MS .NET Coding Standards & Best Practices

Table of Contents

[1 Document Information 4](#_Toc376795936)

[1.1 Introduction 4](#_Toc376795937)

[1.2 Scope 4](#_Toc376795938)

[1.3 Notes 4](#_Toc376795939)

[2 Checklist – C# 4](#_Toc376795940)

[2.1 Naming Conventions 4](#_Toc376795941)

[2.2 .NET Framework Guidelines and Best Practices 5](#_Toc376795942)

[2.3 Comments and embedded documentation 5](#_Toc376795943)

[3 ASP .NET 6](#_Toc376795944)

[3.1 Naming Conventions 6](#_Toc376795945)

[3.2 Performance 7](#_Toc376795946)

[3.3 Threading and Synchronization Features 9](#_Toc376795947)

[3.4 Manage Resources Efficiently 10](#_Toc376795948)

[3.5 Manage Strings 10](#_Toc376795949)

[3.6 Manage Exceptions 11](#_Toc376795950)

[3.7 Optimize Web Pages 11](#_Toc376795951)

[3.8 View State – Not applicable 13](#_Toc376795952)

[3.9 Server Controls 14](#_Toc376795953)

[3.10 Access Data from Aspx Pages 14](#_Toc376795954)

[3.11 Data Binding 14](#_Toc376795955)

[3.12 Calling Unmanaged From Aspx Pages 16](#_Toc376795956)

[3.13 Settings in MACHINE.CONFIG 17](#_Toc376795957)

[4 Intellectual Property Considerations 17](#_Toc376795958)

[5 Development and Deployment Considerations (Entity Framework) – 17](#_Toc376795959)

[6 Linq to SQL Best Practices – 18](#_Toc376795960)

[7 ASP .NET MVC 20](#_Toc376795961)

[8 Windows Communication Foundation (WCF) 21](#_Toc376795962)

[8.1 General Design Guidelines 21](#_Toc376795963)

[8.2 Essentials 22](#_Toc376795964)

[8.3 Service Contracts 22](#_Toc376795965)

[8.4 Data Contracts 23](#_Toc376795966)

[8.5 Instance Management 23](#_Toc376795967)

[8.6 Operations and Calls 23](#_Toc376795968)

[8.7 Faults 24](#_Toc376795969)

[8.8 Concurrency Management 25](#_Toc376795970)

[8.9 Security 25](#_Toc376795971)

# Document Information

## Introduction

The purpose of this document is to define the coding standards to:

* Ensure high degree of code quality for delivering a better product to the customer
* Describe a common understanding among all stakeholders including developers, reviewers, architects, managers and the customer about the acceptance criteria for the code quality.

## Scope

The scope of the review activity is for the .NET project.

## Notes

* These coding standards and best practices are adapted from MSDN and other places.

# Checklist – C#

Code conventions are important to programmers for a number of reasons:

* 80% of the lifetime cost of a piece of software goes to maintenance.
* Hardly any software is maintained for its whole life by the original author.
* Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
* Maintainability, by promoting some proven design principles.
* Maintainability, by requiring or recommending a certain unity of style.
* Rules and recommendations are given that promote reliability and maintainability.
* Performance

For the conventions to work, every person, writing the software must conform to the code conventions.

## Naming Conventions

VS recommends to follow the naming conventions as defined by Microsoft.

Refer to below link for detailed information about naming conventions.

[http://msdn.microsoft.com/en-us/library/ms229002.aspx](about:blank)

## .NET Framework Guidelines and Best Practices

[http://msdn.microsoft.com/en-us/library/ms229042.aspx](about:blank)

## Comments and embedded documentation

|  |  |
| --- | --- |
|  | Each file shall contain a header block.  #region Copyright <CompanyName> <Year>  // All rights are reserved. Reproduction or transmission in whole or in part, in  // any form or by any means, electronic, mechanical or otherwise, is prohibited  // without the prior written consent of the copyright owner.  //  // Date Dev Comments  // --------- --- ---------------------------------------------------------------------------  // 29-June-07 SST Created  #endregion |
|  | Refer to below link for XML documentation  [http://msdn.microsoft.com/en-us/library/b2s063f7.aspx](about:blank) |

# ASP .NET

This link has a collection of all flavours of ASP.Net related to whats new, some best practices and other guidance information.

ASP.NET and Visual Studio

[http://msdn.microsoft.com/en-us/library/dd566231.aspx](about:blank)

