**Standards Document**

**For**

**ASP.NET Coding Standards and Guidelines**

**CONTENTS**

[1 Introduction 6](#_Toc313960049)

[1.1 Audience 6](#_Toc313960050)

[1.2 Scope of the document 6](#_Toc313960051)

[1.3 Scope Exclusions 6](#_Toc313960052)

[1.4 Document Conventions 7](#_Toc313960053)

[1.5 Assumptions 7](#_Toc313960054)

[1.6 Glossary of Terms 7](#_Toc313960055)

[2 Coding Standards 8](#_Toc313960056)

[2.1 File Naming Conventions 8](#_Toc313960057)

[2.2 File Header Information 8](#_Toc313960058)

[2.3 Control Naming Conventions 9](#_Toc313960059)

[3 ASP.NET Programming Guidelines 10](#_Toc313960060)

[3.1 Exception Handling 10](#_Toc313960061)

[3.2 Data Access 12](#_Toc313960062)

[3.3 State Management 14](#_Toc313960063)

[3.4 Caching 17](#_Toc313960064)

[3.5 ASP.NET controls 19](#_Toc313960065)

[3.6 User Interface 22](#_Toc313960066)

[3.7 Others 24](#_Toc313960067)

[4 Microsoft Reference 27](#_Toc313960068)

# Introduction

This document provides standards on which ASP.NET programs should be written and maintained. Close adherence to the rules will help consultants other than the author to understand a piece of code.

The aim is to ensure that code is

* Developed on common standards
* Produced in lesser time
* Accurate
* Reliable
* Easy to understand
* Easy to modify, maintain and test

There is also a need to achieve these aims without unduly restricting the software consultant’s creativity or unnecessarily increasing the quantity of work beyond that merited by the project/product.

## Audience

This document is to be used by all ASP.NET 4.0 developers and reviewers.

## Scope of the document

* Where deviations are permitted, text is included to this effect, with reference made to the fact that a concession note is required in the associated Project Quality Plan.
* Where changes are being made to existing code, the Project Manager must define whether these standards are to be used or compatibility with the existing code is to be maintained. This should be recorded in the Quality Plan for the Project.
* The majority of statements contained within the standard is only for uniformity and maintainability of the code and does not affect the functionality.
* Unless otherwise stated, this document is primarily for ASP.NET 4.

## Scope Exclusions

* This document provides standards and guidelines for ASP.NET program and does not state how a developer should develop an ASP.NET program. While certain capabilities are recommended for ASP.NET specification and syntax for the ways to use these are beyond the scope of this document.
* This document does not provide a tutorial on the use of ASP.NET.

## Document Conventions

* ‘Should’, ‘May’ and ‘Must’: Where recommendations are made, the verb should is used and where alternate approach is provided and no firm recommendations are made, the verb may is used. Where firm recommendations are made the word must is used.

## Assumptions

* The recommended standards and guidelines do not prohibit special considerations for other systems, application specific requirements and hardware constraints.
* Examples are incorporated in this document to enhance clarity and promote understanding. Examples should not be considered as the recommended implementations.
* The examples given indicate that those are for illustration purpose only and does not form a part of the code examples.

## Glossary of Terms

|  |  |
| --- | --- |
| **Term** | **Explanation** |
| ASCII | The dominant standard for coding information on computers and related equipment. The ASCII coding scheme assigns numeric values to letters, numbers, punctuation marks, and certain other characters, enabling computers and computer programs to exchange information. |
| C# | C Sharp |

# Coding Standards

## File Naming Conventions

**Standards**

* There is no limit to the length of a file name except the constraints imposed by the operating system. However, a meaningful name indicating the contents of the file is always preferable.
* The file name should have only one period "." separating the name and the extension.
* The source code filenames should be of the form:[FileName] = [DescriptiveName]**.**[Extension]

Where:

[DescriptiveName] is a meaningful name describing the contents of the file.

[Extension] may be the default extension generated by the application or should be an extension compatible with the existing applications. For example, a text file should have the extension .txt and not .exe, which corresponds to an executable file.

**Example**

Portfolio.aspx

**ASP.NET Project files**

| **Files** | **Convention** | **Example** |
| --- | --- | --- |
| Master Pages | .master | TradingMaster.master |
| CS files in App\_Code | .cs | basePage.cs / Common.cs |
| Web form | .aspx | Trading.aspx |
| Web form code behind | .aspx.cs | Trading.aspx.cs |
| User Controls | .ascx | Trading.ascx |
| Sitemap file | .sitemap | Trading.sitemap |
| Configuration File | Predefined files | Web.config |
| Assembly info File | Predefined files | AssemblyInfo.cs |
| Global application file | Predefined files | Global.asax |

## File Header Information

**Standards**

Any ASP.NET code behind file which can be edited by a text editor (non-binary file) must contain a File Header Information that consists of:

* A short description of the contents of the file
* Author Information such as the author's Name
* Date of creation
* File Modification Log (repeated for each modification) which consists of:
* Author
* Date of Modification
* Modification Description

**Advantages**

* The information about the author and the modification log will make it easier for change tracking and future maintenance and up-gradation of the application.

