# Certificate in C# Programming

## Creating Web Applications in C#

By Randal Root

# Module 04: Error Handling in Data-Driven Web Apps

In this Module, we will see how ASP.NET **Exceptions and Debugging** work for **data-driven Web applications**. We’ll first look at the standard pattern of .NET’s structured exception handling and then look at the ways to debug those errors. This Module **corresponds to the 2nd part of the course Textbook**, Beginning ASP.NET 4.5 in C#, by Matthew MacDonald, ISBN-13: 978-1-4302-4251-2.

*“Part 2: Developing ASP.NET Applications*

*The second part of this book delves into the heart of ASP.NET programming and introduces its event-based model. In Chapter 4, you’ll take a look around the Visual Studio design environment and learn a few fundamentals about web forms, events, and HTML5. In Chapters 5 and 6, you learn how to program a web page’s user interface through a layer of objects called server controls.*

*Next you’ll explore two more essentials of ASP.NET programming.* ***Chapter 7 presents techniques for handling errors.*** *Chapter 8 describes strategies for state management. Taken together, the chapters in this part contain all the core concepts you need to design web pages and create a basic ASP.NET website.” (MacDonald, 2012, p.xxxv)*

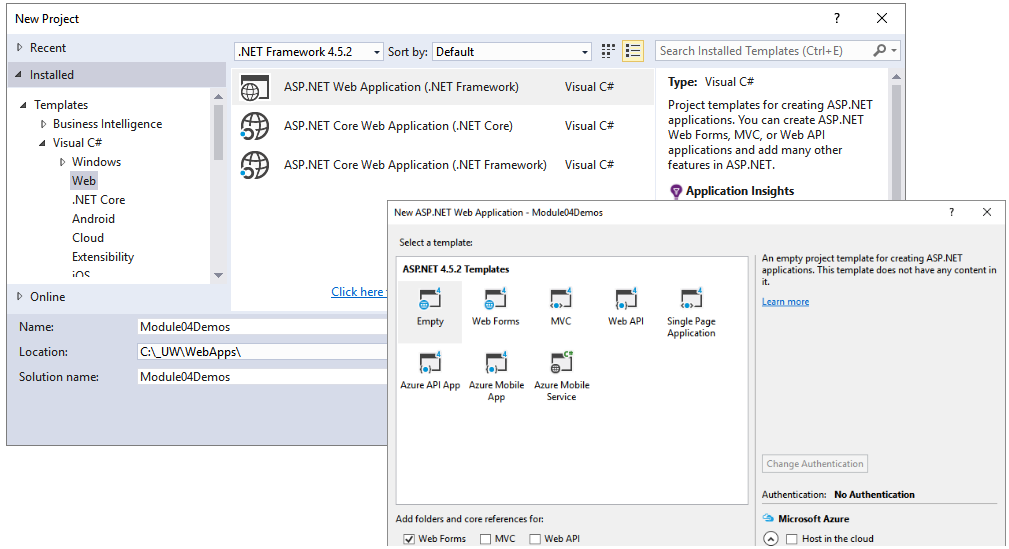
**Note:** This chapter assumes you are familiar with using ADO.NET. If you are not, please review my module05 notes for the 2nd certificate course. These notes are in the [\_NoteFilesFromCourse02.zip](https://canvas.uw.edu/courses/1303393/files/55723024/download?wrap=1) file on our Canvas site.

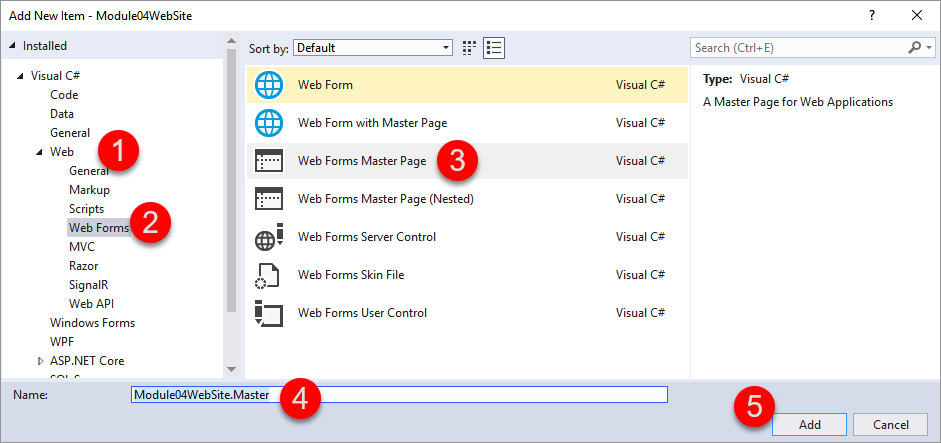
# Understanding Exception Handling

If you have had any C# classes before, you learned about using the **Try-Catch block**. This feature is precisely the **same in ASP.NET**.

“You can catch this object by using an exception handler. If you fail to use an exception handler, your code will be aborted, and the user will see an error page. If you catch the exception, you can notify the user, attempt to resolve the problem, or simply ignore the issue and allow your web page code to keep running.” (MacDonald, 2012, p. 205)

### Using Try-Catch in C#

If you are using C# to create your ASP.Net pages, the code will look similar to the following example.