## Naming Conventions

### Source files

|  |  |
| --- | --- |
| Sr. No. | **Convention** |
|  | Use Camel casing for naming controls. |
|  | Do not use casing to differentiate controls.  (Please note that it is allowed to have identifiers that differ only in case in distinct categories, e.g. a property BackColor that wraps the field  backColor.) |
|  | Do not use underscores in naming controls. |
|  | Name a control according to its meaning and not its type. |
|  | Add EventHandler to delegates related to events.  Delegates that are used to define an event handler for an event must be suffixed with EventHandler. For example, the following declaration is correct for a Close event.  public delegate CloseEventHandler(object sender, EventArgs arguments) |
|  | Add Callback to delegate related to callback methods.  Delegates that are used to pass a reference to a callback method (so not an event) must be suffixed with Callback. For example:  public delegate AsyncIOFinishedCallback(IpcClient client, string message); |
|  | Do not add Callback or similar suffix to callback methods.  Do not add suffixes like Callback or CB to indicate that methods are going to be called through a callback delegate. You cannot make assumptions on whether methods will be called through a delegate or not. An end-user may decide to use Asynchronous Delegate Invocation to execute the method. |
|  | Use a verb (gerund) for naming an event.  Good examples of events are Closing, Minimizing, and Arriving. For example, the declaration for the Closing event may look like this:  public event ClosingEventHandler Closing; Use autogenerated events and do not modify their names. |
|  | Do not add an Event suffix (or any other type-related suffix) to  the name of an event. |
|  | Use an –ing and –ed form to express pre-events and postevents.  Do not use a pattern like BeginXxx and EndXxx. If you want to provide distinct events for expressing a point of time before and a point of time after a certain occurrence such as a validation event, do not use a pattern like BeforeValidation and AfterValidation. Instead, use a Validating and Validated pattern. |
|  | Prefix an event handler with On.  It is good practice to prefix the method that is registered as an event handler with On. For example, a method that handles the Closing event should be named OnClosing().  Exception:  In some situations, you might be faced with multiple classes exposing the same event name. To allow separate event handlers use a more intuitive name for the event handler, as long as it is prefixed with On. |
|  | Suffix exception classes with Exception.  For example: IpcException. |

## Performance

ASP.NET is often the foundation from which other technologies are used. Optimizing ASP.NET performance is critical to ensure optimum application performance. Review the following questions to help assess the efficiency of your ASP.NET applications:

### Caching

|  |  |
| --- | --- |
|  | Do you have too many variations for output caching?  Check your pages that use the output cache to ensure that the number of variations has a limit. Too many variations of an output cached page can cause an increase in memory usage. You can identify pages that use the output cache by searching for the string "OutputCache." |
|  | Could you use output caching?  When reviewing your pages, start by asking yourself if the whole page can be cached. If the whole page cannot be cached, can portions of it be cached? Consider using the output cache even if the data is not static. If your content does not need to be delivered in near real-time, consider output caching. Using the output cache to cache either the entire page or portions of the page can significantly improve performance |
|  | Is there static data that would be better stored in the cache?  Identify application-side data that is static or infrequently updated. This type of data is a great candidate for storing in the cache. |
|  | Do you check for nulls before accessing cache items?  You can improve performance by checking for null before accessing the cached item as shown in the following code fragment.  Object item = Cache["myitem"];  if (item==null)  {  // repopulate the cache  }  This helps avoid any exceptions which are caused by null objects. To find where in your code you access the cache, you can search for the string "Cache". |

### Session State

|  |  |
| --- | --- |
|  | Do you disable session state when not required?  Session state is on by default. If your application does not use session state, disable it in Web.config as follows.  <sessionState mode="Off" />  If parts of your application need session state, identify pages that do not use it and disable it for those pages by using the following page level attribute.  <@% EnableSessionState = "false" %>  Minimizing the use of session state increases the performance of your application. |
|  | Do you have pages that do not write to a session?  Page requests using session state internally use a ReaderWriterLock to manage access to the session state. For pages that only read session data, consider setting EnableSessionState to ReadOnly.  <%@ Page EnableSessionState="ReadOnly" . . .%>  This is particularly useful when you use HTML frames. The default setting (due to ReaderWriterLock) serializes the page execution. By setting it to ReadOnly, you prevent blocking and allow more parallelism. |
|  | Do you check for nulls before accessing items in session state?  You can improve performance by checking for null before accessing the item, as shown in the following code.  object item = Session["myitem"];  if(item==null)  {  // do something else  }  A common pitfall when retrieving data from session state is to not check to see if the data is null before accessing it and then catching the resulting exception. You should avoid this because exceptions are expensive. To find where your code accesses session state, you can search for the string "Session". |
|  | Do you store complex objects in session state?  If yes, first point to note is that these objects should be seriazable.  Avoid storing complex objects in session state, particularly if you use an out-of-process session state store. When using out-of-process session state, objects have to be serialized and deserialized for each request, which decreases performance. |
|  | Do you store STA COM objects in session state?  Storing single-threaded apartment (STA) COM objects in session state causes thread affinity because the sessions are bound to the original thread on which the component is created. This severely affects both performance and scalability.  Make sure that you use the following page level attribute on any page that stores STA COM objects in session state.  <@%Page AspCompat = "true" %>  This forces the page to run from the STA thread pool, avoiding any costly apartment switch from the default multithreaded apartment (MTA) thread pool for ASP.NET. Where possible, avoid the use of STA COM objects. |
|  | For web farming deployment approach, always use SQL server state management approach. |

