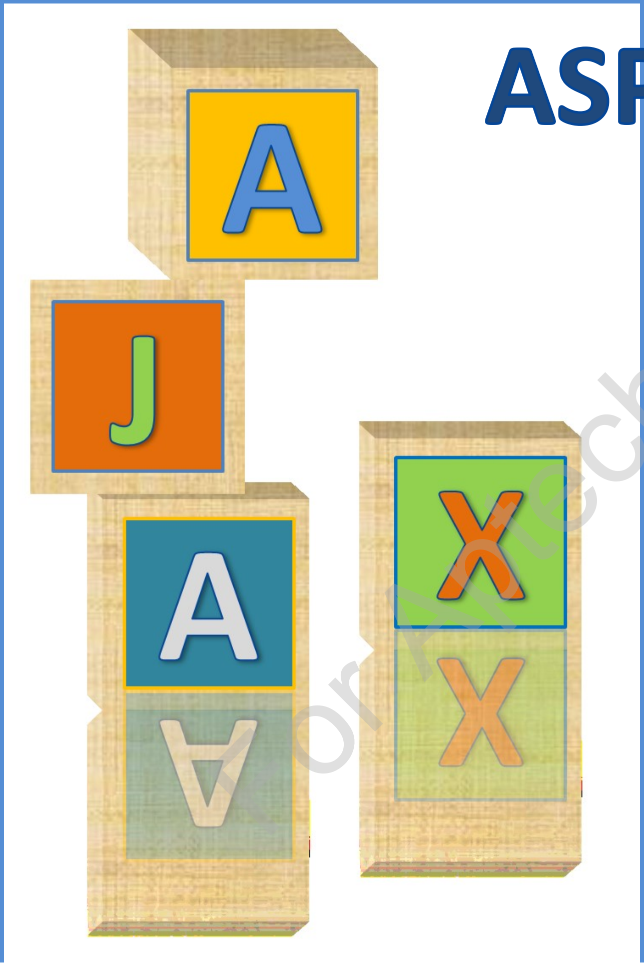
Programming ASP.NET AJAX

Session: 6



ASP.NET AJAX Remote Method Calls

Objectives

- Explain the concept of page methods and the ASP.NET AJAX Web Services
- Describe the process of defining page methods and process of calling remote methods
- Explain Web Services and asynchronous communication

Page Methods

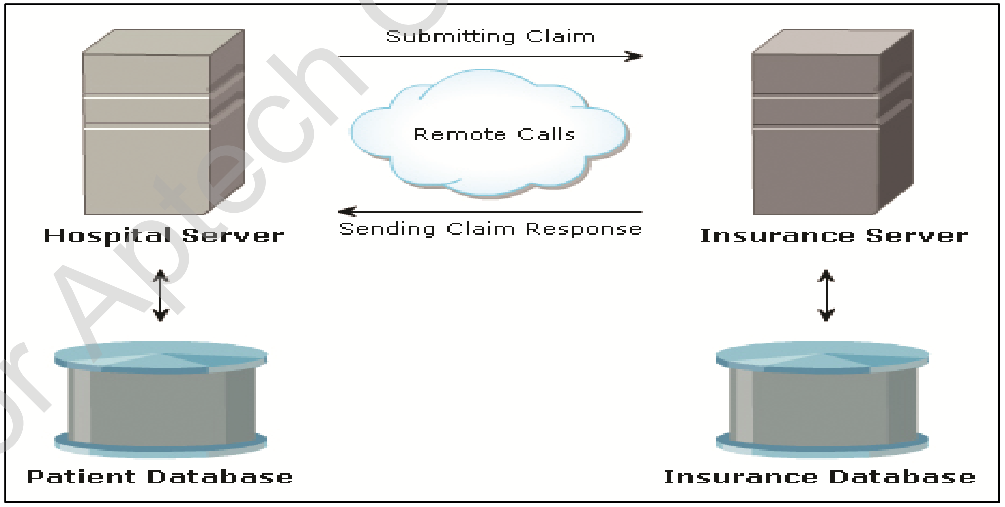
- Local and external asynchronous calls to page and Web services are collectively called remote method calls.
- A JavaScript code triggers the call that applies changes back on the client.
- Remote method calls can be used to asynchronously trigger and control remote operations.
- Page methods can be found within ASP.NET's code-based file page. They are of two types:

  - Static
  - Shared

- The `WebService` attribute applied to them to get the desired results.
- JSON requests and responses are passed back and forth during communication between page methods and ASP.NET AJAX-enabled page.

Remote Methods

- Remote methods make the task of information exchange in the same format easy and faster.
- Different architectures, which can be called by remote methods, are as follows:
  - Common Object Request Broker Architecture (CORBA)
  - Distributed Component Object Model (DCOM)
  - Remote Method Invocation (RMI)
- Remote methods work best in places with the need for heavy information exchange, such as hospitals as shown in the figure.



Calling Remote Methods

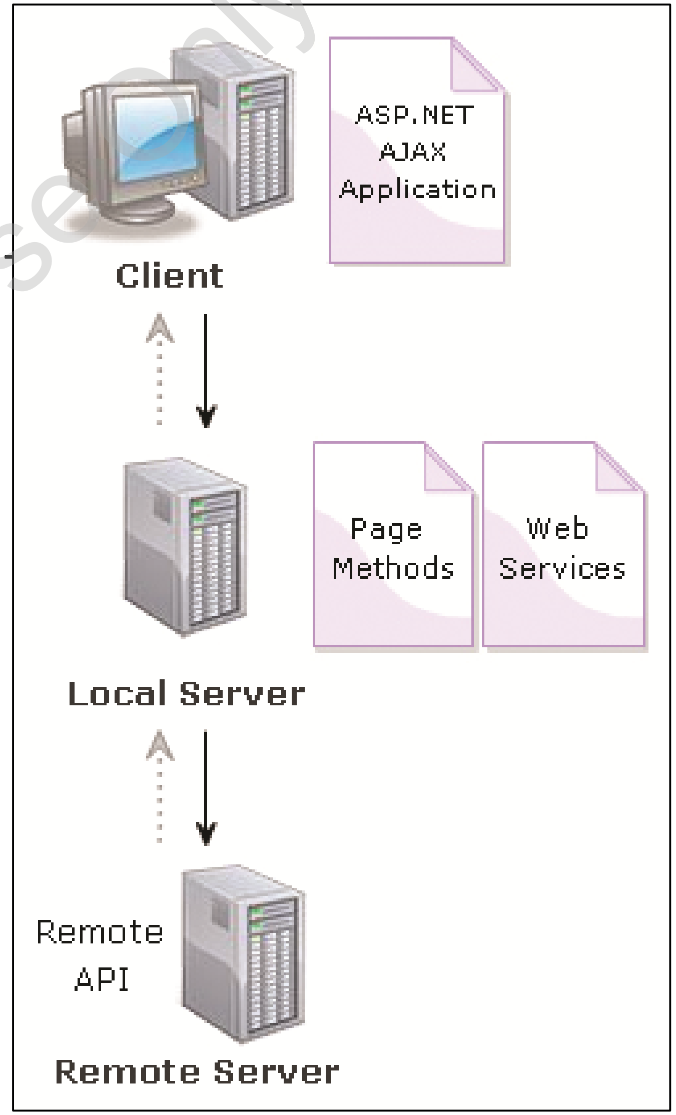
Remote Method Calls in ASP.NET AJAX

AJAX

- This works best when a particular task is to be performed on the server without any adverse effects.
- Calling remote methods requires an Application-Programming Interface (API).
- Two server APIs are provided by ASP.NET AJAX Extensions:

  - Page methods
  - Web services

- An interface groups these methods and properties.
- To implement this interface, a class is created.
- Once these remote APIs are published, the ASP.NET AJAX runtime manages client calls.



Remote Method Calls in ASP.NET AJAX

More About Page Methods

- A postback is dependent on the state of particular instance of the class.
- Use of page methods avoids this dependence.
- Static page methods are exposed by code-behind class.
- Updated information is sent back and forth by the method.
- Page methods are useful because they:

Provide access to session state, cache, and user objects.

Do not require page life cycles.

Async, Await, and TAP

AJAX

Async	• Used to mark task-based asynchronous methods.
Await	• Is a syntactical shorthand. • Indicates part of the code that is to wait asynchronously on other code pieces.
TAP	• New model of asynchronous methods are named Task-based Asynchronous Pattern.

- The `async` and `await` keywords:

Enable working with Task-based objects.

Enhance synchronous support in .NET Framework 4.5.

Simplify the process of writing asynchronous codes.

Defining Page Methods

AJAX

- Page methods, specific to an ASP.NET page are defined and enabled by using the following:

`WebMethod` Attribute

- Attached to the page method.
- Makes the method callable from remote clients.

`EnablePageMethods` Property

- Is set to 'true' to enable page methods.
- Generates `PageMethods` JavaScript client-script proxy class, used to call remote methods.

`ScriptModule` HttpModule

- Allows calling and execution of page methods.
- Available in `web.config` file.

Creating an Asynchronous Gizmos Page

1. • The `async` and `await` keywords are used.
2. • Compiler controls all complex transformations.
3. • Codes are written using C#'s synchronous control flow constructs.
4. • Compiler automatically applies transformations to bypass blocking threads.
5. • Page directive and `async` attributes must be set to `true`.

- Following code snippet creates a `Page` directive.

Code Snippet

```
<%@ Page Async="true" Language="C#" AutoEventWireup="true"
    CodeBehind="GizmosAsync.aspx.cs"
Inherits="WebAppAsync.GizmosAsync" %>
```

Page Methods 1-8

Defining Page Method

A `Greetings` method is defined.

↓

A string is accepted as input parameter.

