TABLE OF CONTENTS

STUDENT ACTIVITY 1.1

Directions:

Match the definitions on the right to the terms on the left. Write the letter of the correct answer next to each problem.

1.	_____	External style	a. A technology developed by the W3C to separate style from content in an HTML page. A style sheet is a file that contains information about the color, font, and layout used on a website.
2.	_____	Internal style	b. All definitions are saved in a separate text file and referenced by one or more of the pages on the website.
3.	_____	Embed	c. Styles are defined within the <style> tag, which is placed in the <head> section of the HTML file.
4.	_____	CSS	d. The name of the HTML tag that is being set by this style.
5.	_____	Table	e. An element used in Web development to include graphics on a Web page.
6.	_____	Inline style	f. The process of including an external image, document, or other object within a Web page.
7.	_____	Image	g. An HTML tag that divides content into rows and data cells.
8.	_____	Property	h. A style that is included in the XHTML tag itself.
9.	_____	Selector	i. The attribute of the HTML tag that is being set by this style.
10.	_____	URL	j. The domain address of the website.

STUDENT ACTIVITY 1.2

Directions:

Review the following example from the Ramp-Up site. Use this code to create a Web page that uses the ASP.NET controls discussed in this lesson. Both Visual Basic and C# versions are included.

Example ASP.NET Web Page

The following code example shows a page that includes the basic elements that constitute an ASP.NET Web page. The page contains static text as you might have in a Hypertext Markup Language (HTML) page, along with elements that are specific to ASP.NET.

```
<%@ Page Language="VB" %>
<html>
<head runat="server">
  <title>Basic ASP.NET Web Page</title>
<script runat="server">
    Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
        Label1.Text = "Welcome, " & TextBox1.Text
    End Sub
</script>
</head>
<body>
  <form id="form1" runat="server">
    <h1>Welcome to ASP.NET</h1>
    <p>Type your name and click the button.</p>
    <p>
      <asp:TextBox ID="TextBox1"
runat="server"></asp:TextBox>
      <asp:Button ID="Button1" runat="server"
        Text="Click" OnClick="Button1_Click" />
    </p>
    <p>
      <asp:Label ID="Label1" runat="server"></asp:Label>
    </p>
  </form>
</body>
</html>
```

Form Elements

If your page includes controls that allow users to interact with the page and submit it, the page must include a **form** element. You use the standard HTML **form** element, but certain rules apply. The rules for using the **form** element are as follows:

- The page can contain only one **form** element.
- The **form** element must contain the **runat** attribute with the value set to **server**. This attribute allows you to refer to the form and the controls on the page programmatically in server code.
- Server controls that can perform a postback must be inside the **form** element.
- The opening tag must not contain an **action** attribute. ASP.NET sets these attributes dynamically when the page is processed, overriding any settings that you might make.

Web Server Controls

In most ASP.NET pages, you will add controls that allow the user to interact with the page, including buttons, text boxes, lists, and other elements. These Web server controls are similar to HTML buttons and **input** elements. However, they are processed on the server, allowing you to use server code to set their properties. These controls also raise events that you can handle in server code.

Server controls use a special syntax that ASP.NET recognizes when the page runs. The following code example shows some typical Web server controls:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server"  Text="Click"
OnClick="Button1_Click" />
```

The tag name for ASP.NET server controls starts with a prefix— in this case, asp:. The prefix might be different if the control is not part of the Microsoft .NET Framework. ASP.NET server controls also include the **runat="server"** attribute and, optionally, an ID that you can use to reference the control in server code.

When the page runs, it identifies the server controls and runs the code that is associated with those controls. Many controls render some HTML or other markup into the page. For example, the **asp:textbox** control renders an **input** element with the **type="text"** attribute into a page. However, there is not necessarily a one-to-one mapping between a Web server control and an HTML element. For example, the **asp:calendar** control renders an HTML table. Some controls do not render anything to the browser; instead, they are processed only on the server, and they provide information to other controls.

HTML Elements as Server Controls

Instead of, or in addition to, using ASP.NET server controls, you can use ordinary HTML elements as server controls. You can add the **runat="server"** attribute and an **ID** attribute to any HTML element in the page. When the page runs, ASP.NET identifies the element as a server control and makes it available to server code. For example, you can add the required elements to an HTML **body** element, as shown in the following code example:

```
<body runat="server" id="body">
```

You can then reference the **body** element in server code—for example, to set the body background color at run time in response to user input or to information from a database.

Server Code

Most ASP.NET pages include code that runs on the server when the page is processed. ASP.NET supports many languages, including C#, Visual Basic, J#, Jscript, and others.

ASP.NET supports two models for writing server code for a Web page. In the single-file model, the code for the page is in a **script** element where the opening tag includes the **runat="server"** attribute. The example earlier in this topic shows the single-file model.

Alternatively, you can create the code for the page in a separate class file, which is referred to as the *code-behind model.* In this case, the ASP.NET Web page generally contains no server code. Instead, the @ **Page** directive includes information that links the .aspx page with its associated code-behind file. The following code example shows a typical @ **Page** directive for a page with a code-behind file.

```
<%@ Page Language="VB" CodeFile="Default.aspx.vb"
Inherits="Default" %>
```

The **CodeFile** attribute specifies the name of the separate class file, and the **Inherits** attribute specifies the name of the class within the code-behind file that corresponds to the page.

Embedded Code Blocks in ASP.NET Web Pages

The default model for adding code to an ASP.NET Web page is to either create a code-behind class file (a code-behind page) or to write the page's code in a **script** block with the attribute **runat="server"** (a single-file page). The code you write typically interacts with controls on the page. For example, you can display

information on the page from code by setting the **Text** (or other) properties of controls.

Another possibility is to embed code directly into the page using an embedded code block.

Embedded Code Blocks

An embedded code block is server code that executes during the page's render phase. The code in the block can execute programming statements and call functions in the current page class.

The following code example shows an ASP.NET page with an embedded code block that displays the results of a loop:

```
<%@ Page Language="VB" %>
<html><head><title></title></head>
<body>
    <form id="form1" runat="server">
    <% For i As Integer = 0 To 5 %>
       <% Response.Write("<br>" & i.ToString())%>
    <% Next%>
    </form>
</body>
</html>
```

The following code example shows an embedded code block that displays the value of a public GetTime() function inside a span element. In embedded code blocks, the syntax <% = expression %> is used to resolve an expression and return its value into the block.

```
<%@ Page Language="VB" %>
<html>
<head>
<script runat=server>
Protected Function GetTime() As String
    Return DateTime.Now.ToString("t")
End Function
</script>
</head>
<body>
    <form id="form1" runat="server">
       Current server time is <b><% =GetTime()%></b>.
    </form>
</body>
</html>
```

Embedded code blocks must be written in the page's default language. For example, if the page's @ **Page** directive contains the attribute language="VB", the page will use the Visual Basic compiler to compile code in any script block marked with runat="server" and any inline code in <% %> delimiters

Uses for Embedded Code Blocks

Embedded code blocks are supported in ASP.NET Web pages primarily to preserve backward compatibility with older ASP technology. In general, using embedded code blocks for complex programming logic is not a best practice because when the code is mixed on the page with markup, it can be difficult to debug and maintain. In addition, because the code is executed only during the page's render phase, you have substantially less flexibility than with code-behind or script-block code in scoping your code to the appropriate stage of page processing.

Some uses for embedded code blocks include:

Setting the value of a control or markup element to a value returned by a function, as illustrated in the preceding example.

Embedding a calculation directly into the markup or control property.

STUDENT ACTIVITY 1.4

Directions:

Arrange the following events in the order that they occur. Indicate the order by placing a number in the space provided.

Order	Page Event
	Page_Load Complete
	Page_PreLoad
	Page_SaveStateComplete
	Page_Load
	Page_Init
	Page_PreRender
	Page_PreInit
	Page_InitComplete
	Control events
	Page_Render
	Page_Unload

STUDENT ACTIVITY 1.6

Directions:

1. Visit the following link and answer the following questions.
   http://support.microsoft.com/kb/815179#3
   A. What type of file is a Web.config file? _____
   B. List two tags that are necessary for the file to function properly within an ASP.NET application. _____
   C. True or False: A Machine.config file contains settings that override the Web.config file. _____
2. Open Visual Studio and create a new ASP.NET project.
3. Add configuration settings to the Web.config file. Most ASP.NET applications come with a prebuilt Web.config file that can be edited with any text editor such as Notepad. Generally, Web.config files contain comments that make editing the file self-explanatory. However, you may have to add configuration items to a Web.config file that does not already have the configuration item defined. To add a standard configuration item to a Web.config file, follow these steps:
   A. Open the Machine.config file in a text editor such as Notepad. The Machine.config file is located in the `%SystemRoot%\Microsoft.NET\Framework\%VersionNumber%\CONFIG\` directory of the 2.0 version installation of the .NET framework. Versions 3.0 and 3.5 are built on 2.0 version installation and use its Machine.config file rather than having their own. Check the documentation if using a different version of .NET. In the Machine.config file, locate the configuration setting that you want to override with your application's Web.config file. Because the Machine .config file must be a well-formed XML document, the elements must be defined using an opening tag (*<element_name>)* and a closing tag (*</element_name>), unless the element is a self-closing tag.* If the element is defined as a self-closing tag, it will look similar to *<element_name attribute1="option" attribute2="option" />*. Any white space is ignored. Therefore, the element may span multiple lines. Often, elements of confi*guration files* may are preceded *by a comment to help describe the element's use and possible values. C*omments are contai*ned inside <!-- and* --> markings. The comments for the Machine.config file are usually included in a separate file named *Machine.config.comments The <trace> configuration eleme*nt example that follows, is an example of a self-closing element tag, has

multiple attributes, spans multiple lines, and is preceded by a comment to explain its purpose and use.