**Example**

////////////////////////////////////////////////////////////////////////////////////

//

// File Description :

// ---------------------------------------------------------------------------------

// Date Created : Month DD, YYYY (e.g. Jan 01, 2006)

// Author : <author name>, <company name>

// ---------------------------------------------------------------------------------

// Change History

// Date Modified : Month DD, YYYY (e.g. Jan 01, 2006)

// Changed By :

// Change Description :

//

////////////////////////////////////////////////////////////////////////////////////

Class, Method and property headers should be placed using the “///” which would document the method name/description and signatures. These comments headers could be later on extracted to MSDN style documentation by compiling the source code using “/d” complier switch.

***Note****: The above file header may be customized to include project specific information such as Copyright Notes etc.*

## Control Naming Conventions

* The control name should be a meaningful name, which gives an indication of the purpose of the control.
* The control name should start with a lower-case. Each identifiable word in the text description should start with an uppercase for clarity. The rest of the text of the word should be in lower case.
* The control name should be suffixed with the control type e.g. loginNameTextBox, saveButton, employeeGridView.

Any development using Visual Studio 2005 should have the HTML compliant to XHTML standards. Few of the examples are listed here below:

* HTML attributes should be in smaller case and should have the values with in double quotes.
* All HTML tags should be in pairs and should have open and closed tags.
* Hierarchy of nested HTML should be followed strictly. (e.g. of a don’t as per XHTML - <p><I></p><I> )

# ASP.NET Programming Guidelines

As with any programming model, writing code to create an ASP.NET Web application has many pitfalls that can cause performance problems. The following guidelines list specific techniques that you can use to avoid writing code that does not perform at acceptable levels.

## Exception Handling

1. **Use try/catch block appropriately**

Know when to set up a try/catch block. For example, you can programmatically check for a condition that is likely to occur without using exception handling. In other situations, using exception handling to catch an error condition is appropriate. Use defensive coding practice.

1. **Do not use exceptions to control the application flow**

Use custom error messages for handling known error conditions instead of relying on exceptions. Use validation code to avoid unnecessary exceptions.

1. **Use the finally block to ensure that the resources are released**

All expensive resources should be released in a suitable finally block or using a using statement in C#. This prevents the expensive resources from being tied up.

**Finally block**

SqlConnection conn = new SqlConnection("...");

try

{

conn.Open();

//.Do some operation that might cause an exception

// Calling Close as early as possible

conn.Close();

// ... other potentially long operations

}

finally

{

if (conn.State == ConnectionState.Open)

{

conn.Close(); // ensure that the connection is closed

}

}

**Using statement**

using (SqlConnection conn = new SqlConnection("..."))

{

conn.Open();

//.Do some operation that might cause an exception

// Calling Close as early as possible

conn.Close();

// ... other potentially long operations

}

1. **Do not catch exceptions that you cannot handle**

Do not catch exceptions unless the code adds some value. Catching and re-throwing exceptions has performance overheads.

1. **Re-throwing exceptions is a costly process**

Exceptions should be wrapped and re-thrown only when additional information is being added to the exception.

1. **Order exceptions in the catch block appropriately**

Always order exceptions in catch blocks from the most specific to the least specific. This technique handles the specific exception before it is passed to a more general catch block.

1. **Log as much information as possible from the exceptions**

Catching exceptions and not doing anything with it can result in loss of important logging information required for diagnosing the application.

1. **Use performance monitor to monitor the CLR exceptions**

Use performance monitor (Perfmon) to monitor the following counters in the CLR exception object:

# of exceptions thrown, # of exceptions thrown per second, # of finallys / sec, throw to catch depth / second.

1. **Use Validation Controls**

Perform user input validations using validation controls. This ensures that the validations are performed on the client as well as the server. Use Page.IsValid method to ensure that the validations are performed on the server side as well.

1. **Implement a Global.asax exception handler**

Implementing a global error handler traps all unhandled exceptions in the application. Inside the handler, you should at the minimum, log the following information to a data store such as a database, the windows event log or a log file:

* The page that the error occurred on
* Call stack information
* The exception name and message

In the Global.asax file, use the Application\_Error event to code the error handling logic.