↓

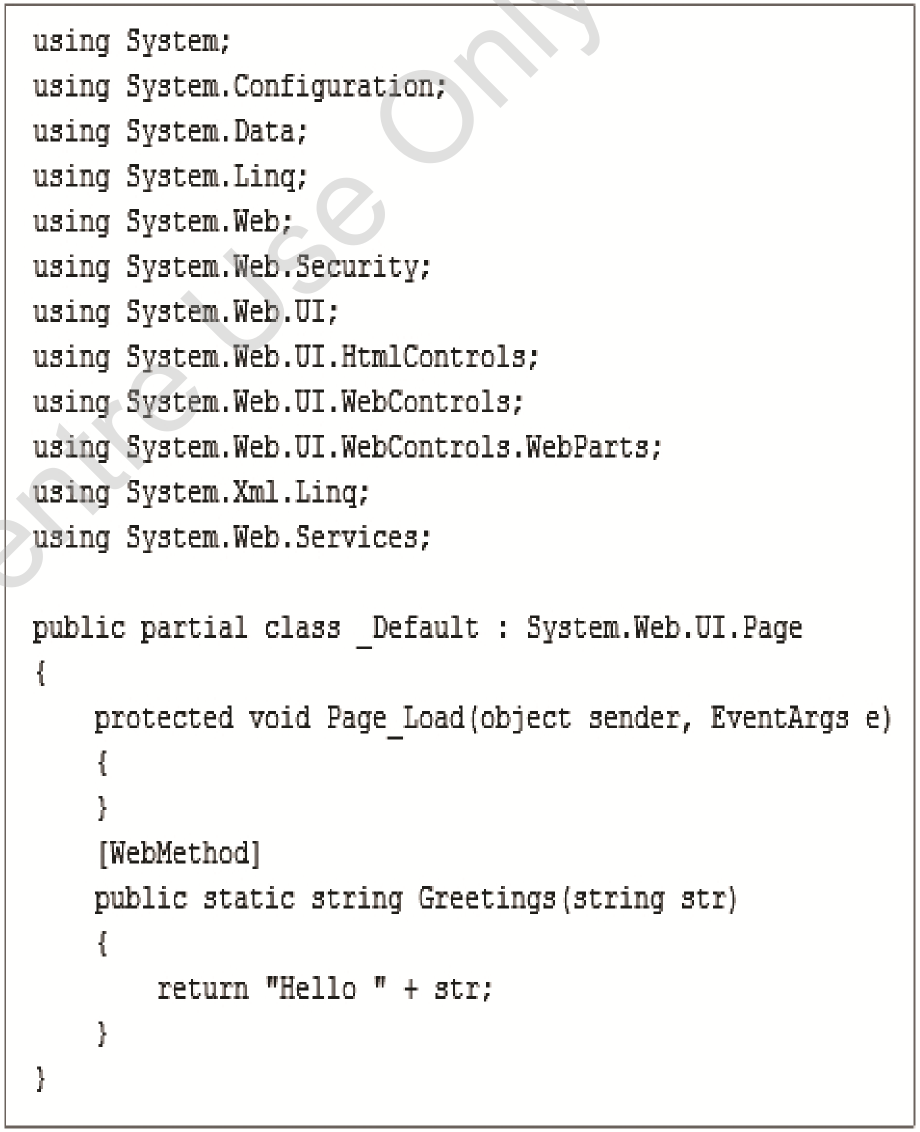
`WebMethod` attribute is added.

↓

`System.Web.Services` namespace is imported.

↓

The method can now be called over the Web.



```
using System;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Web.Services;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    [WebMethod]
    public static string Greetings(string str)
    {
        return "Hello " + str;
    }
}
```

Code for Defining Page Method

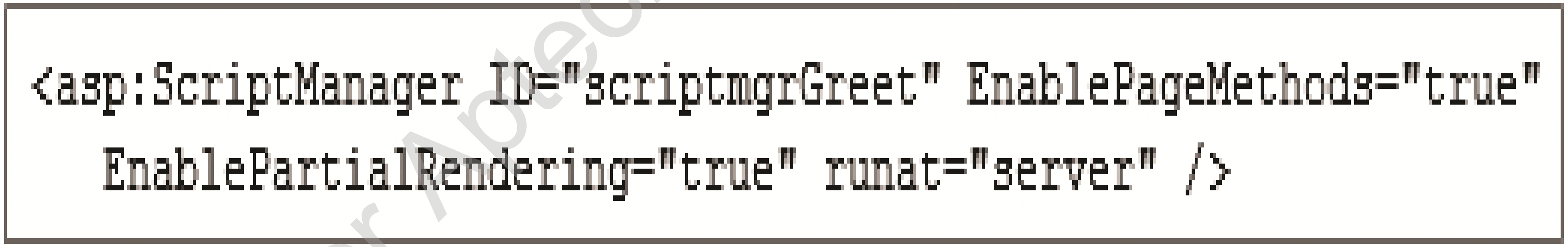
Page Methods 2-8

Enabling Page Method

An element `<asp:ScriptManager>` is included after the `<form>` tag.

↓

`EnablePageMethods` property is set to `True`.



```
<asp:ScriptManager ID="scriptmgrGreet" EnablePageMethods="true"
    EnablePartialRendering="true" runat="server" />
```

Code to Enable Page Method

Page Methods 3-8

AJAX

Providing Runtime Support

`<httpModules>` element configures HTTP modules.

Element is registered in the `web.config` file.

The `<add>` is the sub element that adds `httpModule` elements.

Name of module followed by type is specified.

The type implements the features.

`ScriptModule` is registered in the `<httpModules>` section.

```
<httpModules>
  <add name="ScriptModule" type="System.Web.Handlers.ScriptModule,
       System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
       PublicKeyToken=31BF3856AD364E35"/>
</httpModules>
```

Code to Provide Runtime Support

Page Methods 4-8

AJAX

Creating Giz Synchronous Page

- The giz synchronous Page_load methods and the GizAsync asynchronous page are displayed in the code snippet.

Code Snippet

```
protected void Page_Load(object sender, EventArgs e)
{
   var gService = new GizService();
   GizGridView.DataSource = gService.GetGadgetList();
   GizGridView.DataBind();
}
```

Page Methods 5-8

AJAX

- The asynchronous version is displayed in the code snippet.

Code Snippet

```
protected void Page_Load(object sender, EventArgs e)
{
    RegisterAsyncTask(new PageAsyncTask(GetGizSvcAsync));
}

private async Task GetGizSvcAsync()
{
    var gService = new GizService();
    GizGridView.DataSource = await gService.GetGadgetList();
    GizGridView.DataBind();
}
```

Page Methods 6-8

- The changes made to the `GizAsync` page to be asynchronous are as follows:
  - The `async` attribute in the `Page` directive is set to `true`.
  - An asynchronous task is registered using the `RegisterAsyncTask` method. The `async` keyword is used to mark the new `GetGizSvcAsync` method.
  - Async is also appended to the asynchronous methods name.
  - Task is the return type of the new `new GetGizSvcAsync` method. It shows the ongoing work and also gives method callers a handle to wait for the asynchronous operation's completion.
  - The Web service call uses the `await` keyword.
  - The asynchronous Web service API `GetGizSvcAsync` is called.

Page Methods 7-8

- The code snippet shows the `GetGiz` and `GetGizAsync` methods.

Code Snippet

```
public List<Giz> GetGiz()
{
   var uriServ = Util.getServiceUri("Giz");
   using (WebClient wClient = new WebClient())
   {
        return JsonConvert.DeserializeObject<List<Giz>>(
            wClient.DownloadString(uriServ)
        );
    }
}
public async Task<List<Giz>> GetGizAsync()
{
    var uriServ = Util.getServiceUri("Giz");
    using (HttpClient hClient = new HttpClient())
    {
        var response = await hClient.GetAsync(uriServ);
        return (await response.Content.ReadAsAsync<List<Giz>>());
    }
}
```

Page Methods 8-8

AJAX

RegisterAsyncTasks Notes

- Methods that contain the `RegisterAsyncTask` notes run instantly after PreRender.
  - Following code snippet displays the `Page_Load` event.

Code Snippet

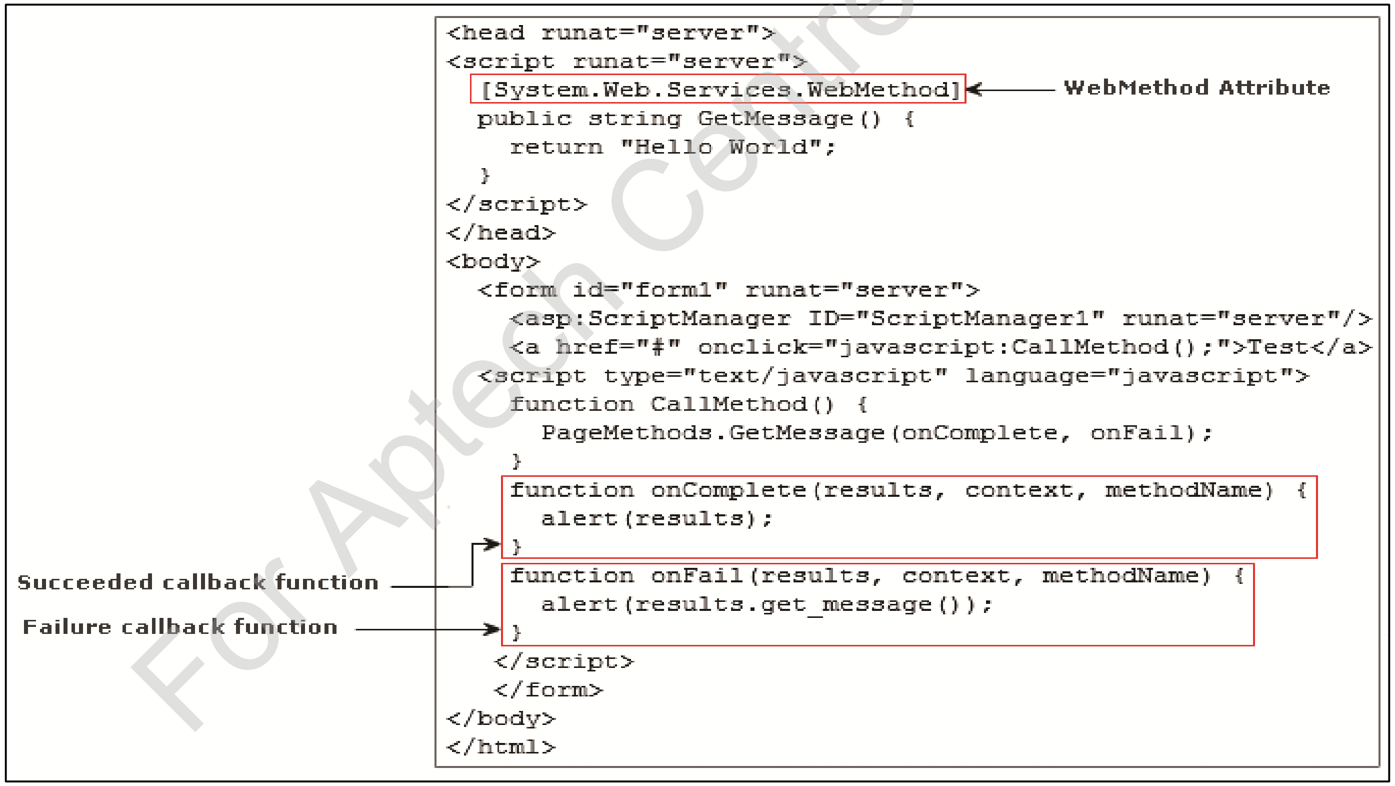
```
protected void Page_Load(object sender, EventArgs e)
{
    await ...;
    // do work
}
```

Proxy Class 1-2

AJAX

- ASP.NET AJAX runtime generates the `PageMethods` proxy class.
- It contains a list of all Web methods.
- The server-side methods contain three additional parameters:
  - Two callback methods – One for success and one for failure
  - One object that represents call context

Following figure displays the code for a proxy class.



```
<head runat="server">
<script runat="server">
  [System.Web.Services.WebMethod]
  public string GetMessage() {
    return "Hello World";
  }
</script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server"/>
    <a href="#" onclick="javascript:CallMethod();">Test</a>
  <script type="text/javascript" language="javascript">
    function CallMethod() {
      PageMethods.GetMessage(onComplete, onFail);
    }
    function onComplete(results, context, methodName) {
      alert(results);
    }
    function onFail(results, context, methodName) {
      alert(results.get_message());
    }
   </script>
   </form>
</body>
</html>
```

Proxy Class 2-2

AJAX

- Following code snippet displays the function.

