**ASP.NET syntax for PHP developers**

[](http://msdn.microsoft.com/en-us/rampup/bb352986.aspx)

Ascend to new heights in your career

[**http://msdn.microsoft.com/rampup**](http://msdn.microsoft.com/rampup)

Contents

[Introduction 3](#_Toc226024783)

[ASP.NET Web Page Syntax Overview 3](#_Toc226024784)

[Example ASP.NET Web Page 4](#_Toc226024785)

[Comparing PHP to ASP.NET 9](#_Toc226024786)

[Architecture Comparison 9](#_Toc226024787)

[Feature Comparison 10](#_Toc226024788)

[Comparing Syntax and Common Tasks 14](#_Toc226024789)

[Variables 15](#_Toc226024790)

[Conditional Processing 18](#_Toc226024791)

[Setting and Retrieving Session Variables 25](#_Toc226024792)

[Data Caching and Page Caching 28](#_Toc226024793)

[XML Manipulation and Web Services 31](#_Toc226024794)

# Introduction

ASP.NET is a unified Web development model that includes the services necessary for you to build enterprise-class Web applications with a minimum of coding. ASP.NET is part of the .NET Framework, and when coding ASP.NET applications you have access to classes in the .NET Framework. You can code your applications in any language compatible with the common language runtime (CLR), including Microsoft Visual Basic, C#, JScript .NET, and J#. These languages enable you to develop ASP.NET applications that benefit from the common language runtime, type safety, inheritance, and so on.Visual Studio Web Development tools

## ASP.NET Web Page Syntax Overview

ASP.NET Web pages are created in a manner similar to static HTML Web pages (pages that do not include server-based processing), but they include extra elements that ASP.NET recognizes and processes when the page runs. The characteristics that distinguish ASP.NET Web pages from static HTML (or other) pages are as follows:

* A file name extension of .aspx instead of .htm, .html, or other file name extensions. The .aspx file name extension causes the page to be processed by ASP.NET. The mapping of file name extensions to ASP.NET is done in Internet Information Services (IIS). By default, .aspx pages are run by ASP.NET and .htm and .html pages are not.
* An optional @ Page directive or other directive, as appropriate to the type of page that you are creating.
* A **form** element that is configured correctly for ASP.NET. The **form** element is required only if the page contains controls whose values you want to use during page processing.
* Web server controls.
* Server code, if you add your own code to the page.
* If you want your pages to conform to XHTML standards, you must include additional elements, such as a DOCTYPE element.

### 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 an HTML page, along with elements that are specific to ASP.NET. The elements that are specific to ASP.NET are highlighted.

**VB**

<b><%@ Page Language="VB" %></b>

<html>

<b><script runat="server"></b>

<b> Sub Button1\_Click(ByVal sender As Object, \_</b>

<b> ByVal e As System.EventArgs)</b>

<b> Label1.Text = "Welcome, " & TextBox1.Text</b>

<b> End Sub</b>

<b></script></b>

<head <b>runat="server"</b>>

<title>Basic ASP.NET Web Page</title>

</head>

<body>

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

<h1>Welcome to ASP.NET</h1>

<p>Type your name and click the button.</p>

<p>

<b><asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></b>

<b><asp:Button ID="Button1" runat="server" </b>

<b>Text="Click" OnClick="Button1\_Click" /></b>

</p>

<p>

<b><asp:Label ID="Label1" runat="server"></asp:Label></b>

</p>

</form>

</body>

</html>

C#  
<b><%@ Page Language="C#" %></b>

<html>

<b><script runat="server"></b>

<b> void Button1\_Click(Object sender, EventArgs e)</b>

<b> Label1.Text = "Welcome, " + TextBox1.Text;</b>

<b> End Sub</b>

<b></script></b>

<head runat="server">

<title>Basic ASP.NET Web Page</title>

</head>

<body>

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

<h1>Welcome to ASP.NET</h1>

<p>Type your name and click the button.</p>

<p>

<b> <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></b>

<b> <asp:Button ID="Button1" runat="server" </b>

<b> Text="Click" OnClick="Button1\_Click" /></b>

</p>

<p>

<b> <asp:Label ID="Label1" runat="server"></asp:Label></b>

</p>

</form>

</body>

</html>

#### @ Directives

ASP.NET pages usually contain directives that allow you to specify page properties and configuration information for the page. The directives are used by ASP.NET as instructions for how to process the page, but they are not rendered as part of the markup that is sent to the browser.

The most commonly used directive is the @ Page directive, which allows you to specify many configuration options for the page, including the following:

* The server programming language for code in the page.
* Whether the page is a page with server code directly in the page, which is called a single-file page, or whether it is a page with code in a separate class file, which is called a code-behind page. In the previous example, the page is a single-file page; the code is directly in the page, and the **@ Page** directive does not include information about linked class files. For more information, see the "Server Code" section later in this topic.
* Debugging and tracing options.
* Whether the page has an associated master page and should therefore be treated as a content page.

If you do not include an **@ Page** directive in the page, or if the directive does not include a specific setting, settings are inherited the from the configuration file for the Web application (the Web.config file) or from the site configuration file (the Machine.config file).

