Programmers Guideline

July 30th 2008
Developer Team Production Pilot

Programmers Guideline

Contents

1. About the Programmers Guideline

This document may be read as a guide to write reliable and readable programs in C# for the XY Project.

2. Basic rules for programming

- Warnings in released code are not allowed!

3. Namespaces and projects

- Every namespace within Pilot starts with the keyword '`Pilot`' (e. g. `Pilot.Production.DAL`).

- Every project (within the Visual Studio) starts with a defined keyword (e. g. `Pilot.Production.DAL`).

- The full namespaces and project descriptions (within Visual Studio) are defined outside this document for every part of the Pilot (e. g. separately for Production and Reporting System).

4. File Organization

4.1 C# Source Files

- Put every class in a separate file and name the file like the class name (with .cs extension of course).

4.2 Directory Structure

- Create a directory for every namespace.

  Example:
  For *Pilot.Production.DAL* use \\*Pilot**Production**DAL* as path!

4.3 Program and configuration files

- The name of every program file (executables, DLLs, ...) and of specific configuration files (initialisation files, ...) has to start with a defined prefix (e. g. '`Pilot.Production`') for the Data Access Layer of the Pilot Production System this results in `Pilot.Production.DAL.dll`).

5. Regions

Every class has to be divided into the following regions, if they are necessary. Between regions must be a space of 2 blank lines.

```
#region Constants
#endregion

#region Enumerations
#endregion

#region Fields
#endregion

#region Constructor / Destructor
#endregion

#region Properties
#endregion

#region Events
#endregion

#region Methods
#endregion

#region Inner Classes
#endregion
```

6. Indentation

6.1 Wrapping Lines

If an expression will not fit on a single line, break it up according to these general principles.

- Break after a comma or operator!
- Align the new line with the beginning of the expression at the same level on the previous line.

  Example: method call

  ```
  longMethodCall(expr1, expr2,
                 expr3, expr4, expr5);
  ```

- Try to avoid breaking arithmetic expressions.
- If this is not possible than try to do it this way:

  Example: arithmetic expression

  Prefer:
  ```
  var = a * b / (c - g + f) +
        4 * z;
  ```

  Bad Style:
  ```
  var = a * b / (c - g +
        f) + 4 * z;
  ```

6.2 White Spaces

Use tabulator for indention! The tabulator size is four characters.

7. Comments

The following rules refer to the comments for the description of classes and the comments within classes and methods. All comments have to be written in English!

7.1 Comments of classes and methods

- Every class and every method must start with an explaining comment. Therefore the possibilities for the creation of HTML-documentations within the Visual Studio have to be used. For detailed information read the appendix (contains a short example) or the corresponding MSDN-Articles (e. g. 'Recommended Tags for Documentation Comments').

  Use the following tags:

  <summary>-tag
  ... to give a **general description** of the class or method. This contains information about the creation and update of a class of method with the name of the programmer and the date.

  <remarks>-tag
  ... to give more **detailed information** of the class or method. At 'class level' this description is appended in the **footer**.

  <param>-tag
  ... to describe every **input parameter** of a method.

  <returns>-tag
  ... to describe the **return value** of a method.

  <exception>-tag
  ... to describe **when and why an exception** will be thrown.

  <br>-tag or **<para>**-tag
  ... to **force** paragraphs.

  <a href ...>
  ... to **refer to** other documents.

  The created web pages have to be saved in a specific sub folder within the project folder.

- For properties use the **<value>**-tag!

7.2 Block Comments

- Block comments should usually be avoided. When you use block comments, then use this style:
```
/* Line 1
Line 2
Line 3 */
```

- If it is less comment, then also this style is possible:
```
// Line 1
// Line 2
```

7.3 Single Line Comments

- Use ‚//' to comment single lines. Align the comment with the code it describes.

- Don't use **/* This is am comment */** as comment style!

- Don't use **#ifdef 0** as comment style!

- No comments at the end of a line!

8. Declarations

8.1 Local Variables

- Local Variables should not be declared at the beginning of a function or method. Try to declare them right before using them! So you avoid the problem of non referenced variables!

- Try to initialize variables together with the declaration (or within the constructor).

  Example:
  ```
  private string name = myObject.Name;
  private int val = time.Hours;
  ```

8.2 Global Variables

- Every definition or declaration of a global variable should stand in an own row and must have a comment!

8.3 Class- and Interface Declarations

- Variables should be initialized in the constructor.

- No space between a method name and the parenthesis ‘(‘ starting its parameter list.

- The parameter names in declaration and definition must be identical.

- The opening and closing brackets ‘{‘ and ‘}’ stand always in a new line.

  Example:
  ```
  public class MySample : MyClass, IMyInterface
  {
     private int _myInt;

     public MySample(int myInt)
     {
           _myInt = myInt;
     }

     private void Inc()
     {
           _myInt++;
     }
  ```

```
        private void EmptyMethod()
        {
        }
    }
```

9. Statements

9.1 Simple Statements

- Each line should contain only one statement!
- The line length should not exceed 100 – 120 characters.
- Keywords in SQL statements have to be capitalized!

9.2 if-else-statements

- If / else / else if statements must have the following form (use always curly brackets!):

```
if (condition)
{
   DoSomething();
   ...
}
else if (condition)
{
   DoSomethingOther();
   ...
}
else
{
   DoSomethingOtherAgain();
   ...
}
```

9.3 for- / foreach- / while- / do-while statement

- They must always have parenthesis (even if they have only one conditional statement).
- ... also for the ';' statement

Examples:

```
do
{
   ...
} while (condition);

for (int i = 0; i < 5; ++i)
{
   ;
}
```

```
while (condition)
{
   DoSomething();
   …
}
```

9.4 switch-statement

- Every switch construct must have a default section. This is always the last one.
- Every case section must end with ‘break’ or a ‘// no break’ comment.