### Application State

|  |  |
| --- | --- |
|  | Store a COM object in Application state if it is not storing any session related state information. |
|  | Do you use the application state dictionary?  You should use application state dictionary for storing read-only values that can be set at application initialization time and do not change afterward. There are several issues to be aware of when using application state in your code, such as the following:   * Memory allocated to the storage of application variables is not released unless they are removed or replaced. * Application state is not shared across a Web farm or a Web garden — variables stored in application state are global to the particular process in which the application is running. Each application process can have different values.   Consider using the following alternatives to application state:   * Create static properties for the application rather than using the state dictionary. It is more efficient to look up a static property than to access the state dictionary. For example, consider the following code.   Application["name"] = "App Name";  It is more efficient to use the following code.  private static String \_appName = "App Name";  public string AppName  {  get{ return \_appName; }  set{\_appName = value; }  }   * Use configuration files for storing application configuration information. * Consider caching data that is volatile enough that it cannot be stored in application state, but needs updates periodically from a persistent medium, in the Cache object. * Use the session store for user-specific information.   You can identify places where your code uses application state by searching for the string "Application”. |

## Threading and Synchronization Features

The .NET Framework exposes various threading and synchronization features, and the way your code uses multiple threads can have a significant impact on application performance and scalability. Use the following review questions to assess how efficiently your ASP.NET code uses threading:

|  |  |
| --- | --- |
|  | Do you create threads on a per-request basis?  Avoid manually creating threads in ASP.NET applications. Creating threads is an expensive operation that requires initialization of both managed and unmanaged resources. If you do need additional threads to perform work, use the CLR thread pool. To find places in your code where you are creating threads, search for the string "ThreadStart." |
|  | Do you perform long-running blocking operations?  Avoid blocking operations in your ASP.NET applications where possible. If you have to execute a long-running task, consider using asynchronous execution (if you can free the calling thread) or use the asynchronous "fire and forget" model. |

## Manage Resources Efficiently

Use the following review questions to assess how efficiently your code uses resources:

|  |  |
| --- | --- |
|  | Do you pool shared resources?  Check that you use pooling to increase performance when accessing shared resources. Ensure that shared resources, such as database connections and serviced components that can be pooled are being pooled. Without pooling, your code incurs the overhead of initialization each time the shared resource is used. |
|  | Do you obtain your resources late and release them early?  Open shared resources just before you need them and release them as soon as you are finished. Holding onto resources for longer than you need them increases memory pressure and increases contention for these resources if they are shared. |
|  | Do you transfer data in chunks over I/O calls?  If you do need to transfer data over I/O calls in chunks, allocate and pin buffers for sending and receiving the chunks. If you need to make concurrent I/O calls, you should create a pool of pinned buffers that is recycled among various clients rather than creating a buffer on a per-request basis. This helps you avoid heap fragmentation and reduce buffer creation time. |

## Manage Strings

Use the following review questions to assess how efficiently your ASP.NET code manipulates strings:

|  |  |
| --- | --- |
|  | Do you use Response.Write for formatting output?  Identify areas in your code where you concatenate output, such as to create a table, and consider using Response.Write instead. Response.Write is the most efficient method for writing content to the client. |
|  | Do you use StringBuilder to concatenate strings?  If the number of appends is unknown and you cannot send the data to the client immediately by using a Response.Write, use the StringBuilder class to concatenate strings. |
|  | Do you use += for concatenating strings?  Identify places in your code where you perform string concatenation by using the += operator. If the number of appends is unknown, or you are appending an unknown size of data, consider using the StringBuilder class instead. |

**Note:**

String Concatenation can decrease the performance of your application if not done correctly. Strings can be concatenated in many ways viz using “+”, using StringBuilder and using String.Format. First is to add the new string to an existing string. However, this operation can become really expensive if you are concatenating the string within a loop or adding many strings. When you add a string to an existing string, the Framework copies both the existing and new data to the memory, deletes the existing string, and reads data in a new string. This can be very time consuming and costly in lengthy string concatenation operations. The second and many times better way to concatenate strings is using the StringBuilder Class. This class modifies the strings directly in memory without having to make copies of them. It is based on correctly estimating the final size of the entire string and the optimization really depends on your accuracy of estimation. These are things you need to consider before jumping to use StringBuilder always. Third is using String.Format, we can actually insert strings in between other strings using a format specifier and specify indicators where the strings should be inserted. By seeing the time this method takes, this approach proves to be slowest amongst all three.

Some good findings about the comparisons are available on the internet (check out: [http://www.heikniemi.net/hc/archives/000124.html](about:blank) ). From such findings, we recommend to use String Builder class when the number of concatenated strings become more than eight since the findings indicate that the processing time is less through direct concatenation when you have around eight strings.

## Manage Exceptions

Use the following review questions to assess how efficiently your code uses exceptions:

|  |  |
| --- | --- |
|  | Have you implemented an error handler in Global.asax?  Although implementing an error handler in Global.asax does not necessarily increase performance, it helps you to identify unexpected exceptions that occur in your application. After you identify the exceptions that occur, take appropriate action to avoid these exceptions.  Also, remember that Application\_Error in Global.asax does not always catch all exceptions. So include the try catch block in most of the places where the exceptions are likely to happen. |