In addition to including an **@ Page** directive, you can include other directives that support additional page-specific options. Other common directives include the following:

* @ Import   This directive allows you to specify namespaces that you want to reference in your code.
* @ OutputCache   This directive allows you to specify that the page should be cached, along with parameters for when and how long to cache the page.
* @ Implements   This directive allows you to specify that the page implement a .NET interface.
* @ Register   This directive allows you to register additional controls for use on the page. The **@ Register** directive declares the control's tag prefix and the location of the control's assembly. You must use this directive if you want to add user controls or custom ASP.NET controls to a page.

Certain types of ASP.NET files use a directive other than **@ Page**. For example, ASP.NET master pages use an @ Master directive, and ASP.NET user controls use an @ Control directive. Each directive allows you to specify different options that are appropriate for the file.

#### 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 so on. 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 .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 on the server only, 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.  
  
**VB**  
<%@ Page Language="VB" CodeFile="Default.aspx.vb" Inherits="Default" %>  
  
**C#**

<%@ Page Language="C#" CodeFile="Default.aspx.cs" 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.

## Comparing PHP to ASP.NET

### Architecture Comparison

As you will learn from the syntax and language comparison and the end of this paper, PHP and ASP.NET are relatively similar with analogous functionality and syntax. PHP, however, is very different from ASP.NET at a lower architectural level. PHP is based on a platform-independent processor/engine that parses PHP scripts and provides for database connections, Internet protocol compliance, and numerous other tasks common to most Web application platforms.

ASP.NET is a framework built upon a series of technologies such as the CLR and offers an extensive series of well-organized class libraries that provide for most every conceivable set of functionality that would be used in a Web application. It also allows for the easy and simple creation of components to extend the framework.

While PHP offers similar things, such as the PEAR libraries, PHP and ASP.NET are not truly analogous because the ASP.NET framework is built from the ground up on an OOP paradigm and OOP concepts; PHP is not. This difference is most apparent in the ways you access classes and objects in PHP and ASP.NET.

#### Object-Oriented Programming in PHP and ASP.NET

Both PHP and ASP.NET offer OOP paradigms to application development, but their support for various OOP concepts, such as encapsulation and polymorphism differ. For example, PHP only supports partial encapsulation (such as support for declaring methods and fields in the class) and partial polymorphism (no overloading, no abstraction). PHP also lacks support for such concepts and accessibility in that there is no concept of private, public, or protected functions in classes as well as the Overloading. While OOP purists may debate that ASP.NET and the various languages do not fully support every concept in the OOP paradigm, this is true of most languages considered OOP, such as C++ and Java.

This has both an upside and a downside. The downside is that for some Web developers there is a steeper learning curve for ASP.NET versus PHP, which offers a scripting paradigm that developers have traditionally employed for building Web sites. However, developers who have a background in OOP languages and/or Vwill find ASP.NET intuitively familiar and easy to learn.

The upside to ASP.NET's support of OOP concepts means that ASP.NET applications for the most part result in better designed code, have clear separation of content, logic, and data and thus are generally easier to support over the long term of a applications life cycle. In addition, ASP.NET's native support for enterprise technologies such as Message Queuing, Transactions SNMP, and Web Services, makes it simple to develop highly scalable and robust applications.

#### Compilation

PHP

When a PHP page is requested, the HTML and inline PHP script is compiled to Zend Opcodes. Opcodes are low-level binary instructions that will be used to serve the PHP page. After compilation, the Zend Engine runs the opcodes (similar to the way Java's runtime engine runs byte code), and then HTML is generated and served to the client.

There are a number of commercial products that can be used to speed up the execution of a PHP page by optimizing these opcodes. Other ways to increase performance of PHP scripts include caching the opcode and caching the generated HTML.

ASP.NET

When a request is made to IIS (Internet Information Services) or another Web server for an .aspx page (or any other extension supported by ASP.NET), the request is passed to ASP.NET for processing. If this is the first time the page has been requested, ASP.NET compiles the page to MSIL (Microsoft intermediate language). This MSIL code is then processed by the CLR (common language runtime) to machine code. Then the request is run using this compiled code. Subsequent requests are served from this same machine code assuming the page has not been modified.

It is important to note that the binary code that is generated by the CLR is already as optimized as possible; no add-on product is necessary to achieve maximum performance.

It is also important to note that everything in ASP.NET is compiled to machine code before being run. Even HTML text is converted to a string literal control and inserted in proper order into the control tree.