1. **Write code that avoids exceptions**

* Check for NULL values to avoid exceptions.
* Suppress the internal call to Response.End. The Server.Transfer, Response.Redirect, Response.End all raise exceptions. Each of these methods internally callResponse.End. The call to Response.End in turn causes ThreadAbortException exception. If you need to call Response.Redirect, consider using the overloaded method and passing false as the second parameter to suppress the internal call to Response.End.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Can errors occur?** | **Recoverable?** | **Can useful context information be added?** | **Cleanup required?** | **Recommended combination of try, catch and finally** |
| No | N/A | N/A | No | None |
| No | N/A | N/A | Yes | try and finally only |
| Yes | No | No | No | None |
| Yes | No | No | Yes | Try and finally only |
| Yes | No | Yes | No | Try and catch only |
| Yes | No | Yes | Yes | Try, catch and finally |
| Yes | Yes | N/A | N/A | Try and catch only |

## Data Access

1. **Use stored procedures for data access**

When the SQL query is submitted to the database server, the query is first parsed, query plan is prepared and optimized and then the query is executed. A stored procedure is a compiled version of the query statements that are pre-parsed and pre-optimized by the query processor. Thus using stored procedures gives higher performance than a normal Sql query. This is very useful in case of complex queries.

1. **Use Paging for large result sets**

Paging large query result sets can significantly improve the performance of an application. The paging solution reduces the back end work on the database. The paging solution also reduces the size of the data that is sent to the client. Paging can be implemented easily using datasource controls such as SQLDataSource and ObjectDataSource controls.

1. **Use ADO.NET Asynchronous Data Access for executing the commands that run for a long time**

ADO.NET provides true asynchronous support for two specific scenarios: opening connections and executing commands. ADO.NET includes methods with Begin and End keyword to support asynchronous execution. We can implement callback mechanism using this approach. With asynchronous pattern Begin method takes all the input parameters, and the End method provides all the output parameters, as well as the return value.

1. **Use a DataReader for fast and efficient binding**

The DataReader is the optimal choice for retrieving read-only data in a forward only manner.

1. **Prevent users from requesting too much data**

* Implement a paging mechanism.
* Enable users to filter data

1. **Consider caching data**

If data is static and can be used application wide, cache the data.

1. **Use output parameters wherever possible**

Use output parameters and ExecuteNonQuery to return small amounts of data instead of returning a result set that contains a single row. When you use output parameters and ExecuteNonQuery to return small amounts of data, you avoid the performance overhead that is associated with creating the result set on the server.

The following code fragment uses a stored procedure to retrieve the product name and unit price for a specific product that is contained in the Products table in the Northwind database.

void GetProductDetails(int ProductID,

outstring ProductName, outdecimal UnitPrice)

{

using (SqlConnection conn = new SqlConnection(

"server=(local);Integrated Security=SSPI;database=Northwind"))

{

//Set up the command object used to run the stored procedure.

SqlCommand cmd = new SqlCommand(

"spGetProductDetailsSPOutput", conn);

cmd.CommandType = CommandType.StoredProcedure;

//Establish stored procedure parameters.

//@ProductID int INPUT

//@ProductName nvarchar(40) OUTPUT

//@UnitPrice money OUTPUT

//Must explicitly set the direction of the output parameters.

SqlParameter paramProdID =

cmd.Parameters.Add("@ProductID", ProductID);

paramProdID.Direction = ParameterDirection.Input;

SqlParameter paramProdName =

cmd.Parameters.Add("@ProductName",

SqlDbType.VarChar, 40);

paramProdName.Direction = ParameterDirection.Output;

SqlParameter paramUnitPrice =

cmd.Parameters.Add("@UnitPrice", SqlDbType.Money);

paramUnitPrice.Direction = ParameterDirection.Output;

conn.Open();

//Use ExecuteNonQuery to run the command.

//Although no rows are returned, any mapped output parameters

//(and potential return values) are populated

cmd.ExecuteNonQuery();

//Return output parameters from stored procedure.

ProductName = paramProdName.Value.ToString();

UnitPrice = (decimal)paramUnitPrice.Value;

}

}

1. **Specify parameter types**

When you create a new parameter, use the relevant enumerated type to specify the data type of the parameter. Use an enumerated type such as SqlDbType or OleDbType. This prevents unnecessary type conversions that are otherwise performed by the data provider.

1. **DataSet vs. DataReader**

Use a DataReader when the following conditions are true:

* You need forward-only, read-only access to data, and you want to access the data as quickly as possible, and you do not need to cache it.
* You have a data container such as a business component that you can put the data in.

Use a DataSet when the following conditions are true:

* You have to cache or pass the data between layers.
* You require an in-memory relational view of the data for XML or non-XML manipulation.
* You want to update some or all the retrieved rows, and you want to use the batch update facilities of the SqlDataAdapter class.
* You have to bind data to a control type that the DataReader cannot be bound to.
* You have to access multiple sets of data at the same time, and you do not want to hold open server resources.

1. **Working with DataReader**

* Close your DataReader object as soon as you are finished with it, either by calling its Close method or by calling its Dispose method.
* If you need to return a DataReader from a method, consider using the CommandBehavior.CloseConnection method to ensure that the associated connection is closed when the DataReader is closed.
* Using an index or ordinal-based lookup is faster than using string-based column names. Get the ordinal using the GetOrdinal method of the DataReader.
* Use data type specific methods of the datareader for e.g. GetString, GetInt32 etc to read data from the datareader instead of the generic GetValue.

