



Introduction to ASP.NET



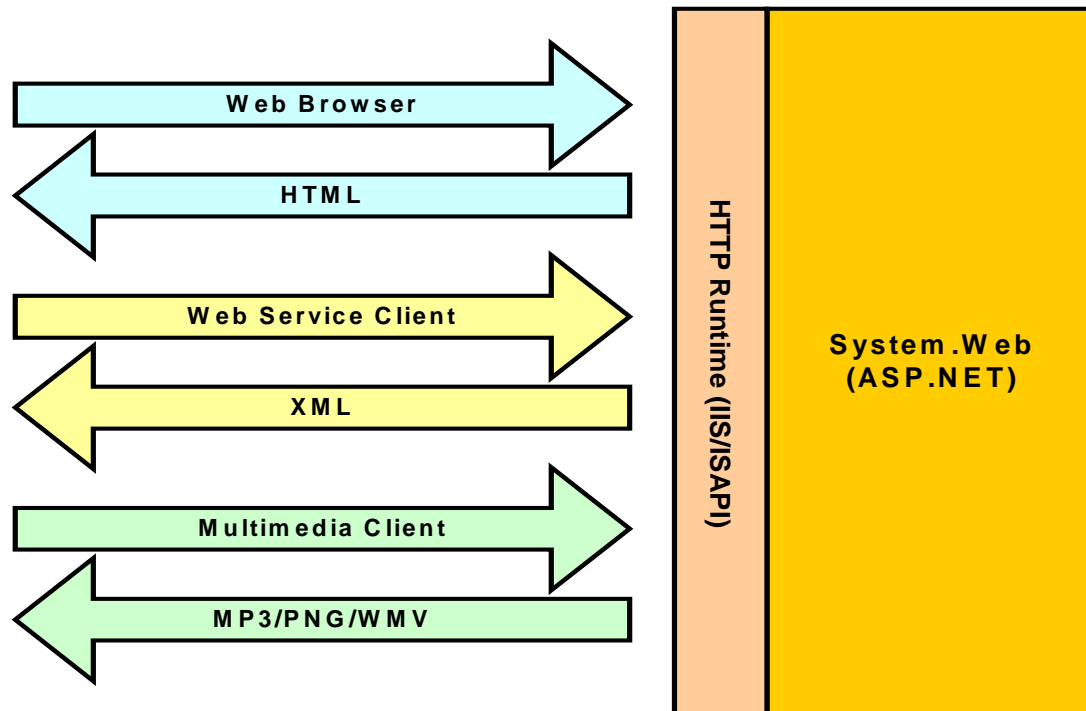
ASP.NET

- *ASP.NET is a managed framework that facilitates building server-side applications based on HTTP, HTML, XML and SOAP. To .NET developers, ASP.NET is a platform that provides one-stop shopping for all application development that requires the processing of HTTP requests.*



A platform on a platform

- ASP.NET is a managed framework that facilitates building server-side applications based on HTTP, HTML, XML and SOAP
 - ASP.NET supports building HTML-based applications with Web forms, server-side controls and data binding
 - ASP.NET supports building non-visual request handlers and Web services