## Feature Comparison

**Table 1. Comparing the features of PHP and ASP.NET**

|  |  |  |
| --- | --- | --- |
| **Feature** | **PHP** | **ASP.NET** |
| Coding Language | C, C++ style scripting language with older ASP style mark up. Supports some OOP concepts. | Supports more than 25 languages, but the 2 that are most-commonly used are Visual Basic .NET and C#. Most developers pick one language but can consume components written in any of the other supported languages. |
| Compiled Application Logic | Compilable and can be run as a executable | Supported, in both dynamically-compiled and precompiled modes. |
| Full-Page Output Caching | No native support | Supported, caches different versions of the page based on one or more URL parameters, browser type, a custom function, or any combination. |
| Partial-Page Output Caching | No native support | Built-in support through use of User Controls. Data and other objects can be cached with sophisticated expiration rules using the Cache API. |
| Database Access | Has drivers for most databases on the market as well as open-source databases | Supports OLE-DB and ODBC directly, and includes native drivers for Microsoft SQL Server™ and Oracle. |
| Database Output | Datasets are returned as PHP variables and can be outputted like any other variable | Templated data binding to server-side controls for ease of development, or manual looping if that is preferred. |
| External Components | Can call a variety of packages and the Zend engine can be modified directly by the developer because it is open source. The programmers responsible for Zend are also working on allowing .NET objects to be called from PHP | Very good support for native C libraries and COM objects, as well as assemblies written in any .NET-compliant language, including Managed C++. No built-in support for CORBA objects or Java classes. |
| XML/XSLT | Supported in add-on packages and libraries | Comprehensive and easy-to-use support is provided for XML DOM, XSLT, validation, and lightweight stream-oriented parsing of XML documents. |
| XML Web Services | PHP packages are currently being developed to support Web services based on Apache AXIS engine as well as others. | The current release provides extensive and flexible standards-compliant support and makes it extremely easy to both publish and consume Web services. |
| Session State | Cookie based session management | Cookie-based or cookieless session state, using a single-server in-memory store, a centralized state server, or a database back-end. In addition, the extensible architecture allows for custom session-state modules to replace the built-in options. Cookieless sessions require only a configuration change to enable. |
| Built-In Functionality | PHP has built-in functions to cover many common tasks that a Web-based application may need to perform. It can also access Java class libraries with some extra work. | ASP.NET has direct access to the entirety of the .NET Framework class libraries, which encompass a vast amount of functionality. |
| Regular Expressions | Supports POSIX and Perl compatible regular expression syntax. | Supports Perl-5-compatible regular expressions, with additional features such as right-to-left matching, precompiled expressions, named groups, full Unicode support. Also allows the user to specify a function to be called during a regular expression replace operation. |
| Debugging | PHP does not offer extensive debugging, although various products from third parties and Zend allows for increased debugging and testing. | Includes extensive tracing and environment information that can be included in the page or displayed in a separate page. Microsoft Visual Studio® .NET allows for easy interactive debugging of pages as they execute, in addition to debugging of client-side scripting and SQL Server stored procedures. |
| Error Handling | Does not support error trapping but has various error-handling functionality and logging, | Supports structured exception handling (with the addition of a "finally" block for code that executes regardless of whether or not an error occurs), raising custom exceptions, and specifying custom error pages for different types of unrecoverable errors. |
| Image Manipulation | No built-in support, although third-party components are available. | Includes extensive image creation and manipulation facilities. |
| Code Re-Use | User-defined functions, ability to create Classes, and included files. | User Controls, Server Controls, custom classes, and included files. |
| Threading | PHP has a good threading model | Fully supported. Not only do simultaneous page requests happen in separate threads, but each page can (if needed) spawn its own threads to perform simultaneous or asynchronous work. |
| Data Caching | Has limited data native Data Caching | Includes an extensive cache API that allows nearly any type of data (including database query results) to be stored, with expiration based upon time, usage, or dependency upon a file or another cached item. Also allows for a user-defined function to be called when a given item is removed from the cache. |
| Internationalization | Full support for Unicode | Supports Unicode strings and various character encodings. Date, number, and currency functions are all culture-aware and alter their output depending on what the current culture is set to (rather than requiring different functions to be called). Support for using resource files to dynamically localize an application is included. |
| SMTP, HTTP, FTP, POP3 | Native support for a wide variety of Internet protocols | SMTP support depends upon the IIS SMTP Service. HTTP has very good support. FTP and POP3 are not supported in .NET, but free and commercial third-party components are available for this functionality. |
| Integrated Development Environment | Numerous development tools with a broad range of capabilities are available at price ranges from free to several hundred dollars. | There is a free tool for ASP.NET development available from Microsoft called ASP.NET Web Matrix  The most popular tool is Visual Studio.NET which has full support for all .NET languages, database tools for creation of SQL and testing databases, Web Design tools, integration with version control, advanced debugging and numerous other features for a full list see the MSDN® Visual Studio Developer Center.  Other tools, including Borland C# Builder and Macromedia Dreamweaver MX, also support ASP.NET. |
| Web Server Support | Supports almost every Web server. | Supports IIS and Covalent's commercial version of Apache 2.0. |
| Operating System Support | There are ports to almost every commonly used OS including Microsoft Windows®, Mac, OS X, Amiga, Solaris, Free BSD, Linux, AIX and more. | Currently supports Windows 2000, Windows XP, and Windows Server 2003. |

### Comparing Syntax and Common Tasks

The following sections provide comparisons between PHP and .NET syntax as well as how to accomplish some of the more common programming tasks.

#### Comments

PHP allows you to insert comments in your code using C, C++ and Unix shell-style syntax, and anything within those comment indicators will not be executed.

In general, to comment out Visual Basic .NET code in ASP.NET you just need to use <%-- to open a comment block and --%> to close the block.

Code Sample 1 shows comments in each environment.

**Code Sample 1. Server-side comments in PHP**

/\*

This is a block of text

That has been commented out

\*/

**Code Sample 1. Server-side comments in ASP.NET**

<%--

This is a comment.