1. Use ExecuteNonQuery for commands that do not return data.
2. Use ExecuteScalar to return single values.
3. Retrieve only the columns and rows you need. Avoid “Select \*”.
4. Always close database connection after use.
5. Reduce round trips to the database.

## State Management

1. **Use appropriate container for storing state information**

* Use Session state to store per-user state on the server. The state information is tracked by using a session cookie or mangled URL. ASP.Net session scales across server farm, provided the state management mode is not In-Proc.
* Use ViewState to store per-page state information. The view state is included with every HTTP POST request and response. However as ViewStategrows larger, it affects the performance of garbage collection. Please refer point 4 under this section for usage guidelines.
* Store simple state on clients wherever possible. Use cookies, query strings and hidden controls for storing lightweight, user-specific state that is not sensitive such as personalization data. Do not use cookies, query strings or hidden controls to store sensitive data.
* Use static properties instead of the application object to store the application state. This increases the performance as the static variables can be accessed faster than the Application dictionary.
* Use application state to store static and read only data. Since, application state is specific to a server, it is best to store static read only data in the application state to avoid server affinity.
* In-process mode offers fastest access to the session state information, as the session state information is stored within the managed memory of the ASP.Net worker process. However, the in-process mode limits the scalability of the application as it results in server affinity.

In the State Server mode, the session state is stored in a Win32 service that could be run on the same server as the web/application server or a separate server. State Server mode scales well as the state can be accessed across web/application servers but the performance is less as compared to the In-process mode due to the additional serialization and communication costs.

SQL Server provides a highly scalable state management solution. The performance of the SQL Server is slightly lower than that of State Server. SQL Server is well suited to large amount of session state.

1. **Prefer basic types to reduce serialization costs**

To minimize the serialization costs, prefer basic types to complex types. Complex types are serialized using a relatively slow BinaryFormatter object. For complex types, you can use the Serializable attribute or you can implement the ISerializable attribute.

1. **Avoid storing large objects in Session**

* For improving performance, avoid storing large amounts of data in Session. Disable session state if it is not used in the application. Also, it can be turned off for specific pages (using EnableSessionState), instead of for every page.
* To make sessions persist or to store relatively large objects, use Microsoft SQL Server as a centralized storage location of session state. ASP.NET 4.0 provides a new property called Compression that stores values in serialized form in SQL Server with GZip compression resulting in better performance.

1. **Disable the session state if you do not use it**

To disable the session state for all applications on your server, use the following element on your machine.config:

<sessionStatemode="Off"/>

To disable session state for a specific application, use the following element in the Web.config file of the application:

<sessionStatemode="Off"/>

You can also remove the session state module from <httpModules> to completely remove session processing overhead.

To disable session state for a specific web page, use the following page setting:

<%@PageEnableSessionState="false"%>

1. **Disable the view state if not required**

View state is not required in the following cases:

* If the page does not post information back to itself, if the page is used only for output and if the page does not rely on response processing, you do not need the view state.
* If your server controls do not handle events and if your server controls do not have any dynamic or data bound property values, you do not need the view state.
* If you ignore old data and if you repopulate the server controls every time the page is refreshed, you do not need the view state.

To disable view state for all application on a Web Server, configure the <pages> element in machine.config as follows:

<pagesenableViewState="false"/>

To disable view state for a single page, use the @Page directive as follows:

<%@PageEnableViewState="false"%>

To disable view state for a single control on a page, set the **EnableViewState** property of the control to false.

* To use opt-in state management (turn view state off for entire page and turn on view state selectively for one or multiple controls):
  + Set **ViewStateMode** of the page to Disabled - <%@ Page Language="C#" ViewStateMode="Disabled" ... %>
  + Now, to opt in viewstate for a particular control in the page, set ViewStateMode to Enabled for that control –

<asp:LabelViewStateMode="Enabled" ... />

1. **Use View State Chunking**

Some proxy servers, firewalls, and mobilebrowsers refuse to let pages through if they have view state fields greater than a certain size. As a workaround, use view state chunking, this automatically divides view state into multiple fields to ensure that no hidden field exceeds a particular size threshold.To use this functionality, set the **maxPageStateFieldLength** attribute of the <pages>element in the web.config file. This specifies the maximum view state size, in bytes.

<configuration>

<system.web>

<pages maxPageStateFieldLength="1024" />

...

</system.web>

</configuration>

When a page that generates a view state larger than that specified in web.config, several hidden input fields willbe created:

<input type="hidden" name="\_\_VIEWSTATEFIELDCOUNT" value="3" />

<input type="hidden" name="\_\_VIEWSTATE" value="..." />

<input type="hidden" name="\_\_VIEWSTATE1" value="..." />

<input type="hidden" name="\_\_VIEWSTATE2" value="..." />

1. **Profile v/s Session**

Profile and session is used to store per-user data but there are certain differences between them which should be considered while deciding which one to use.