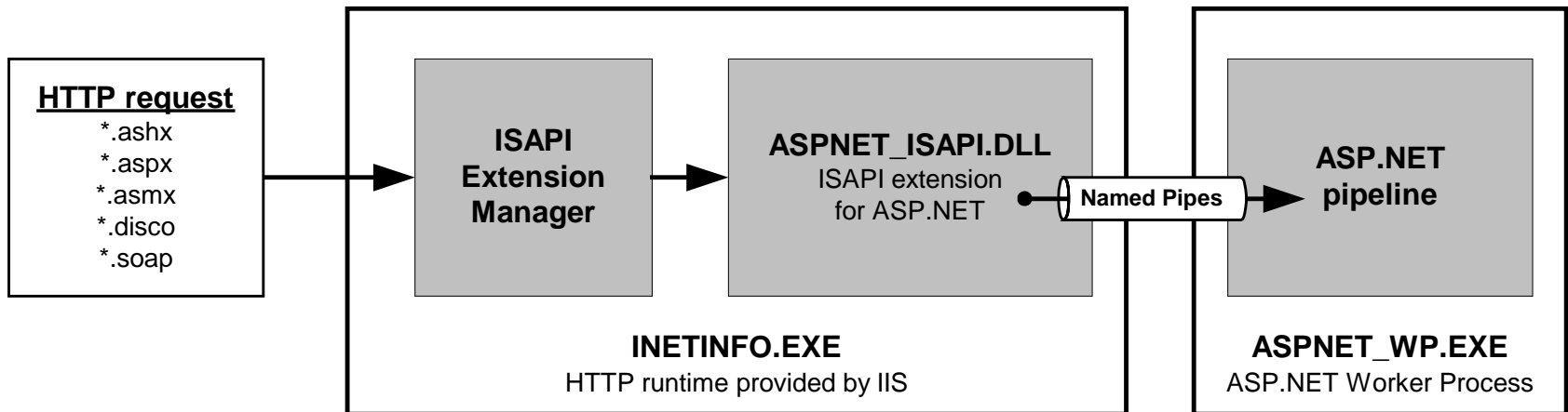
A new and better version of ASP

- ASP.NET is a much improved replacement for original ASP framework
 - All ASP.NET code is fully compiled prior to execution
 - ASP.NET doesn't require the use of scripting languages
 - ASP.NET allows for total separation of code from HTML
 - ASP.NET state management works in a Web farm environment



ASP.NET Internals

- ASP.NET uses a new CLR-based framework to replace ISAPI/ASP architecture
 - ASP.NET requests are initially handled by the HTTP runtime of IIS
 - ASP.NET ISAPI extension DLL redirects requests to ASP.NET worker process





HttpContext Properties

Type	Name	Description
HttpContext	Current (Shared)	Context for request currently in progress
HttpApplicationState	Application	Application-wide property-bag
HttpApplication	ApplicationInstance	Application context (modules)
HttpSessionState	Session	Per-client session state
HttpRequest	Request	The HTTP request
HttpResponse	Response	The HTTP response
IPrincipal	User	The security ID of the caller
IHttpHandler	Handler	The handler dispatched to the call
IDictionary	Items	Per-request property-bag
HttpServerUtility	Server	URL Cracking/Sub-handlers
Exception	Error	The 1st error to occur during processing



Programming against the `HttpContext` object

```
Imports System.Web

Public Module PipelineUtilities
    Sub SayHello()
        Dim ctx As HttpContext = HttpContext.Current
        If ctx Is Nothing
            '*** executing outside the scope of HTTP pipeline
        Else
            ctx.Response.Write("Hello, ")
            Dim name As String = ctx.Request.QueryString("name")
            ctx.Response.Output.WriteLine(name)
        End If
    End Sub
End Module
```



Integrated compilation support

- ASP.NET provides integrated compilation support to build source files into DLLs in just-in-time fashion
 - Auto-compilation supported for `.aspx`, `.ascx`, `.asmx` and `.ashx` files
 - The generated assembly is stored in subdirectory off of `CodeGen` directory
 - ASP.NET automatically references all assemblies in `\bin` directory
 - ASP.NET automatically references several commonly-used .NET assemblies
 - Other assemblies can be referenced through custom configuration
 - Shadow-copying allows DLLs to be overwritten while application is running



Hello, World using VB.NET in an ASP.NET page

```
<!-- MyPage1.aspx -->
<%@ page language="VB" %>
<html><body>
  <h3>My Page</h3>
  This page contains boring static content and
  <%
    Dim s As String = "<i>exciting dynamic content</i>"
    Response.Write(s)
  %>
</body></html>
```



System.Web.UI .Page

- An **.aspx** file is transformed into a class that inherits from **Page** class
 - **Page** class provides HTTP handler support
 - **Page** class members are accessible from anywhere inside **.aspx** file
 - Much of **Page** class dedicated to generation of HTML
 - Static HTML in **.aspx** file becomes part of page's **Render** method
 - Code render blocks in **.aspx** file become part of page's **Render** method
 - Members defined in code declaration blocks become members of page class
 - WebForms and server-side controls build on top of **Page** class architecture