## Optimize Web Pages

Use the following review questions to assess how efficiently your code uses web pages:

|  |  |
| --- | --- |
|  | Have you taken steps to reduce your page size?  Try to keep the page size to a minimum. Large page sizes place increased load on the CPU because of increased processing and a significant increase in network bandwidth utilization, which may lead to network congestion. Both of these factors lead to increased response times for clients. Consider the following guidelines to help reduce page size:   * Use script includes (script tags rather than interspersing code with HTML). * Remove redundant white space characters from your HTML. * Disable view state for server controls where it is not needed. * Avoid long control names. * Minimize the use of graphics, and use compressed images. * Consider using cascading style sheets to avoid sending the same formatting directives to the client repeatedly. |
|  | Is buffering disabled? Need to discuss with team  Ensure that you have buffering enabled. Buffering causes the server to buffer the output and send it only after it has finished the processing of the page. If buffering is disabled, the worker process needs to continuously stream responses from all concurrent requests; this can be a significant overhead on memory and the processor, especially when you use the ASP.NET process model.  To find out if you have buffering disabled, you can search your code base for the following strings: "buffer" and "BufferOutput."  Make sure that the buffer attribute is set to true on the <pages> element in your application's Web.config file.  <pages buffer="True"> |
|  | Do you use Response.Redirect?  Search your code for "Response.Redirect" and consider replacing it with Server.Transfer. This does not incur the cost of a new request because it avoids any client-side redirection.  You cannot always simply replace Response.Redirect calls with Server.Transfer calls because Server.Transfer uses a new handler during the handler phase of execution. Response.Redirect generates a second request. If you need different authentication and authorization, caching, or other run-time devices on the target, the two mechanisms are not equivalent. Response.Redirect causes an extra request to be sent to the server. Response.Redirect also makes the URL visible to the user. This may be required in some scenarios where you require the user to bookmark the new location.  Use the overloaded Server.Transfer method Server.Transfer(String,bool) instead of Server.Transfer, Response.Redirect, and Response.End to avoid exceptions. |
|  | Do you use Page.IsPostBack?  Check that the logic in your page uses the Page.IsPostBack property to reduce redundant processing and avoid unnecessary initialization costs. Use the Page.IsPostBack property to conditionally execute code, depending on whether the page is generated in response to a server control event or whether it is loaded for the first time. |
|  | Do you validate user input? – We do not validate data on server side  Check that you validate user input on the client to reduce round trips to the server. This also provides better feedback to the user.. |
|  | Have you disabled debugging for production?  Check your Web.config file and ensure debug is set to false in the <compilation> section and check your .aspx pages to ensure debug is set to false. If debugging is enabled, the compiler does not generate optimized code and pages are not batch compiled.  You can check your .aspx pages by using the Findstr.exe file with regular expressions.  C:\Brijesh>findstr /i /r /c:"<%.\*@.\*page.\*debug=.\*true\*.\*%>" \*.aspx  login.aspx:<%@ page Language="VB" Debug="True" %>  main.aspx:<%@ Page Language="c#" Debug="True" %> |
|  | Do you set aggressive timeouts?  Set timeouts aggressively and tune accordingly. Evaluate each page and determine a reasonable timeout. The default page timeout is 90 seconds specified by the executionTimeout attribute in Machine.config. Server resources are held up until the request is processed completely or the execution times out, whichever is earlier.  In most scenarios, users do not wait for such a long period for the requests to complete. They either abandon the request totally or send a new request which further increases the load on the server. |

## View State – Not applicable

Use the following review questions to asses how efficiently your applications use view state:

|  |  |
| --- | --- |
|  | Do you disable view state when it is not required?  Evaluate each page to determine if you need view state enabled. View state adds overhead to each request. The overhead includes increased page sizes sent to the client as well as a serialization and deserialization cost. You do not need view state under the following conditions:   * The page does not post back to itself; the page is only used for output and does not rely on response processing. * Your page's server controls do not handle events and you have no dynamic or data-bound property values (or they are set in code on every request). * If you are ignoring old data and repopulating the server control every time the page is refreshed. |
|  | Have you taken steps to reduce the size of your view state?  Evaluate your use of view state for each page. To determine a page's view state size, you can enable tracing and see each how each control uses it. Disable view state on a control-by-control basis. |

## Server Controls

Use the following review questions to review how efficiently your ASP.NET applications use server controls:

|  |  |
| --- | --- |
|  | Do you use server controls when you do not need to?  Evaluate your use of server controls to determine if you can replace them with lightweight HTML controls or possibly static text. You might be able to replace a server control under the following conditions:   * The data being displayed in the control is static, for example, a label. * You do not need programmatic access to the control on the server side. * The control is displaying read-only data. * The control is not needed during post back processing. |
|  | Do you have deep hierarchies of server controls?  Deeply nested hierarchies of server controls compound the cost of building the control tree. Consider rendering the content yourself by using Response.Write or building a custom control which does the rendering. To determine the number of controls and to see the control hierarchy, enable tracing for the page. |

## Access Data from Aspx Pages

Some form of data access is required by most ASP.NET applications. Data access is a common area where performance and scalability issues are found. Review the following questions to help improve your application's page level data access:

|  |  |
| --- | --- |
|  | Do you page large result sets?  Identify areas of your application where large result sets are displayed and consider paging the results. Displaying large result sets to users can have a significant impact on performance. |