Following table lists the difference between profile and session:

|  |  |
| --- | --- |
| **Profile** | **Session** |
| **Persistent:** When you modify the state of the profile object, the modifications are saved between visits to the website. | **Non-persistent:** The session object is lost after you leave the web site. |
| **Storage:** Uses provider model to store information. The database used for storing profile information can be configured in the configuration file. | **Storage:** Provides following storage types:   * In Process - default * State Server (Out of Process) * SQL Server (Out of Process) |
| Profile is strongly typed. | Session is not strongly typed. |

1. **Minimize the number of objects stored in the view state**

Avoid storing large objects due to associated serialization and de-serialization overheads.

## Caching

1. **Cache the Right Data**

* Cache application-wide data and data that is used by multiple users.
* Cache static data and dynamic data that is expensive to create or retrieve.
* If the data is used more often than it is updated, it is also a candidate for caching.
* Do not cache expensive resources that are shared, such as database connections.
* Avoid storing DataReader objects in the cache because these objects keep the underlying connections open. It is better to pool these resources.
* Do not cache per-user data that spans requests — use session state for that.

1. **Cache items selectively**

* Always identify items in the solutions that need to be cached. It is easy to go overboard and try to cache everything. There is a cost for caching an item, especially in memory utilization
* Do not create cache for items with fast expiration. Fast expiring cached items often cause high churn rate of cache and more work for cache cleanup code. A high rate could be something of concern as this may occur due to low server memory when items are removed before their expiration date/time.

1. **Separate volatile data from the non-volatile data**

Use user controls to segregate static data such as navigational menus, page headers and footers from non-static data. This helps in caching the static data.

1. **Choose the right caching mechanism**

User specific data is best stored in the Session object.Static pages and a few types of dynamic pages such as non-personalized pages that are served to a larger audience could be cached by using output caching and response caching. Static content in a page can be cached by using a combination of output cache and user controls.

1. **Use the caching API to cache application wide data that is used by multiple users**

* Page headlines and product catalogs are good candidates for the cache API.
* Avoid using the cache API if the data to be cached is changing real time.

1. **Cache the Appropriate Form of the Data**

* If you want to cache rendered output, you should consider using output caching or fragment caching.
* Use the cache API to programmatically cache application-wide data that is shared and accessed by multiple users.

1. **Refresh your cache appropriately**

The expiration algorithm attributes that would be considered for each expiration scenario would be:

* Least recently used
* Least frequently used
* Absolute expiration after fixed interval
* Caching expiration based on a change in an external dependency, such as a file
* Cleaning up the cache if a resource threshold is reached

1. **Removing Client side cache**

Since the natural tendency is to have the web application’s client data and resources cached on the local browser, it’s an obvious source of confusion. Inorder to work around this and prevent any sort of client side cache, Microsoft recommends to add the below meta tags onto the head tag of the HTML.

<metacontent="pragma"name="no-cache">

Incase of pages with competitively smaller rendering size, the problem may occur even after adding the above Meta tags. Microsoft recommendation for this problem is to add the below tags after the body tag towards the end of the HTML document.

<head>

<metacontent="pragma"name="no-cache">

</head>

*Note:* Meta tag is not supported by all browsers. The supported browsers are Internet Explorer and Netscape Navigator.

1. **Leverage SQL Cache dependency**

SQL cache dependency helps you to invalidate the cache if the underlying data changes.

It can be used to invalidate ASP.NET application cache as well as output cache.

SQL Cache dependency is compatible with SQL Server 7, 2000 and 2005.

Configuring SQL cache dependency in Web.config:

<configuration>

<connectionStrings>

<addname="Northwind"

connectionString="server=localhost;database=northwind;..." />

</connectionStrings>

<system.web>

<caching>

<sqlCacheDependencyenabled="true"pollTime="5000">

<databases>

<addname="Northwind"connectionStringName="Northwind" />

</databases>

</sqlCacheDependency>

</caching>

</system.web>

</configuration>

**Using SqlCacheDependency with the Application Cache**

Cache.Insert ("Products", products, new SqlCacheDependency ("Northwind", "Products");

**Using SqlCacheDependency with the Output Cache**

<%@OutputCacheDuration="60"VaryByParam="None"SqlDependency="Northwind:Products"%>

**Using SqlCacheDependency with SqlDataSource**

<asp:SqlDataSourceID="Countries"runat="server"ConnectionString="server=localhost;database=northwind;..."

SelectCommand="select distinct country from customers order by country"EnableCaching="true"

CacheDuration="60000"SqlCacheDependency="Northwind:Customers"/>

<asp:DropDownListID="MyDropDownList"DataSourceID="Countries"DataTextField="country"

AutoPostBack="true"runat="server"/>

## ASP.NET controls

1. Use Server controls in appropriate circumstances

Review your application code to make sure that your use of ASP.NET server controls is necessary. Even though they are extremely easy to use, server controls are not always the best choice to accomplish a task, since they use server resources. In many cases, a simple rendering or data-binding substitution will do. The following example demonstrates a situation in which using a server control is not the most efficient way to substitute values into the HTML sent to the client. Each method sends the path to an image to be displayed by the requesting browser, but using server controls is not the most expedient method shown, since the Page\_Load event requires a call to the server for processing.

