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 id="form1" runat="server">

<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);

}

}

}

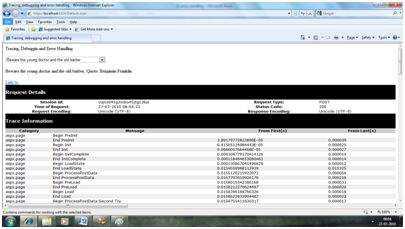
Tracing

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:



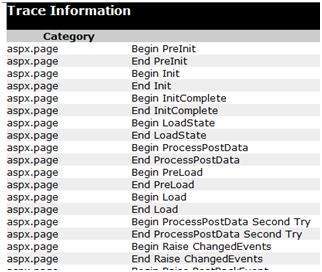
It provides the following information at the top:

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

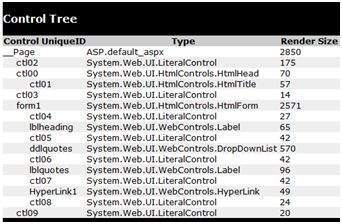
The status code sent from the server, each time the page is requested shows the name and time of error if any. The following table shows the common HTTP status codes:

|  |  |
| --- | --- |
| **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.



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



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");

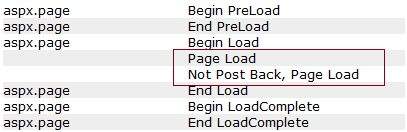
string[,] quotes =

.......................

}

}

Run to observe the effects:



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.

Tracing Info5

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 (yellow pages)
  + **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" %>