**Note**: This is not the same as a "Web Form with Master Page"!

#### Module04WebSite.Master

<%@ Master Language="C#" %> <%--Removed Code Behind page--%>

<!DOCTYPE html>

<html>

<head runat="server">

<title></title>

<asp:ContentPlaceHolder ID="head" runat="server">

</asp:ContentPlaceHolder>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

</asp:ContentPlaceHolder>

</div>

</form>

</body>

</html>

With the Master page created, we make a new Content page like this one:

#### 01TryCatch.aspx

<%@ Page Title="01TryCatch" Language="C#" MasterPageFile="~/Module04WebSite.Master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<script runat="server">

protected void ServerCodeButton\_Click(object sender, EventArgs e)

{

System.Data.OleDb.OleDbConnection objOleCon = new System.Data.OleDb.OleDbConnection();

System.Data.OleDb.OleDbCommand objCmd = new System.Data.OleDb.OleDbCommand();

**try**

{ //1. Make a Connection

string strOledbConnection = @"Provider=SQLOLEDB;

Data Source=.\SQLExpress;

Integrated Security=SSPI;

Initial Catalog=Master";

objOleCon.ConnectionString = strOledbConnection;

objOleCon.Open();

//2. Issue a Command

objCmd.Connection = objOleCon;

objCmd.CommandText = "Select Count(\*) From Sys.Databases";

int intOrderCount = (int)objCmd.ExecuteScalar();

//3. Process the Results

Label1.Text += "<b>" + intOrderCount.ToString() + "</b>";

}

**catch** (Exception ex) { Label1.Text += "<b>" + ex.ToString() + "</b>"; }

**finally** { objOleCon.Close(); } //4. Run clean up code

}

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

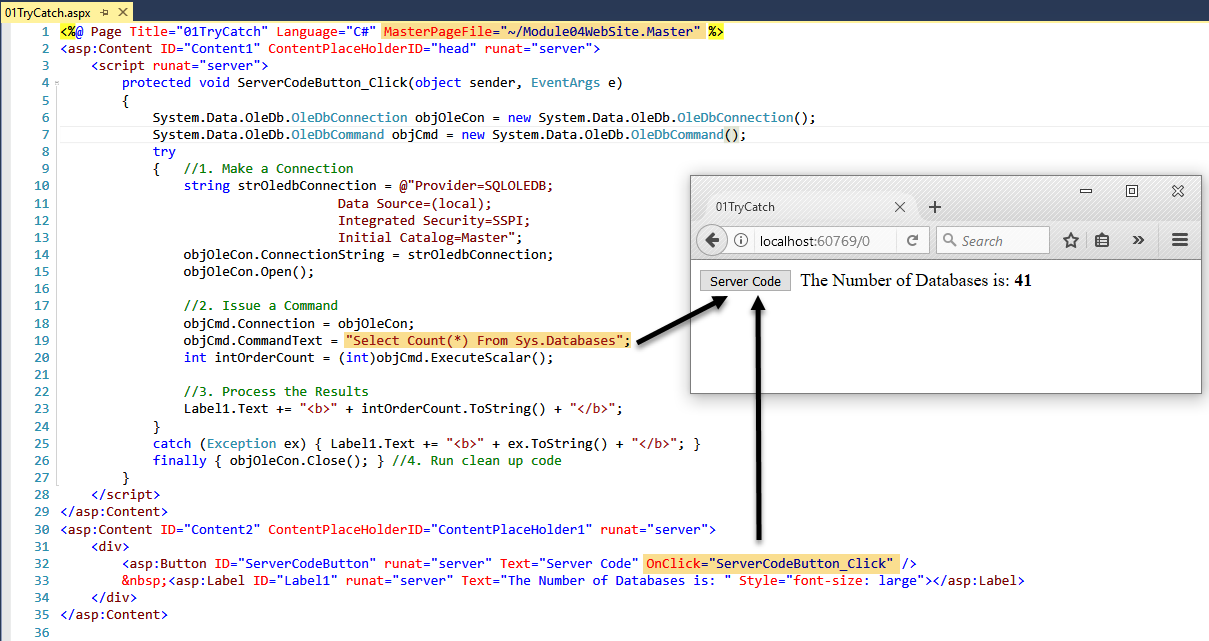
<div>

<asp:Button ID="ServerCodeButton" runat="server" Text="Server Code" OnClick="ServerCodeButton\_Click" />

&nbsp;<asp:Label ID="Label1" runat="server" Text="The Number of Databases is: " Style="font-size: large"></asp:Label>

</div>

</asp:Content>



### LAB 01: ASP.NET Server Try-Catch Code

In this lab you will:

#### LAB SETUP: Your instructor will walk you through setting up the SQL database and table using this code:

#### SQL Lab Setup Code.sql

Create Database ASPNetLab04Demos;

Go

Use ASPNetLab04Demos;

Go

CREATE TABLE [dbo].[SimpleLoginRequests](

[LoginId] [int] IDENTITY(1,1) NOT NULL,

[Name] [nvarchar](50) NOT NULL,

[EmailAddress] [nvarchar](50) NOT NULL,

CONSTRAINT [PK\_LoginRequests] PRIMARY KEY CLUSTERED

( [LoginId] ASC )

);

Go

Insert Into [SimpleLoginRequests]

([Name],[EmailAddress])

Values('Bob Smith', 'BSmith@Gmail.com');

Go

Select [LoginId],[Name],[EmailAddress]

From [SimpleLoginRequests];

Go

Select Count([LoginId]) From [SimpleLoginRequests];

Go

-- Lab 05

Select Count([LoginId]) From [SimpleLoginRequests];

Update [SimpleLoginRequests] Set Name = 'Bob' Where LoginID = 1;

Go

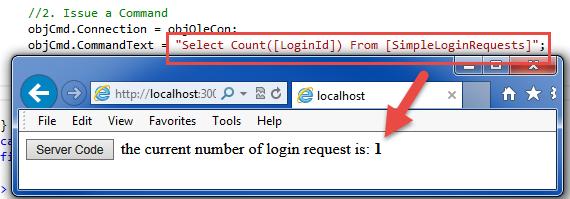
Select \* From [SimpleLoginRequests];

1) Create a new Visual Studio solution with an ASP.NET Empty Web Application project called **MYModule04Labs**.

2) Create a new Master Page called **LabSite.Master**. The code in the new master page should look like Module04WebSite.Master example you saw earlier.

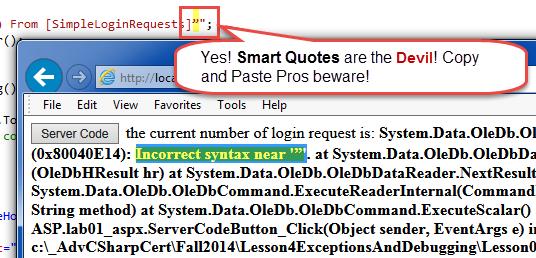
3) Create a new Content Page and call it **Lab01**.