```
<!--
trace Attributes:
enabled="[true|false]" - Enable application tracing
localOnly="[true|false]" - View trace results from localhost
only
pageOutput="[true|false]" - Display trace ouput on
individual pages
requestLimit="[number]" - Number of trace results available
in trace.axd
traceMode="[SortByTime|SortByCategory]" - Sorts trace result
displays based on Time or Category
-->
    <trace
enabled="false"
localOnly="true"
pageOutput="false"
requestLimit="10"
traceMode="SortByTime"
/>
```

B. Copy the whole configuration element and any associated comments to the clipboard.

4. Determine the element's nested location within the Machine.config file.
   The Machine.config file is hierarchical, and the configuration elements must be nested properly within other elements. When you copy a configuration element from the Machine.config file to the Web.config file, you must nest that configuration element in the same element that it was copied from. To determine the element of the Machine.config file that the configuration element is contained in, scroll up in the Machine.config file until you find an element that is opened, not closed. The containing element is simple to identify because higher-level elements are typically shown with less indentation.
   Most ASP.NET configuration items are contained in the `<system.web></system.web>`
   Note the element that your configuration element is contained in. You must paste that element in the same element in the Web.config file. A configuration element may be nested in multiple elements. You must create all higher-level elements in the Web.config file.
5. Close the Machine.config file, and then use your text editor to open the Web.config file in the root directory of your ASP.NET application.
6. Paste the configuration element between the beginning and the end of the element that you identified in step 4.
   For example, if the configuration item is contained in the <*system.web*> element, the configuration item must be pasted immediately after the opening line of the **<***system.web***>** element and before the **</***system.web***>** closing line.

7. Modify your Web application's Web.config file to override the server's Machine.config file's configuration element which you chose in step 'b' to override.Modify the setting for that element in your Web application's Web.config file. This setting now applies to the folder that contains the Web.config file and all its subfolders.

STUDENT ACTIVITY 2.1

Directions:

1. Find the error:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
      <to>Jane</to>
      <from>Tammy</Ffrom>
      <heading>Reminder</heading>
      <body>Don't forget to do your homework for English
tomorrow!</body>
</note>
```

2. Create a simple XML document using notepad to keep track of your CD collection. You want to keep track of the following information for each CD: Title, Artist, Year, Number of Tracks, Cost. Include data for at least five CDs in your document. Save the file as CD_Catalog.xml

3. Validate an XML document:

   A. Visit the following website:

   http://www.w3schools.com/XML/xml_validator.asp

   B. Following the instructions for validating an XML page using the XML document you created in problem 2 of this activity.

   C. A message will indicate the result of the validation process. Correct as necessary.

   D. Remove some of the closing tags or misspell a tag and record the results of the validation process for each introduced error.

