**Version History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Ver. No.*** | ***Authors*** | ***Date*** | ***Reviewers*** | ***Review Date*** | ***Release Date*** |
| 1.0 | Application Development Team | 27-Aug-2018 | QMF | 31-Aug-2018 | 03-Sep-2018 |
| 2.0 | Application Development Team | 10-Dec-2019 | QMF | 13-Dec-2019 | 16-Dec-2019 |
| 3.0 | Application Development Team | 02-Nov-2020 | QMF | 06-Nov-2020 | 10-Nov-2020 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Change History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Ver. No.*** | ***Section*** | ***Date*** | ***Change Information*** | ***RFC No.*** |
| 1.0 | All | 03-Sep-2018 | New Release | - |
| 2.0 | All | 16-Dec-2019 | Annual Review | - |
| 3.0 | All | 02-Nov-2020 | Annual Review |  |
|  |  |  |  |  |
|  |  |  |  |  |

# STANDARDS FOR CODING IN DOT NET

1. **Objective**

The objective of this document is to define the coding standards to be followed while developing application using Microsoft dotnet frame work.

1. **Scope**

This standard applies to all products / projects developed using VB.Net, ASP.Net, C#.

1. **References to (checklists, forms, guidelines, lists, standards, Templates, other processes)**

| ***Item*** | ***Description*** | ***ID*** |
| --- | --- | --- |
| Checklists | -- | -- |

1. **Document Structure, Glossary of Terms, etc.**

This document describes various formats for:

* File Naming
* Capitalization Styles
* Namespaces
* Classes
* Interfaces
* Enums
* Static Fields
* Parameters
* Methods
* Properties
* Events
* Case Sensitivity
* Avoiding Type Name Confusion
* Word Choice
* Variable Naming Convention
* Error Trapping and Messaging

1. **File Naming**

* The following file type tags must be used when specifying file 'Name' attributes

|  |  |
| --- | --- |
| ***File Type Tags*** | ***Type / Meaning*** |
| frm | Form module (.frm). |
| mod | Code module (.bas). |
| Cls | Private class module (.cls). |
| <none> | Public class module (.cls). |
| I | Public class module (.cls) used to define an interface only. |
| Dsr | Designer module (.dsr). |
| uctl | User control module (.ctl). |
| udoc | User document module (.dob). |

* The body element of source file 'Name' attributes must always begin with a capital letter, use mixed case and contain no underscore characters. Eg. frmCustomerDetails

1. **Capitalization Styles**

* The following section describes different ways of capitalizing identifiers

## Pascal Casing

* This convention capitalizes the first character of each word. Eg. color Bit Converter

## Camel Casing

* This convention capitalizes the first character of each word except the first word
* Eg : backgroundColor, totalValueCount

## All Uppercase

* The syntax for declaring a function is,
* Only use all uppercase letters for an identifier if it contains an abbreviation.
* Eg. System.IO, System.WinForms.UI
* The following table summarizes the capitalization style for the different kinds of identifiers:

| *Type* | *Case* | *Notes* |
| --- | --- | --- |
| Namespace | PascalCase | CompanyName.TechnologyName E.g.Microsoft.Office.PowerPoint |
| Class | PascalCase |  |
| Attribute Class | PascalCase | Has a suffix of Attribute |
| Exception Class | PascalCase | Has a suffix of Exception |
| Constant | PascalCase |  |
| Enum type | PascalCase |  |
| Namespace | PascalCase | CompanyName.TechnologyName  E.g. Microsoft.Office.PowerPoint |
| Class | PascalCase |  |
| Attribute Class | PascalCase | Has a suffix of Attribute |
| Exception Class | PascalCase | Has a suffix of Exception |
| Constant | PascalCase |  |
| Enum type | PascalCase |  |
| Enum values | PascalCase |  |
| Event | PascalCase |  |
| Interface | PascalCase | Has a prefix of I |
| Local variable | camelCase |  |
| Method | PascalCase |  |
| Namespace | PascalCase |  |
| Property | PascalCase |  |
| Public Instance Field | PascalCase | Rarely used (use a property instead) |
| Protected Instance Field | camelCase | Rarely used (use a property instead) |
| Parameter | camelCase |  |