4) Add and test code from 01TryCatch.aspx in your new content page. Then, modify it so the following results are displayed when using this SQL statement: “**Select Count([LoginId]) From [SimpleLoginRequests];**”



5) Troubleshoot any errors you have

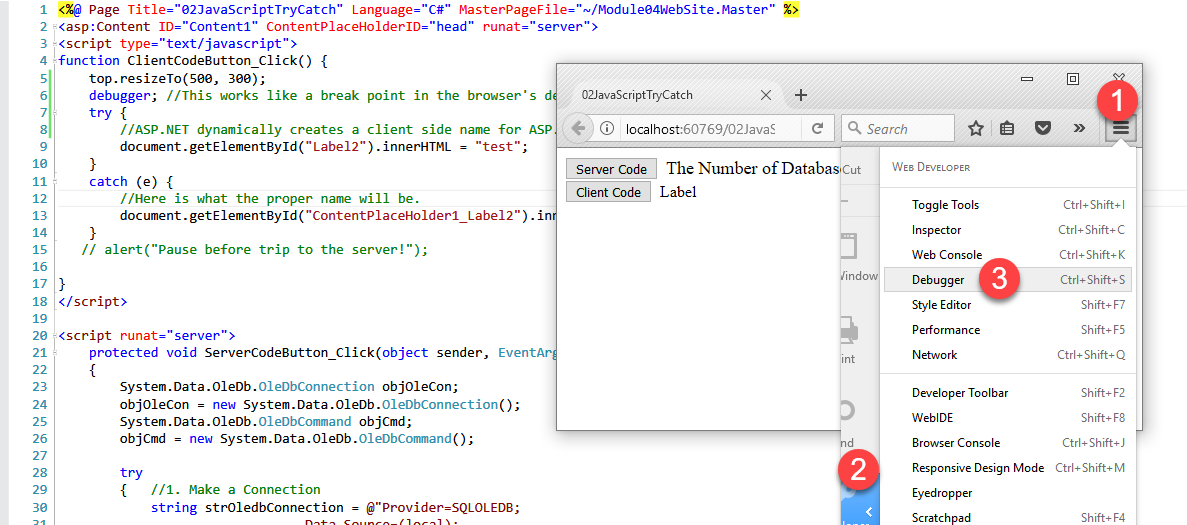
**NOTE:** Be careful whenever you copy and paste code out of MS Word, since it may add “Smart Quotes” instead of normal quotation marks. These have a different numeric value and will cause errors in your program!



**This lab should take about 5 to 10 minutes**

## Using Try-Catch in JavaScript

JavaScript also uses Try-Catch block for client-side code. You can also add the debugger command to act as a breakpoint in your JavaScript code. JavaScript can be debugged using the Browers developer tools.



#### 02JavaScriptTryCatch.aspx

<%@ Page Title="02JavaScriptTryCatch" Language="C#" MasterPageFile="~/Module04WebSite.Master" %>

<script runat="server">

protected void Button1\_Click(object sender, EventArgs e)

{

string Msg = @"Server Side Changes overwrite any Client Side Changes!

So all you see is a flash and this message.

<p><b>

Note: Use the Browser's debugging tools to test the page!

As we will see in the next section!<b></p>";

Label2.Text = Msg;

}

</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<script type="text/javascript">

function ClientCodeButton\_Click() {

debugger; //This works like a breakpoint in the browser's development tools!

try {

//NOTE: ASP.NET dynamically creates a client side name for ASP.NET controls.

document.getElementById("Label2").innerHTML = "test"; //But, this ain't it!

}

catch (e) {

//Here is what the proper name will be.

document.getElementById("ContentPlaceHolder1\_Label2").innerHTML = e.message;

}

alert("Pause before trip to the server!\n(Note how the label changes back after the PostBack)");

}

</script>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">

<div>

<asp:Button ID="Button1" runat="server" Text="Button"

OnClientClick="ClientCodeButton\_Click()" OnClick="Button1\_Click"></asp:Button>

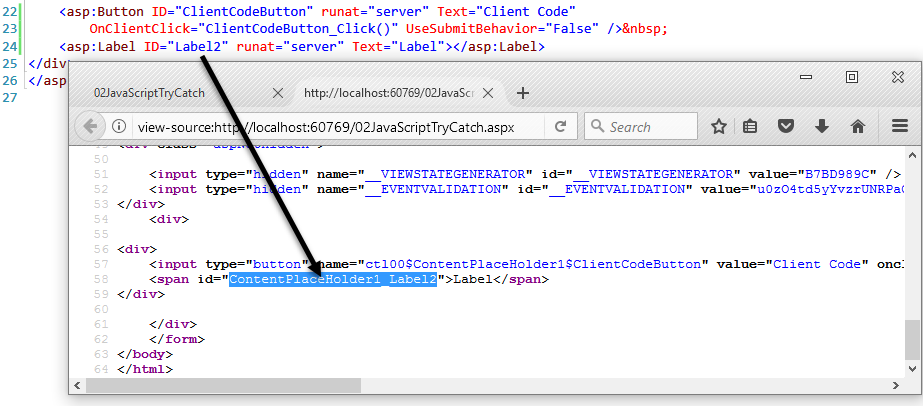
<br />

<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>

</div>

</asp:Content>

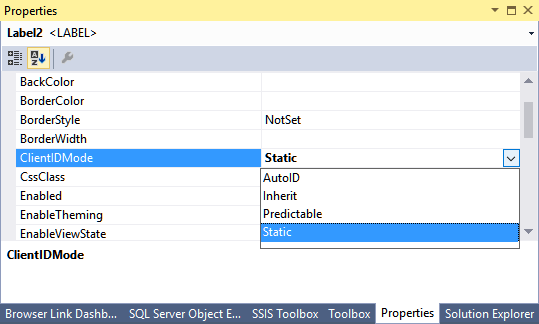
In the preceding code, there was an **intentional error** where the **server side control name** was used instead of the **client side control name**. This occurs because ASP.NET dynamically creates a client side name for ASP.NET controls. If you use the server side name in your JavaScript code, you will get an error like this one! Viewing the Source of the web page when you browse it is an easy way to figure out what the client side name is.



This behavior is controlled using the ClientIDMode **property of the control**.