4. Use this html page to test your new CD Collection xml file. Make sure both file are in the same directory.

```
<html>
<body>
<script type="text/javascript">
var xmlDoc=null;
if (window.ActiveXObject)
{
// code for IE
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
}
else if (document.implementation.createDocument)
{
// code for Mozilla, Firefox, Opera, etc.
xmlDoc=document.implementation.createDocument("","",null);
}
else
```

```
{
alert('Your browser cannot handle this script');
}
if (xmlDoc!=null)
{
xmlDoc.async=false;
xmlDoc.load("cd_catalog.xml");
var x=xmlDoc.getElementsByTagName("cd");
document.write("<table border='1'>");
document.write("<thead>");
document.write("<tr><th>Artist</th><th>Title</th></tr>");
document.write("</thead>");
document.write("<tfoot>");
document.write("<tr><th colspan='2'>This is my CD
collection</th></tr>");
document.write("</tfoot>");
for (var i=0;i<x.length;i++)
{
document.write("<tr>");
document.write("<td>");
document.write(x[i].getElementsByTagName("artist")[0].childN
odes[0].nodeValue);
document.write("</td>");
document.write("<td>");
document.write(x[i].getElementsByTagName("title")[0].
childNodes[0].nodeValue);
document.write("</td>");
document.write("</tr>");
}
document.write("</table>");
}
</script>
</body>
</html>
```

5. Use the links below to review the process of creating an XML document and using it in conjunction with the Web services listed in the Lesson Objective:

   http://quickstarts.asp.net/QuickStartv20/howto/doc/Xml/OverviewofXML.aspx

   http://msdn.microsoft.com/en-us/data/bb190600.aspx

   http://msdn.microsoft.com/en-us/library/aa468547.aspx

STUDENT ACTIVITY 2.2

Directions:

Part 1

1. Open Visual Studio and create a new ASP.NET website project.
2. Complete the following activity for creating a DataSet at the following link:

   http://msdn.microsoft.com/en-us/library/04y282hb%28VS.80%29.aspx

Part 2

1. Open Visual Studio and create a new ASP.NET website project.
2. Complete the following activity demonstrating the use of DataReader at the following link:

   http://support.microsoft.com/kb/310107

Part 3

1. Reflect on the two activities and summarize the key aspects of each.
2. Describe how each example demonstrates these points.

Use these sites for additional information on DataSet and DataReader:

http://msdn.microsoft.com/en-us/library/haa3afyz.aspx

http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx

http://msdn.microsoft.com/en-us/magazine/cc188717.aspx#S4

STUDENT ACTIVITY 2.3

Directions:

Part 1: Write a simple .asmx Web service

1. Open Visual Studio 2008.
2. Create a new ASP.NET Web service project. Name the Web service MathService and point the location to an appropriate Web server that is running ASP.NET if necessary.
3. Change the name of the Solution file to MathService for consistency.
4. Change the name of the default Web service that is created from Service1.asmx to MathService.asmx.
5. At this point, the code will be displayed. You can change the name of the class from Public Class Service1 to Public Class MathService, but you must also change the name of the class referenced in the asmx file in its markup view.
6. Define methods that encapsulate the functionality of your service. Each method that will be exposed from the service must be flagged with a WebMethod attribute in front of it. Without this attribute, the method will not be exposed from the service.

   Note: Not every method needs to have the WebMethod attribute. It is useful to hide some implementation details called by public Web service methods or for the situation in which the WebService class is also used in local applications. A local application can use any public class, but only WebMethod methods can be remotely accessed as Web services.
7. Add the following methods to the MathServices class that you just created. These methods must go inside the class body and replaced any placeholder method that already exists, such as Hello World.(This example is using Visual Basic.)

```
<WebMethod()> Public Function Add(a As Integer, b As Integer)
As Integer
   Return(a + b)
End Function
<WebMethod()> Public Function Subtract(A  As System.Single, B
As System.Single) As System.Single
       Return A - B
End Function
<WebMethod()> Public Function Multiply(A As System.Single, B
As System.Single) As System.Single
       Return A * B
End Function
<WebMethod()> Public Function Divide(A As System.Single, B As
System.Single) As System.Single
```

```
If B = 0
  Return -1
End If
Return Convert.ToSingle(A / B)
End Function
```

8. Select Build on the Build menu to finalize the Web service.
9. Browse to the MathService.asmx Web service page to test the Web service. If you set the local computer to host the page, the URL is *http://localhost/MathService/MathService.asmx*

   The ASP.NET runtime returns a Web Service Help Page that describes the Web service. This page also enables you to test different Web service methods.