Using reflection from an ASP.NET page

```
<%@ page language="VB" %>
<html><body>
  <%
    '*** note that this variable refers to subtype of Page
    Dim MyType As String = Me.GetType().ToString()
    Dim MyDadsType As String = Me.GetType().BaseType.ToString()

    '*** note that Response is a property of Page
    Me.Response.Write("page type: " & MyType & "<br>")
    Me.Response.Write("page's base type: " & MyDadsType & "<br>")
  %>
</body></html>
```



System.Web.UI.Page

```
Imports System.Web.UI

Class System.Web.UI.Page

    '*** functionality similar to what's in ASP
    ReadOnly Property Application As HttpSessionState
    ReadOnly Property Session As HttpSessionState
    ReadOnly Property Request As HttpRequest
    ReadOnly Property Response As HttpResponse
    ReadOnly Property Server As HttpServerUtility
    Property Buffer As Boolean
    Function MapPath(virtualPath As String) As String
    Sub Navigate(url As String)

    '*** functionality new to ASP.NET
    Property ClientTarget As ClientTarget
    ReadOnly User As IPrincipal
    ReadOnly Property IsPostBack As Boolean
    Protected Overridable Sub Render(writer As HtmlTextWriter)
    Event Load As EventHandler

End Class
```



Sample aspx file customizing Page

```
<!-- MyPage2.aspx -->
<%@ Page Language="VB" %>

<html><body>
<h2>Customer list</h2>
<ul>
<%
    '*** code render block becomes part of Render method
    Dim i As Integer
    For i = 0 to (m_values.Count - 1)
        Response.Write("<li>" & m_values(i).ToString & "</li>")
    Next
%>
</ul>
</body></html>

<!-- code declaration block -->
<script runat=server>
    '*** becomes a custom field in page-derived class
    Private m_values As New ArrayList()
    '*** becomes a custom event in page-derived class
    Sub Page_Load(sender As Object, e As EventArgs)
        If Not Page.IsPostBack Then
            m_values.Add("Bob Backerrach")
            m_values.Add("Mary Contrary")
            m_values.Add("Dirk Diggler")
        End If
    End Sub
</script>
```



ASP.NET directives

- ASP.NET supports several important page-level directives
 - `@Page` directive controls how the page file is compiled
 - `@Page` directive can take many different attributes
 - `@Assembly` directive replaces the `/r` command-line switch to `VBC.EXE`
 - `@Import` directive replaces the VB.NET `Imports` statement



@Page Directives

Imports AcmeCorp
'*** code goes here

```
<!-- MyPage4.aspx -->  
<%@Page Language="VB" Explicit="True" %>  
  
<%@Assembly name="MyLibrary" %>  
<%@Import namespace="AcmeCorp" %>  
  
<!-- code/content goes here -->
```

```
vbc.exe /optionexplicit+ /r:MyLibrary.dll HashedNameForMyPage4.vb
```

Directive Name	Description
@Page	Compilation and processing options
@Assembly	Replaces the VBC.EXE /r: switch
@Import	Replaces the VB.NET Imports statement



@Page Directives

Name	Description
Language	Programming language to use for <%
Buffer	Response buffering
ContentType	Default Content-Type header (MIME)
EnableSessionState	Turns session state on/off
EnableViewState	Turns view state on/off
Src	Indicates source file for code-behind
Inherits	Indicates base class other than Page
ErrorPage	URL for unhandled exception page
Explicit	Turns Option Explicit on/off
Strict	Turns Option Strict on/off
Debug	Enables/disables compiling with debug symbols
Trace	Enables/disables tracing
CompilerOptions	Enabled passing switches directly to VBC.EXE

```
<%@Page Language="VB" Buffer="true" Explicit="True" Strict="True" %>
```