<http://msdn.microsoft.com/en-us/library/system.web.ui.control.clientidmode(v=vs.110).aspx>

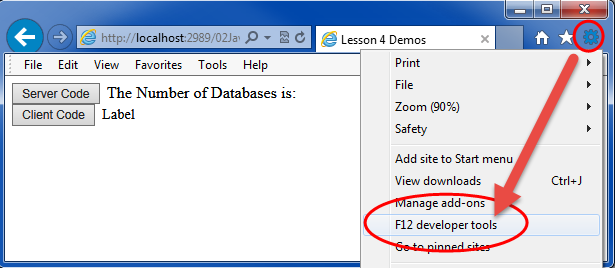
Setting it to Static will make is use the actual ID attributes value.



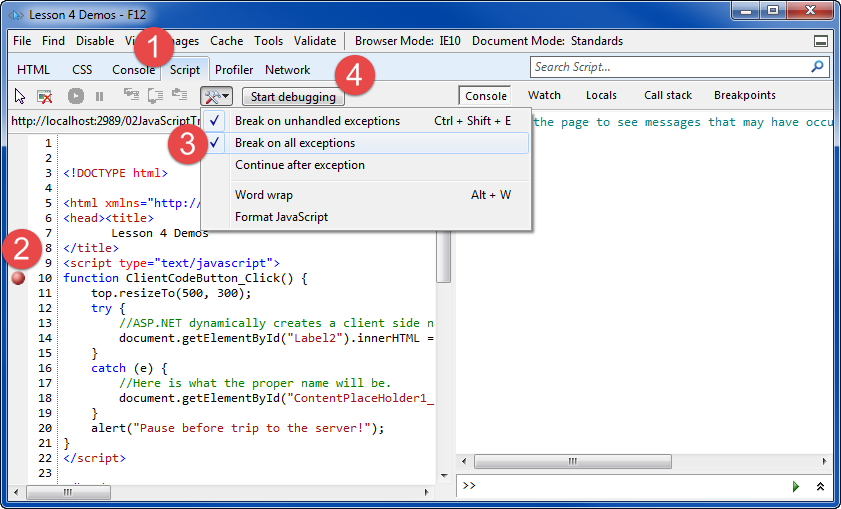
## Using A Web Browser’s Debugging Tools

Most modern browsers have Debugging tool for handling JavaScript errors. As you might guess they are not designed for debugging Server Side code (they do not have access to it).

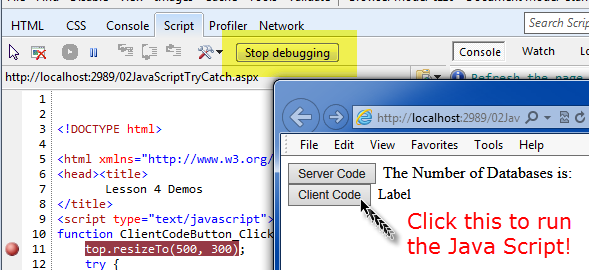
All of the major browsers have tools like these, and they all work similarly. In Microsoft’s Internet Explorer you can use the “F12” tools.



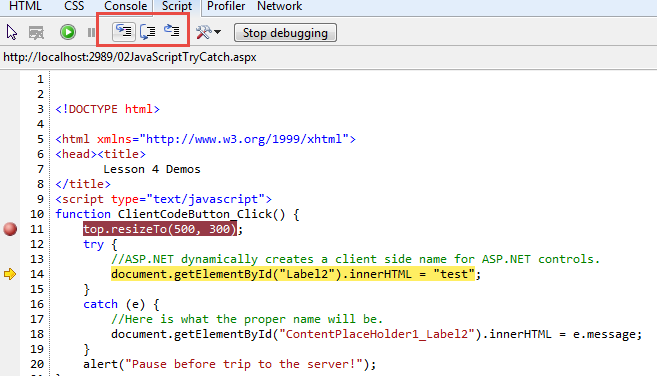
Once these open you can use the script tag to place breakpoints in the JavaScript. Then start the debugger.



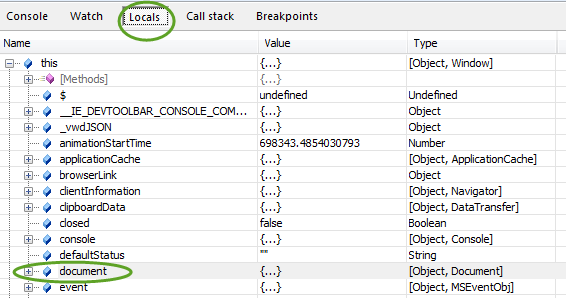
The JavaScript will not run until a function is called, even if the debugger is running. You must interact with a page so that it fires a function before debugging can begin.



Once the debugger starts, you walk through the code using the "step" commands, just as you would in Visual Studio!

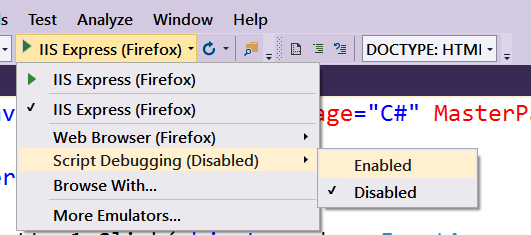


There is a lot to explore here. So, give yourself time to get familiar with these tools! For example, make sure you drill down into the Document Object Model (DOM) using the Locals window!

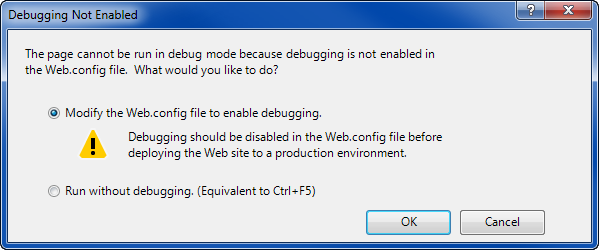


## Using Visual Studio's Debugging Tools

You can use Visual Studio to debug both Client-Side and Server-Side code. To do so, you place breakpoints in the designer, just as you would with Windows or Console application development. You must also enable it, since nowadays it is disabled by default.

