Title:

Hope Technik C# Programming Language Standard

Reference Number:	**RND-REF-LTK-001**
Issue/Revision Index:	**1.0**
Last Change:	**10-10-15**

WRITTEN BY :

Lavon Teng

Contents

Distribution

H.Lim
P.Sheng
L.Teng

Change Record

Version	Date
1.0	19/10/15

Terms, Acronyms and Abbreviations

C#	C Sharp

Applicable Documents

1	n/a
2	

Reference Documents

1	n/a
2	

Purpose

The purpose of this document is to provide a standard for Hope Technik code written in the C# programming language.

The standard is intended to increase readability, reliability, testability, maintainability and portability of C# programs. Using a consistent style produces a final product which appears as the creation of a team working together as opposed to individuals working independently and attempting to combine their efforts late in the process.

Naming Conventions

These are the industry-accepted naming conventions for .NET programs. For additional information, please see the MSDN help documentation:

https://msdn.microsoft.com/en-us/library/ms229002.aspx

Note: Microsoft only suggests conventions for public and protected items.

Type	Standard / Convention	Example

Namespaces	Pascal Case, no underscores.
	
	Use CompanyName.TechnologyName as root.
	
	Note that any acronyms of three or more letters should be pascal case (Xml instead of XML) instead of all caps.
		`AppliedIS.TimeCard.BusinessRules`
		`IrritatedVowel.Controllers`
		`PeteBrown.DotNetTraining.InheritanceDem o PeteBrown.DotNetTraining.Xml`

Assemblies	If the assembly contains a single name space, or has an entire self-contained root namespace, name the assembly the same name as the namespace.	`AppliedIS.TimeCard.BusinessRules.dll` `IrritatedVowel.Controllers.dll`

Classes and Structs	Pascal Case, no underscores or leading "C" or "cls".
	
	Classes may begin with an "I" only if the letter following the I is not capitalized, otherwise it looks like an Interface. Classes should not have the same name as the namespace in which they reside. Any acronyms of three or more letters should be pascal case, not all caps. Try to avoid abbreviations, and try to always use nouns.
		`Widget`
		`InstanceManager`
		`XmlDocument`
		`MainForm`
		`DocumentForm`
		`HeaderControl`
		`CustomerListDataSet (typed dataset)`

Collection Classes	Follow class naming conventions, but add Collection to the end of the name	`WidgetCollection`
Delegate Classes	Follow class naming conventions, but add Delegate to the end of the name	`WidgetCallbackDelegate`
Exception Classes	Follow class naming conventions, but add Exception to the end of the name	`InvalidTransactionException`
Attribute Classes	Follow class naming conventions, but add Attribute to the end of the name	`WebServiceAttribute`
Interfaces	Follow class naming conventions, but start the name with "I" and capitalize the letter following the I	`IWidget`

Enumerations	Follow class naming conventions. Do not add "Enum" to the end of the enumeration name. If the enumeration represents a set of bitwise flags, end the name with a plural.	`SearchOptions (bitwise flags)`
		
		`AcceptRejectRule (normal enum)`

Functions and Subs	Pascal Case, no underscores except in the event handlers. Try to avoid abbreviations. Many programmers have a nasty habit of overly abbreviating everything. This should be discouraged. Functions and subs must differ by more than case to be usable from case-insensitive languages like Visual Basic .NET	VB: `Public Sub DoSomething(...)` C#: `public void DoSomething(...)`
Properties and Public * Member Variables	Pascal Case, no underscores. Try to avoid abbreviations. Members must differ by more than case to be usable from case-insensitive languages like Visual Basic .NET.	VB: `Public Property RecordID As Integer` C#: `public int RecordID`
Parameters	Camel Case. Try to avoid abbreviations. Parameters must differ by more than case to be usable from case-insensitive languages like Visual Basic .NET.	VB: `ByRef recordID As Integer` C#: `ref int recordID`
Procedure-Level Variables	Camel Case	VB: `Dim recordID As Integer` C#: `int recordID ;`
Class-Level Private and Protected Variables	Camel Case with Leading Underscore. In VB.NET, always indicate "Protected" or "Private", do not use "Dim". Use of "m_" is discouraged, as is use of a variable name that differs from the property by only case, especially with protected variables as that violates compliance. Keep all potentially protected member variables private, and supply protected accessors and mutators instead.	VB: `Private _recordID As Integer` C#: `private int _recordID ;`

Constants	Capitalised. Use underscores (if needed) to enhance readability.	`const int MAX_ATTEMPTS = 3;`

Controls on Forms	Modified Hungarian Using .NET Class Names, or a common "control" prefix for all controls. This keeps the controls together in intellisense and makes UI programming much easier.	`Button btn,`
		`TextBox txt,`
		`Label lbl,`
		`MainMenu menu,`
		`CheckBox chk,`
		`RadioButton rdo,`
		`GroupBox grp,`
		`PictureBox pic,`
		`Panel panel,`
		`DataGrid grid,`
		`ListBox lst,`
		`CheckedListBox chkLst,`
		`ComboBox cbo,`
		`ListView lstView,`
		`TreeView tree,`
		`TabControl tab,`
		`DateTimePicker dateTimePicker,`
		`MonthCalendar monthCalendar,`
		`HScrollBar hScrollBar,`
		`VScrollBar vScrollBar,`
		`Timer timer,`

```
Splitter splitter,

DomainUpDown domUpDown,,

NumericUpDown numUpDown,

TrackBar trackBar,

ProgressBar progBar,

RichTextBox richTxt,

ImageList imgList,

HelpProvider helpProv,

ToolTip toolTip,

ContextMenu contextMenu,

ToolBar toolBar,

StatusBar statusBar,

NotifyIcon notifyIcon,

OpenFileDialog openFileDlg,

SaveFileDialog saveFileDlg,

FontDialog fontDlg,

ColorDialog colorDlg,

PrintDialog printDlg,

PrintPreviewDialog printPrevDlg,

PrintPreviewControl printPrevCtrl,

ErrorProvider errProvider,

PrintDocument printDoc,
                              PageSetupDialog pageSetupDlg,

                              TabPage tabPage,

                              MenuItem menuItem
```

Note: The absence of Hungarian Notation except in visual controls.