Code Snippet

```
function method(result, context, methodName)
```

where,

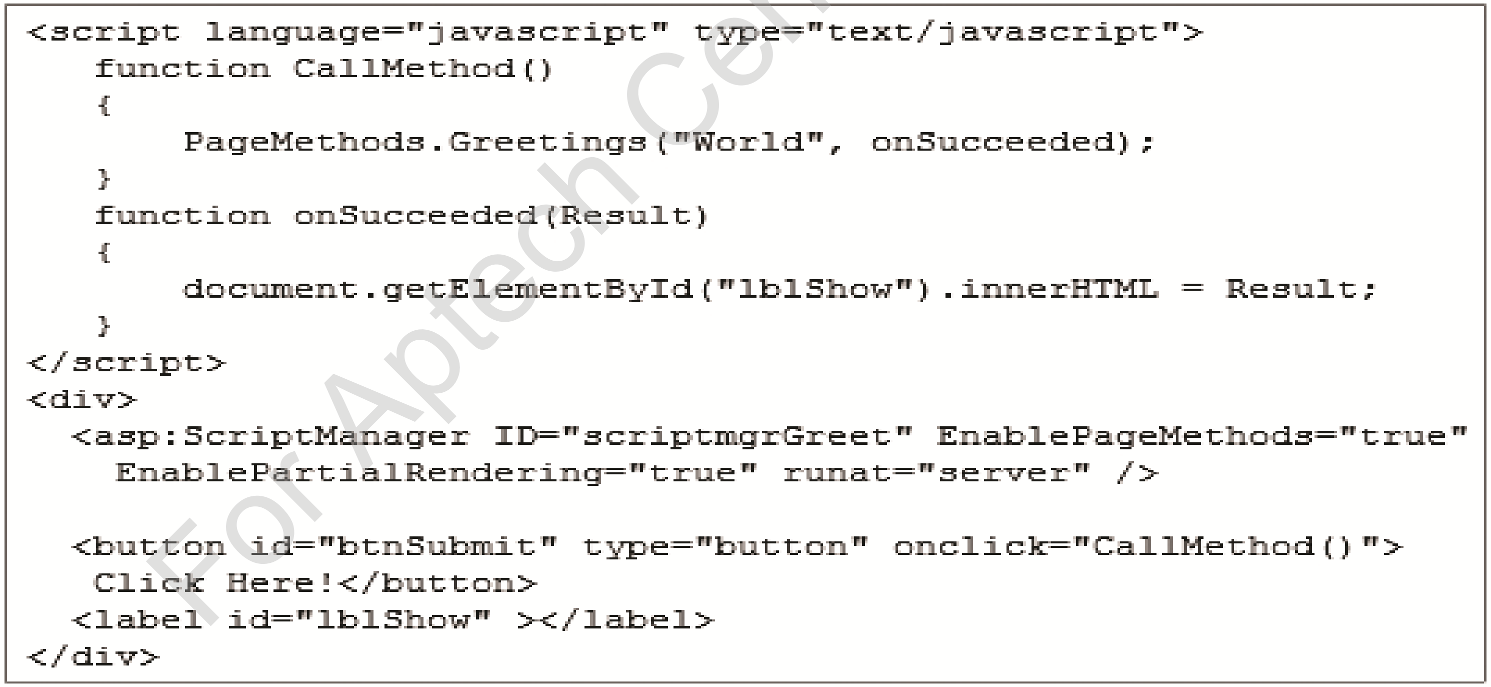
- `result` – Specifies the return value from the method if it is success, and returns JavaScript error object if it fails.
- `context(optional)` – Specifies the context information passed to the callback function.
- `methodName(optional)` – Specifies the name of the Web method that is invoked.

Using Page Methods

AJAX

- JavaScript function `CallMethod()` calls the `Greetings()` method using the `PageMethods` proxy class.
- The function is associated with the button event handler, `onclick`.
- The function takes two arguments:
  - A string as an input parameter
  - The succeeded callback function

Following figure displays the use of `PageMethods`.



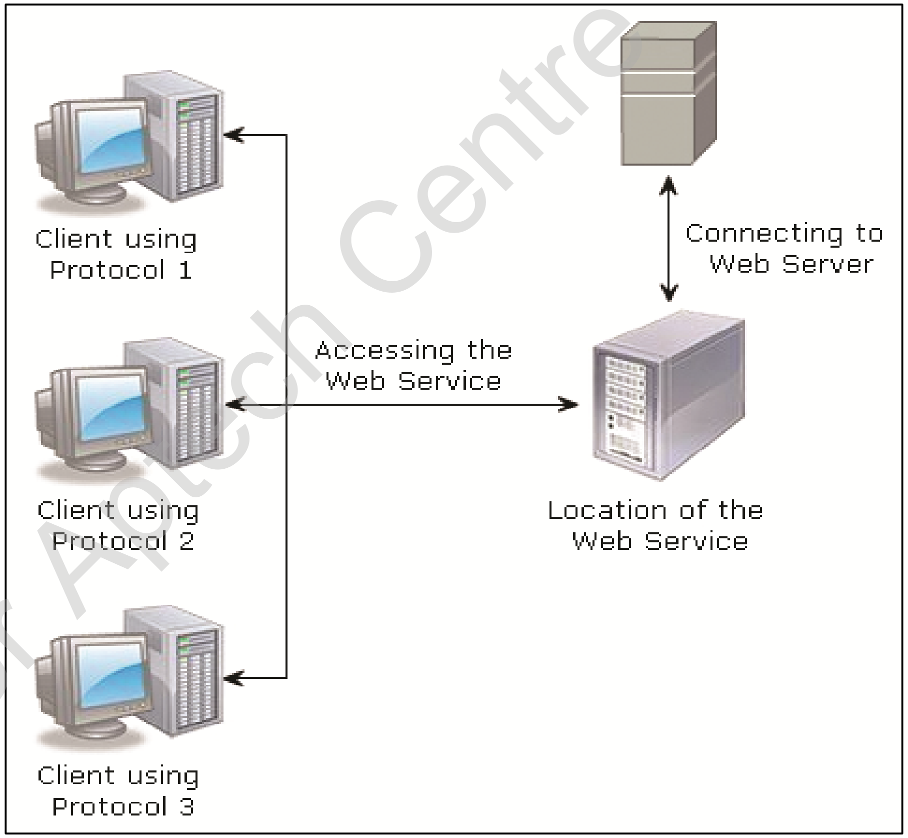
```
<script language="javascript" type="text/javascript">
   function CallMethod()
   {
       PageMethods.Greetings("World", onSucceeded);
   }
   function onSucceeded(Result)
   {
       document.getElementById("lblShow").innerHTML = Result;
   }
</script>
<div>
  <asp:ScriptManager ID="scriptmgrGreet" EnablePageMethods="true"
    EnablePartialRendering="true" runat="server" />

  <button id="btnSubmit" type="button" onclick="CallMethod()">
   Click Here!</button>
  <label id="lblShow" ></label>
</div>
```

ASP.NET AJAX Web Services - Purpose

AJAX

- A Web Service is a reusable component providing services on the Web.
- The service works in distributed environment irrespective of the operating system and communication protocols in use.
- It functions irrespective of the parties software and hardware platforms.

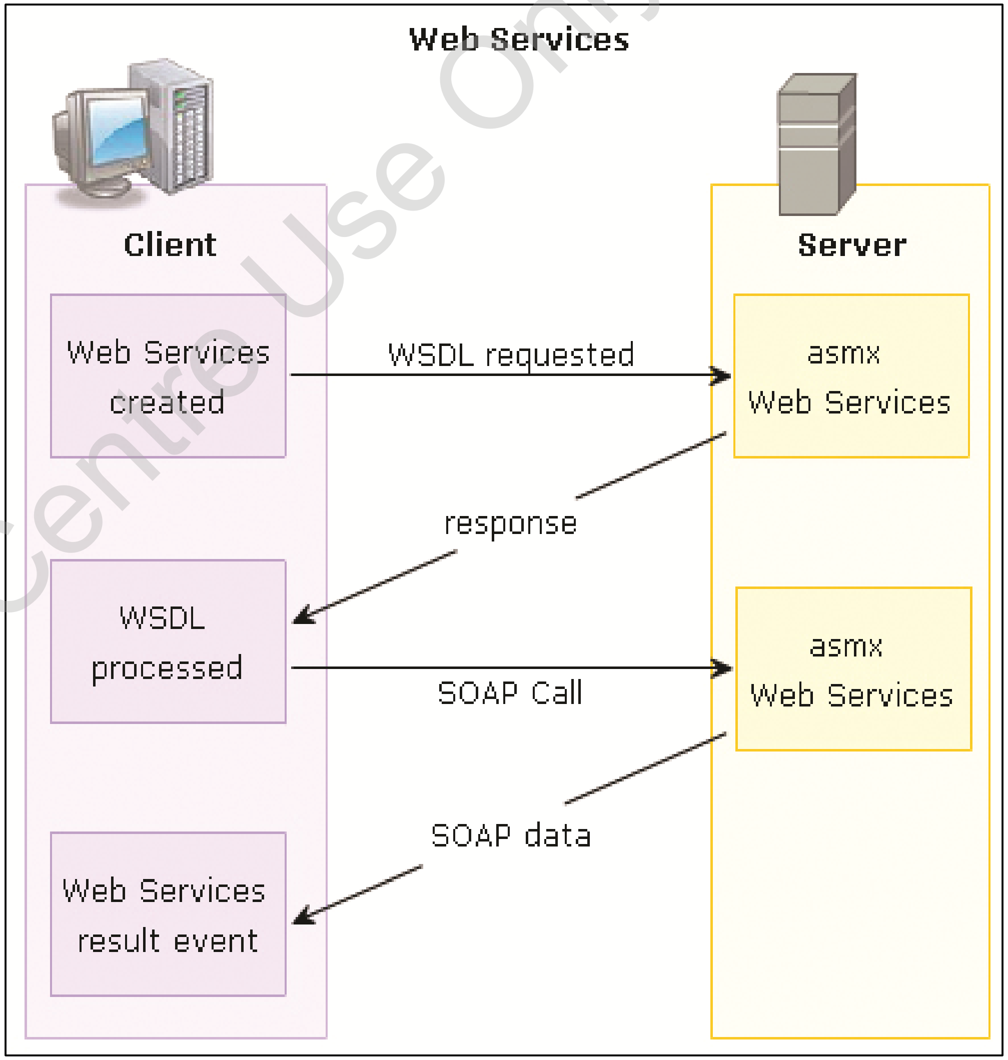
Following figure displays the Web Services.