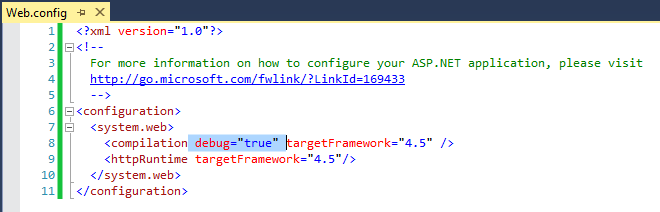


NOTE: The first time you launch the debugger, you may see this message, depending on your version of Visual Studio.

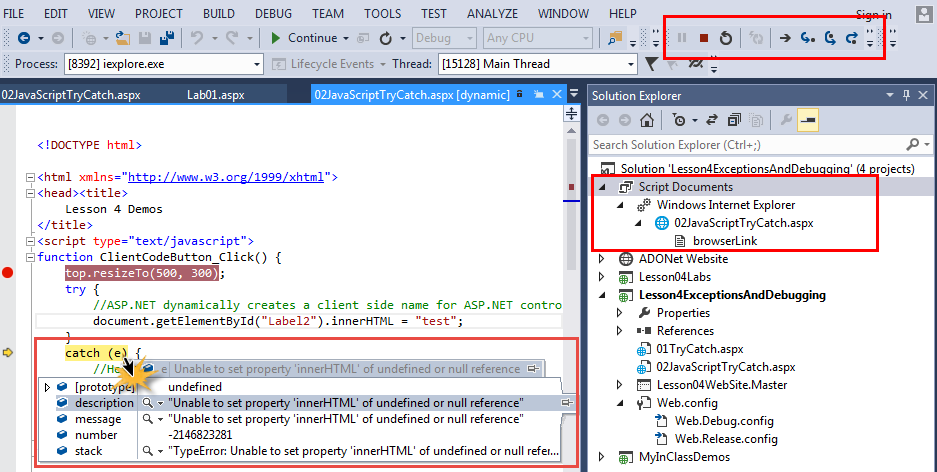


**Figure 8: Allowing modification of the Web.Config file to enable debugging**

Saying OK to this dialog will add the following setting to the web.config file.

Figure 9: The debug attribute in the Web.Config file

With that configuration in place, you can start the debugging session by pressing the F5 key, using the Debug toolbar, or using the Debug menu item.



**Figure 10: A debugging example**

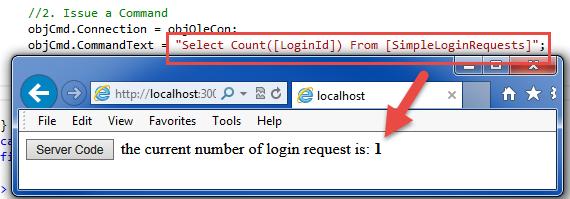
The client-side Debugging tools were made to work with **Window’s Internet Explorer (IE) only ☹.** **So, Client-Side code in Visual Studio only works with IE or Edge(partially)**.

### LAB 02: Debugging

In this lab you will:

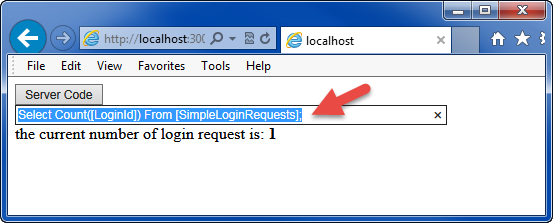
1) Create a new Content Page called **Lab02** by copying and pasting the Lab01 **page** **into** your existing **project**.

2) Verify that the page still works as it did before.



#### ***Figure 6: The Results of Lab2 after step 2***

3) Modify the **Server-Side** code so that the SQL Statement is coming from a Textbox! Then test that it still works!



#### ***Figure 7: The Results of Lab2 after step 3***

***Warning: This is a really bad Idea in real life, as we will see later***

4) Add **Client-Side** code to raise (throw) an error if the Textbox is empty!

Here is an example: <http://www.w3schools.com/js/tryit.asp?filename=tryjs_throw_error>

4a) Add code to the button to call a JavaScript function when the button is clicked.

OnClientClick=" return ValidateTextBox()"

4b) Add the following code to your page, then modify it to use the correct names for the label and textbox.

function ValidateTextBox() {

debugger;

var objLabel1 = null;

var objTextBox1 = "";

var blnReturn = true;

objLabel1 = document.getElementById("<REPLACE WITH LABLE NAME>");

objLabel1.innerHTML = "";

strText = document.getElementById("<REPLACE WITH TEXTBOX NAME>").value;

try {

if (strText == "") throw "The textbox cannot be Empty";

}

catch (ex) {

objLabel1.innerHTML = "Input Error: " + ex.toString();

blnReturn = false;// This will stop the Submit to the server

}

return blnReturn;

}

5) Test that the page works as expected.

**This lab should take about 10 to 15 minutes**

## Using Page Tracing

Microsoft includes tracing features in ASP.NET. Tracing is in some ways similar to debugging but can be turned on for a single page or the entire website. Also, tracing may work when the debugging tools of Visual Studio are not available.

“Visual Studio’s debugging tools and ASP.NET’s detailed error pages are extremely helpful when you’re testing an application. However, sometimes you need a way to identify problems after you’ve deployed an application, when you don’t have Visual Studio to rely on.” *(MacDonald, 2012, p.219)*

### The Trace Setting

Add the Trace directive to that page to enable tracing on a particular page.

<%@ Page Title="" Language="C#" MasterPageFile="~/LabSite.Master" trace="true" %>

Another option is to programmatically add it to an event handler.

