Code Repository Exercises

Error Handling and Tracing

Purpose: The purpose of this exercise is to familiarize you with Exceptions and how to handle them properly in a Web application. You will learn the purpose of try/catch/finally blocks and see how exceptions can be thrown, caught, and customized. You will also see how to implement a custom error page and enable page tracing as a diagnostic tool.

Directory Name: ErrorHandling

Instructions:

Part 1 – Error Handling

1. Create a new .aspx page called *ErrorHandling1.aspx*. This page should include two Textboxes that allow the user to enter two values, and a button that gets the values entered by the user and divides the first value by the second value, as shown in the figure on page 213 of the MacDonald book. Write the code for the code-behind file as shown on page 213. This code attempts to perform the division in a try block, and then catches any exception that may be thrown. Run the page to verify that exceptions are caught as anticipated.
2. Copy the *ErrorHandling1.aspx* page and paste it in as a new file. Rename the file *ErrorHandling2.aspx*. Change the code-behind method as follows.
    a. Create a *DivideNumbers* private method like the one shown in the second example on the bottom of page 215. This method should throw a DivideByZeroException with a custom error message if the divisor is zero.
    b. Change the line of code in the try block that says **`result = a/b`** to **`result = DivideNumbers(a, b)`**
    c. Modify the catch block so that it catches a DivideByZeroException explicitly, rather than a generic Exception.
    d. Create an additional catch block by copying the catch block and paste it in again below the existing catch block. Change the type of the exception caught in this new catch block to a FormatException (this will handle the case where the user enters a non-numeric value in either of the textboxes)
    e. Run the page to verify that your exceptions are caught as anticipated.
3. Copy the *AuthorManager_DataSource.aspx* page from your DataBinding exercise and paste it into the directory created for this exercise. Rename the file *AuthorManager_Exception.aspx*. Do the following:
    a. In the second SqlDataSource (the one that is bound to the Detailsview control), change the UpdateCommand in some way as to make the command **invalid** (e.g. change the name of the table from 'Authors' to 'Author'). This will force a failure of the update command.

b. Create an event handler method for the *Updated* event of the SqlDataSource. In this event handler method, check whether an exception has been thrown, and, if so, notify the user of the problem by setting the text of the *lblResults* label to an appropriate message. See the code shown on page 505 of the MacDonald book for an example. (Your code will do the same thing, but in the Updated event handler rather than the Selected event handler.)

Part 2 – Error Pages

1. Create a new **html** page called *Error.html* in the directory created for this exercise. Add some simple text stating that an error has occurred.
2. Configure the <customErrors> element in your web.config file (this element is likely already there, but just commented out). Set the *defaultRedirect* attribute to point to the *Error.html* page you just created (remember to include the subdirectory name). Set the *mode* attribute to "RemoteOnly"
3. Copy the *ErrorHandling1.aspx* page from part 1 and paste it in again as a new file. Rename this file to *UnhandledError.aspx*. Modify the code behind file by removing the entire catch block, including all code inside this block. Preserve the code in the try block, but remove the declaration of the try block and the braces that contain it (you are simply removing exception handling from this page.)
4. Run the *UnhandledError.aspx* page and attempt to divide by zero. You should see the standard ASP.NET error page that reports the details of the exception.
5. Change the mode attribute of the <customErrors> element to "On". Run the *UnhandledError.aspx* page again and attempt to divide by zero. You should now be redirected to the *Error.html* page you created in step 1.
6. Change the *mode* attribute of the <customErrors> element back to "RemoteOnly". This will display the detailed ASP.NET error page on the local machine, but will display your custom Error.html page to any remote user.

Part 3 – Tracing

1. Enable tracing in the *ErrorHandling2.aspx* page by setting the *Trace* attribute in the Page directive to "true"
2. Insert three Trace.Warn() statements in the code-behind file for *ErrorHandling2.aspx*:
    a. Right before the DivideNumbers() method is called in the try block to notify that the division is about to take place.
    b. First-thing inside the DivideNumbers() method to notify that division is being attempted
    c. Right after the DivideNumbers() method is called in the try block to notify that the division has completed successfully
3. Run the page and attempt a division both with a zero divisor and with a valid non-zero divisor. Note the appearance of the Trace.Warn statements in each case. Note also the other content included in the trace.

Header on Default.aspx: Error Handling and Tracing

Pages linked from Default.aspx: *ErrorHandling1.aspx*, *ErrorHandling2.aspx*, *AuthorManager_Exception.aspx, UnhandledError.aspx*