Web Services

AJAX

- Are independent modular components.
- Provide services on LAN, WAN, MAN, and Internet.
- Designed to provide 100 percent interoperability.
- Use Simple Object Access Protocol (SOAP) to communicate.
- Are the ideal solution to all the problems of traditional distributed computing.
- Figure displays the working of Web services.



Working of Web Services

Asynchronous Communication Layer 1-4

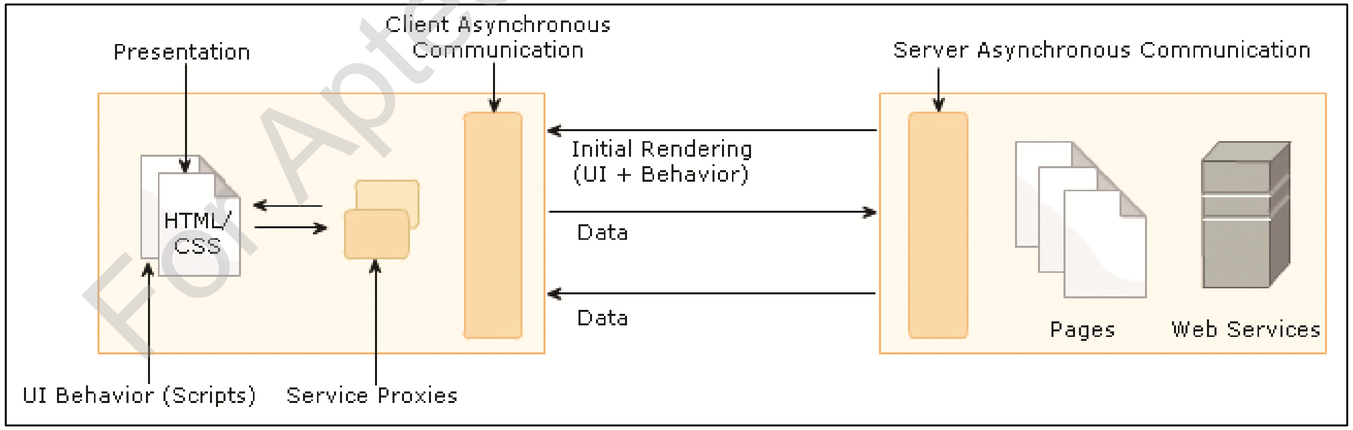
AJAX

- Is the base of Microsoft AJAX library.
- Contains a networking layer known as asynchronous communication layer.

Client-Server Communication

- It enables a rich user interface.
- The server handles business logic and data-related actions.
- Asynchronous communication layer supports the client.
- It is divided into two:
  - Client architecture
  - Server architecture

Following figure displays the Client-Server communication.


Presentation
Client Asynchronous Communication
Server Asynchronous Communication
HTML/ CSS
Initial Rendering (UI + Behavior)
Data
Data
Pages
Web Services
UI Behavior (Scripts)
Service Proxies

Asynchronous Communication Layer 2-4

Client Architecture

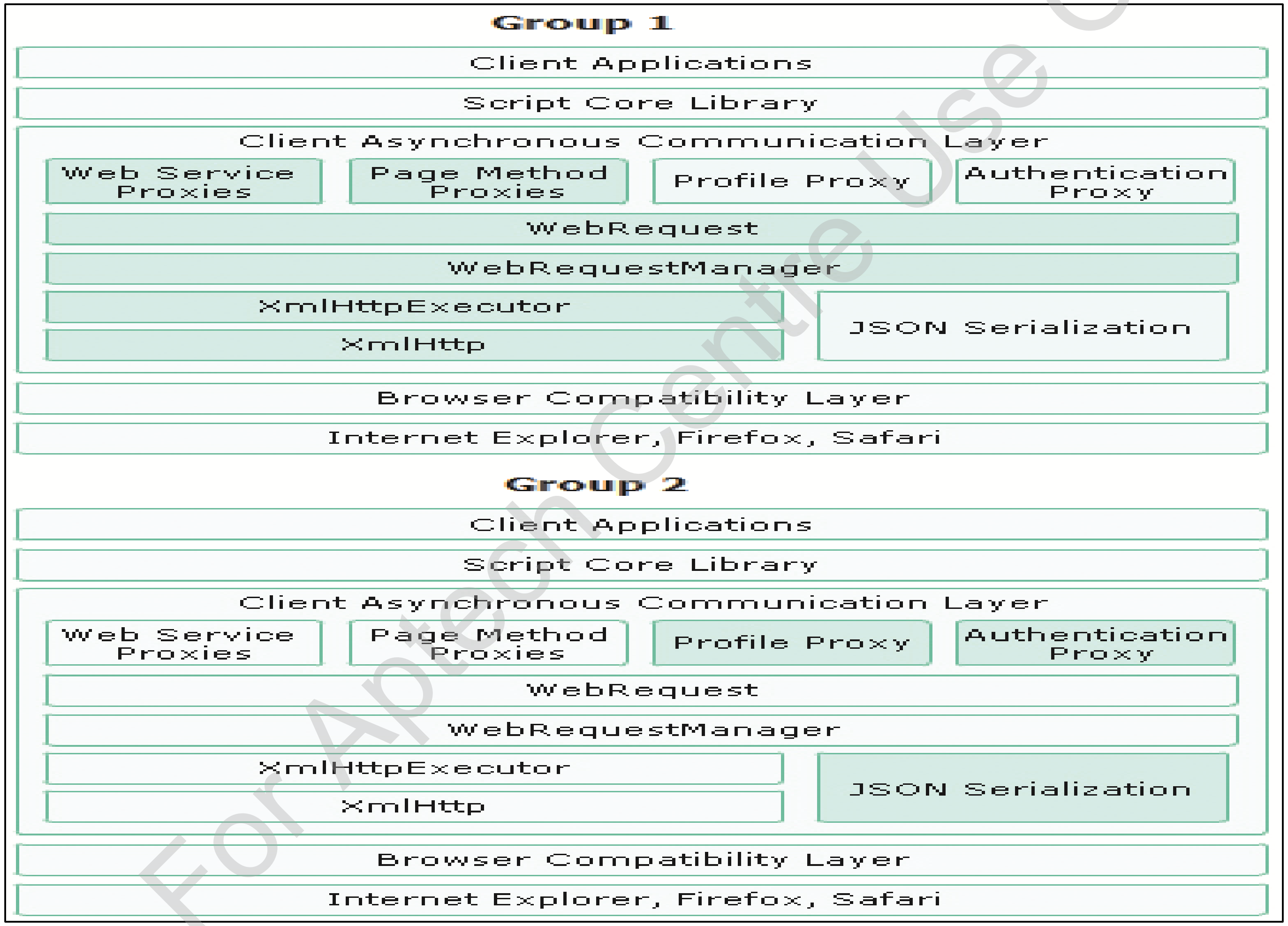
- It contains two groups:

Communication Group
Web Service Proxies
Page Method Proxies
WebRequest
WebRequestManager
XMLHttp
XMLHttpExecutor

Support Group
Profile Proxy
Authentication Proxy
JSON Serialization

Asynchronous Communication Layer 3-4

Following figure displays the client architecture of an asynchronous communication layer.



Asynchronous Communication Layer 4-4

AJAX

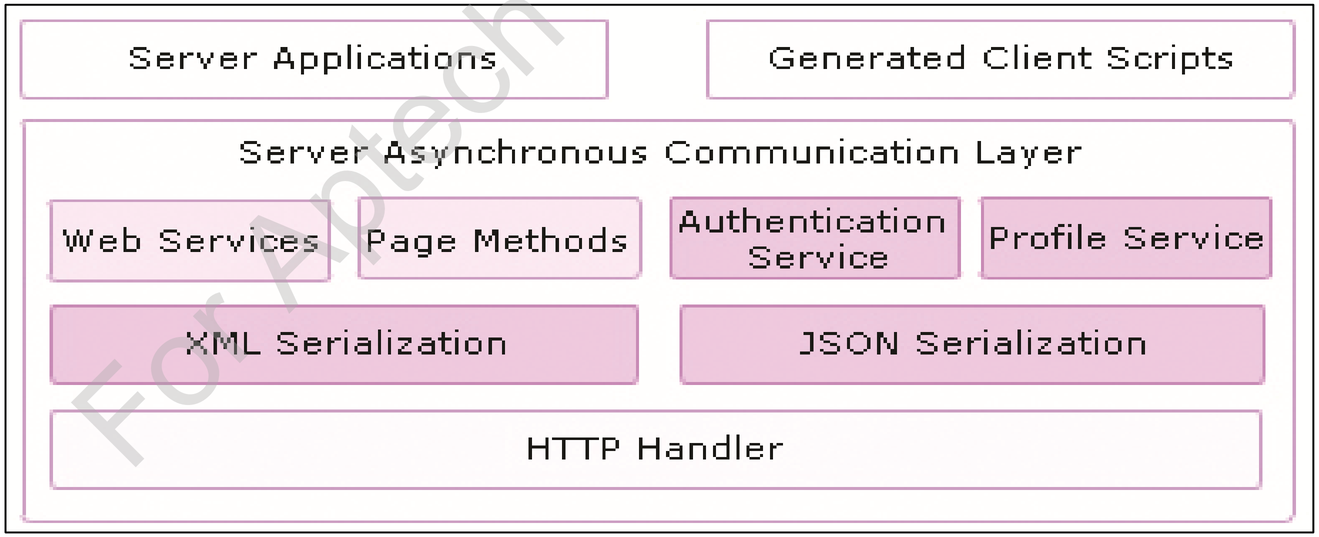
Server Architecture

- It contains two groups:

Communication Group
Web Services
Page Methods

Support Group
Authentication Service
Profile Service
JSON Serialization
XML Serialization

Following figure displays the server architecture.