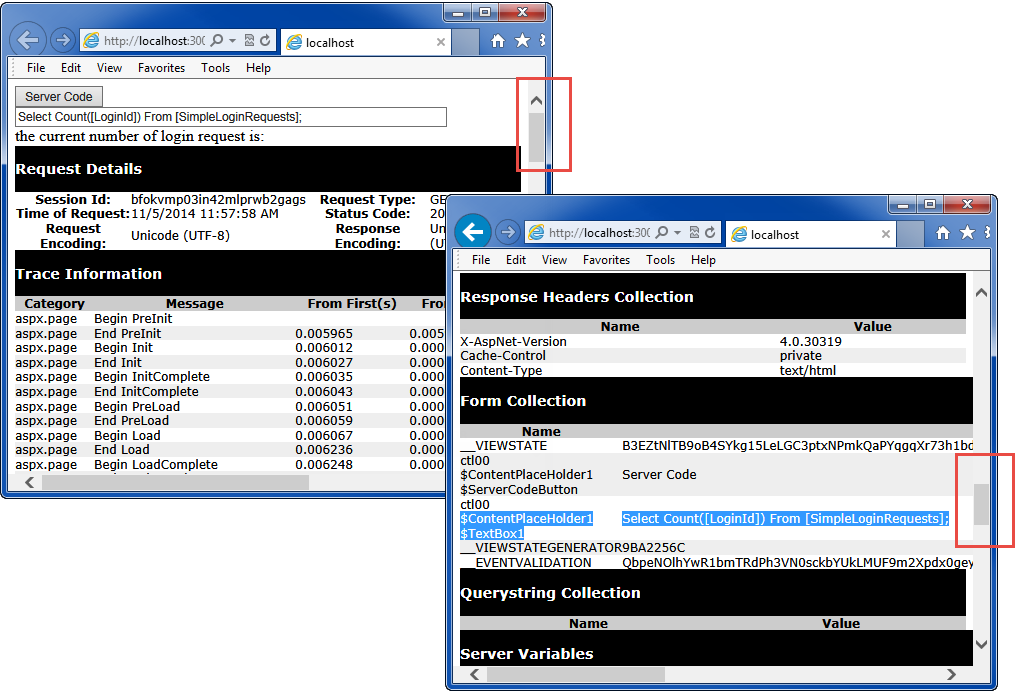
protected void Page\_Load(Object sender, EventArgs e)

{

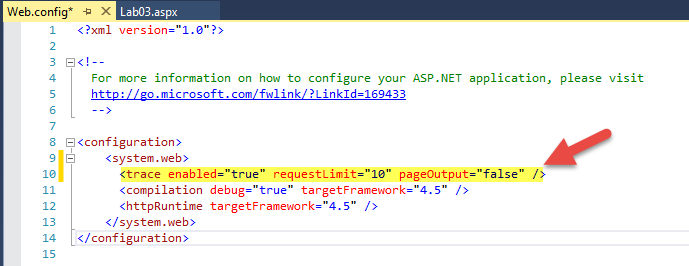
if (IsPostBack == true) Trace.IsEnabled = true;

}

Either way, your page will now return Trace information rendered at the bottom of the Web Form.

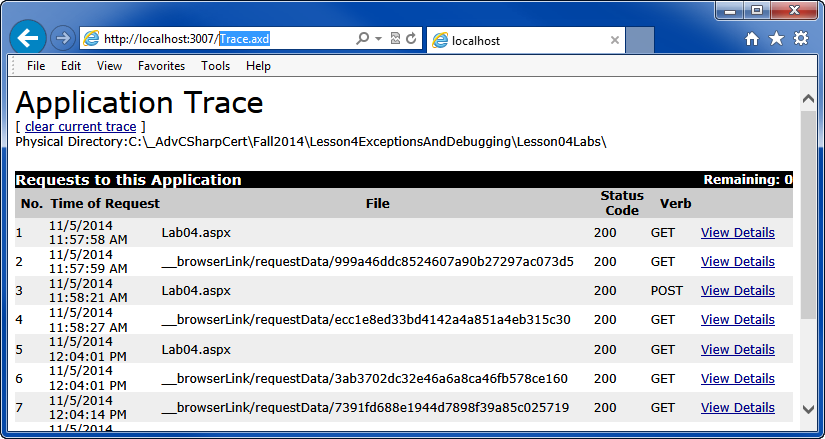


If you want to turn it on for the whole site, you change the web.config file as shown here.

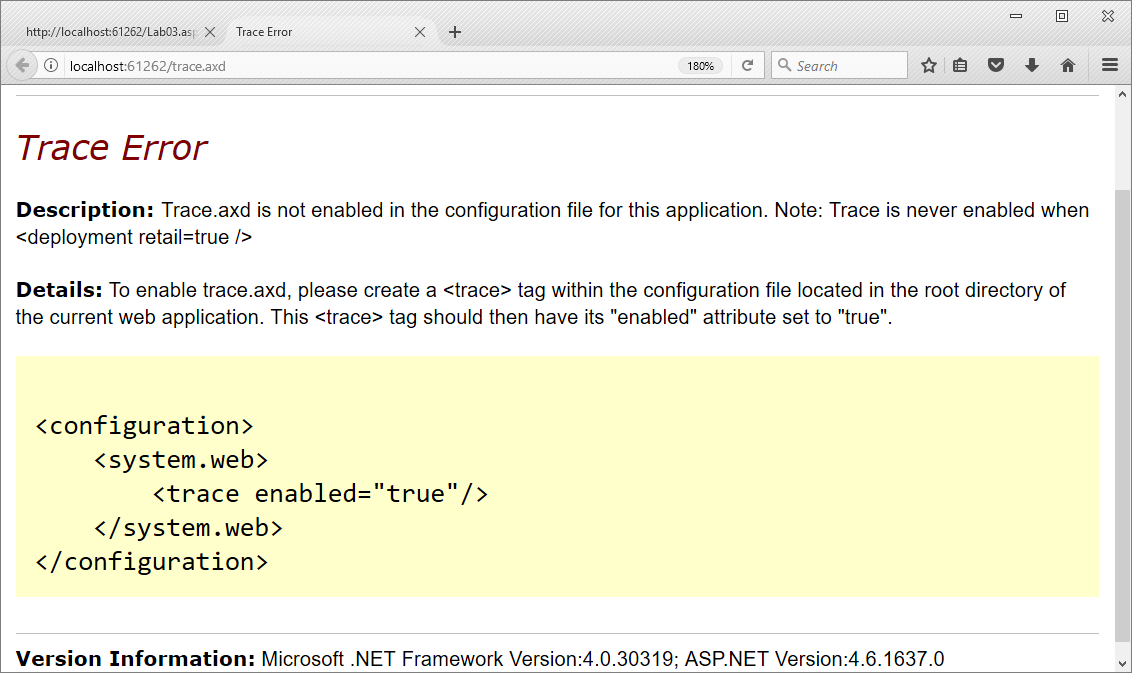


### The Trace.axd page

You can view trace information for the whole website using the Trace.axd file. This file is dynamicly created for you upon request, but it will only do this if the web.config file has the pageOutput setting at false. After that you access the page by typing in the URL as shown here:



Tracing must be turned on for a site. If you do not do so you receive this error:



### Trace after deployment

Tracing is considered a way to track what is going on with a page **after deployment**. Here is an article that talks about these differences.