## Data Binding

Use the following review questions to review your code's use of data binding:

|  |  |
| --- | --- |
|  | Do you use Page.DataBind?  Avoid calling Page.DataBind and bind each control individually to optimize your data binding. Calling Page.DataBind recursively calls DataBind on each control on the page. |
|  | Do you use DataBinder.Eval?  DataBinder.Eval uses reflection, which affects performance. In most cases DataBinder.Eval is called many times from within a page, so implementing alternative methods provides a good opportunity to improve performance.  Avoid the following approach.  <ItemTemplate>  <tr>  <td><%# DataBinder.Eval(Container.DataItem,"field1") %></td>  <td><%# DataBinder.Eval(Container.DataItem,"field2") %></td>  </tr>  </ItemTemplate>  Use explicit casting. It offers better performance by avoiding the cost of reflection. Cast the Container.DataItem as a DataRowView if the data source is a DataSet.  <ItemTemplate>  <tr>  <td><%# ((DataRowView)Container.DataItem)["field1"] %></td>  <td><%# ((DataRowView)Container.DataItem)["field2"] %></td>  </tr>  </ItemTemplate>  Cast the Container.DataItem as a String if the data source is an Array or an ArrayList.  <ItemTemplate>  <tr>  <td><%# ((String)Container.DataItem)["field1"] %></td>  <td><%# ((String)Container.DataItem)["field2"] %></td>  </tr>  </ItemTemplate> |

## 

## Calling Unmanaged From Aspx Pages

Use the following review questions to review your code's use of interoperability:

|  |  |
| --- | --- |
|  | Have you enabled AspCompat for calling STA COM components?  Make sure that any page that calls an STA COM component sets the AspCompat page level attribute.  <@%Page AspCompat = "true" %>  This instructs ASP.NET to execute the current page request using a thread from the STA thread pool. By default, ASP.NET uses the MTA thread pool to process a request to a page. If you are using STA components, the component is bound to the thread where it was created. This causes a costly thread switch from the thread pool thread to the thread on which the STA object is created. |
|  | Do you create STA COM components in the page constructor?  Check your pages to ensure you are not creating STA COM components in the page constructor. Create STA components in the Page\_Load, Page\_Init or other events instead.  The page constructor always executes on an MTA thread. When an STA COM component is created from an MTA thread, the STA COM component is created on the host STA thread. The same thread (host STA) executes all instances of apartment-threaded components that are created from MTA threads. This means that even though all users have a reference to their own instance of the COM component, all of the calls into these components are serialized to this one thread, and only one call executes at a time. This effectively bottlenecks the page to a single thread and causes substantial performance degradation.  If you are using the AspCompat attribute, these events run using a thread from the STA thread pool, which results in a smaller performance hit due to the thread switch. |
|  | Do you use Server.Create object?  Avoid using Server.CreateObject and early bind to your components at compile time wherever possible. Server.CreateObject uses late binding and is primarily provided for backwards compatibility.  Search your code base to see if you use this routine and as an alternative, create an interop assembly to take advantage of early binding. |

## Settings in MACHINE.CONFIG

This section is for Information only, since we would never configure these settings ourselves.

Use the following review questions to review your application's deployment plan:

|  |  |
| --- | --- |
|  | Is the thread pool tuned appropriately?  Proper tuning of the CLR thread pool tuned improves performance significantly. Before deploying your application, ensure that the thread pool has been tuned for your application.  Controlling or changing the thread pool count might not be in your hands always. |
|  | Is the memory limit configured appropriately?  Configuring the ASP.NET memory limit ensures optimal ASP.NET cache performance and server stability. In IIS 5.0 or when you use the ASP.NET process model under IIS 6.0, configure the memory limit in Machine.config. With IIS 6.0, you configure the memory limit by using the IIS MMC snap-in. |
|  | Have you removed unnecessary HttpModules?  Including HttpModules that you do not need adds extra overhead to ASP.NET request processing. Check that you have removed or commented out unused HttpModules in Machine.config. |

# Intellectual Property Considerations

|  |  |
| --- | --- |
|  | The implementation MUST NOT USE ANY downloaded source code and design artifacts from any sources including but not limited to Internet for customer projects without a written authorization of the Pune architect. Without architect's approval, even the customer's code and designs must not be used. The architect will have a better idea about the applicability.  If there is no architect assigned to the project, then an approval from Director - Technology or the Pune architect who works in the relevant technology is necessary. Budding architects or technical leads can't approve these artifacts. Obviously, the Program Manager / US Project Manager / US Architect can't approve these changes either. |

# Development and Deployment Considerations (Entity Framework) –

[http://msdn.microsoft.com/en-us/library/ee518967.aspx](about:blank)

