­

Site Binder Project

*Sept. 25, 2014*

­

Coding Standards

**Prepared By:**

**ORACULAR IS**

300 Ohio Street

Oshkosh, Wisconsin 54902

## Revision History

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Description** |
| Sep 25th 2014 | Haris Ahmad | First Iteration |

Contents

[Revision History 1](#_Toc399338926)

[Naming Guidelines 3](#_Toc399338927)

[Capitalization Conventions 3](#_Toc399338928)

[Pascal Casing 3](#_Toc399338929)

[Camel Casing 3](#_Toc399338930)

[DO 3](#_Toc399338931)

[General Naming Conventions 3](#_Toc399338932)

[Word Choice 3](#_Toc399338933)

[DO 3](#_Toc399338934)

[DO NOT 3](#_Toc399338935)

[Names of Assemblies and DLL 4](#_Toc399338936)

[Name of Namespaces 4](#_Toc399338937)

[Do 4](#_Toc399338938)

[Do Not 4](#_Toc399338939)

[Names of Classes, Structs and Interfaces 4](#_Toc399338940)

[Do 4](#_Toc399338941)

[Do Not 5](#_Toc399338942)

[Name of Type Members 5](#_Toc399338943)

[Name of Methods 5](#_Toc399338944)

[Name of Properties 5](#_Toc399338945)

[Name of Variables 5](#_Toc399338946)

[Name of Parameters 5](#_Toc399338947)

[Type Design Guidelines 5](#_Toc399338948)

[Choosing Between Class and Struct 5](#_Toc399338949)

[Abstract Class Design 6](#_Toc399338950)

[Static Class Design 6](#_Toc399338951)

[Do 6](#_Toc399338952)

[Do Not 6](#_Toc399338953)

[Struct Design 6](#_Toc399338954)

[Enum Design 6](#_Toc399338955)

[Member Design Guidelines 6](#_Toc399338956)

[Member Overloading 7](#_Toc399338957)

[Property Design 7](#_Toc399338958)

[Constructor Design 7](#_Toc399338959)

[Design Guidelines for Exceptions 7](#_Toc399338960)

[Exception Throwing 7](#_Toc399338961)

[Using Standard Exception Types 7](#_Toc399338962)

[Do Not Throw 7](#_Toc399338963)

[ASP.NET MVC Standards 8](#_Toc399338964)

[Best Practices 8](#_Toc399338965)

[Isolate Controllers 8](#_Toc399338966)

[No Magic String 8](#_Toc399338967)

[Create ViewModel for each view 8](#_Toc399338968)

[HTML Helper 8](#_Toc399338969)

[Action Methods 8](#_Toc399338970)

[Caching 8](#_Toc399338971)

[Controller and Domain logic 8](#_Toc399338972)

[Domain logic in Views 8](#_Toc399338973)

[Views should not contain presentation logic 8](#_Toc399338974)

[Use POST for "Delete" links instead of GET 9](#_Toc399338975)

[Database Standards 10](#_Toc399338976)

[Database 10](#_Toc399338977)

[Tables 10](#_Toc399338978)

[Columns 10](#_Toc399338979)

[Views 10](#_Toc399338980)

[Stored Procedures 10](#_Toc399338981)

# Naming Guidelines

## Capitalization Conventions

### Pascal Casing

* Used for all identifiers except parameter names and variables.
* Capitalize first letter of each word
  + PropertyDescriptor
  + HtmlTag
  + IOStream

### Camel Casing

* Used only for parameter names and variables
* Capitalize first letter of each word except first word
  + propertyDescriptor
  + ioStream
  + htmlTag

### DO

* Use PascalCasing for all public members, type, and name space names
* User camelCasing for parameter names and variables

## General Naming Conventions

### Word Choice

* Choose easy to understand identifier names
  + e.g. HorizontalAlignment is more comprehensible than AlignmentHorizontal
* Favor comprehension over brevity
  + e.g. CanScrollHorizontally is better than ScrollableX

### DO

* Use semantically relevant names rather than language-specific keywords
  + e.g. GetLength is better name than GetInt