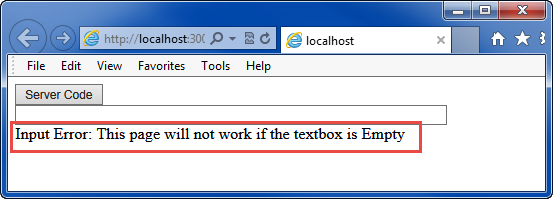
<http://www.codeproject.com/Articles/149251/Debugging-Tracing-and-Instrumentation-in-NET-and-A>

### LAB 03: Using Trace

In this lab, you will:

1) Create a new Content Page called **Lab03** by copying and pasting the Lab02 page into your existing project.

2) Verify that the page still works as it did before.



3) Add this code to the web page:

protected void Page\_Load(Object sender, EventArgs e)

{

if (IsPostBack == true)

{

Trace.IsEnabled = true;

Trace.Write("Adding a Session veriable for Student ID");

Session["StudentID"] = 1;

Trace.Warn("Student ID ( "

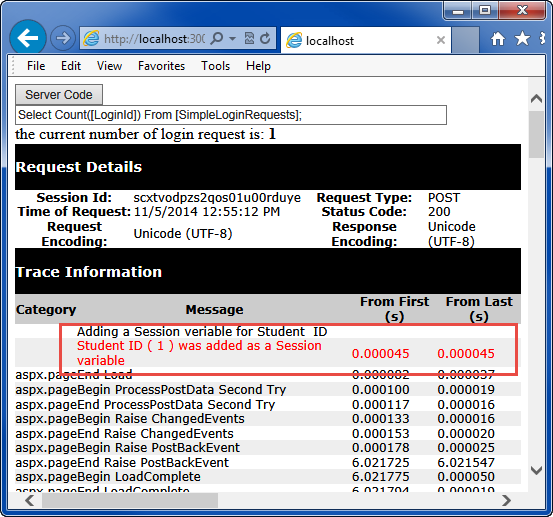
+ Session["StudentID"]

+ " ) was added as a Session variable");

}

}

4) Test that the trace information is displaying data about the Session Variable.



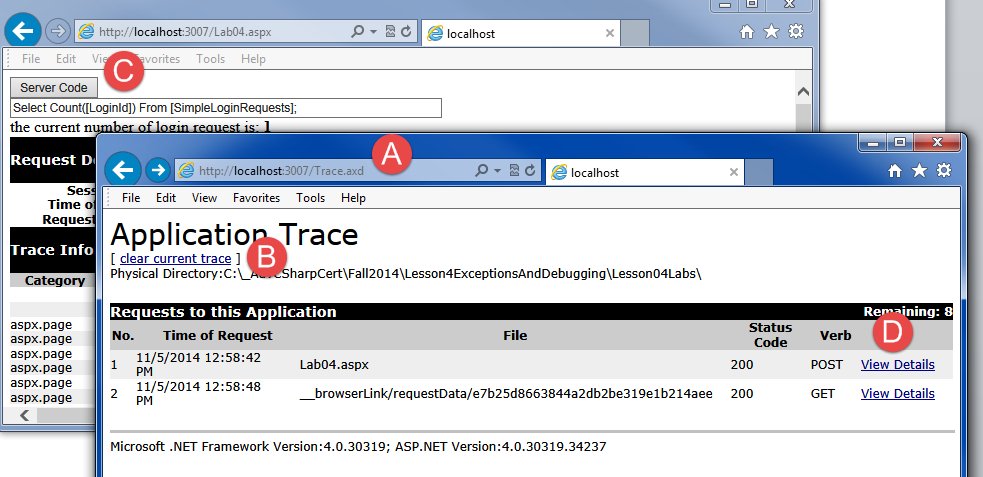
5) Now display the Trace data using the Trace.axd page.

5a) Navigate the page in a separate tab or browser window.

5b) Clear the current trace data.

5c) Send a PostBack to the server.

5d) View the details about the PostBack.



#### ***Figure 23: Trace information in the Trace.axd page***

**This lab should take about 5 to 15 minutes**

# Glimpse

MCV does not support for the Trace pages, so third-party options are used. Glimpse was/is a popular Open Source option that works with both MVC and WebForms.

*"Install Glimpse on each project you want to use it on (rather than once per server). The quickest way to get Glimpse up and running is by*[*using one of the NuGet packages*](https://github.com/Glimpse/Glimpse/wiki/Installing#installing-with-nuget)*, but you can also*[*install Glimpse manually*](https://github.com/Glimpse/Glimpse/wiki/Installing#installing-manually)*." (*[*https://github.com/Glimpse/Glimpse/wiki/Installing*](https://github.com/Glimpse/Glimpse/wiki/Installing)*, 2019)*

***Note:*** *Sadly, and UN-Officially, Microsoft bought and "Killed" Glimpse to make way for…*

# [Visual Studio Application Insights](https://azure.microsoft.com/services/application-insights) (Now Azure Monitor)

Microsoft has created its own debugging and performance system, **but it pushes you into Azure hosting to get its full benefit**. Still, it **can work "On-Premises"** too and may well turn out to be an excellent option (**one day**).