# Linq to SQL Best Practices –

|  |  |
| --- | --- |
| 6.1 | Turn off [ObjectTrackingEnabled](about:blank) property of Data Context if not necessary –  If you are trying only to retrieve data as read only, and not modifying anything, you don’t need object tracking. This will allow you to turn off the unnecessary identity management of the objects – hence Data Context will not have to store them because it will be sure that there will be no change statements to generate. |
| 6.2 | Do NOT dump all your DB objects into one single DataContext –  [DataContext](about:blank) represents a single unit of work, not all your database. If you have several database objects that are not connected, or they are not used at all (log tables, objects used by batch processes,etc..). These objects just unnecessarily consume space in the memory hence increasing the identity management and object tracking costs in CUD engine of the DataContext. Instead think of separating your workspace into several DataContexts where each one represents a single unit of work associated with it. You can still configure them to use the same connection via its constructors to not to lose the benefit of connection pooling. |
| 6.3 | Use [CompiledQuery](about:blank) wherever needed –  When creating and executing your query, there are several steps for generating the appropriate SQL from the expression. With [CompiledQuery](about:blank), you compile your query once and store it somewhere for later usage. This is achieved by static [CompiledQuery.Compile](about:blank) method. If you don’t call the query don’t worry about compilation too, since it will be compiled whenever the query is first executed. |
| 6.4 | Filter data down to what you need using [DataLoadOptions.AssociateWith](about:blank) –  When we retrieve data with [Load](about:blank) or [LoadWith](about:blank) we are assuming that we want to retrieve all the associated data those are bound with the primary key (and object id). But in most cases we likely need additional filtering to this. Here is where [DataLoadOptions.AssociateWith](about:blank) generic method comes very handy. This method takes the criteria to load the data as a parameter and applies it to the query – so you get only the data that you need. |
| 6.5 | Turn [Optimistic Concurrency](about:blank) off unless you need it –  LINQ to SQL comes with out of the box [Optimistic Concurrency](about:blank)support with SQL timestamp columns which are mapped to [Binary](about:blank) type. You can turn this feature on and off in both mapping file and attributes for the properties. If your application can afford running on “last update wins” basis, then doing an extra update check is just a waste. |
| 6.6 | Constantly monitor queries generated by the DataContext and analyze the data you retrieve –  As your query is generated on the fly, there is this possibility that you may not be aware of additional columns or extra data that is retrieved behind the scenes. Use [Data Context’s Log](about:blank) property to be able to see what SQL are being run by the Data Context. An example is as follows:  using (NorthwindDataContext context = new NorthwindDataContext())  {  context.Log = Console.Out;  }  Using this snippet while debugging you can see the generated SQL statements in the Output Window in Visual Studio and spot performance leaks by analyzing them. Don’t forget to comment that line out for production systems as it may create a bit of an overhead. |
| 6.7 | Avoid unnecessary attaches to tables in the context –  Since Object Tracking is a great mechanism, nothing comes for free. When you  [Attach](about:blank) an object to your context, you mean that this object was disconnected for a while and now you now want to get it back in the game. DataContext then marks it as an object that potentially will change - and this is just fine when you really intent to do that.  But there might be some circumstances that aren’t very obvious, and may lead you to attach objects that arent changed. One of such cases is doing an AttachAll for collections and not checking if the object is changed or not. For a better performance, you should check that if you are attaching ONLY the objects in the collection those are changed. |
| 6.8 | Be careful of entity identity management overhead –  During working with a non-read only context, the objects are still being tracked – so be aware that non intuitive scenarios this can cause while you proceed. Consider the following DLINQ code:  using (NorthwindDataContext context = new NorthwindDataContext())  {  var a = from c in context.Categories  select c;  }  Very plain, basic DLINQ isn’t it? That’s true; there doesn’t seem any bad thing in the above code. Now let’s see the code below:  using (NorthwindDataContext context = new NorthwindDataContext())  {  var a = from c in context.Categories  select new Category  {  CategoryID = c.CategoryID,  CategoryName = c.CategoryName,  Description = c.Description  };  }  The intuition is to expect that the second query will work slower than the first one, which is **WRONG**. It is actually much [faster than the first one](about:blank).  The reason for this is in the first query, for each row the objects need to be stored, since there is a possibility that you still can change them. But in the 2nd one, you are throwing that object away and creating a new one, which is more efficient. |
| 6.9 | Retrieve only the number of records you need –  When you are binding to a data grid, and doing paging – consider the easy to use methods that LINQ to SQL provides. These are mainly [Take](about:blank) and [Skip](about:blank) methods. |
| 6.10 | Don’t misuse [CompiledQuery](about:blank) –  If you are using [CompiledQuery](about:blank)make sure that you are using it more than once as it is more costly than normal querying for the first time. That’s because the resulting function coming as a CompiledQuery is an object, having the SQL statement and the delegate to apply it. [It is not compiled like the way regular expressions are compiled](about:blank). And your delegate has the ability to replace the variables (or parameters) in the resulting query. |

# ASP .NET MVC

ASP.NET MVC

[http://www.asp.net/mvc/](about:blank)

ASP.NET WEB API

[http://www.asp.net/web-api](about:blank)

