventArgs e) in

Application level tracing applies to all the pages in the web site. It is implemented by putting the following code lines in the web.config file:

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

Error Handling

Although ASP.NET can detect all runtime errors, still some subtle errors may still be there. Observing the errors by tracing is meant for the developers, not for the users.

Hence, to intercept such occurrence, you can add error handing settings in the web.config file of the application. It is application-wide error handling. For example, you can add the following lines in the web.config file:

```
<configuration>
   <system.web>

      <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">
         <error statusCode="403" redirect="NoAccess.htm" />
         <error statusCode="404" redirect="FileNotFound.htm" />
      </customErrors>

   </system.web>
<configuration>
```

The <customErrors> section has the possible attributes:

- **Mode** : It enables or disables custom error pages. It has the three possible values:
  - **On** : displays the custom pages.
  - **Off** : displays ASP.NET error pages *yellowpages*
  - **remoteOnly** : It displays custom errors to client, display ASP.NET errors locally.
- **defaultRedirect** : It contains the URL of the page to be displayed in case of unhandled errors.

To put different custom error pages for different type of errors, the <error> sub tags are used, where different error pages are specified, based on the status code of the errors.

To implement page level error handling, the Page directive could be modified:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
   Inherits="errorhandling._Default" Trace ="true" ErrorPage="PageError.htm" %>
```

Because ASP.NET Debugging is an important subject in itself, so we would discuss it in the next chapter separately.