### DO NOT

* Do not Use underscores, hyphens, or any other non-alphanumeric character. Exception: Database model created by Entity Framework will have underscores
* Do not Use Hungarian notation
  + Boolean : bStillGoing
  + Char: cLetterGrade
* Do not Use abbreviations or contractions as part of identifier names
* Avoid Identifier names that conflict with keywords

## Names of Assemblies and DLL

* <Company>.<Component>.dll
  + e.g. Oracular.Admin.dll or OIS.Admin

## Name of Namespaces

The following are examples:

* Oracular.Admin or OIS.Admin
* Oracular.Admin.Menus

### Do

* Prefix namespace names with a company name prefix
* Use a stable, version-independent product name at second level of namespace name.
* Namespace should be used to categorize similar functionality together.
* Consider using plural namespace names where appropriate
  + e.g. System.Collections

### Do Not

* Do not use organizational hierarchies as the basis of names in namespace hierarchies
* Do not use the same name for a namespace and a type in that namespace
  + For example, do not use Debug as a namespace name and then also provide a class named Debug in the same namespace. Several compilers require such types to be fully qualified.
* Do not introduce generic type namespace such as Element, Node, Log and Message

## Names of Classes, Structs and Interfaces

### Do

* Name classes and structs with nouns or noun phrases using PascalCasing
* Name interfaces with adjective phrases
* Consider the ending name of derived classes with the name of base class
  + e.g. AuthorizeAttribute, with its kind of Attribute
* Prefix interface with prefix letter I, to indicate that the type is an interface
* Name Generic Type parameters with descriptive names unless a single-letter name is completely self-explanatory
* Add the suffix “Attribute” to names of custom attribute classes
* Add the suffix “EventHandler” to names of delegates that are used in events
* Add the suffix “Callback” to names of delegates other than those used as event handlers
  + Do not use suffix “Delegate” to a delegate
* User a singular type name for an enumeration unless its values are bit fields
* Use a plural type name for an enumeration with bit fields as values

### Do Not

* Do not give class names a prefix (e.g. “C”)
* Do not use “Enum” suffix in enum type names
* Do not use a prefix on enumeration value names

## 

## Name of Type Members

### Name of Methods

* Give methods’ names that are verbs or verb phrases
  + e.g. CompareTo(…)

### Name of Properties

* Name properties using a noun, noun phrase, or adjective
* Name collection properties with a plural phrase describing the items in the collection instead of singular phrase followed by “List” or “Collection”
  + i.e. Students instead of StudentList

### Name of Variables

The variable-naming guidelines apply to static public and protected and local variables.

* Use camelCasing
* Name variables using a noun, noun phrase, or adjective
* Use descriptive names.

## Name of Parameters

* Use camelCasing
* Use descriptive parameter names
* Consider using names based on parameter’s meaning rather than type

# Type Design Guidelines

## Choosing Between Class and Struct

As a rule of thumb, the majority of types in a framework should be classes. There are, however, some situations in which the characteristics of a value type make it more appropriate to use structs. Examples of situations are listed below:

* If instances of the type are small and commonly short-lived or are commonly embedded in other objects
* It logically represents a single value
* It has an instance size under 16 byte
* It is immutable
* It will not have to be boxed frequently

In all other cases, you should define your types as classes.

## Abstract Class Design

* Do not define public or protected internal constructors in abstract types
* Define a protected or an internal constructor in abstract classes
* Provide at least one concrete type that inherits from each abstract class that you ship

## Static Class Design

* A static class is defined as a class that contains only static members
* Static classes are a compromise between object-oriented design and simplicity

### Do

* Use static classes sparingly
  + Should be used only as a supporting classes for the object-oriented core of the framework
* Declare static classes as sealed, abstract

### Do Not

* Do not treat static classes as a miscellaneous bucket
* Do not override instance members in static classes

## Struct Design

* The general-purpose value type is most often referred to as a struct
* Do Not provide a default constructor for a struct
* Do Not define mutable value types

## Enum Design

* Use an enum to strongly type parameters, properties, and return values that represent sets of values
* Favor using an enum instead of static constants
* Do not use an enum for open sets (such as the operating system version, names of your friends, etc.).