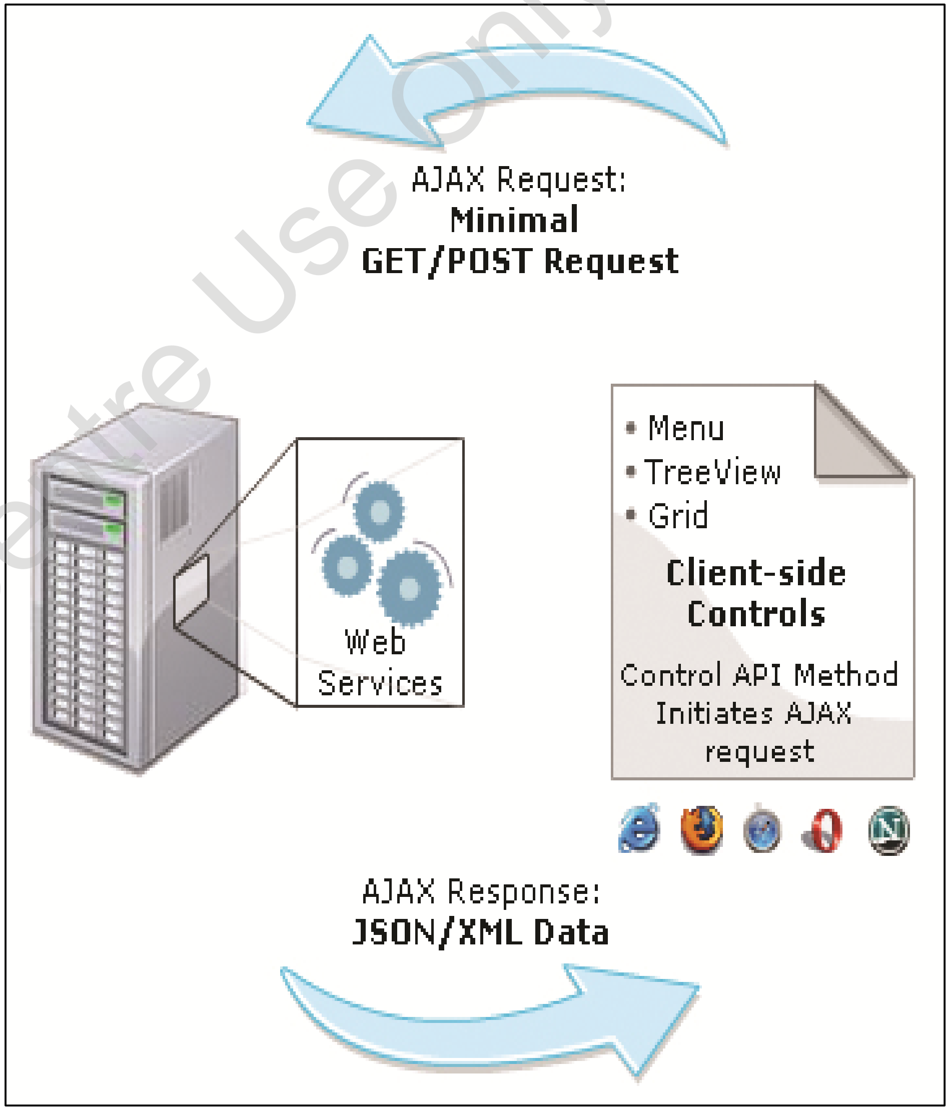


ASP.NET AJAX Web Services

AJAX

- They differ from ASP.NET Web Services in two ways:
  - Two new attributes for methods and the class that are not available in ASP.NET Web Service are used.
  - New serialization for transportation of messages named JSON is used.
- JSON, the default serialization of AJAX Web Services, is a subset of JavaScript available in text format and is easier to read and write.
- They have two public interfaces:
  - SOAP-based
  - JSON-based

Figure displays the ASP.NET AJAX Web Service.



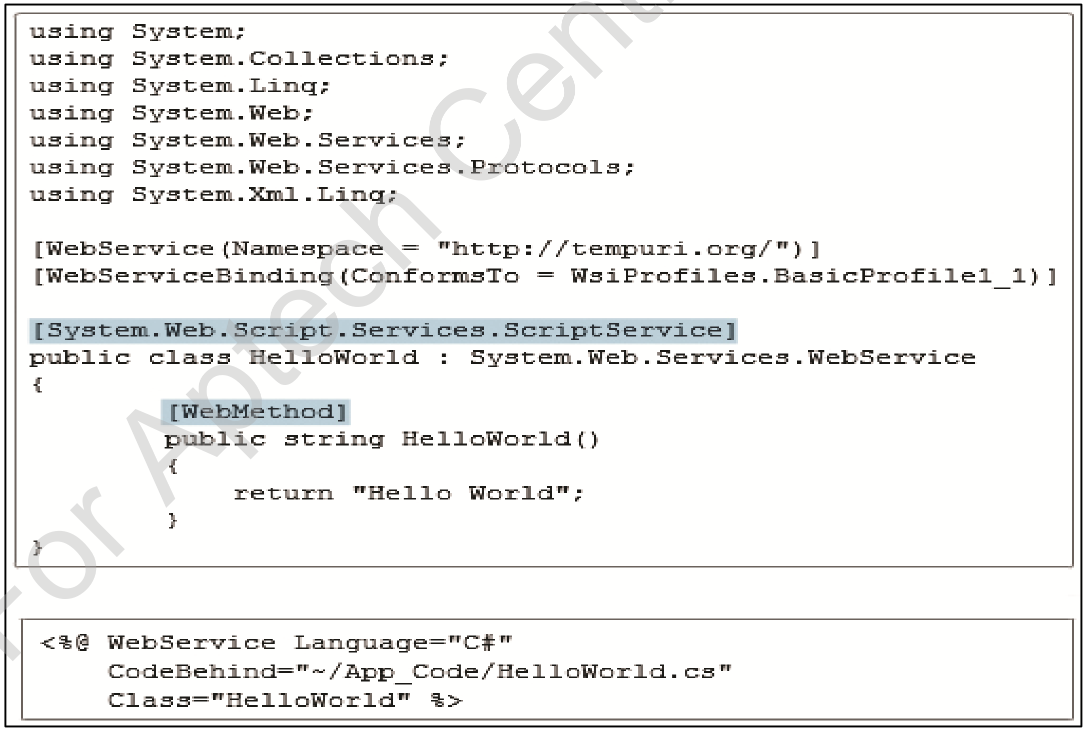
Creating an AJAX Web Services

AJAX

In the **New Web Site** dialog box, select the **ASP.NET Web Service** template.

In the **Solution Explorer**, right-click the application path name, click the **Add New Items** command on the displayed menu, and click the **Web Service** template.

Following figure displays the Web service.



```
using System;
using System.Collections;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]

[System.Web.Script.Services.ScriptService]
public class HelloWorld : System.Web.Services.WebService
{
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
}
```

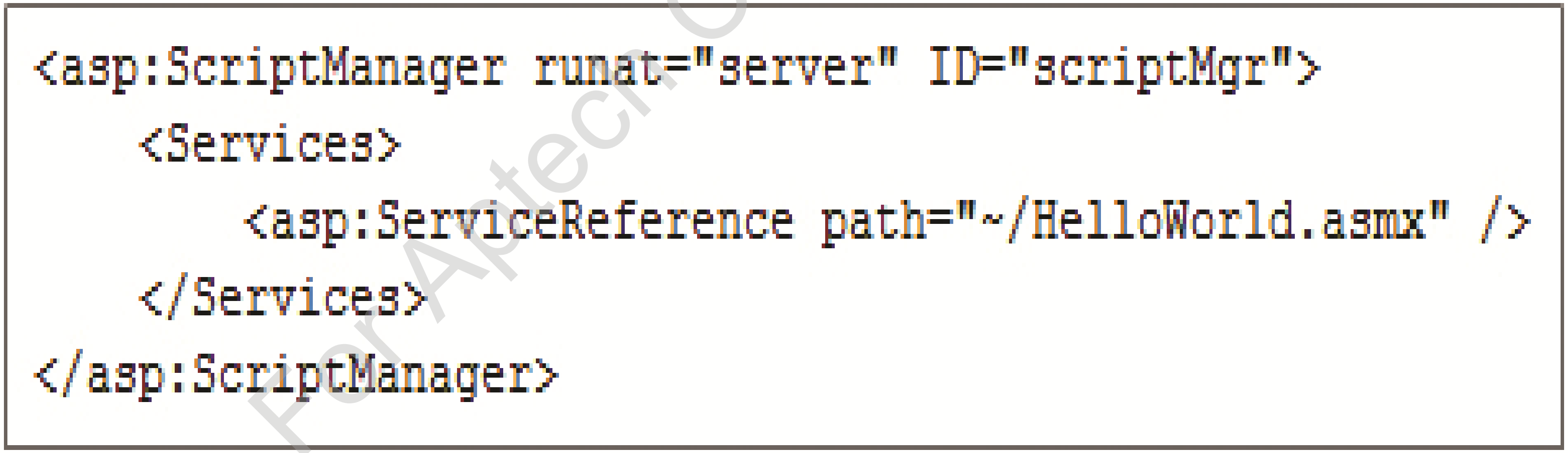
```
<%@ WebService Language="C#"
    CodeBehind="~/App_Code/HelloWorld.cs"
    Class="HelloWorld" %>
```

Using an AJAX Web Services 1-3

AJAX

Registering Web Service

- The `ScriptManager` control enables a Web service call.
- The `<asp:ServiceReference>` is the child element of the `ScriptManager` control.
- The path attribute specifies the URL of the Web service file.
- ASP.NET page creates a JavaScript proxy class to call Web service from the client-side script. Following figure registers a AJAX Web service.

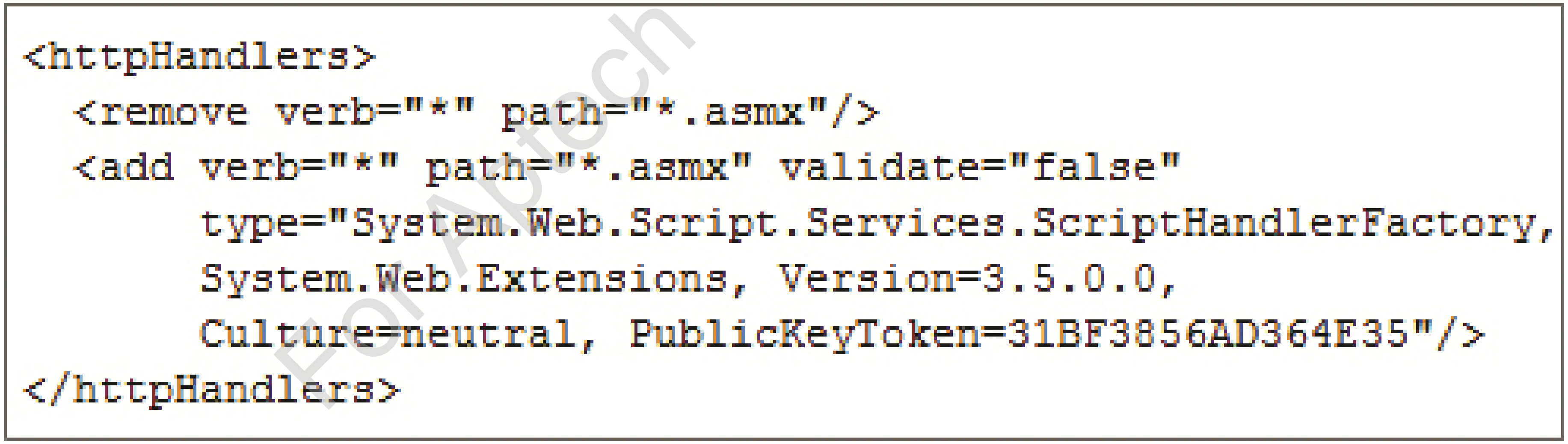


```
<asp:ScriptManager runat="server" ID="scriptMgr">
    <Services>
        <asp:ServiceReference path="~/HelloWorld.asmx" />
    </Services>
</asp:ScriptManager>
```

Using an AJAX Web Services 2-3

Configuring Application

- ASP.NET AJAX application is configured by registering the `ScriptHandlerFactory` HTTP handler.
- This registration is done in a `web.config` file.
- The handler factory differentiates JSON calls from the regular SOAP-based calls.
- These configuration settings are present in the `web.config` file by default as shown in the figure.