Using a code-behind source file

- ASP.NET provides various code-behind techniques promote a separation of code and HTML
 - Types inside a code-behind file are accessible from a `.aspx` file
 - Pre-compiled code-behind files are placed in the `\bin` directory
 - Code-behind source files can compiled on demand using the `Src` attribute
 - Code-behind source files have extensions such as `.vb` or `.cs`
 - Compiling code-behind files on demand is useful during development
 - Compiling code-behind files on demand shouldn't be used in production



ASP.NET code can be partitioned using code-behind features

```
<!-- MyPage4.aspx -->
<%@ page language="VB" src="MyLibrary.vb" %>

<%@ Import Namespace="AcmeCorp" %>

<html><body>
  <h3>Splitting code between a .aspx file and a .vb file</h3>
  This text was generated by MyPage3.aspx<br>
  <%
    Dim obj As New ContentGenerator
    Response.Write(obj.GetContent())
  %>
</body></html>
```



ASP.NET code used as code-behind for an ASP.NET page

```
*** source file: MyLibrary.vb
Imports System

Namespace AcmeCorp
    Public Class ContentGenerator
        Function GetContent() As String
            Return "This text was generated by MyLibrary.vb"
        End Function
    End Class
End Namespace
```



web.config

- ASP.NET configuration is controlled by a XML-based file named `web.config`
 - `web.config` must be placed in IIS application's root directory
 - `compilation` section controls compiler stuff
 - `httpRuntime` section controls request execution
 - `pages` section controls how pages are compiled and run
 - Many other ASP.NET-specific section are available



An example web.config file

```
<?xml version="1.0" encoding="UTF-8" ?>

<configuration>

  <system.web>

    <compilation debug="true" explicit="true" strict="true">
      <assemblies>
        <add assembly="System.Runtime.Serialization.Formatters.Soap"/>
      </assemblies>
    </compilation>

    <httpRuntime
      executionTimeout="90"
      maxRequestLength="4096"
    />

    <pages
      buffer="true"
      enableSessionState="true"
      enableViewState="true"
    />

  </system.web>

</configuration>
```



Creating a custom HTTP handler

- **IHttpHandler** defines the semantics for an object that will serve as the end point for an HTTP request in the ASP.NET framework
 - **IHttpHandler** is defined by ASP.NET in the **System.Web** namespace
 - Any **IHttpHandler**-compatible object can be used as an ASP.NET handler
 - It's simple to write and configure a handler as a plug-compatible component
 - **web.config** file can be configured to dynamically load assembly/class
 - Use of **.ashx** files eliminates need to configure **web.config** file



IHttpHandler is implemented by objects that service HTTP requests

```
Namespace System.Web
```

```
Public Interface IHttpHandler
```

```
Sub ProcessRequest(context As HttpContext)
```

```
ReadOnly Property IsReusable As Boolean
```

```
End Interface
```

```
End Namespace
```



Classes that implement IHttpHandler are plug compatible

```
*** source file: MyLibrary1.vb
*** build target: MyLibrary1.dll

Imports System.Web

Public Class MyHandler1 : Implements IHttpHandler

    Sub ProcessRequest(context As HttpContext) _
        Implements IHttpHandler.ProcessRequest
        *** return friendly message in body of HTTP response
        context.Response.Output.Write("Hello world")
    End Sub

    ReadOnly Property IsReusable As Boolean _
        Implements IHttpHandler.IsReusable
        Return True
    End property

End Class
```

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <httpHandlers>
      <add verb="*"
          path="*"
          type="MyHandler1, MyLibrary1" />
    </httpHandlers>
  </system.web>
</configuration>
```



Using a .ashx file

```
<%@ webhandler language="VB" class="Class1" %>

'*** everything from here down is sent directly to VBC.EXE

Imports System.Web

Public Class Class1 : Implements IHttpHandler

    Sub ProcessRequest(context As HttpContext) _
        Implements IHttpHandler.ProcessRequest
        context.Response.Output.Write("Hello world")
    End Sub

    ReadOnly Property IsReusable As Boolean _
        Implements IHttpHandler.IsReusable
        Return True
    End property