--%>

### Variables

While PHP and Visual Basic .NET have similar language constructs, they are very different syntax for them. Since Visual Basic .NET is built upon an OOP model, variable declaration is much more rigorous than in PHP where a variable is declared simply by adding a dollar sign ($) before the variable name.

In Visual Basic .NET you declare a variable by specifying its name and characteristics. The declaration statement for variables is the Dim keyword. Its location and contents determine the variable's characteristics. Variables have levels such as local and module, data types, lifetimes and finally accessibility.

While this approach may seem more complex at first than variable assignment in PHP it actually makes a developer's life easier. ASP.NET focuses on helping developers build robust applications—and specifying data types makes tasks such as variable clean up, debugging, exception and error handling, and code maintenance much easier.

Code Sample 2 shows examples of declaring variables in each environment.

**Code Sample 2. Variable declaration in PHP**

$head\_count

$foo

$X

$obj

**Code Sample 2. Variable declaration in Visual Basic .NET**

Dim head\_count As Integer

Dim foo As String

Dim X As Date

Dim Obj As object

#### Declaring Data Types

The AS clause in the declaration statement allows you to define the data type or object type of the variable you are declaring. You can specify any of the following types for a variable:

* An elementary data type, such as Boolean, Long, or Decimal
* A composite data type, such as an array or structure
* An object type, or class, from Visual Basic or another application, such as Label or TextBox

You can declare several variables of the same type in one statement without having to repeat the data type. In the following statements, the variables numStudents, numGTA and numProfessors are declared as type Integer:

Dim numStudents, numGTA , numProfessors As Integer

' All three are Integer variables.

#### Declaring Lifetime

The lifetime of a variable is the period of time during which it is available for use. A local variable declared with a Dim statement exists only as long as its procedure is executing. When the procedure terminates, all its local variables disappear and their values are lost.

The concept of lifetime is extremely useful in that it allows developers to build applications without having to concern themselves with many issues that occur in large-scale applications such as efficient memory management. By selecting the correct lifetime for a variable you can allow .NET to perform clean up operations on variables that are not being used.

#### Declaring Scope

A local variable is one that is declared within a procedure (a **procedure** is analogous to a function). A non-local variable is one that is declared outside a procedure, but within a class or structure.

In a class or structure, the category of a non-local variable depends on whether or not it is shared. If it is declared with the Shared keyword, it is a shared variable, and it exists in a single copy shared among all instances of the class or structure. Otherwise it is an instance variable, and a separate copy of it is created for each instance of the class or structure. A given copy of an instance variable is available only to the instance for which it was created.

The scope of a variable is the set of all code that can refer to it without qualifying its name. A variable's scope is determined by where the variable is declared. Code located in a given region can use the variables defined in that region without having to qualify their names. When declaring scope, the following rules apply:

* The scope of a shared or instance variable is the structure or class in which it is declared.
* The scope of a local variable is the procedure in which it is declared.

However, if you declare a local variable within a block, its scope is that block only. A local variable is active within the defining control block. The control block can be a procedure, an if statement, a loop statement and so on.

#### Declaring Accessibility

.NET supports the idea of accessibility to variables, which allows you, the developer to control what code can access specific variables. For example if you wanted to set some constants for a formula and make sure that your constant never gets changed by other code outside of its class you could declare that variable private like this:

Private myConstant As Integer

A variable's accessibility is determined by which keyword or keywords—Dim, Public, Protected, Friend, Protected Friend, or Private—you use in the declaration statement. In general you will only use public and private in your development.

You can declare a module, structure, class, or instance variable with any of these keywords. Within a procedure, only the Dim keyword is allowed, and the accessibility is always private.

#### Output

The typical way of outputting data in PHP is through the echo() language construct. The closest analogue to this in ASP.NET is the Response.Write() method, or the <%= %> construct, which is simply shorthand for Response.Write(). Code Sample 3 shows basic syntax for writing a value to the page.

**Code Sample 3. Basic output in PHP**

<?php

$hello = "hi how are you\n";

echo $hello;

?>

**Code Sample 3. Basic output in Visual Basic .NET**

<%

Dim Hello As String = "Hi how are you" & vbcrlf

Response.Write(Hello)

%>

However, these methods for sending output to the browser exist primarily for backwards compatibility with classic ASP. ASP.NET's new control-based, event-oriented model allows for data to be output to the browser by simply setting properties on server controls. This technique allows for clean separation of layout and code and can make maintenance easier, requiring significantly less code in complex situations than PHP.

<script language="VB" runat="server">

Sub Page\_Load(sender As Object, e As EventArgs)

TheDate.Text = DateTime.Now

End Sub

</script>

The current date is: <asp:Label id="TheDate" runat="server"/>

This example declares a server-side Label control called TheDate, and during the page's Load event, sets the Text property of the label to the current date and time. The HTML output of this code is identical to the other two versions, except that the Label control renders itself as a span tag containing whatever was set as the label's text.

### Conditional Processing

IF/ELSE

PHP has several conditional processing expressions such as for, while, switch, and foreach but the most common is the if/else expression. Visual Basic .NET has very similar constructs with similar syntax. Code Sample 4 providesa comparison of equivalent conditional logic in PHP and Visual Basic .NET.