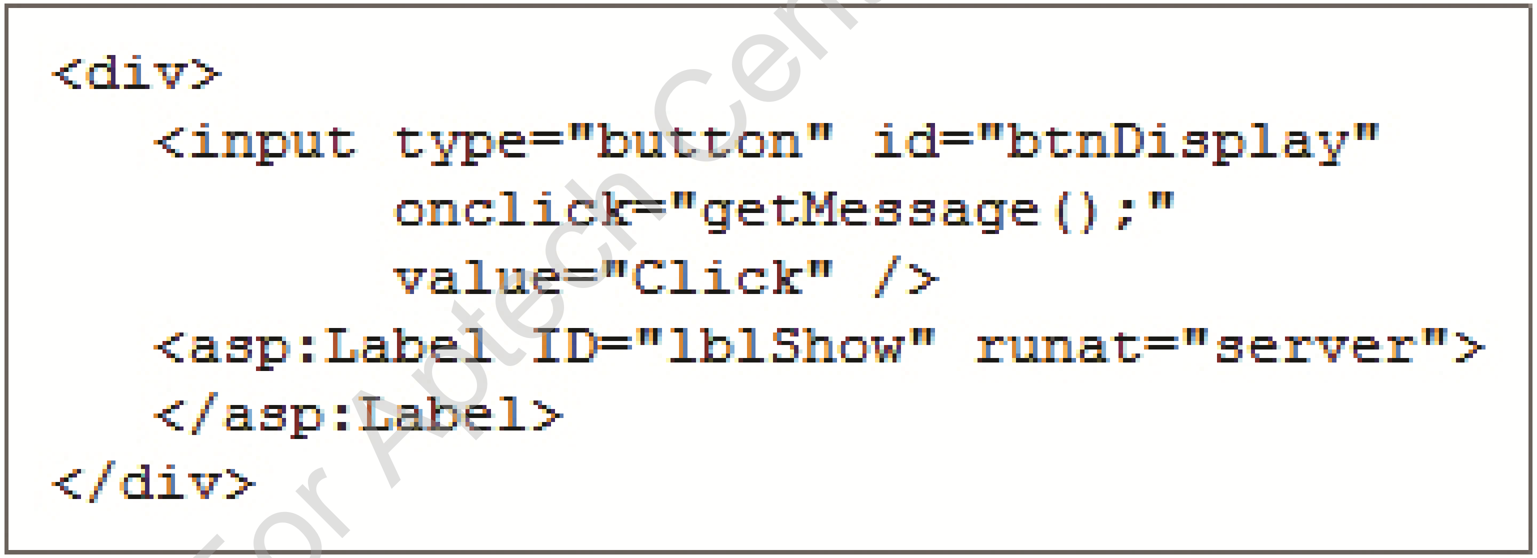
```
<httpHandlers>
  <remove verb="*" path="*.asmx"/>
  <add verb="*" path="*.asmx" validate="false"
       type="System.Web.Script.Services.ScriptHandlerFactory,
       System.Web.Extensions, Version=3.5.0.0,
       Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
</httpHandlers>
```

Using an AJAX Web Services 3-3

AJAX

Executing the Web Service

- The Web service method executes when the user performs some action.
- Event handlers are required to associate with JavaScript function.
- The JavaScript function `getMessage()` is associated with this event handler as shown in the figure.



```
<div>
   <input type="button" id="btnDisplay"
          onclick="getMessage();"
          value="Click" />
   <asp:Label ID="lblShow" runat="server">
   </asp:Label>
</div>
```

HTTP Requests and Responses 1-5

AJAX

- ASP.NET 4.5 enhances the ability to read and access the HTTP request entity.
- It uses the `HttpRequest.GetBufferlessInputStream` method.
- No threads are tied up and user can asynchronously read streams.
- It also simplifies the integration with .aspx page handlers and ASP.NET MVC controllers.
- The asynchronous `Stream` methods enable asynchronous reading of request entity.
- `HttpRequest.GetBufferlessInputStream` stream reference supports synchronous and asynchronous read methods.
- The `Stream` object carries out two methods:
  - `BeginRead`
  - `EndRead`

HTTP Requests and Responses 2-5

AJAX

Supporting `Await` and Task-based Asynchronous Modules and Handlers

- Task is an asynchronous programming concept in the .NET Framework 4.5.
- The Task type represents it and other related types in the `System.Threading.Tasks` namespace.
- The two keywords in .NET Framework 4.5 that simplify the process of using the `Task` object are:
  - `await`
  - `async`

HTTP Module

- These are assemblies that are called on each request that is made to the application.
- They can access life cycles events.
- They also examine and act upon incoming requests.
- Outbound responses can also be examined and modified.

HTTP Requests and Responses 3-5

AJAX

Asynchronous Method

- An asynchronous method is defined to make an asynchronous call for downloading the Microsoft Home page as shown in the code snippet.

Code Snippet

```
private async Task
readHtmlPage(object caller, EventArgs e)
 {
      WebClient wClient = new WebClient();
      var result = await
wClient.DownloadStringTaskAsync("http://www.LearningCurve.co
m");
      // Do something with the result
}
```

- .NET Framework 4.5 automatically handles unwinding of the call stack which is restored after the download.

HTTP Requests and Responses 4-5

AJAX

Asynchronous HTTP Modules

- `EventHandlerTaskAsyncHelper` helper method and `TaskEventHandle` delegate type make it possible to use the same asynchronous method in an asynchronous ASP.NET HTTP module as shown in the code snippet.

Code Snippet

```
public void Init(HttpApplication context)
 {
   // Wrap the Task-based method to use with
   // the older async programming model.
   EventHandlerTaskAsyncHelper help =
            new EventHandlerTaskAsyncHelper(readHtmlPage);
            // The helper object makes it easy to extract Begin/End methods 
out of
            // a method that returns a Task object. The ASP.NET pipeline 
calls the
            // Begin and End methods to start and complete calls on 
asynchronous
            // HTTP modules.
            context.AddOnPostAuthorizeRequestAsync(
                    help.BeginEventHandler, help.EndEventHandler);
        }
```

HTTP Requests and Responses 5-5

AJAX

Asynchronous HTTP Handlers

- Asynchronous HTTP handlers can now be easily created due to ASP.NET 4.5 `HttpTaskAsyncHandler` asynchronous and abstract base type as shown in the code snippet.

Code Snippet

```
public class NewAsyncHandler : HttpTaskAsyncHandler
{
        // ...

        // ASP.NET automatically takes care of integrating the Task based 
override
        // with the ASP.NET pipeline.
        public override async Task ProcessRequestAsync(HttpContext context)
        {
           WebClient wClient = new WebClient();
           var result = await 

wClient.DownloadStringTaskAsync("http://www.microsoft.com");
           // Do something with the result
        }
}
```

Async and Await 1-5

AJAX

- Asynchrony is vital for applications to access the Web.
- It enables the application to continue working.
- It makes it independent of the Web resource till the time the task finishes.
- Asynchrony is useful in applications which access the UI thread.
- Asynchrony avoids the blocking of processes due to synchronous application.
- Under asynchronous methods, the applications continuously respond to the UI.

Async and Await 2-5

AJAX

Control Flow Move in Asynchronous Programming

- Each of the labeled locations, 'ONE' through 'SIX,' displays information about the current state of the program as shown in the figure.

```
public partial class MainWindow : Window
{
    private async void startButton_Click(object sender, RoutedEventArgs e)
    {
        // ONE
        Task<int> getStringLengthTask = ConnectToNetAsync();

        // FOUR
        int DataLength = await getLengthTask;

        // SIX
        resultsTextBox.Text +=
            String.Format("\r\nLength of the downloaded string: {0}.\r\n", DataLength);
    }


    async Task<int> ConnectToNetAsync()
    {
        // TWO
        HttpClient client = new HttpClient();
        Task<string> getStringTask =
            client.GetStringAsync("http://www.LearningCurve.com");

        // THREE
        string urlContents = await getStringTask;

        // FIVE
        return urlContents.Length;
    }
}
```

Async and Await 3-5

AJAX

API Async Methods

- The .NET Framework 4.5 also contains members that work along with the `async` and `await`.
- Such members are identified using the `Async` suffix added to the member name and the return type of Task.

Threads

- The use of `await` expression inside an `async` method helps to keep the current thread running without blocking.
- The `async` and `await` keywords do not create any additional threads.
- There is no necessity for multithreading.
- They are superior to `BackgroundWorker` in CPU-bound operations.

Async and Await 4-5

AJAX

Return Types and Parameters

- A Task or Task<Result> is returned from the async method.
- The await operator is assigned to the task that is returned from the call as shown in the following code snippet.

Code Snippet

```
// Signature specifies Task<TResult>
async Task<int> TaskOfTResult_ GetTaskByAsync()
{
    int hours;
    // . . .
    // Return statement specifies an integer result.
    return hours;
}

// Calls to TaskOfTResult_MethodAsync
Task<int> recievedtaskTResult = GetTaskByAsync();
```

Async and Await 5-5

AJAX

Return Types and Parameters

- Following code snippet displays the return types and parameters.