1. **Name space**

* Use plural namespace names where appropriate. Eg. System.Collections
* Exceptions to this rule are brand names and abbreviations. Eg System.IO
* Do not have namespaces and classes with the same name

1. **Classes**

* Name the classes with nouns or noun phrases.
* Never use any prefix which are reserved names
* Never use any underscores.

public class FileStream { … }  
public class Button { … }  
public class String { … }

1. **Interfaces**

* Name interfaces with nouns or noun phrases, or adjectives describing behavior. Eg. IComponent (descriptive noun), ICustomAttributeProvider (noun phrase), and IPersistable (adjective).
* Never use any underscores.
* Never prefix interface with reserved names or beginning with the letter “I”, to indicate that the type is an interface.
* Use similar names when defining a class/interface pair where the class is a standard implementation of the interface. The names should differ only by the “I” prefix in the interface name. This approach is used for the interface IComponent and its standard implementation, Component.

public interface IComponent { … }  
public class Component : IComponent { … }  
public interface IServiceProvider{ … }  
public interface IFormatable { … }

1. **Enums**

* Use a singular name for enums
* Use a plural name for bit fields
* Define enumerated values using an enum if they are used in a parameter or property. This gives development tools a chance at knowing the possible values for a property or parameter

public enum FileMode{  
 Create,  
 CreateNew,  
 Open,  
 OpenOrCreate,  
 Truncate  
}

* Do use the Flags custom attribute if the numeric values are meant to be bitwise ored together

[Flags]  
public enum Bindings {  
 CreateInstance,  
 DefaultBinding,  
 ExcatBinding,  
 GetField,  
 GetProperty,  
 IgnoreCase,  
 InvokeMethod,  
 NonPublic,  
 OABinding,  
 SetField  
 SetProperty,  
 Static  
}

* Use int as the underlying type of an enum. (An exception to this rule is if the enum represents flags and there are more than 32 flags, or the enum may grow to that many flags in the future, or the type needs to be different from int for backward compatibility.)
* Use enums only if the value can be completely expressed as a set of bit flags. Do not use enums for open sets (such as operating system version).
* Never use a family-name prefix on enum.
* Never use any “Enum” suffix on enum types.

1. **Static fields**

* Name static members with nouns, noun phrases, or abbreviations for nouns.
* Never use Hungarian-type prefixes on static member names.

1. **Parameters**

* Use descriptive names such that a parameter’s name and type clearly imply its meaning
* Prefer names based on a parameter’s meaning, to names based on the parameter’s type. It is likely that development tools will provide the information about type in a convenient way, so the parameter name can be put to better use describing semantics rather than type
* Never use reserve parameters for future use. If more data is need in the next version, a new overload can be added
* Never use Hungarian-type prefixes

Type GetType (string typeName)  
string Format (string format, object [] args)

1. **Methods**

* Name the methods with verbs or verb phrases

RemoveAll()

GetCharArray()

Invoke()

1. **Properties**

* Consider having a property with the same as a type. When declaring a property with the same name as a type, also make the type of the property be that type. In other words, the following is okay

public enum Color {...}  
public class Control {  
 public Color Color { get {...} set {...} }  
}

* But this is not

public enum Color {...}  
public class Control {  
 public int Color { get {...} set {...} }  
}

* In the latter case, it will not be possible to refer to the members of the Color enum because Color.Xxx will be interpreted as being a member access that first gets the value of the Color property (of type int) and then accesses a member of that value (which would have to be an instance member of System.Int32)

1. **Events**

* Name the events with a verb.
* Name the event handlers with the “EventHandler” suffix.