Part 2: Consume a Web service

1. Start Visual Studio 2008.
2. Create a new Console application project. Select either VB or C#.
3. Add a reference for the MathService Web Service to the new Console application.
   This step creates a proxy class on the client computer. After the proxy class exists, you can create objects based on the class. Each method call that is made with the object then goes out to the uniform resource identifier (URI) of the Web service (usually as a SOAP request).
   A. In the Solution Explorer, right click on References, choose 'Add Service Reference'. From the dialog box, choose 'Advanced', and then click the 'Add Web Reference' button at the bottom of the dialog box. In the Add Service Reference dialog box, type the URL for the Web service in the Address text box and press Enter. If you set the local computer to host *the Web service, the URL is* *http://localhost/MathService/MathService.asmx*
   B. Click Add Reference.
   C. Expand the Web References section of Solution Explorer and note the namespace that was used.
4. Create an instance of the proxy object that was created. Place this code in the Main procedure of the Module1 module:

```
Dim myMathService As localhost.MathService = New
localhost.MathService()
```

5. In the main method of your console application code, invoke a method on the proxy object created in the previous step, such as:

```
Console.Write("2 + 4 = {0}", myMathService.Add(2,4))
```

6. Save the project. Compile and run the application.

STUDENT ACTIVITY 2.4

Directions:

1. Visit the Web tutorials listed below to review DataSource controls.
   - Displaying Data with the ObjectDataSource
     http://www.asp.net/learn/data-access/tutorial-04-vb.aspx
   - Demonstration: XmlDataSource
     http://quickstarts.asp.net/QuickStartv20/aspnet/doc/ctrlref/data/xmldatasource.aspx
   - Example: Using SqlDataSource
     http://www.asp.net/%28S%28pdfrohu0ajmwt445fanvj2r3%29%29/learn/data-access/tutorial-47-vb.aspx
   - Example: Using LinqDataSource
     http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.linqdatasource.aspx
2. Write a one- to two-sentence summary describing the features of each DataSource control.

STUDENT ACTIVITY 2.6

Directions:

Review the information and example from the following link. Summarize your learning by answering the following questions.

"Walkthrough: Data Binding to a Custom Business Object"
http://msdn.microsoft.com/en-us/library/1se6685s.aspx

Questions:

1. What data is used by the component in the example presented?

2. What is important to remember with respect to the *filePath* variable?

3. What is important to remember when naming the *SelectMethod* property?

4. Describe how you would create an App_Code folder.

As time permits or as homework, review the information and examples from the following link.

"Binding to Data Using a Data Source Control"
http://msdn.microsoft.com/en-us/library/ms228089.aspx

STUDENT ACTIVITY 3.1

Directions:

1. Create a new Visual Studio ASP.NET website project.
2. Use that project to complete the activity described at the following link: *http://support.microsoft.com/kb/316726*
3. Use that same project to complete the activity for ASP.NET AJAX Debugging and Tracing at *http://www.asp.net/ajax/documentation/live/overview/aspnetajaxdebuggingandtracingoverview.aspx*

Questions:

1. What attribute-value pair must be added to the page directive at the top of the Code window to turn on the tracing functionality?

2. What does page-level tracing do?

3. What statements are placed throughout the code to track variable values and execution paths during the debugging of a traditional Active Server Pages (ASP) application?

4. Identify two locations from which a Web application can be configured to allow tracing.

Now you will review how to use tracing in a website. Start by opening either Visual Studio or Microsoft Visual Web Developer. Explore the following websites and answer the questions about each.