|  |  |
| --- | --- |
| 7.1 | Create Extension methods of UrlHelper to generate your url from Route. |
| 7.2 | [Create Extension Method of UrlHelper to map your JavaScript, Stylesheet and Image Folder](about:blank#urlHelperExtension). |
| 7.3 | [Use Bootstrapper in Global.asax](about:blank#bootstrapper) - Create individual task for specific part. Do use Bootstrapper to execute those. This makes your code lot more clean and testable. This will be also helpful when developing a portal kind of app in asp.net mvc where each module can have some startup initialization without affecting others. But if you are developing a small app, you can surly go ahead with the default global.asax. |
| 7.4 | [Decorate your Action Methods with Proper AcceptVerbs Attribute](about:blank#acceptVerbs) - ASP.NET MVC is much more vulnerable comparing to Web Forms. Make sure the action methods that modifies the data only accepts HttpVerbs.Post. Thumb rule is to use HttpVerbs.Post for all data modification actions and HttpVerbs.Get for data reading operations. |
| 7.5 | [Decorate your most frequent Action Methods with OutputCache Attribute](about:blank#outputCache) - Use OutputCache attribute when you are returning the less frequent updated data, prime candidate may be your home page, feed etc etc. You can use it for both Html and Json data types. |
| 7.6 | [Keep your controller free from HttpContext and its tail](about:blank#httpContext) - Make sure your controller does not have to refer the HttpContext and its tail. it will make your life easier when unit testing your Controller. If you need to access anything from HttpContext like User, QueryString, Cookie etc use custom action filter or create some [interface and wrapper](about:blank) and pass it in the constructor. |
| 7.7 | [Use Action Filter to convert to compatible Action Methods parameters](about:blank#actionFilterConvert) |
| 7.8 | [Action Filter Location](about:blank#actionFilterLocation) - If you need the same action filter to all of your controller action methods, put it in the controller rather than each action method. If you want to apply the same action filter to all of your controller create a base controller and inherit from that base controller. |
| 7.9 | [Controller will not contain any Domain logic](about:blank#controller) - Controller should be only responsible for:   * Validating Input * Calling Model to prepare the view * Return the view or redirect to another action   If you are doing any other thing you are doing it in a wrong place, it is rather the model responsibility which you are doing in controller. |
| 7.10 | [Avoid ViewData, use ViewData.Model](about:blank#viewDataDictionary) - Depending upon the dictionary key will not only make your code hard to refactor, also you will have to write the casting code in your view. It is completely okay even you end up per class for each action method of your controller. |
| 7.11 | [Create new ActionResult if required](about:blank#actionResult) - ASP.NET MVC has quite a number of ActionResult for different purposes, but still we might need new ActionResult. For example xml, rss, atom etc. In those cases, instead of using the generic ContentResult, create new ActionResult. |
| 7.12 | [Split your View into multiple ViewUserControl](about:blank#userControl) - Split your view into multiple ViewUserControl when it is getting bigger, it really does not matter whether the same UserControl is reused in another page, it makes the very view much more readable. |
| 7.13 | [HtmlHelper extension](about:blank#htmlHelper) - |
| 7.14 | [Encode](about:blank#encode) - Whatever you receive from the User always use Html.Encode(“User Input”) for textNode and Html.AttributeEncode(“User Input”) for html element attribute. |
| 7.15 | [Do not put your JavaScript codes in your View](about:blank#javascript) - Do not intermix your javascript with the html, create separate js files and put your java script in those files. Some time, you might need to pass your view data in your java script codes, in those cases only put your initialization in the view. |
| 7.16 | [Use jQuery and jQuery UI](about:blank#jQuery) |
| 7.17 | Use validators to validate all inputs coming from user. This is required to avoid possibilities of SQL and JavaScript ingestions. |
| 7.18 | Use error page for showing errors\exception details etc. when any exception occurs. |
| 7.19 | Use resource files to display messages, instructions etc. (anything that is displayed on UI for user to read). This is required for supporting internationalization for the application. |

# Windows Communication Foundation (WCF)

WCF Guidelines and Best Practices

[http://msdn.microsoft.com/en-us/library/ms731197.aspx](about:blank)

## General Design Guidelines

|  |  |
| --- | --- |
|  | All services must adhere to these principles:   * Services are secure. * Service operations leave the system in a consistent state. * Services are thread-safe and can be accessed by concurrent clients. * Services are reliable. * Services are robust. |
|  | Services can optionally adhere to these principles:   * Services are interoperable (services meant to be exposed for B2B communication). * Services are scale-invariant. * Services are available. * Services are responsive. * Services are disciplined and do not block their clients for long. |

## Essentials

|  |  |
| --- | --- |
|  | Place service code in a class library and not in any hosting EXE. |
|  | Provide a parameterized constructor to a service class to aid automated unit testing by passing mock dependent classes. |
|  | Provide a meaningful namespace for contracts.  Use your client’s URL or equivalent URN with a year and month to support versioning.  For example:  [ServiceContract(Namespace = "http://www.sionicmobile.com/2010/05")]  interface IMyContract  {...} |
|  | On Windows Server 2008 or later, prefer WAS hosting to self-hosting. |
|  | Enable metadata exchange. |
|  | Always name all endpoints in the client config file. |
|  | Do not use SvcUtil or Visual Studio 2008 to generate a config file. |
|  | When using Visual Studio to generate the proxy, do clean up the proxy. |
|  | Do not duplicate proxy code. If two or more clients use the same contract, factor the  proxy to a separate class library. |
|  | Always close or dispose of the proxy. |

## Service Contracts

|  |  |
| --- | --- |
| 8.3.1 | Always apply the ServiceContractAttribute on an interface, not a class.  //Avoid  [ServiceContract]  class MyService  {  [OperationContract]  public void MyMethod()  {...}  }  //Correct  [ServiceContract]  interface IMyContract  {  [OperationContract]  void MyMethod();  }  class MyService : IMyContract  {  public void MyMethod()  {...}  } |
| 8.3.2 | Prefix the service contract name with **I**  [ServiceContract]  Interface IMyContract  {...} |
| 8.3.3 | Avoid property-like operations:  //Avoid  [ServiceContract]  interface IMyContract  {  [OperationContract]  string GetName();    [OperationContract]  void SetName(string name);  } |
| 8.3.4 | Create a separate service contract for each functional group of service operations. |