# Member Design Guidelines

## Member Overloading

* Avoid being inconsistent in the ordering of parameters in overloaded members. Parameters with the same name should appear in the same position in all overloads.
* DO NOT use ref or out modifiers to overload members.
* Use member overloading rather than defining members with default arguments.

## Property Design

* Create get-only properties if the caller should not be able to change the value of the property.
* DO NOT use properties with a public setter and a protected getter
* Provide sensible default values for all properties, ensuring that the defaults do not result in a security hole or terribly inefficient code
* Allow properties to be set in any order even if this results in a temporary invalid state of the object
* Preserve the previous value if a property setter throws an exception
* AVOID throwing exceptions from property getters.

## Constructor Design

* CONSIDER providing simple, ideally default, constructors.
* DO minimal work in the constructor.
* DO throw exceptions from instance constructors, if appropriate.
* AVOID explicitly defining default constructors on structs.
* AVOID calling virtual members on an object inside its constructor.
* DO NOT throw exceptions from static constructors.

# Design Guidelines for Exceptions

## Exception Throwing

* DO NOT return error codes
* Do report execution failures by throwing exceptions
* DO NOT use exceptions for the normal flow of control, if possible.
  + For example, you can provide a way to check preconditions before calling a member so users can write code that does not throw exceptions
* DO NOT have public members that can either throw or not based on some option
* DO NOT have public members that return exceptions as the return value or an out parameter
* Consider using exception builder methods
* Do not throw exception from exception filter block
* AVOID explicitly throwing exceptions from final blocks. Implicitly thrown exceptions resulting from calling methods that throw are acceptable.

## Using Standard Exception Types

### Do Not Throw

* System.Exception or System.SystemException
* ApplicationException
* InvalidOperationException
* ArgumentException
* StackOverflowException
* Explicitly throw OutOfMemoryException
* Explicitly throw COMExcpetion, ExecutionEngineExecption and SEHException

# ASP.NET MVC Standards

## Best Practices

### Isolate Controllers

Isolate controllers from dependencies on Http Context, data access classes, configuration, logging etc.

### No Magic String

Never use magic strings in your code. This means hard-coding view names, link text etc.

### Create ViewModel for each view

* Create a specialized view model for each view.
* The role of ViewModel should only be data-binding
* It should not contain any presentation logic.

### HTML Helper

* For generating view html use HtmlHelper
* If the current HtmlHelper is not sufficient, extend it using extension methods. This will keep the design in check

### Action Methods

Decorate your action methods with appropriate verbs like Get or Post as applicable

### Caching

Decorate your most used action methods with OutputCache attribute.

### Controller and Domain logic

Try to keep domain logic away from controller. Controller should only be responsible for:

* Input validation and sanitization.
* Get view related data from the model.
* Return the appropriate view or redirect to another appropriate action method.

### Domain logic in Views

There should be no domain logic in the views. Views must be only responsible for showing the data.

### Views should not contain presentation logic

Views should not contain any presentation logic. For example, if a "Delete" button is to be displayed only for "Admin" role, this should be abstracted away in an Html Helper. This is just an example. There will be many scenarios which will require this abstraction for easy maintenance of views.

### Use POST for "Delete" links instead of GET

Using Delete links (GET) is more vulnerable than using POST.

# Database Standards

## Database

* Database names should use Pascal Casing e.g OracularIntranet
* Name should be a noun.

## Tables

* Table names should use Snake Case e.g EMPLOYEE\_DETAIL.
* All words should be capitalized.
* Table names should be descriptive, and should not use abbreviations.

## Columns

* Column names should use PascalCase e.g EmployeeName
* Column names should be descriptive.
* Column names should not include the type of column e.g txt\_name

## Views

1. View names should use Snake Case e.g Employee\_Detail\_View
2. View names should be suffixed by “View” to distinguish between tables and views.

## Stored Procedures

1. Stored Procedures should be prefixed by sp\_ and should use camel case e.g sp\_getEmployeeDetail.