Code Snippet

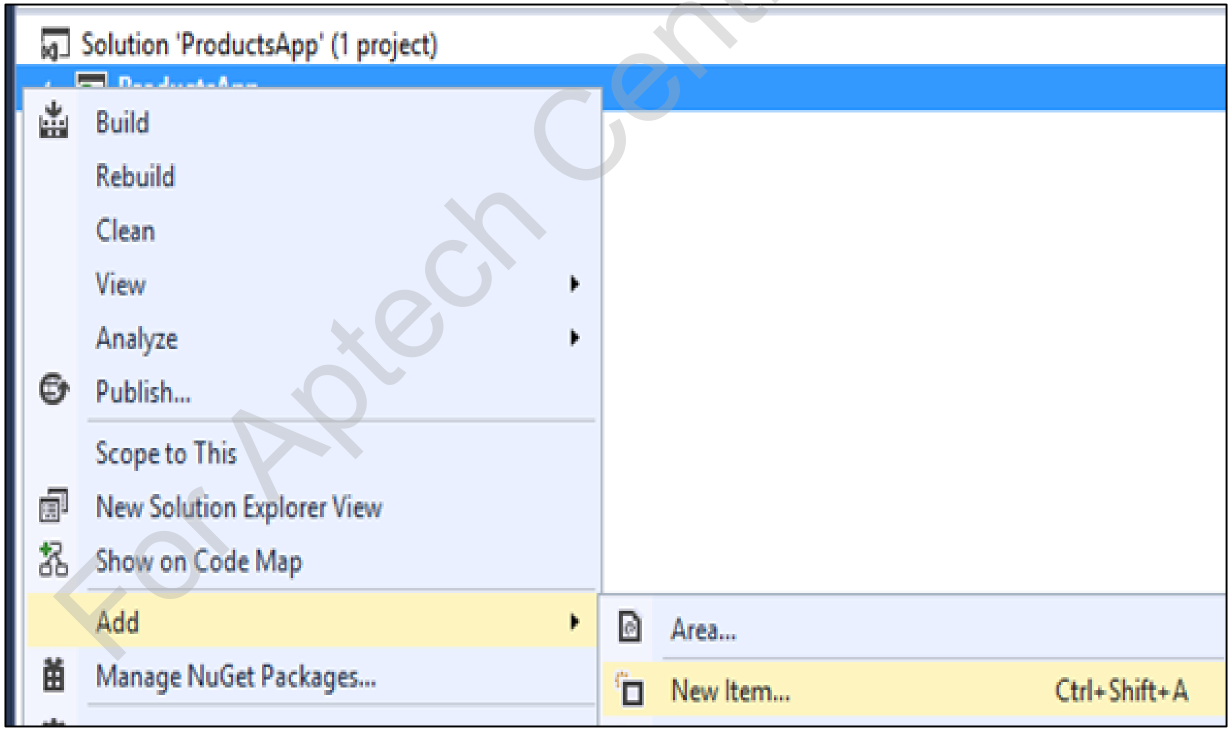
```
int intResult = await recievedtaskTResult;
// or, in a single statement
int intResult = await GetTaskByAsync();
  // Signature specifies Task
async Task GetTaskByAsync()
{
    // . . .
    // The method has no return statement.
}
 // Calls to Task_MethodAsync
Task recievedTask = GetTaskByAsync ();
await recievedTask;
// or, in a single statement
await GetTaskByAsync();
```

Web API 1-8

- Simplifies building HTTP services and enables a wider reach among clients.
- It is an optimal platform to build RESTful applications.
- HTTP is a strong and effective platform to build APIs which disclose data and services.

Calling WEB API with JavaScript and jQuery

1. Right-click the project under **Solution Explorer**. Select **Add** and then, **New Item** as shown in the figure.

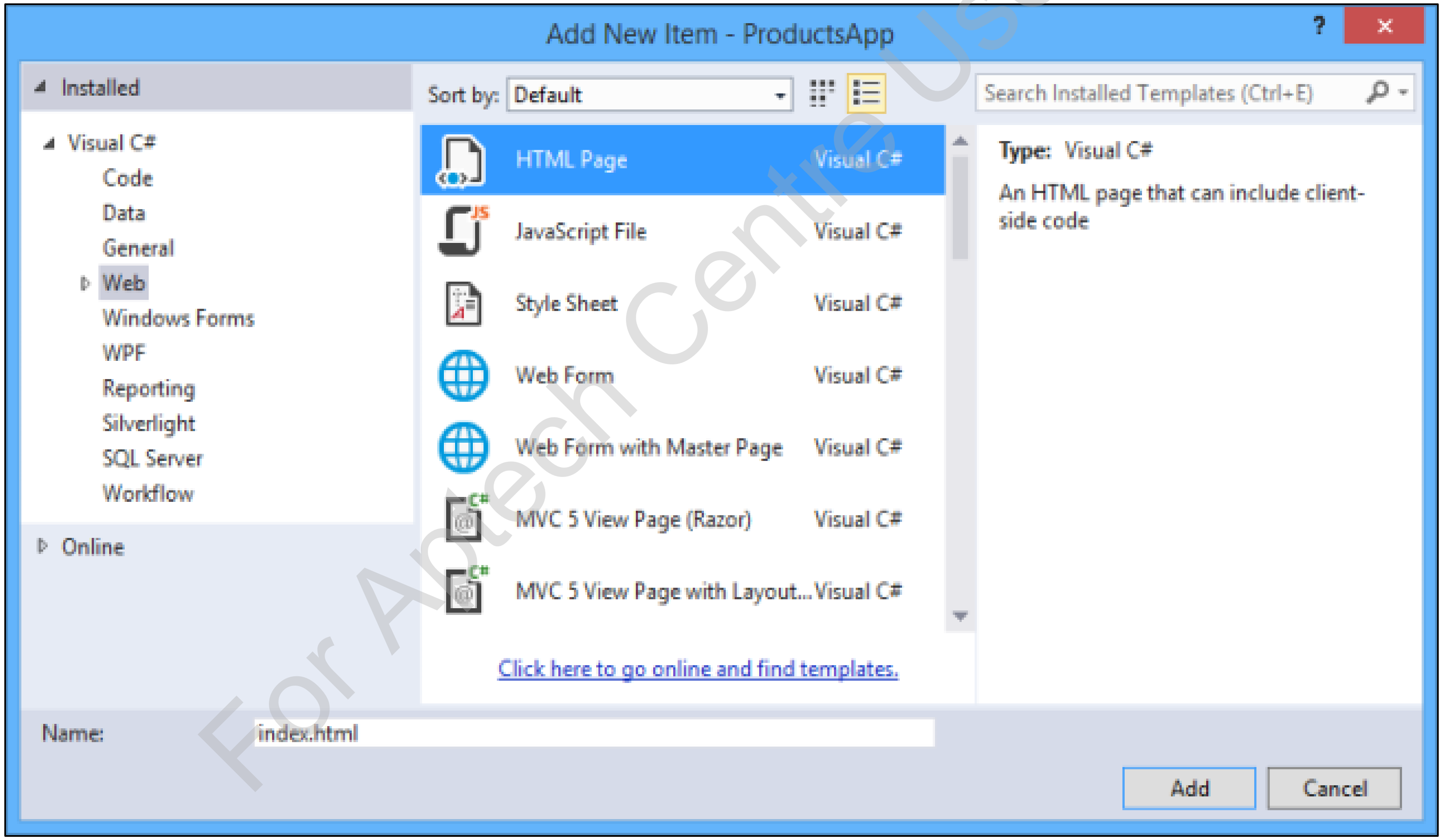


Web API 2-8

AJAX

Calling WEB API with JavaScript and jQuery

2. Select **Web** node under **Visual C#** in the **Add New Item** dialog box. Next, select **HTML Page** and name the page as 'index.html' as shown in the figure.



Add New Item - ProductsApp

Sort by: Default

Search Installed Templates (Ctrl+E)

- Installed
  - Visual C#
    - Code
    - Data
    - General
    - Web
    - Windows Forms
    - WPF
    - Reporting
    - Silverlight
    - SQL Server
    - Workflow
- Online

HTML Page	Visual C#
JavaScript File	Visual C#
Style Sheet	Visual C#
Web Form	Visual C#
Web Form with Master Page	Visual C#
MVC 5 View Page (Razor)	Visual C#
MVC 5 View Page with Layout...	Visual C#

Click here to go online and find templates.

Type: Visual C#

An HTML page that can include client-side code

Name: index.html

Web API 3-8

Calling WEB API with JavaScript and jQuery

3. Replace the text/code with the following code snippet displayed here.

Code Snippet

```
<!DOCTYPE html>
<html>
<head>
  <title>Product List</title>
</head>
<body>
  <div>
    <h2>All Products</h2>
    <ul id="products" />
  </div>
  <div>
    <h2>Search by ID</h2>
    <input type="text" id="productId" size="5" />
  <input type="button" value="SearchProduct" onclick=" SearchProd();" />
    <p id="product" />
  </div>
```

Web API 4-8

AJAX

Calling WEB API with JavaScript and jQuery

Code Snippet

```
<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-
2.0.3.min.js"></script>
  <script>
    var uri = 'api/products';
    $(document).ready(function () {
      // Send an AJAX request
      $.getJSON(uri)
          .done(function (data) {
            // On success, 'data' contains a list of products.
            $.each(data, function (key, item) {
              // Add a list item for the product.
              $('<li>', { text: UpdateFormat(item)
}).appendTo($('#products'));
            });
          });
    });
```

Web API 5-8

AJAX

Calling WEB API with JavaScript and jQuery

Code Snippet

```
     function Updateformat(item) {
       return item.Name + ': $' + item.Price;
     }

     function SearchProd() {
       var id = $('#prodId').val();
       $.getJSON(uri + '/' + id)
           .done(function (data) {
             $('#product').text(formatItem(data));
           })
           .fail(function (jqXHR, textStatus, err) {
             $('#product').text('Error: ' + err);
           });
     }
   </script>
 </body>
 </html>
```

Web API 6-8

AJAX

Accessing the List of Products

- A HTTP GET request is sent to the "/api/products“.
- An AJAX request sent by the jQuery `getJSON` function, obtains a response containing array of JSON objects.
- The DOM is updated with the information about the product in the callback as shown in the code snippet.

Code Snippet

```
$(document).ready(function () {
    // Send an AJAX request
    $.getJSON(apiUrl)
        .done(function (data) {
            // On success, 'data' contains a list of products.
            $.each(data, function (key, item) {
                // Add a list item for the product.
                $('<li>', { text: Updateformat(item)
}).appendTo($('#products'));
            });
        }); });
```

Web API 7-8

AJAX

Accessing a Product by ID

- A HTTP GET request is sent to "/api/products/ID" to access a product by its ID as shown in the code snippet.

Code Snippet

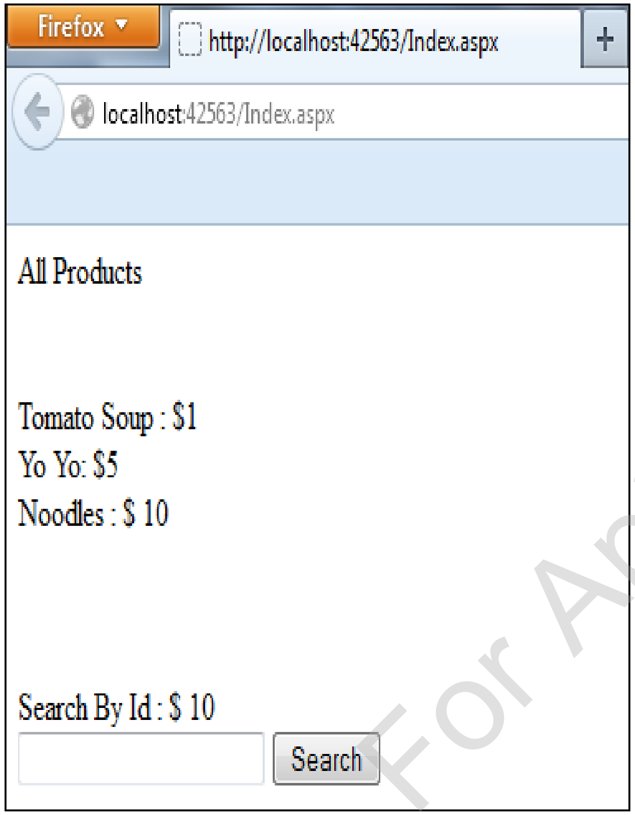
```
function find() {
    var id = $('#prodId').val();
    $.getJSON(apiUrl + '/' + id)
        .done(function (data) {
            $('#product').text(UpdateItem(data));
        })
        .fail(function (jqXHR, textStatus, err) {
            $('#product').text('Error: ' + err);
        });
}
```

Web API 8-8

AJAX

Running the Application

- To debug the application, press **F5**.
- To access a product by its ID, enter the ID and click **Search**.
- In case of an invalid ID, an HTTP error is returned by the server as shown in the following figures.