PART 1—ENABLE TRACING FOR AN ASP.NET PAGE

HTTP://MSDN.MICROSOFT.COM/EN-US/LIBRARY/94C55D08.ASPX

Questions:

1. What file was updated with the “@ Page” directive? Where is the new code located in the file?

2. What specific code is used?

Part 2—Enable Tracing for an ASP.NET Application
http://msdn.microsoft.com/en-us/library/0x5wc973.aspx
Questions:

1. What file was used in this example to enable application tracing?

2. What is the specific code used to enable tracing in an application?

3. What must be done to make the trace information appear at the end of the page that is associated with it?

Part 3—How to: View ASP.NET Trace Information with the Trace Viewer
http://msdn.microsoft.com/en-us/library/wwh16c6c.aspx
Questions:

1. Describe how you would view trace details for a specific website request.

2. How would you clear requests from the trace viewer?

STUDENT ACTIVITY 4.1

Directions:

1. View the video: *http://www.asp.net/learn/videos/video-275.aspx*
2. Answer the following questions:
   a. Explain how the hidden field changes its value after clicking the Get Data button in the first portion of the video.

   b. What advantages does the developer get when the JavaScript code is moved to an external JavaScript file?

   c. Explain how the developer is using the ClientScriptManager class?

   d. Download the VB or C# version of the code. Run the programs making changes and setting breakpoints to test your understanding of client-side scripting. Record your steps, experiences, and problems.

STUDENT ACTIVITY 4.2

Directions:

Explore the tutorial at *http://www.asp.net/learn/Ajax/tutorial-01-cs.aspx* using Microsoft Visual Web Developer *or the Express version* to complete the *tutorial* activity. Answer the following questions.

Questions:

1. List problems with server-side processing that AJAX addresses.

2. List some of the benefits of using AJAX to do partial- page updates.

3. What is the role of the *ScriptManager* control?

STUDENT ACTIVITY 5.1

Directions:

The following section contains tutorials that demonstrate the use of Windows authorization (Part 1), and Forms authentication (Part 2). You may choose one or both, depending on the areas in which you want more practice. Work on the activities as class time allows, and complete outside of class as needed.

Part 1—Windows Authorization

1. Open Visual Studio and create a new ASP.NET website project.
2. Complete the following activity in either Visual Basic or C#:
   Windows Authorization (Visual Basic version)
   http://www.asp.net/learn/mvc/tutorial-18-vb.aspx

Part 2—Forms Authorization

1. Open Visual Studio and create a new ASP.NET website project.
2. Complete the following activity in either Visual Basic or C#:
   Forms Authorization (Visual Basic version)
   http://www.asp.net/learn/security/tutorial-02-vb.aspx

Once you have completed your work, print the code from the Web.config file as the result of your work.

STUDENT ACTIVITY 5.2

Directions:

View the video *How Do I Determine Whether to Use a Web Site or a Web Application Project* (http://www.asp.net/learn/videos/video-410.aspx). Complete the chart below to demonstrate your understanding of the differences and similarities between Visual Studio Web Site and Web Application Projects.

Attribute	**Visual Studio Web Site Project Type**	**Visual Studio Web Application Project Type**
Historical background		
Creation process		
Folder arrangement		
Namespaces		
Classes		
Master Pages		
Referencing a user control		
Compiling		
Previewing site		
When this is the best choice	Best when:	Best when:

STUDENT ACTIVITY 5.3

Directions:

Using Microsoft Visual Web Developer or the Express version, complete the following activity to review deploying a Web Application in Part 1, and creating a website using IIS in Part2. Follow the link below to the tutorial. Then answer the questions below on a separate piece of paper. You may do either the Visual Basic or C# version. Note: you do not have to perform the steps; just review the process.

Part 1: Deploying a Web Application

Visual Basic Version—*http://www.asp.net/learn/hosting/tutorial-04-vb.aspx*

Reflection:

Summarize or outline the steps needed to deploy a Web application, as described in the tutorial.

Part 2: Creating a Website Using IIS

http://technet.microsoft.com/en-us/library/cc772350%28WS.10%29.aspx

Reflection:

Summarize or outline the steps to set up a website using Internet Information Services (IIS), as described in the tutorial.

STUDENT ACTIVITY 5.4

Directions to the student:

Review the steps to create application pools. Start by visiting the link to review the tutorial. Then answer the following questions.

CREATING APPLICATION POOLS

http://www.iis.net/ConfigReference/system.applicationHost/applicationPools

1. Read the “Overview” section on the page.
2. Go to the section called “How to create a new application pool” and continue reviewing from there.