public delegate void MouseEventHandler(object sender, MouseEvent e);

* Name the event argument classes with the “EventArgs” suffix.

public class MouseEventArgs : EventArgs {  
 int x;  
 int y;  
 public MouseEventArgs(int x, int y)   
 { this.x = x; this.y = y; }  
 public int X { get { return x; } }   
 public int Y { get { return y; } }   
}

* Name the event names that have a concept of pre- and post-operation using the present and past tense (do not use BeforeXxx/AfterXxx pattern). For example, a close event that could be canceled would have a Closing and Closed event

public event ControlEventHandler ControlAdded {  
 //..  
}

* Use two parameters named sender and e. The sender parameter represents the object that raised the event, and this parameter is always of type object, even if it is possible to employ a more specific type. The state associated with the event is encapsulated in an instance e of an event class. Use an appropriate and specific event class for its type

public delegate void MouseEventHandler(object sender, MouseEvent e);

1. **Case Sensitivity**

* Never use names that require case sensitivity. Components might need to be usable from both case-sensitive and case-insensitive languages. Since case-insensitive languages cannot distinguish between two names within the same context that differ only by case, components must avoid this situation
* Never do the following

namespace ee.cummings;  
namespace Ee.Cummings;

void foo(string a, string A)

System.WinForms.Point p;  
System.WinForms.POINT pp;

int Foo {get, set};  
int FOO {get, set}

void foo();  
void Foo();

1. **Avoiding type name confusion**

* In a multi-language environment, take care to avoid language-specific terminology. This section describes a set of rules that help avoid type name confusion
* Use semantically interesting names rather than type names
* In the rare case that a parameter has no semantic meaning beyond its type, use a generic name. For example, a class that supports writing a variety of data types into a stream might have:

void Write(double value);  
void Write(float value);  
void Write(long value);  
void Write(int value);  
void Write(short value);

* Rather than a language-specific alternative such as:

void Write(double doubleValue);  
void Write(float floatValue);  
void Write(long longValue);  
void Write(int intValue);  
void Write(short shortValue);

* In the extremely rare case that it is necessary to have a uniquely named method for each fundamental data type, do use the following universal type names: Sbyte, Byte, Int16, UInt16, Int32, UInt32, Int64, UInt64, Single, Double, Boolean, Char, String, and Object. For example, a class that supports reading a variety of data types from a stream might have:

double ReadDouble();  
float ReadSingle();  
long ReadIn64();  
int ReadInt32();  
short ReadInt16();

* Rather than a language-specific alternative such as:

double ReadDouble();  
float ReadFloat();  
long ReadLong();  
int ReadInt();  
short ReadShort();

1. **Word Choice**

* Do avoid using class names duplicated in heavily used namespaces. For example, don’t use the following for a class name

System Collections Forms UI

* Do not use abbreviations in identifiers.
* If you must use abbreviations, do use camelCase for any abbreviation containing more than two characters, even if this is not the usual abbreviation

1. **Variable Naming Convention**

## Database

|  |  |
| --- | --- |
| Table | “AW” + table name |
| Char or VarChar | field name |
| Integer / Numeric | field name |
| Bit | field name |
| Date | : field name |
| Stored Procedure | “SP\_” + stored procedure name (E.g.: sp\_Insert\_AWWarehouse ) |

## Coding

| ***Variable Type*** | ***Abbreviation*** |
| --- | --- |
| String | **“str”** |
| Bolean | **“bln”** |
| Variant | **“var”** |
| Integer | **“int”** |
| Float | **“flt”** |
| Long | **“lng”** |
| Double | **“dbl”** |
| Date | **“dt”** |
| Array | **“arr”** |
| Arraylist | **“arrlst”** |
| Connection | **“con”** |
| CommandObject | **“cmd”** |
| Dataset | **“ds”** |
| DataReader | **“dr”** |
| DataAdapter | **“da”** |
| DataView | **“dv”** |
| DataTable | **“dt”** |
| DataColumn | **“dcol”** |
| **DataRow** | **“drow”** |