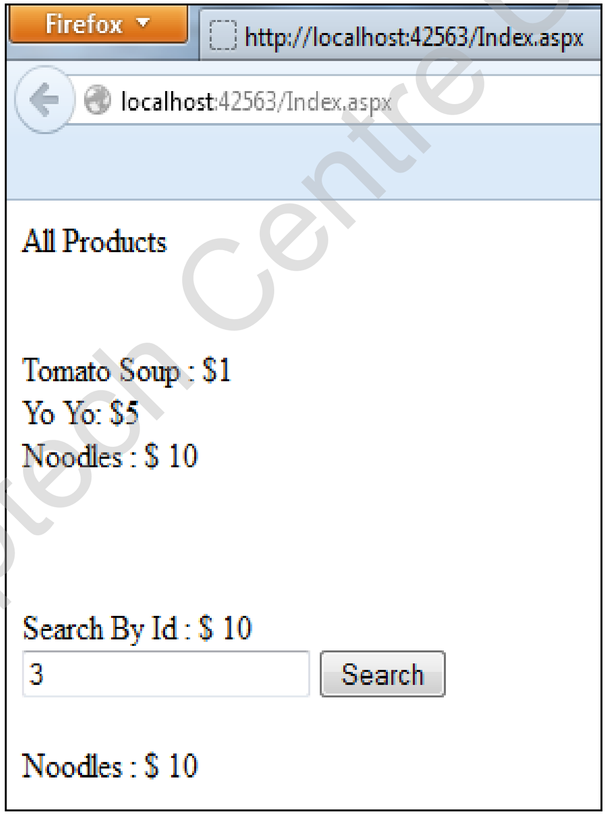

Firefox http://localhost:42563/Index.aspx
localhost:42563/Index.aspx

All Products

Tomato Soup : $1
Yo Yo: $5
Noodles : $ 10

Search By Id : $ 10
Search

Web Page


Firefox http://localhost:42563/Index.aspx
localhost:42563/Index.aspx

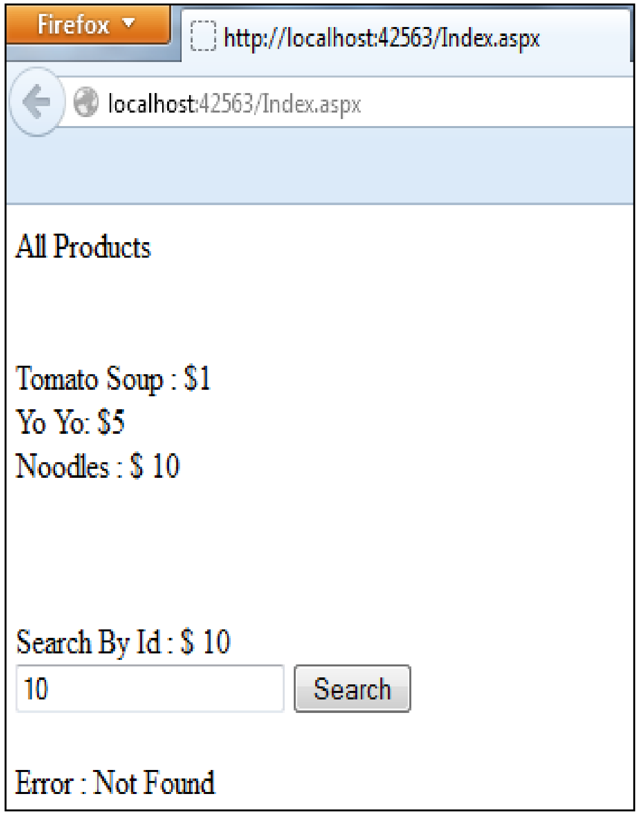
All Products

Tomato Soup : $1
Yo Yo: $5
Noodles : $ 10

Search By Id : $ 10
3 Search

Noodles : $ 10

Search Web Page


Firefox http://localhost:42563/Index.aspx
localhost:42563/Index.aspx

All Products

Tomato Soup : $1
Yo Yo: $5
Noodles : $ 10

Search By Id : $ 10
10 Search

Error : Not Found

Http Error Displayed

JSON

- JavaScript Object Notation stores data in an organized manner.
- This data can later be easily and logically accessed by humans as shown in the code snippets.

Code Snippet

```
{
"employees": [
{ "firstName":"Dan"
,
"lastName":"Brown"
},
{
"firstName":"Andy"
,
"lastName":"Garcia"
},
{
"firstName":"David"
,
"lastName":"Coleman
"  }
]
}
```

```
[DataContract]
    class Person
    {[DataMember]
     internal string
name;

     [DataMember]
     internal int age;
    }
```

```
Person p = new
Person();
//Set up Person
object...
MemoryStream stream1 =
new MemoryStream();
DataContractJsonSerial
izer ser = new
DataContractJsonSerial
izer(typeof(Person));
ser.WriteObject(stream
1, p);
```

REST

AJAX

- Representational State Transfer uses disturbed services and helps exchange data in such an environment.
- HTTP protocols, involving the CRUD operations, can be used to perform different operations in such a resource.
- CRUD - Create, Retrieve, Update, and Delete. Following table displays the HTTP protocols.

HTTP Protocol	Mapped to CRUD Operation	Uses
GET	R (Retrieve)	Retrieves required information from remote resource
POST	U (Update)	Updates current representation of data in the remote server
C (Create)	PUT	Creates a new entry for current data sent to the server
D (Delete)	DELETE	Deletes specified data from the remote server

AJAX Web Services

- Web services in the ASP.NET AJAX framework are called using `AutoCompleteExtender` control and JavaScript among many other tools.
- Creating AJAX-enabled service is very easy and is supported by ASP.NET Web services.
- The `WebMethod` attribute must be applied for the method to be called by Web service consumers.
- `ScriptMethod` attribute can also be used with the `UseHttpGet` property to call Web services as shown in the following code snippets.

Code Snippets

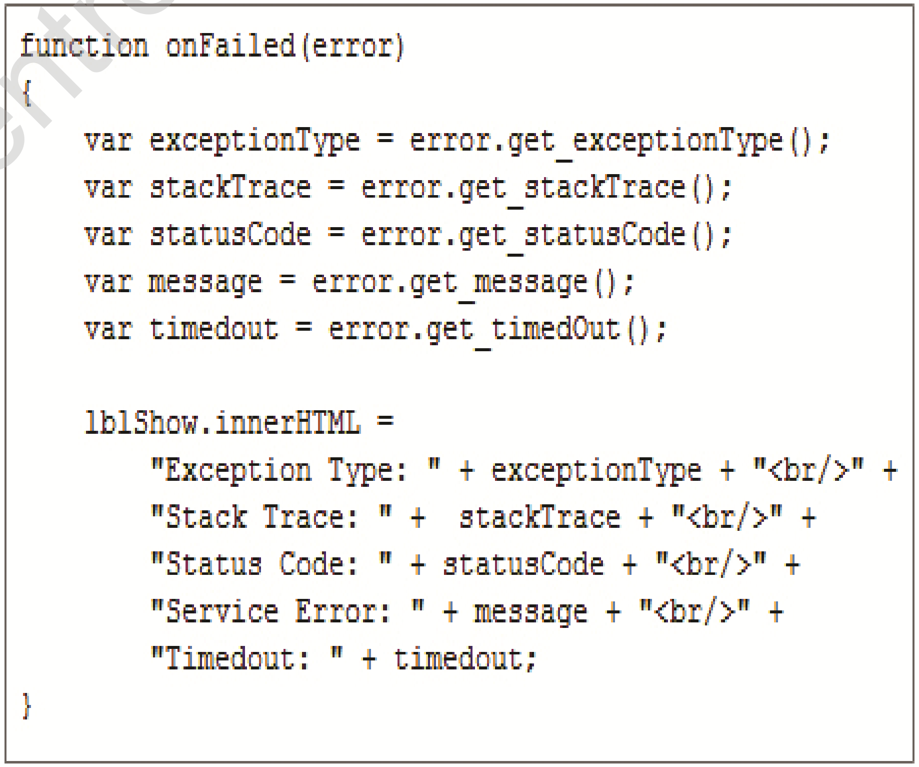
```
[WebMethod] public Customer[]
GetCustomersByCountry(string country) {    return
Biz.BAL.GetCustomersByCountry(country); }
```

```
[WebMethod] [ScriptMethod(UseHttpGet = true)] public
string HttpGetEcho(string input) {      return input;
}
```

Error Handling

- Failed callback is invoked if the call to Web services fails.
- The resultant parameter is a JavaScript error object.
- The second parameter is the method context.
- The third parameter is the method name.
- Error objects are resided in the class `Sys.Net.WebServiceError`.
- Following table displays the properties. Following figure displays the `onFailed` function.

Properties	Description
`exceptionType`	Returns the exception type
`stackTrace`	Returns the stack trace from the server
`statusCode`	Returns an HTTP error status code for the response
`Message`	Returns the error message from the server
`Timedout`	Returns true values if the Web Service call is timed out



```
function onFailed(error)
{
    var exceptionType = error.get_exceptionType();
    var stackTrace = error.get_stackTrace();
    var statusCode = error.get_statusCode();
    var message = error.get_message();
    var timedout = error.get_timedOut();

    lblShow.innerHTML =
        "Exception Type: " + exceptionType + "<br/>" +
        "Stack Trace: " +  stackTrace + "<br/>" +
        "Status Code: " + statusCode + "<br/>" +
        "Service Error: " + message + "<br/>" +
        "Timedout: " + timedout;
}
```

Summary

- Page methods use ASP.NET AJAX technologies to enable asynchronous communication with the server. They also expose Web methods to the client script.
- JavaScript Object Notation, JSON, is a subset of JavaScript that is present in text format.
- The await keyword is used to indicate that part of a code should asynchronously wait on another part of code.
- The async keyword shows a hint that the user can utilize to mark methods as task-based asynchronous.
- Async methods are non-blocking operations. An await expression in an async method doesn’t block the current thread while the awaited task is running.
- ASP.NET Web API is a framework that supports building of Web APIs on top of the .NET Framework.
- HTTP modules enable examining of incoming requests and taking actions based on the request.