* Public class-level variables are generally frowned upon. It is considered to be a better practice to use property procedures to provide read and/or write access to a private member variable. If you must expose a member variable to other classes using "Public", follow the property naming conventions.

Layout

The order of code within a class shall generally adhere to the following order:

1. Constants and Read-Only
2. Enumerations
3. Events
4. Member variables
5. User Interface Controls
6. Properties
7. Constructor
8. Dispose Function
9. Member Functions

Indentation – Tabs and Spaces

Indentation

In general braces will be aligned to indicate the next innermost layer of code.

Each new layer of code shall be indented by a single tab.

IF statements

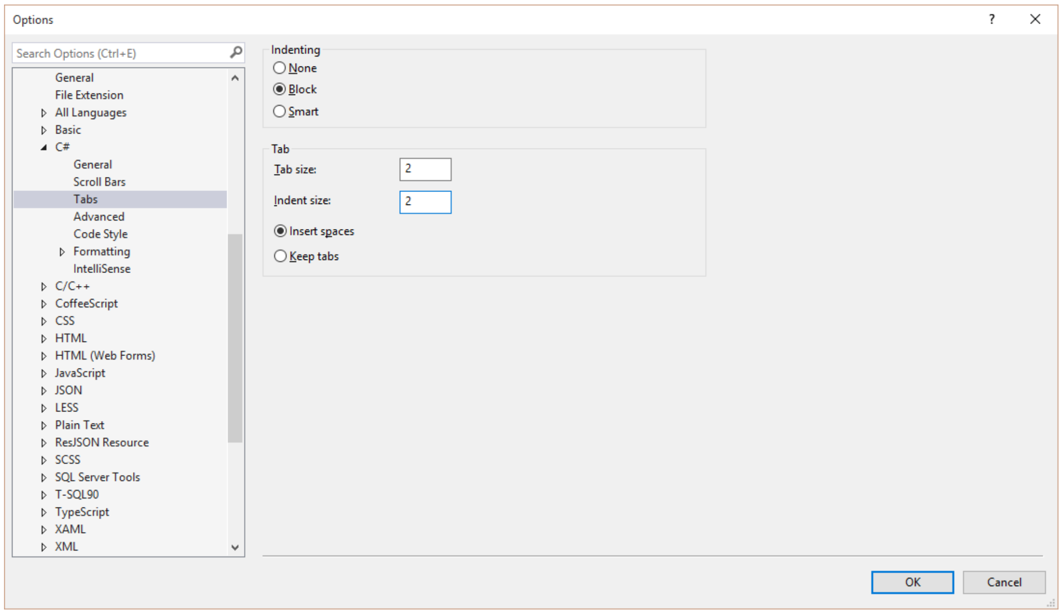
```
if(condition)
{
        do this code…
}
else
{
        else do this…
}

NextFunction( );
```

SWITCH Statements:

```
switch(letter)
{
  case('a'):
    doThis( );
    break;
  case('b'):
    break;
}
```

Tab Settings

The settings for TABS in the Visual Studio Development environment shall be set to the following:



Indenting
None
[✓ Block]
Smart

Tab
Tab size: 2
Indent size: 2
[✓ Insert spaces]
Keep tabs

Comments

Comments shall not simply restate the C# syntax or semantics, but shall clarify the associated objects and functions at a more descriptive level than the source code. Comments should be grammatically correct, and shall address a reader who is a C# programmer.

Certain comments, such as for class member data and global constants and variables, which will be incorporated into the documentation should be written as a sentence with appropriate capitalisation and periods.

Function Comments

All member functions, properties and member variables that are created will have an associated Smart Comment Editing section (///). This will facilitate simple creation of Class Documentation.

Comments shall not simply restate the C# syntax or semantics, but shall clarify the associated objects and functions at a more descriptive level than the source code. Comments should be grammatically correct, and shall address a reader who is a C# programmer.

Block Commentary

Block commentary shall be used for prologs and any other multi-line comments.
Prologs shall be aligned at the left margin. Block comments associated with compound statements shall be aligned at the same level of indentation as the compound statement being explained.

Double slash ("`//`") shall be placed at the left edge of all lines in block commentary.

The slash-asterisk style of commentary ("`/*..... */`") shall NOT be used.

In-Line Commentary

All data type, variable and constant declarations should be commented when such commentary aids in understanding the code or when required to support automated document generation.

Comment statements shall be located either before the statement being clarified, or to the right of the statement, based on available space and programmer preference. Comments which are not to the right of code shall be placed at the same indentation level as the surrounding code.

For comments that must follow a statement and have alignment requirements, comments should be on the following line and indented to the correct tab stop.

Double slash ("`//`") is the required style of commenting. The slash-asterisk style ("`/* ..... */`") is not allowed.

Commentary examples:

```
// This is an example of in-line commentary
…..code;        // This is also a comment

int var1;        // This is a data comment.
int variable1;   // This is the comment for variable1 which lines up
                 // with the comment for var1.
```