## Front End

| ***Artifact*** | ***Naming*** |
| --- | --- |
| Project | **“prj” + project name** |
| Forms | **“frm” + functionality name e.g. frmSupplierMaster.aspx** |
| AdRotator | **“adrot” + field name** |
| Button | **“cmd”+ field name e.g. cmdSave** |
| CheckBox | **“chk” + field name** |
| CheckBoxList | **“chklst” + field name** |
| Class | **“cls” + functionality name** |
| ClassLibrary | **“clslib” + functionality name** |
| Calender | **“cld” + field name** |
| CompareValidator | **“cmpvld” + field name** |
| CustomValidator | **” cusvld” + field name** |
| DropDownList | **“cmb”+ field name** |
| DataGrid | **“dtg” + table/collection name e.g. dtgBuyer** |
| DataList | **“dtlst” + table/ collection name** |
| Lable | **“lbl”+ field name** |
| ListBox | **“lst” + field name** |
| LinkButton | **“lnkbtn” + field name** |
| Link | **: “lnk”+ field name** |
| HyperLink | **“hlnk” + field name** |
| Multipage | **: “mup” + Fieldname** |
| Image | **: “img” + field name** |
| Object | **: “obj” + functionality name** |
| Panel | **: “pnl”+ functionality name** |
| Repeater | **: “rpt” + table/ collection name** |
| RadioButton | **: “rdo” + field name** |
| RadioButtonList | **: “rdolst” + field name** |
| TexBox | **: “txt”+ field name** |
| RequiredFieldValidator | **: “reqvld” + field name** |
| RangeValidator | **: “rngvld” + field name** |
| RegularExpressionValidator | **” regexpvld” + field name** |
| Tab Strip | **“tab” + fieldname** |
| **ValidationSummary** | **“vldsum” + functionality name** |

## Error Trapping and Messaging

| ***Error Code*** | ***Error Description*** |
| --- | --- |
| E001 | **Error while adding record.** |
| E002 | **Error while updating record.** |
| E003 | **Error while deleting record.** |
| E004 | **Error while displaying records.** |
| E005 | **Login Failed.** |
| E006 | **User Name and Old password do not match.** |
| E007 | **The Employee has been Already selected for the project.** |
| E008 | **should be at least** |
| E009 | **should contain only characters.** |
| E010 | **should be numeric.** |
| E011 | **should contain alphanumeric characters.** |
| E012 | **Special characters not allowed in** |
| E013 | **incorrect format.** |
| E014 | **Page expired, Re-Login.** |
| E015 | **Page cannot be displayed.** |
| E016 | **Enter value for** |
| E017 | **Should not be left blank.** |
| E018 | **Invalid** |
| E019 | **Please Select** |
| E020 | **New password and Confirm Password does not match** |
| E021 | **Reference Of Value To be Deleted Found. Cannot Delete** |
| E022 | **Cannot exceed the maximum length of** |
| E023 | **Only nummeric and '-' characters are allowed for** |
| M001 | **You are successfully logged on.** |
| M002 | **Your password has been changed successfully.** |
| M003 | **Your password has been reset.** |
| M004 | **Search is complete.** |
| M005 | **No results found for your search.** |
| M006 | **successfully added.** |
| M007 | **successfully modified.** |
| M008 | **successfully deleted.** |
| M009 | **No records found for the selected criteria** |
| M010 | **Record Already Exists** |
| M011 | **You are successfully logged off** |
| M012 | **Enter only alphabets** |
| M012 | **Enter only alphabets** |
| M013 | **Enter only Numeric values** |
| M014 | **Please Enter Company Name.** |
| M015 | **Current Log Prefix is Already Existing** |
| **M016** | **No Permissions are assigned to the Form and hence it is Deleted from the List** |