**Code Sample 4. Basic conditional logic in PHP**

if ($a > $b) {

print "a is bigger than b";

} elseif ($a == $b) {

print "a is equal to b";

} else {

print "a is smaller than b";

}

**Code Sample 4. Basic conditional logic in Visual Basic .NET**

If a > b

Response.Write ("a is bigger than b")

ElseIf a = b Then

Response.Write ("a is equal to b")

Else

Response.Write ("a is smaller than b")

End If

Switch

Switch statements are common language constructs for most programming languages when you wish to test a single expression for multiple values. They are commonly used to replace if statements that contain multiple elseif/else blocks.

Code Sample 5 shows a comparison between PHP's switch statement and Visual Basic's Select Case statement.

**Code Sample 5. A switch statement in PHP**

switch ($i) {

case 0:

print "i equals 0";

break;

case 1:

print "i equals 1";

break;

case 2:

print "i equals 2";

break;

default:

print "i is not equal to 0, 1 or 2";

}

**Code Sample 5. A Select Case statement in Visual Basic .NET**

Select Case Number i

Case 0

description = "0"

Wesponse.Write ("i equals 0")

Case 1

description = "1"

Response.Write ("i equals 1")

Case 2

description = "2"

Response.Write ("i equals 2")

Case Else

description = " i is not equal to 0, 1 or 2"

Response.Write ("i is not equal to 0, 1 or 2 ")

End Select

Looping

Another extremely common control construct is the loop. There are several different commonly recognized types of loops that both PHP and .NET support.

**Code Sample 6. For loop in PHP**

for ($i = 1; $i <= 100; $i++) {

print $i;

}

**Code Sample 6. For loop in Visual Basic .NET**

Dim sum As Integer = 0

Dim counter As Integer

For counter = 1 To 100 Step 5

sum += counter

Next

For i = 1 To 100

Response.Write (i)

Next I

In Visual Basic, this type of loop is known as a For...Next loop, and in PHP it is simply called a For loop. In this example, the += operator is used as shorthand for sum = sum + counter. In PHP one may break out of a loop with the Break; statement. A For...Next loop can be broken out of with the Exit For statement.

Conditional Loop

A conditional loop iterates over a set of instructions as long as a condition evaluates to true. Code Sample 7 shows an example of a basic conditional loop in each language.

**Code Sample 7. Conditional loop in PHP**

$i = 1;

while ($i <= 10):

print $i;

$i++;

endwhile;

**Code Sample 7. Conditional loop in Visual Basic .NET**

Dim counter i As Integer = 1

Do While counter i <= 10

Response.Write(i)

counter i += 1

Loop

In Visual Basic, this type of loop is known as a Do...Loop statement, or a while loop. PHP also supports Do...While loops, which are very similar to while loops, except the truth expression is checked at the end of each iteration instead of at the beginning. The main difference from regular while loops is that the first iteration of a Do...While loop is guaranteed to run (the truth expression is only checked at the end of the iteration), whereas it may not necessarily run with a regular while loop. (The truth expression is checked at the beginning of each iteration; if it evaluates to FALSE right from the beginning, the loop execution ends immediately.)

Here is an example of this in PHP:

$i = 0;

do {

print $i;

}

while ($i>0);

This loop runs exactly one time, since after the first iteration, when the truth expression is checked, it evaluates to FALSE ($i is not bigger than 0) and the loop execution ends.

In Visual Basic .NET, you can do much the same thing:

Dim counter i As Integer = 0

Do

Response.Write (i)counter

Loop While counter i > 0

However, Visual Basic .NET supports a built-in looping capability that PHP does not support, which is to evaluate a condition until it is true.

Foreach Loop

PHP 4 (not PHP 3) includes a foreach construct, much like ASP.NET and some other languages. This simply gives an easy way to iterate over arrays. foreach works only on arrays and will issue an error when you try to use it on a variable with a different data type or on uninitialized variables. In Visual Basic .NET, the equivalent is the For Each...Next statement. Code Sample 8 shows an example of looping over an array in each language.

**Code Sample 8. foreach loop in PHP**

$i = 0;

foreach($a as $v) {

print "\$Key[$i]$v \n";

$i++;

}

**Code Sample 8. For Each loop in Visual Basic .NET**

For Each v In a

Response.Write a(v) (v & vbcrlf)

Next

Arrays

Arrays in PHP function very differently than arrays in Visual Basic .NET. In PHP, arrays are actually associative arrays but can be used like index or associative arrays. In Visual Basic .NET arrays are index arrays. Visual Basic .NET does not support associative arrays as such (although you can build them your self—see [collections](http://msdn.microsoft.com/en-us/library/aa479002.aspx#aspnet-migratingphp-aspnet_collections) below). Other ASP.NET languages do support these types of arrays but Visual Basic .NET does not, and this can provide some challenges to PHP developers who are not used to working with the more common index array and might wish to model an associative array in Visual Basic .NET. Code Sample 9 shows an example of a simple array in PHP and Visual Basic .NET.

