Agenda

- Naming Guidelines
- Coding Guidelines
- Language Guidelines

Naming Guidelines (1 of 3)

- General Guidelines:
  - Always use Camel Case or Pascal Case. Never use Hungarian notation.
  - Do not use a prefix for member variables (_, m_, s_, etc.). Use “this” in C# to distinguish between local and member variables.
  - Choose meaningful and specific names.
  - Avoid using abbreviations unless the variable name is very long.
  - Do not use names that begin with a numeric character.
  - Do not use C# reserved words as names.
  - Avoid naming conflicts with existing .NET Framework namespaces or types.
  - Avoid adding redundant or meaningless prefixes and suffixes to identifiers.
  - Do not include the parent class name within a property name.

Naming Guidelines (2 of 3)

Identifier Naming Usage

Identifier	Naming Convention	Example
Project File	Pascal Case	CustomerManagement.Web.csproj
Source File	Pascal Case	Login.cs → public class Login {…}
Namespace	Pascal Case	namespace System.Drawing
Class	Pascal Case	public class CurrencyCalculator {….}
Interface	Pascal Case	interface IBook Always prefix interface name with capital I
Method	Pascal Case	public void GetCustomer {…}

Property	Pascal Case	public int Price
		{
		get {..}
		set{..}
		}

Naming Guidelines (3 of 3)

Identifier Naming Usage (cont.)

Identifier	Naming Convention	Example
Constants	Upper Case	public const string AUTHORNAME
Enum type	Pascal Case	public enum BorderColor {…}
Enum Values	Pascal Case	RedColor, BlueColor
Event	Pascal Case	public event EventHandler LoadPlugIn;
Exception class	Pascal Case	WebException Will always end with suffix Exception.
Parameter	Camel Case	public void Display(string bookName) {…}
Variable	Camel Case	redValue , totalAmount Use meaningful names. Avoid single character variable names.

Coding Guidelines (1 of 3)

- Importance of Coding Structure:
  - Improper styling of code not only makes it difficult for others to understand, it also makes it difficult to maintain.
  - A consistent layout, format of code, and proper organizing of code:
    - Helps in creating maintainable code.
    - Helps in creating a readable, clearer code that is easier to understand.

Coding Guidelines (2 of 3)

- Formatting:
  - Declare one namespace per file.
  - Avoid putting multiple classes in a single file.
  - Place curly braces { and } on a new line.
  - In conditional statements, always use curly braces ({ and }).
  - Use a tab and indentation size of 4 always.
  - Declare each variable independently not in a single line (statement).
  - Place namespace “using” statements together at the top of file.
  - Group internal class implementation by type in the following order:
    - Member variables.
    - Constructors and finalizers.
    - Properties.
    - Methods.
  - Segregate interface implementation by using #region statements:
    - Nested enums, structs, and classes.
  - Recursively indent all code blocks contained within braces.
  - Use white space (CR/LF, tabs, etc) liberally to separate and organize code.

Coding Guidelines (3 of 3)

- Commenting Guidelines:
  - Each file should start with a copyright notice similar to:

```
//------------------------------------------------------------------------
// <copyright file="FileName.cs" company="Accenture">
//     Copyright (c) Accenture.  All rights reserved.
// </copyright>
//------------------------------------------------------------------------
```

  - Use ‘//’ for inline comments and ‘/* .. */’ for block comments.
  - Use inline-comments to explain assumptions, known issues, and algorithm insights.
  - Only use C# comment-blocks for documenting the API.
  - Always use ‘///’ for header comments (for example, method header comments).
  - Always add CDATA tags to comments containing code and other embedded markup in order to avoid encoding issues.

Language Guidelines (1 of 4)

- Variables and Types:
  - Try to initialize variables where you declare them.
  - Use the simplest data type, list, or object required:
    - For example, use int over long unless you know you need to store 64 bit values.
  - Always use the built-in C# data type aliases, not the .NET Common Type System (CTS):
    - For example, use short instead of System.Int16; Use int instead of System.Int32.
  - Declare member variables as private only:
    - Use properties to provide access to them.
  - Avoid specifying a type for an enum:
    - Use default of int unless there is an explicit need for long.
  - Avoid declaring inline string literals:
    - Use constants or resources instead.
  - Avoid direct casts. Instead, use the as operator and check for null.
  - Floating point values should include at least one digit before the decimal place and one after:
    - Example: totalPercent = 0.05f.

Language Guidelines (2 of 4)

- Try to use the “@” prefix for string literals instead of escaped strings.
- Prefer StringBuilder over string concatenation.
- Never concatenate strings inside a loop.
- Always use descriptive variable names which would clearly explain the purpose of the variable. Never use i, j etc. for variable names.
- Avoid using foreach to iterate over immutable value-type collections (e.g. String arrays):
  - Do not modify enumerated items within a foreach statement.
- Only use switch/case statements for simple operations with parallel conditional logic.
- Never declare an empty catch block.
- Avoid nesting a try/catch within a catch block.
- Only use the finally block to release resources from a try statement.
- Always check event and delegate instances before invoking.
- Use the default EventHandler and EventArgs for most simple events.

Language Guidelines (3 of 4)

- Spacing Guidelines
  - Do use a single space after a comma between function arguments.

    Right: `Read(myChar, 0, 1);`

    Wrong: `Read(myChar,0,1);`
  - Do not use a space after the parenthesis and function arguments.

    Right: `CreateFoo(myChar, 0, 1)`

    Wrong: `CreateFoo( myChar, 0, 1 )`
  - Do not use spaces between a function name and parenthesis.

    Right: `CreateFoo()`

    Wrong: `CreateFoo ()`
  - Do not use spaces inside brackets.

    Right: `x = dataArray[index];`

    Wrong: `x = dataArray[ index ];`
  - Do use a single space before flow control statements.

    Right: `while (x == y)`

    Wrong: `while(x==y)`
  - Do use a single space before and after comparison operators.

    Right: `if (x == y)`

    Wrong: `if (x==y)`

Language Guidelines (4 of 4)

- General Guidelines:
  - Have only one public class per '.cs' file.
  - Keep each line of code under 120 characters.
  - Break a line preferably at the following places to improve readability:
    - At a logical condition if any.
    - A ';' if any.
    - After a ',' in case of multiple parameters.
  - Do not leave commented code in the file unless it will be used in the future.
  - Only uncommented and compilable code should be kept before the final build or final delivery.