End Class
```



Debugging ASP.NET applications

- ASP.NET provides extensive debugging support
 - `compilation` section in `web.config` must enable debugging
 - You can use either Visual Studio .NET or `DbgClr.exe`
 - Choose **Debug Processes** from the **Tools** menu
 - Attach debugger to ASP.NET worker process **ASPNET_WP.EXE**
 - Open source files inside debugger and set breakpoints
 - Execute page by submitting request as usual with browser



Creating HTML-based applications

- ASP.NET is more than just ASP 4.0
 - Unlike ASP, ASP.NET is HTML-aware
 - Lots of internal **Page** class code dedicated to forms/control processing
 - Code-behind techniques encourage better separation of code from HTML
 - Extensible, server-side control architecture
 - Server-side data binding model
 - Form validation architecture



The "Hello world" WebForm example

```
<%@Page language="VB" %>

<html><body>
  <form runat="server">
    <p><ASP:Textbox id="txtValue1" runat="server" /></p>
    <p><ASP:Textbox id="txtValue2" runat="server" /></p>
    <p><ASP:Button text="Add Numbers"
      runat="server"  OnClick="cmdAdd_OnClick" /> </p>
    <p><ASP:Textbox id="txtValue3" runat="server" /></p>
  </form>
</body></html>

<script runat="server">
  Sub cmdAdd_OnClick(sender As Object , e As System.EventArgs)
    Dim x, y As Integer
    x = CInt(txtValue1.Text)
    y = CInt(txtValue2.Text)
    txtValue3.Text = CStr(x + y)
  End Sub
</script>
```



Using code-behind inheritance

- An ASP.NET page can inherit from another custom **Page**-derived class
 - Code-behind inheritance allows you to remove all code from **.aspx** files
 - Your code goes into a custom class that inherits from the **Page** class
 - Custom page class goes into a separate source file
 - The **.aspx** file uses **Inherits** attribute inside **@Page** directive
 - Separation of code and HTML (i.e. presentation) significantly improve the maintainability of an HTML-based application



```
<%@Page Inherits="MyPageClass" Src="MyLibrary.vb" %>

<html><body>
  <form runat="server">
    <p><ASP:Textbox id="txtValue1" runat="server" /></p>
    <p><ASP:Textbox id="txtValue2" runat="server" /></p>
    <p><ASP:Button id="cmdAdd" text="Add Numbers"
      runat="server" OnClick="cmdAdd_OnClick" /> </p>
    <p><ASP:Textbox id="txtValue3" runat="server" /></p>
  </form>
</body></html>
```

```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls

Public Class MyPageClass : Inherits Page

  Protected txtValue1 As Textbox
  Protected txtValue2 As Textbox
  Protected txtValue3 As Textbox
  Protected WithEvents cmdAdd As Button

  Sub cmdAdd_OnClick(sender As Object, e As System.EventArgs)
    Dim x, y As Integer
    x = CInt(txtValue1.Text)
    y = CInt(txtValue2.Text)
    txtValue3.Text = CStr(x + y)
  End Sub

End Class
```



User versus Programmatic Interfaces

- ASP.NET applications may expose both user and programmatic interfaces
 - User interface expressed through HTML
 - Handled by `System.Web.UI.*`
 - Programmatic interfaces exposed via XML/SOAP
 - Programmatic interface handled by `System.Web.Services.*`



Web Methods

- Web services are .NET methods marked with the `WebMethod` attribute
 - Authored like any other method you might write, with a file extension of `.asmx`
 - ASP.NET takes care of mapping incoming HTTP requests onto calls on your object
 - .NET provides proxy classes on the client side that make calling web services as trivial as calling other functions
 - Client-side proxy takes care of generating HTTP request to server, and parsing response as return value



Web Method within a .asmx file

```
<%@ WebService language="C#" Class="Calculator"%>

using System;
using System.Web.Services;

public class Calculator : WebService {

    [ WebMethod ]
    public int Add( int x, int y ) {
        return x + y;
    }

    [ WebMethod ]
    public int Multiply( int x, int y ) {
        return x * y;
    }
}
```