**Code Sample 9. Example of array creation in PHP**

$a = array (0,1,2);

**Code Sample 9. Example of array creation in Visual Basic .NET**

Dim MySingleArraya() As Integer = New Integer (2) {0,1,2}

In a Visual Basic .NET array, variables are declared the same way as other variables, using the Dim statement. You follow the variable name with one or more pairs of parentheses to indicate that it is an array rather than a scalar (a variable containing a single value).

Visual Basic .NET arrays also must be declared as holding a specific type of data when they are created. If the type specified is Object (the generic type underlying every object type in .NET), then the array can hold any type of data, but values must be converted to their original type when they are retrieved from the array.

Visual Basic .NET arrays can be either nested arrays-of-arrays, or multi-dimensional arrays. There are a variety of functions to manipulate arrays that are comparable to PHP, with one exception. Because Visual Basic .NET has no notion of associative arrays, there are no functions to access or index or doing anything by an array's "Key," which does not exist in Visual Basic .NET.

While we have made several references to Visual Basic .NET not supporting associative arrays, it is possible to create what is called a collection as an alternative to an array. Collections work somewhat like associative arrays in that they can be used to solve similar problems.

In some circumstances, it can be more efficient to store items in a collection than in an array.

You might want to use a collection if you are working with a small, dynamic set of items To create a collection all you need to do is declare and instantiate a Collection variable as shown in the sample code below:

Dim myCollection As New Collection()

You then can use the Add method to add members to the collection. In this example, we create four strings and add them to the collection. A unique String value may optionally be added as the key for the members of your collection. This value is passed to the collection as the second argument of the Add method.

Dim w, x, y, z As String

w = "key1"

x = "key2"

y = "key3"

z = "key4"

myCollection.Add(w, "1")

myCollection.Add(x, "2")

myCollection.Add(y, "3")

myCollection.Add(z, "4")

While this may seem a lot like creating an associative array in PHP, a collection is a very different animal in that it is an object in and of itself. For PHP developers moving to ASP.

Managing State

A common task in any Web application is the management of state, which is usually done using cookies or an application state management construct such as Session variables. Visual Basic .NET has similar methods to PHP for handling state.

Setting and Retrieving Cookies

Setting cookies in both environments is relatively trivial. Code Sample 10 showsan example of writing and then reading a cookie in each language.

**Code Sample 10. Setting and retrieving cookies in PHP**

<?php

$value = 'something from somewhere';

setcookie ("TestCookie", $value,time()+3600); /\* expire in 1 hour \*/

?>

/\* and to retive the set cookie \*/

<?

echo $\_COOKIE["TestCookie"];

?>

**Code Sample 10. Setting and retrieving cookies in Visual Basic .NET**

Dim value as string = "something from somewhere"

Dim myCookie As New HttpCookie = New HttpCookie("Something from somewhereTestCookie")

Dim now as DateTime = DateTime.Now

myCookie.Value = now.ToString()value

myCookie.Expires = now.AddHour(1)

Response.Cookies.Add(myCookie)

'and to retrieve the set cookie

Response.Write(Request.Cookies["What we setTestCookie"].Value)

### Setting and Retrieving Session Variables

Session variables in ASP.NET are very similar to PHP session variables. Session variables in both environments provide handling and cookie manipulation for you to provide persistence through a Web application visit.

The last important difference is that when a value is retrieved from the ASP.NET session object, it is returned as the generic System.Object type, which can hold any type of data. This value must be converted back to its original underlying type before it can be used. Code Sample 11 shows someexamples of session variable usage.

**Code Sample 11. Session variable usage in PHP**

<?PHP

session\_start();

session\_register('today');

$today = getdate();

?>

<?= $today ?>

**Code Sample 11. Session variable usage in Visual Basic .NET**

Session("Today") = DateTime.Now

Dim today As Date

today = CDate(Session("Today"))

Response.Write(today)

Response.Write (session("Today"))

ASP.NET also has another form of state management called Application State that is analogous to session variables but persists for the lifetime of an application. This allows you to store various things, such as configuration information or database connection strings that would not change while the application is running.

Regular Expressions

ASP.NET supports most of the popular features of other regular expression implementations such as those in Perl and awk. It is designed to be compatible with Perl 5 regular expressions. ASP.NET also supports regular expression features not yet seen in other implementations, such as right-to-left matching and on-the-fly compilation. Since ASP.NET is compatible with Perl regular expressions and since most PHP developers use Perl-compatible regular expressions, there is usually no need for translating your syntax from one form to the other.

Exception Handling

The ASP.NET framework includes support for structured exception handling following familiar language constructs Try/Catch, which provides the ability to catch exceptions that may arise in code. This is something that is missing in PHP and will be added in PHP 5.

Below is an example of how this is done in Visual Basic .NET:

Try

' code that might cause an error here

Catch e As ExceptionType

' code to handle the error

' Optional: More Catch blocks here

Finally

' code that is always executed

End Try

It is worth noting that a Try block can have either one or more Catch blocks, or a Finally block, or both. In other words, in situations in which you know that there is no way to correct for an error, but you still need to clean up some objects regardless of whether an error occurs or not, you can use a Try...Finally block with no Catch statement.

Querying a Database

In PHP there are generally two common ways to access a database: by using a database specific extension or by using the database independent PEAR DB library.