## Data Contracts

|  |  |
| --- | --- |
| 8.4.1 | Avoid inferred data contracts (POCO). Always be explicit and apply the DataContract attribute. |
| 8.4.2 | Use the DataMemberAttribute on properties or read-only public members only. |
| 8.4.3 | Avoid explicit XML serialization on your own types. |
| 8.4.4 | When using the Order property, assign the same value to all members coming from the same level in the class hierarchy. |
| 8.4.5 | Do not mark delegates and events as data members. |
| 8.4.6 | Do not pass .NET specific types such as Type as operation parameters. |
| 8.4.7 | Do not accept or return ADO.NET DataSet and DataTable (or their type-safe subclasses) from operations. Return a neutral representation such as an array. |
| 8.4.8 | Suppress the generation of a generic type parameter hash code and provide a legible type name instead. |
| 8.4.9 | Share data contract across projects in a solution when possible. |

## Instance Management

|  |  |
| --- | --- |
| 8.5.1 | Prefer the per-call instance mode when scalability is a concern. |
| 8.5.2 | If selecting SessionMode.NotAllowed on the contract, always configure the service instancing to InstanceContextMode.PerCall. |
| 8.5.3 | Do not mix sessionful contracts and session-less contracts on the same service. |
| 8.5.4 | Use ordered delivery with a sessionful service. |
| 8.5.5 | Avoid instance deactivation with a sessionful service. |
| 8.5.6 | With durable services, always designate a completing operation. |

## Operations and Calls

|  |  |
| --- | --- |
| 8.6.1 | Do not treat one-way calls as asynchronous calls. |
| 8.6.2 | Do not treat one-way calls as concurrent calls. |
| 8.6.3 | Expect exceptions out of a one-way operation. |
| 8.6.4 | Enable reliability even on one-way calls. Use of ordered delivery is optional for one-way calls. |
| 8.6.5 | Avoid one-way operations on a sessionful contract. If used, make it the terminating  operation:  [ServiceContract(SessionMode = SessionMode.Required)]  interface IMyContract  {  [OperationContract]  void MyMethod1();    [OperationContract(IsOneWay = true, IsTerminating = true)]  void MyMethod2();  } |
| 8.6.6 | Name the callback contract on the service side after the service contract suffixed by Callback:  interface IMyContractCallback  {...}  [ServiceContract(CallbackContract = typeof(IMyContractCallback))]  interface IMyContract  {...} |
| 8.6.7 | Strive to mark callback operations as one-way. |
| 8.6.8 | Use callback contracts for callbacks only. |
| 8.6.9 | Avoid mixing regular callbacks and events on the same callback contract. |
| 8.6.10 | Event operations should be well designed:   * void return type * No out parameters * Marked as one-way operations |
| 8.6.11 | Always provide explicit methods for callback set-up and teardown:  [ServiceContract(CallbackContract = typeof(IMyContractCallback))]  interface IMyContract  {  [OperationContract]  void DoSomething();    [OperationContract]  void Connect();    [OperationContract]  void Disconnect();  }  interface IMyContractCallback  {...} |

## Faults

|  |  |
| --- | --- |
| 8.7.1 | Never use a proxy instance after an exception even if you catch that exception. |
| 8.7.2 | Do not reuse the callback channel after an exception even if you catch that exception as the channel may be faulted. |
| 8.7.3 | Use the FaultContractAttribute with exception classes as opposed to mere serializable types:    //Avoid  [OperationContract]  [FaultContract(typeof(double))]  double Divide(double number1, double number2);    //Correct:  [OperationContract]  [FaultContract(typeof (DivideByZeroException))]  double Divide(double number1, double number2); |
| 8.7.4 | With both service classes and callback classes set IncludeExceptionDetailInFaults to true in debug sessions, either in the config file or programmatically:  public class DebugHelper  {  public const bool IncludeExceptionDetailInFaults =  #if DEBUG  true ;  #else  false;  #endif  }  [ServiceBehavior(IncludeExceptionDetailInFaults = DebugHelper.IncludeExceptionDetailInFaults)]  class MyService : IMyContract  {...} |
| 8.7.5 | In release builds do not return unknown exceptions as faults except in diagnostic scenarios. |

## Concurrency Management

|  |  |
| --- | --- |
| 8.8.1 | Always provide thread-safe access to:   * Service in-memory state with sessionful or singleton services. * Client in-memory state during callbacks. * Shared resources such as files. * Static variables. |
| 8.8.2 | Prefer ConcurrencyMode.Single (the default). It enables transactional access, and it is thread-safe without any effort. |
| 8.8.3 | Keep operations on single-mode sessionful and singleton services short in order to  avoid blocking other clients for long. |

## Security

|  |  |
| --- | --- |
| 8.9.1 | Always protect the message and provide for message confidentiality and integrity. |
| 8.9.2 | With Internet clients, always use message security. |
| 8.9.3 | Allow clients to negotiate the service certification (the default). |
| 8.9.4 | Validate certificates using peer-trust. |