[C#]

<scriptlanguage="C#"runat="server">

publicstring imagePath;

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

//...Retrieve data for imagePath here...

DataBind();

}

</script>

<!-- The span and img server controls are unnecessary...-->

The path to the image is: <spanid="Span1"innerhtml='<%# imagePath %>'runat="server"/><br>

<imgid="Img1"src='<%# imagePath %>'runat="server"/>

<br><br>

<%-- Or use a simple rendering expression...--%>

The path to the image is: <%= imagePath %><br>

<imgsrc='<%= imagePath %>'/>

There are many other cases in which rendering or data binding are more efficient than using server controls, even when you use server control templates. However, if you want to programmatically manipulate server control’s properties, handle server control events or take advantage of view-state preservation, then a server control would be appropriate.

1. To navigate to a URL without posting form, use hyperlink control

The hyperlink control immediately navigates to the target URL, without posting the form to the server. So, if server side processing is not necessary, then use the hyperlink control.

1. Use ASP.NET navigation control

ASP.NET site navigation control can be used to navigate across pages. The navigational control provides a consistent way for users to navigate the site. As the site grows and the pages are moved around in the site, it becomes difficult to manage all the linkages. ASP.NET site navigation enables to store links to all the pages in a central location and render those links in lists or navigation menus on each page by including a specific web server control.

Before a navigation user interface is created for anywebsite, the site's logical structure must first be defined. This logical structure is referred to as a site map. Use ASP.NET 2.0's default XML-based method for specifying your website's site map (.sitemap file) or, with code.

ASP.NET ships with a number of navigation Web controls for displaying the site map. These are SiteMapPath, TreeView and Menu.

1. Use ObjectDatasource control to bind ASP.NET controls to data components

* ObjectDataSource permits you to practice declarative data binding without sacrificing the ability to use middle-tier data access components to access data sources. Thus it helps in keeping data access code separate from UI layer.
* Helps to isolate the logic for interacting with the database and other data sources in a data access layer.
* Supports two way data binding through SelectMethod, InsertMethod, UpdateMethod and DeleteMethod methods.
* Supports parameterized operation.
* Supports caching of query results.
* When using ObjectDatasource, use ObjectCreated and ObjectDisposing events to perform specialized initialization or clean-up work on data components.

1. Use GridView instead of DataGrid

GridView provides additional features apart from those provided by DataGrid such as:

* Declarative sorting, paging, selecting, updating, and deleting support with the help of data source controls such as ObjectDataSource control. The data source controls have no user interface but supply data to data-bound controls. Data source controls reduce the quantity of code you’re called upon to write in data-driven pages. Data source controls intelligently communicate with made-to-measure data-bound controls to provide data to display and save changes back to the physical data source in a two-way data binding mechanism.
* Supports a richer assortment of column types than DataGrid. For e.g. ImageField, CheckBoxField.
* The GridView control simplifies the implementation of some features such as the definition of edit and select buttons. To enable editing and selection, you simply set a couple of Boolean properties with the GridView, whereas the DataGrid requires you to add specific columns and handle related events.

**Usage Recommendation:**

**GridView:**

Always use the GridView in new ASP.NET 4 code and as much as possible in migrated code. For ASP.NET 1.x applications being upgraded and maintained, moving to the GridView doesn’t pose any significant issues and, more importantly, it positions you well for future enhancements.

**DataGrid:**

The DataGrid remains available in ASP.NET 2.0 onwards mostly to ensure compatibilityreasons so that existing applications can continue to work when simply ported and recompiled in ASP.NET 2.0 or greater.

1. Master-Detail views

Implement master detail views using GridView and DetailsView control.

| **Feature** | **Repeater** | **DataList** | **DataGrid** | **GridView** |
| --- | --- | --- | --- | --- |
| Default appearance | None (template driven) | Table | Table | Table |
| Automatically generates columns from the data source | No | No | Yes | Yes |
| Header can be customized | Yes | Yes | Yes | Yes |
| Data row can be customized | Yes | Yes | Yes | Yes |
| Supports alternating row customization | Yes | Yes | Yes | Yes |
| Supports customizable row separator | Yes | Yes | No | No |
| Footer can be customized | Yes | Yes | Yes | Yes |
| Supports pagination | No | No | Yes | Yes |
| Supports editing contents | No | Yes | Yes | Yes |
| Supports selecting a single row | No | Yes | Yes | Yes |
| Supports arranging data items horizontally or vertically (from left-to-right or top-to-bottom) | No | Yes | No | No |
| In-built support for Master/Detail views | No | No | No | Yes |
| In-built support for binding columns containing image | No | No | No | Yes |

## User Interface

1. Template based approach using master pages

Master pages provide a key component to any web application, namely the ability to organize a consistent layout into reusable templates. These master page templates offer full designer support and programmatic interfaces to meet the demands of most applications.