In ASP.NET, database access is performed through a set of objects known as ADO.NET, which serves much the same function as the PEAR DB library. The actual execution of a database query is accomplished with a set of connection, command, parameter, and data-adapter objects. There are multiple versions of each of these objects, depending on the type of database being accessed. For example, there is a set for databases with OLE-DB drivers (such as Microsoft Access), as well as a set for databases that have ODBC drivers but lack OLE-DB drivers. There are also specialized data providers for both Oracle and Microsoft SQL Server that are optimized to provide high-performance access to each of those specific databases. Third parties also provide support for other databases such as MySQL. The examples in this section will use the SQL Server objects, as that is one of the most-commonly used databases with ASP.NET.

System.Data, System.Data.SqlClient, and System.Data.oledb are the namespaces that define data base access in ADO.NET. To give your page access to the classes, you need to import the System.Data and [System.Data.SqlClient](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.aspx) namespaces into your page.

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.SqlClient" %>

Code Sample 12 shows an example of executing a query in each language. With PHP we have shown a connection using PEAR, which is not only one of the most popular methods of connecting to a DB, it is the most analogous to ADO.NET.

**Code Sample 12. Executing queries in PHP**

<?php

//connect

require\_once('DB.PHP');

$db=DB::connect ("mysql://mydbvie w:user@localhost/mydb");

if (DB::iserror($db)) {

die($db->getMessage());

$sql = "select \* from mytable";

$q= $db->query($sql);

if (DB::iserror($q)) {

die($q->getMessage());

}

<tr>

<td><?= $row[0] ?></td>

<td><?= $row[1] ?></td>

<td><?= $row[2] ?></td>

</tr>

**Code Sample 12. Executing queries in Visual Basic .NET**

<script runat="server">

Sub Page\_Load(Sender As Object, E As EventArgs)

Dim myConnection As New SqlConnection("server=(local)\NetSDK;database=mydb;Trusted\_Connection=yes")

Dim myCommand As New SqlDataAdapter("select \* from mytable", myConnection)

Dim ds As New DataSet()

myCommand.Fill(ds, "myDataset")

myDataGrid.DataSource = myDataset

myDataGrid.DataBind()

End Sub

</script>

<%--- outputting the resutlt ---%>

<form runat="server">

<asp:DataGrid id="myDataGrid" runat="server" />

</form>

In PHP, your query result is stored in a variable called a **result set**, while in ADO.NET it is called a Dataset object. **result set** is a read-only view of the data returned while the .NET Dataset is actually an in-memory and read-write view of the data allowing .NET developers to easily manipulate data returned from a data source.

When outputting data, ASP.NET offers several methods for display of data to the user or client. The first is similar to PHP, which is to loop through the result set using a SQLDataReader object to write out the data we wish to display from the query. The more common way, which does not have an analog in PHP, is ASP.NET's data binding. This allows developers to build User Interface and Display controls that can be used and reused throughout an application and allows for greater abstraction of display from data and logic. Data binding's flexible syntax allows you to bind not only to data sources, but also to simple properties, collections, expressions, and even results returned from method calls.

To use data binding, you need to assign some data source such as query results to the DataSource property of a data-aware server control (such as the DataGrid). Optionally, you can provide some additional formatting information for each column and call the DataBind() method. The server control will take care of the rest.

For example, in Code Sample 12 we used the data binding syntax to output the result of our query like this:

<%--- In page load event

---%>

myDataGrid.DataSource = myDataset

myDataGrid.DataBind()

<%--- outputting the resutlt ---%>

<asp:DataGrid id="myDataGrid" runat="server" />

Data-aware server controls will provide additional functionality, such as support for paging or in-line editing of the data being displayed. For more information and examples, please refer to [Data Binding Server Controls](http://samples.gotdotnet.com/QuickStart/aspplus/default.aspx?url=/quickstart/aspplus/doc/webdatabinding.aspx).

### Data Caching and Page Caching

Caching frequently accessed data can dramatically improve the response time of a site, as the page processing doesn't need to wait on a database query. Caching the HTML generated by a page request can improve response time even more, as the cached page doesn't need to be processed at all. Both environments provide support for both caching strategies; however, ASP.NET has many more methods for caching and managing data than PHP allowing developers to pick which method and strategy suites the needs of their application performance need.

##### Page Caching

Caching the HTML output of a page request is a common method of reducing load on a Web application. PHP does not natively support Page Caching, but it can be performed programmatically or by downloading third party packages. Usually page caching is performed on the server in numerous ways, from caching the compiled code to actually writing out the output of the page to a separate file that is updated whenever the code is updated.

In ASP.NET, page caching can be performed via either the low-level OutputCache API or the high-level @ OutputCache directive. When output caching is enabled, an output cache entry is created on the first GET request to the page. Subsequent GET or HEAD requests are served from the output cache entry until the cached request expires.

The output cache respects the expiration and validation policies for pages. If a page is in the output cache and has been marked with an expiration policy that indicates that the page expires 60 minutes from the time it is cached, the page is removed from the output cache after 60 minutes. If another request is received after that time, the page code is executed and the page can be cached again. This type of expiration policy is called absolute expiration—a page is valid until a certain time.