  Example:

```
switch (condition)
{
   case A:
        // no break
   case B:
        ...
        break;
   default:
        ...
        break;
}
```

9.5 goto-statement

The goto-statement has not to be used (in release code)!

9.6 try- / catch-mechanism

The try/ catch- mechanisms should be used as follows:

```
try
{
      ...
}
catch (Exception)
{
}
```

or

```
try
{
      ...
}
catch (Exception e)
{
      ...
}
```

or

```
try
{
     ...
}
catch (Exception e)
{
     ...
}
finally
{
     ...
}
```

10. White Spaces

- After a comma and a semicolon must be a white space ...
- ... but not after an opening and before a closing parenthesis.

  Example 1:

  Prefer:
  ```
  TestMethod(a, b, c);
  ```

  Bad Style:
  ```
  TestMethod(a,b,c);
  TestMethod( a, b, c );
  ```

- ... between operators (except the operator ‘*’, ‘!’ and increment respectively decrement operators)

  Example 2:

  Prefer:
  ```
  a = b;
  for (int i = 0; i < 10; ++i) ...
  ```

  Bad Style:
  ```
  a=b;
  for(int i=0; i<10; ++i) ...
  ```

11. Naming Conventions

11.1 Capitalization Styles

‘Pascal Casing’:

- This convention capitalizes the first character of each word.

  Example:
  `TestCounter`

‘Camel Casing’:

- This convention capitalizes the first character of each word except the first one.

  Example:
  `testCounter`

Capitalization:

- A variable with less than 3 characters can be written with capital letters.

  Example:

```
Public Class Math
{
   public const PI = …;
   public const E = …;
   public const senderNummer = …;
}
```

11.2 Naming Guidelines (arranged alphabetically)

Classes:

- Use ‘Pascal Casing’!
- … without prefix ‚C’!
- The starting letter ‚I’ can only be used for interfaces (except it belongs to the name of the class, e. g. `Item`).

Collections:

- Self-defined classes for Collections have the word ‘Collection’ at the end (e. g. `PersonCollection`).

DataSet classes:

- Self-defined DataSet classes (e. g. typed DataSets) have the word ‘DataSet’ at the end (e. g. `CoreManagementDataSet`).

Enumerations:
- Use for and within enumerations ‚Pascal Casing'!
- … without prefix.

Events:
- Use ‚Pascal Casing'!
- Every event handler has the suffix 'EventHandler'.
- Use the parameters ‚sender' und 'e'. The parameter has the suffix 'EventArgs'

Exception classes:
- Self-defined exception classes have the word 'Exception' at the end (e. g. `AddNewHouseholdException`).

GUI:
- All GUI elements like buttons, list boxes, etc. have their type at the end of their name (no abbreviation).

  Example:
  ```
  System.Windows.Forms.Button      cancelButton;
  System.Windows.Forms.TextBox     nameTextBox;
  ```

- Abbrevations are allowed, if the name of the GUI element is too long!

Interfaces:
- Use ‚Pascal Casing'!
- Interfaces have a prefix 'I', followed by a capital letter (e. g. `IComponent`, `IEnumerable`).

Methods:
- Use ‚Pascal Casing'!
- Use verbs and adjectives (e. g. `CalculateInvoiceTotal()`)!
- Use descriptive method names (e. g. `GetNextStudent()` is better than `GetNextArrayElement()`)!

Properties:
- Use ‚Pascal Casing'!

Variables:
- Use 'Camel Casing'!
- There is no tag for the type (e.g. n for Integer, c for Character, …).
- Every private variable which is valid over the validity time of a class has to begin with an underscore (e. g. `_counter`).
- Arrays have the suffix 'Array' (e. g. `int[] customerArray = {1, 2, 3};`)!

Web Service classes:

- Self-defined classes for Web Services have the word ‘Service’ at the end (e. g. `CoreManagementService`).

12 Source Analysis Rules

Furthermore the following source analysis rules are defined and shall be applied in your code!

13.1 Documentation Rules

- All elements must be documented and must not be empty.
  This rule contains all elements, sub-elements (enumeration items), parameters and return values
- Single line comments must not use documentation style slashes.
- Element documentation headers must not be followed by a blank line.

13.2 Layout Rules

- Elements and statements must not be on a single line.
- Curly brackets must not be omitted.
- Curly brackets for multiline statements must not share a line.

13.3 Maintainability Rules

- Access modifiers must be declared.

13.4 Naming Rules

- Public and internal elements must begin with an upper-case letter.
- Interface names must begin with an upper-case I.
- Private fields must begin with an underscore followed by a lower-case letter.
- Constant names must use upper-case letters delimited by underscores if useful.
- The following elements must always begin with an upper-case letter:
  Namespaces, Enumerations, Structures, Classes, Properties, Methods, Events, Delegates

13.5 Readability Rules

- The C# build-in type aliases must always be used. E.g. ‘int’ instead of ‘System.Int32’
- Opening parenthesis must be on declaration line.
- Closing parenthesis must be on same line as the last parameter.
- Commas must be on same line as the previous parameter.
- Statements must not use unnecessary parenthesis. E.g. return (this);
- Code must not contain empty statements.
- Code must not contain multiple statements on one line.

13.6 Spacing Rules

- Elements must be spaced correctly. (one whitespace only)
- Commas and semicolons must be placed correctly. Eg. “, ”
- Always use tabs for indentation.