Master pages (.master files) allows to create the common page layouts and incorporate the actual web form contents (separated and placed in .aspx files) to the layout using the content place holder.

The advantages are:

* Clean separation of layout or common HTML from actual form contents.
* Reusability of template or master pages across many web pages with same layout.
* Capability to access properties in master page from the content page using the object model.

However master page brings in some changes in the way web form controls are accessed in the pages as it appends “<Master page Form name>\_<Content place holder ID>\_” to the actual control name.

This also means that the length of the names of content place holder controls, master pages, user controls can impact the page size a lot while rendering. Because these names get appended in the actual page source with validation controls etc, the length of these control names should be kept to a maximum of 5.

1. Skinning and theming for personalization

Skins and themes are for defining the style and attributes of the web forms. This allows consolidation of the styles and even values of attributes of server controls into culture or user specific themes folder. This also allows defining the styles using conventional style sheet approach which means the style can be read from style sheets using skins.By defining skin IDs and associating with controls, it offers flexibility to control the attribute values of server side controls. So, with capability to control the themes folder association dynamically, it makes personalization lot easier.

1. Setting Focus

Use Page.SetFocus method to programmatically set focus on the required control on the server side when the page loads.

The following attributes allow setting of cursor focus on the screen:

Validation failure – FocusOnError attribute of validation control to be set true

Default focus – Form and each control has a new property “Default Focus”

Use Panel.DefaultButton property to designate a button to be “clicked” when Enter is pressed with focus in panel.

1. Applying themes dynamically

Store the active Theme in a [User Profile](http://www.asp.net/QuickStart/aspnet/doc/profile/default.aspx) if there is requirement for dynamically applying a theme based on user preferences.

**Example**

To apply a Theme from code, you set the Theme property of the Page object at runtime. A Theme must be applied to a page very early in the request lifecycle, in the PreInit event. The example below shows a Theme dynamically applied in this event when the user selects a Theme name from a DropDownList control.

protectedvoidPage\_PreInit()

{

if (Profile.FavoriteColor != "")

Page.Theme = Profile.FavoriteColor;

}

1. Use Web.config for associating Default Master pages with Web forms

In case of default master pages, use Web.config to assign master page for all web pages in the web project. We can do this using the <pages> element. An example excerpt from web.config is shown below:

<configuration>

<system.Web>

<pagesmaster="defaultMaster.master" />

</system.Web>

</configuration>

1. Use combination of Themes and CSS for applying styles

If web site requires use of default CSS throughout the entire web pages, add Cascading Style Sheet (CSS) to a Theme by placing it under the named Theme subdirectory. The CSS stylesheet will be applied to all pages with that theme applied, provided the page has a <head runat="server"/> control defined.

1. HTML tips

* Each web page or .aspx file must have a title mentioned for it.
* The maxlength property should be specified for all editable controls on the page.
* Access key (works for IE 5.0 +), Title and tooltips (works for IE 5.0+) should be specified for each control in order to meet the accessibility standards.
* document.GetElementByID only should be used against conventional script routines like document.alletc to ensure compliance to different browsers.
* Set the‘controlRenderingCompatbilityVersion’ flag to “4.0” in the web.config, all server controls will strictly adhere to the XHTML 1.1 specification and have cleaner client IDs and have redundant inline styles removed.

1. Cleaner Markup for Controls

In order to have clean, predictable, ID attributes on rendered HTML elements, use the ‘ClientIDMode’ property of ASP.NET server controls. Set this property to ‘Static’ to render whatever is set as the ID of the control during design time (example: id=”TextBox1”). This will help to write client-side JavaScript against the output, makes it easier to style elements using CSS, and on large pages can help reduce the overall size of the markup generated.

1. WebPart Framework

Use the new ASP.NET WebPart framework to build highly customized and personalized user interfaces.

* Use PersonalizationScope enumeration to indicate whether a web part property is per-user or shared.
* Although any control can serve as a Web Part, use controls derived from WebPart which can better leverage the Web Parts infrastructure.
* For sensitive webpart properties, set the IsSensitive flag in the Personalizable attribute to true. This will prevent persisting of sensitive properties used in the webpart. e.g. password, SSN etc.
* Prevent persistence of sensitive properties.

publicclassMyWebPart : WebPart

{

public MyWebPart ()

{

ExportMode = WebPartExportMode.NonSensitiveData;

}

// This property will be exported

[Personalizable (PersonalizationScope.User, false)]

publicstring ZipCode

{ ... }

// This one will not be exported

[Personalizable (PersonalizationScope.User, true)]

publicstring SocialSecurityNumber

{ ... }

...

}

## Others

1. Centralizing imports section using web.config

Imports sections can be included in the web.config to avoid duplication of import statements across pages. In order to achieve easy modification of namespaces, the code should have fully qualified access to objects.