In addition to output caching an entire page, ASP.NET provides a simple way for you to cache just specific portions of a page, which is called fragment caching. You delineate regions of your page with a user control, and mark them for caching using the @ OutputCache directive introduced in the previous section. This directive specifies the duration (in seconds) that the output content of the user control should be cached on the server, as well as any optional conditions by which it should be varied.

Data Caching

There are a variety of ways to cache query results in PHP programmatically but not natively to the environment. Building Data Caching classes or systems in PHP can be done simply for small amounts of information with Session variable and/or cookies or for larger and more complex information by building your own Data caching classes. The problem is that when you are working with large types of complex data this can be inefficient, error prone, and somewhat complex to program.

ASP.NET offers a system-wide method for caching data (DataSets, arrays, collections, XML objects, and so on.) through the Page.Cache object. For applications that need more sophisticated functionality, ASP.NET cache supports three specific types of cache: expiration, scavenging, and file and key dependencies.

* An expiration type allows developers control when a cached item expires. This can be defined as a specific time, such as 01:00, or it can be relative to an item's last use, such as expire 20 minutes after the item was last accessed. After an item has expired, it is removed from the cache and future attempts to retrieve it return the null value unless the item is reinserted into the cache.
* A scavenging type cache attempts to remove infrequently used or unimportant items when memory becomes scarce. Developers have the power to control how the scavenging occurs and can provide hints to the scavenger when items are inserted into the cache that indicate the relative cost of creating the item and the relative rate at which the item must be accessed to remain useful.
* A file and key dependencies type allow the validity of a cache item to be based on an external file or on another cache item. If a dependency changes, the cache item is invalidated and removed from the cache. An example of how you might use this functionality in an application is if you had a large report that is periodically updated and downloadable to your staff. The application processes the data in the file and the report. The application caches that data and inserts a dependency on the file from which the data was read. When the file is updated, the data is removed from the cache and the application can reread it and reinsert the updated copy of the data.

ASP.NET Data Caching provides programmers many different methods to manage their applications and make them more responsive and efficient. For more information, see the Cache Class documentation for the Cache object.

Sending E-mail

Both PHP and ASP.NET have built in support for doing e-mail programmatically. To send e-mail with ASP.NET in this example you need to setup the IIS SMTP service, which must be installed because the built-in mail objects in .NET depend on objects included with the service. .NET though allows you to work with any SMTP server or mail server like PHP. Code Sample 13 compares basic syntax of each environment.

**Code Sample 13. Sending e-mail in PHP**

$to = "test@atnoaddress.com";

$from = "me@nosuchaddress.com"; $subject = "hi";

$message = "just wanted to say hi";

mail($to,$subject,$message, $from)

**Code Sample 13. Sending e-mail in Visual Basic .NET**

Dim myMail As MailMessage = New MailMessage()

myMail.From = "me@nosuchaddress.com"

myMail.To = "test@atnoaddress.com"

myMail.Subect = "hi"

myMail.Body = "just wanted to say hi"

SmtpMail.Send(myMail)

### XML Manipulation and Web Services

Built-in support for parsing and manipulating XML in PHP is rather poor. While developers can use it for parsing and traversing XML, it lacks support for DOM parsing, which while slower than PHP's SAX parser, is much easier to work with. PHP also does not support the ability to natively validate XML documents against a DTD or XML SCHEME, and PHP does not support XSL/XSLT, as well as numerous other technologies that are common to many Web application products on the market. While there are numerous PHP packages that allow PHP to accomplish many XML-related tasks .NET and ASP.NET have extensive built-in support for working with XML. XML is one of the technologies at the heart of the .NET platform.

The .NET Framework has extremely comprehensive support for all XML recommendations as defined by the W3C and supports XSL/XSLT, XPath, XQuery, as well as a host of other technologies, such as UDDI, WSDL, SOAP for Web services.

While it is possible to create XML-RPC type mechanisms in PHP, it is much harder to create Web services, which allow developers to exchange data and procedures using common protocols and standards to provide for discovery, data binding, and description. .NET has extensive support for Web services and related technologies, such SOAP, WSDL and UDDI. .NET also makes the creation and development of Web services a trivial matter for the developer. For example, here is code to create a simple hello world Web service:

<%@ WebService Language="VB" Class="HelloWorld" %>

Imports System

Imports System.Web.Services

Public Class HelloWorld :Inherits WebService

<WebMethod()> Public Function SayHelloWorld() As String

Return("Hello World")

End Function

End Class

The .NET Framework SDK allows you to generate your proxy classes using the command-line Web Services Description Language tool (WSDL.exe). To create a proxy class called HelloWorld.cs for the above example, you could enter:

WSDL http://someDomain.com/someFolder/HelloWorld.asmx?WSDL

This class would look very similar to the class created in the previous section. It would contain a method called SayHelloWorld that returns a string. Compiling this proxy class into an application and then calling this proxy class's method results in the proxy class packaging a SOAP request across HTTP and receiving the SOAP-encoded response, which is then marshaled as a string.

From the client perspective, the code would be simple, as shown in the following example:

Dim myHelloWorld As New HelloWorld()

Dim sReturn As String = myHelloWorld.SayHelloWorld()

And that's all there is to creating a simple Web service. For more information on XML in general and Web services in specific.