*"*[*Visual Studio Application Insights*](https://azure.microsoft.com/services/application-insights)*is an extensible analytics service that helps you understand the performance and usage of your live web application. It's designed for developers, to help you continuously improve the performance and usability of your app.*

*It works with web apps on .NET or J2EE, hosted on-premises or in the cloud. You can also apply it to Windows background services and desktop apps."* ***(https://www.visualstudio.com/en-us/docs/insights/application-insights)***

**Application Insights Home page:** <https://azure.microsoft.com/en-us/services/application-insights/>

**Watch this video**: <https://analytics.applicationinsights.io/demo#/discover/home>

**How to set it up:** <https://docs.microsoft.com/en-us/azure/application-insights/app-insights-asp-net>

# SQL Profiler

When you are working with a SQL Server database, you can use Microsoft’s SQL Profiler to trace communication between the client and the server. Here is a quote from one of my books, on this subject (Pro SQL Server 2012 BI Solutions, by Randal Root and Caryn Mason, ISBN-13: 978-1430234883).

“Measuring Performance with SQL Profiler

Microsoft provides a number of tools that can help measure performance. One very useful tool that comes with SQL Server is SQL Server Profiler. To access SQL Profiler, navigate to Start ➤ MS SQL Server 2012 ➤ Performance Tools ➤SQL Server Profiler. This starts the application.

Creating a Profiler Trace

Once SQL Profiler starts, you need to create a new SQL trace. Traces are collections of monitoring objects that track activity on your SQL Server or SSAS database engines.

The first step to creating a trace is to access the File ➤ New Trace menu option. A dialog window appears asking which database engine you want to connect to. To select the SQL Server database engine, use the “Server type” dropdown box and choose Database Engine, as shown in Figure 18-14. Use the same dropdown box to select the Analysis Server database engine.



Figure 18-14. Creating a new trace with SQL Server Profiler

After selecting which database engine you want to connect to, you are able to select the properties of the profiler trace using the properties dialog window shown in Figure 18-15. Common configurations include the trace name, a template, and whether you will save the trace data into a file or a SQL Server database table.

In our example, we have typed ReportQueryPerformance in the “Trace name” textbox. We also selected Tuning from the “Use the template” dropdown box. Once you have made the selections, click Run to start the trace.

Running a Trace

As a trace starts, you will see a new dialog window appear that displays the activity occurring on the connected database engine from the various clients. For example, if we were to create a new SSRS report and preview the results, the SELECT statement used in the SSRS report would run and be captured by the SQL trace, as shown in Figure 18-16. Information about how long it took to resolve the query is shown in the Duration column. In this example, the group was resolved in 20 milliseconds, which of course is not much time, but for larger Datasets and more complex queries, you can expect the number to increase.

****

Figure 18-15. Selecting a template for the new trace with SQL Server Profiler



Figure 18-16. An example of an SSRS report query captured with SQL Server Profiler

Database administrators can use this tool to monitor traffic between the SSIS, SSAS, and SSRS servers to look for queries that have the highest duration and target them for additional tuning. During development, there are no users with activity to track. Therefore, developers use this tool to identify the current performance of statements between the three servers and try to improve performance by making changes to the current BI solution by adding indexes or modifying queries.” (Root, 2012, p. 749)

### LAB 04: Tracing with Profiler

In this lab, you will:

1) Create a new Content Page called **Lab04** by copying and pasting the Lab03 page into your existing project.

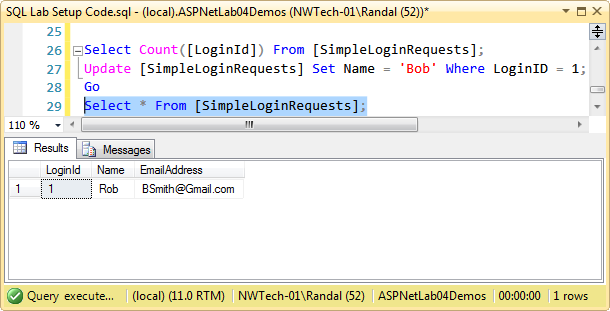
2) Verify that the page still works as it did before.

3) Modify the SQL Statement is coming from a Textbox to the following SQL Code

Select Count([LoginId]) From [SimpleLoginRequests]; Update [SimpleLoginRequests] Set Name = 'Rob' Where LoginID = 1;

***NOTE: This is an example of a SQL Injection attack!***

4) Look at the table to see if the data has changed.

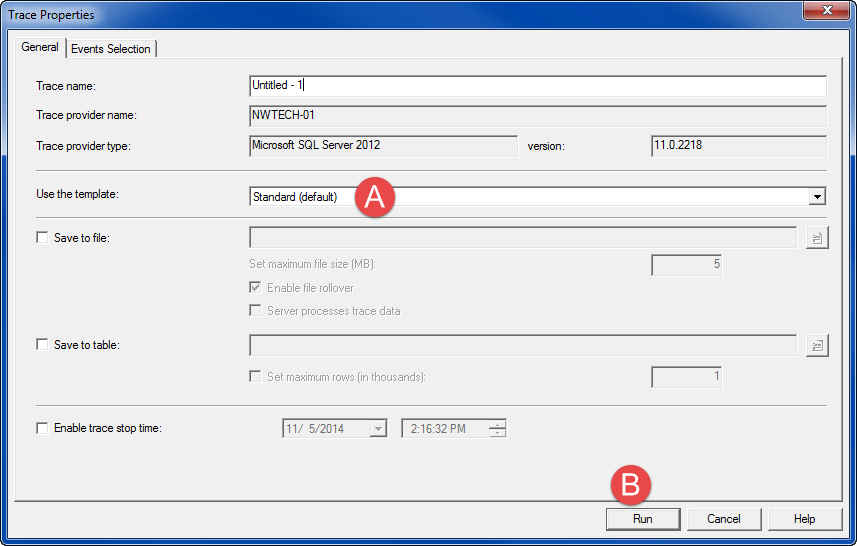


#### Figure 24: Viewing the changed data

5) Use SQL Profiler to see what is being sent from the Web Application to the Database Server.

5a) Start SQL Profiler and chose the Standard template option.

5b) Press the Run button to start the trace.



#### ***Figure 25: Staring a SQL Profiler Trace***

5c) Use the web page to send the SQL command to the database server.

5d) View the trace information in SQL Profiler.

#### C:\Users\Randal\AppData\Local\Temp\SNAGHTML63c74c2.PNG

#### ***Figure 26: Monitoring communication between IIS and SQL***

**This lab should take about 5 to 15 minutes**