1. Configuring and securing connection strings

Web.configprovides <connectionstrings> configuration sections which are dedicated for storing database connection parameters. The advantages are:

* Clean separation of database credentials from other app settings.
* Capability to encrypt sections of web.config storing the credentials.
* SetReg utility can be used along with this to configure web.config to read encrypted credentials from registry.

1. Enabling ASP.NET application to avoid queuing of web service requests

The ASP.NET 1.1 engine would queue requests from the application to any web service calls incase of concurrent requests of more than 4 calls at a time. This is true in case of asynchronous calls also.

ASP.NET web.config allows a new <connections> section where the number of concurrent connections as well as the IP of the web service URL could be configured to scale up further from default.

1. Web.config settings to support web farming

Most important feature to enable this is the capability to configure ASP.NET session store into various source like Inproc, SQL session state and State server.

Since ASP.NET allows deploying applications across web farms, there will be probabilities of application firing subsequent requests across servers in the farm and leading a view state validation error. In order to work around this problem, ASP.NET supports <machineKey> settings in the web.config which would create a similar view state encryption token across all the servers allowing the successfully validation of the view state.

1. Asynchronous Programming

Since ASP.NET does allowing returning the page response before ending all the different threads opened from the application, the below options can be considered for allowing parallel processing and also for responding to the end user with out really waiting for all the operations which have to be really complete:

* Implement the parallel functionality as a web service accessed from the web application asynchronously using Begininvoke. This can be done only if the application doesn’t need to wait for the result of the method.
* Use AJAX approach which would use a JavaScript call back method and implement the functionality as a web service. This would also allow user to proceed working with the form as it don’t involve any refresh of the page.
* Use ASP.NET client callbacks to refresh part of the page without making a postback to the server.

1. Implement Client Callbacks

In certain situations, it is useful to run server code from the client without performing a postback as page postbacks introduce processing overhead that can decrease performance and force the user to wait for the page to be processed and re-created.To avoid the processing overhead of a server round-trip and to avoid losing client state, code an ASP.NET web page so that it can perform client callbacks.

There are options for automating client callbacks in an ASP.NET Web page:

* UpdatePanel control - helps to develop web pages with complex client behavior that can refresh selected parts of the page instead of refreshing the whole page with a postback. It can automate asynchronous partial-page updates.
* ASP.NET AJAX automates asynchronous Web service calls. Data is exchanged asynchronously between client and server, typically in JSON format.

1. Use Page.IsPostback to avoid performing unnecessary processing on a round trip

If you are handling server control postbacks, you often need to execute different code the first time the page is requested from the code you do use for the round trip when an event is fired. If you check the Page.IsPostBack property, your code can execute conditionally, depending on whether there is an initial request for the page or a responce to a server control event.Use the Page.IsPostback property to conditionally execute code depending on whether the page is generated in response to a server control event.

Example:

protectedvoid Page\_Load(Object sender, EventArgs e)

{

if (!Page.IsPostback)

{

string query = "select \* from Authors";

// ...Set up a connection and command here....

myCommand.Fill(ds, "Authors");

gridView.DataSource = ds;

gridView.DataBind();

}

}

Since the Page\_Load event executes on every request, this code checks whether the IsPostBack property is set to false. If so, the code executes; otherwise, it does not.Thus the retrieval of data is executed only for the first time the page is requested.

1. Grouping Validation Controls

“Validation group” is allows to group validation controls. The same attribute can be associated to the control causing the post back which would fire the validation for only the associated group.

1. Leverage App\_Code folder to place common code in the web project.

Code files in the App\_Code directory will be dynamically compiled at runtime and made available to the application. It is possible to place files of more than one language under the App\_Code directory, provided they are partitioned in subdirectories.

The App\_Code directory is recommended to share common code in standalone files across several pages in your application.

1. Consider using RedirectPermanent() instead of Redirect()

* **Redirect()** method issues an HTTP 302 Found (temporary redirect) response, which results in an extra HTTP round trip. ASP.NET 4 adds a new **RedirectPermanent()** helper method that issues HTTP 301 Moved Permanently response.
* Use Redirect() for normal navigation and control of flow in an application
* Use RedirectPermanent() if an old URL is requested, which was supported in the past but is no longer in use. This call is specifically done in the**Application\_BeginRequest()** method in the global.asax file.
* If a search engine’s web crawler is exploring a website and it receives the 301 status code, it will update the search catalog with the new URLinformation.

# Microsoft Reference

For Design guidelines, Performance and Security refer the following Microsoft links:

* What's New in ASP.NET 4 and Visual Web Developer: http://msdn.microsoft.com/en-us/library/s57a598e.aspx
* Design Guidelines for Developing Class Libraries:

<http://msdn.microsoft.com/en-us/library/ms229042.aspx>

* Improving .NET Application Performance and Scalability: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenet.asp>
* Improving ASP.NET Performance: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt06.asp>
* Patterns & practices ASP.NET 2.0 Security Guidance Index: http://msdn.microsoft.com/en-us/library/ff648125.aspx