Programming Guidelines

2019-07-24

## Purpose of coding standards and best practices

To develop reliable and maintainable applications, you must follow coding standards and best practices.

The naming conventions, coding standards and best practices described in this document are compiled from our own experience and by referring to various Microsoft and non-Microsoft guidelines.

## General Rules

A method should do only one job. Do not combine more than one job in a single method, even if those jobs are very small.

Avoid writing very long methods. A method should typically have 1-25 lines of code. If a method has more than 25 lines of code, you may wish to consider refactoring it into separate methods.

File names should match the class names. For example, for the class HelloWorld, the file name should be helloworld.cs or, helloworld.vb.

There should be one and only one single blank line between each method inside the class.

Indentation should be four spaces.

Use #region to group related pieces of code together.

Look for and remove dead code from time to time, such as unnecessary “using” statements, methods that are no longer called, and unneeded parameters.

## Names of variables and other identifiers

In most modern programming languages there are two ways to group the characters that make up a variable name. One is to begin with a capital letter, and the other is to use an underscore. This allows us to use two levels of separation.

We use capital letters to separate words and underscores to separate important sections of the name.

PathedFileName\_Original

PathedFileName\_Old

PathedFileName\_New

PathedFileName\_Backup

Use the prefix I for interfaces.

IEntity

One-letter variable names should only be used for iterations through short loops.

for (int i = 0; i > wb.Worksheets.Count; i++)

{

wb.Worksheets[i].Cells.ApplyStyle(allRowStyle, allRowStyleFlag);

}

Do not use variable names that resemble keywords. For example, “DESC” is a reserved word in SQL, so a description should be named Descr rather than Desc.

Do not use two consecutive underscores, because these are reserved in .net for compiler-generated identifiers.

In the past, standards such as Hungarian notation were used to provide information about a variable, such as its type and scope. In Visual Studio, you can now just hover the mouse pointer over a variable name to get this information. Still, there are cases where information about the kind of variable helps. For example, when SQL code is stored in variables, the variable names should be prefixed with “sql\_”. UI elements on the aspx pages should be prefixed by the type of control, such as the following:

btn\_Start

textbox\_Name

grid\_People

pic\_Photo

## Abbreviations

Use meaningful, descriptive words to name variables. Most English words should be spelled out rather than abbreviated. The drawbacks of having longer variable names are greatly outweighed by having more consistent, readable code. When words are abbreviated, it should usually be obvious what the abbreviation stands for. The Abbr.ods spreadsheet contains a list of the standard abbreviations we use.

Using standardized abbreviations has some great benefits, but few programmers bother with it.

Say, for example, you needed to search the code for anything related to the fiscal year. Without standardized abbreviations, you would need to search for “fiscal year”, “FiscalYear”, “FiscalYr”, “FiscYr”, “FY”, and possibly others, and you still might not catch all the occurrences you need.

These standardized abbreviations apply to variables and function names in the program code, and to database table and field names. They generally do NOT apply to text on the user interface.

## Comments

In places where the functionality of the code is not obvious, comments should be used to describe what the code is doing. A comment which describes what the line after it is doing should be a sentence ending with a period. A comment describing what a block of code is doing should end with a colon.

'// If we have 3 monitors, use the one on the left:

Dim NumberOfMonitors As Integer = Screen.AllScreens.Length

If NumberOfMonitors > 2 Then

'// Set the screen boundaries.

Me.Bounds = Screen.AllScreens(1).Bounds

End If

'// Maximize the application.

Me.WindowState = FormWindowState.Maximized

Alternatively, you may place a short comment at the end of the line.

e.Node.Nodes.Clear() '// Clear all items.

AddAllSubFolders(e.Node, CStr(e.Node.Tag)) '// Add all folders.

Symbols used in comments:

† or ‡ exception

↑ jump out

Comments not directly related to the program flow should be placed in parentheses or marked as a note. These comments typically describe why you are doing something.

b\_Success = UpdateRecord.Update\_BPlAllotment(

Convert.ToInt32(ID),

0,

"",

Convert.ToDecimal(Salary),

Convert.ToDecimal(Travel),

Convert.ToDecimal(Contracts));

// Note: We do not pass the Budget\_ID or BPL\_ID to the function, because these values cannot be changed.

// Calculate the percentages:

int Percentage1 = (int)Convert.ToDecimal(row["Units1"].ToString());

int Percentage2 = (int)Convert.ToDecimal(row["Units2"].ToString());

int Percentage3 = (int)Convert.ToDecimal(row["Units3"].ToString());

// ( The "Units" data fields contain the percentages. )

Avoid commenting out code. Large amounts of commented-out code clutter things up and reduce the code’s readability. If you *do* comment out lines of code, be sure to mark the reason for commenting it out. Examples could include the following:

// no longer used //

// probably not needed //

// DEBUG //

// TEST //

// possible alternate solution //

// could not get this to work //

If there is no reason to save the commented-out code, delete it.

## Conditional statements

### Form 1

if (oRadioButton\_Empty.Checked == true && chkbox\_SetAsCurrentBudget.Checked == false)

// If the user selected "Create an empty budget", but not "Set as current budget" ...

{

FiscalYear\_NewBudget = FiscalYear\_CurrentBudget + 1;

}

else

// If the user selected "Copy an existing budget" OR "Set as current budget" OR both ...

{

FiscalYear\_NewBudget = FiscalYear\_CurrentBudget;

}

This is generally the preferred format in C#, since it looks clean and the flow easy to understand.

### Form 2

if (cols != null) {

foreach (string cl in cols) {

if (!dictionary.ContainsKey(cl)) {

copyFTable = AddStrColumn(copyFTable, cl, "");

} else {

copyFTable = AddStrColumn(copyFTable, “2”, "");

}

}

}

Avoid using this form, with the inline braces, in C#. In some situations, it may be used in JavaScript.

### Form 3

if ((bool)rbAutoA.IsChecked)

TblName = Constants.AUTO\_A\_FS;

else if ((bool)rbCarwash.IsChecked)

TblName = Constants.CARWASH\_FS;

else if ((bool)rbLongReach.IsChecked)

TblName = Constants.LONGREACH\_FS;

else if ((bool)rbGenFac.IsChecked)

TblName = Constants.GENFACILITY\_FS;

This abbreviated form can only be used when there is just one statement per condition. We generally avoid using this form in C#, since it is prone to errors. A case where you have a long list of “else if” clauses with only one statement each might be an exception, since it shortens the code considerably.

### Form 4

if (Insps.StartsWith("/")) Insps = Insps.Substring(1);

if (Insps.Length == 0) return;

When you have a short condition followed by a short statement, this is a good format to use. It is less prone to errors than the previous one.

### Form 5

string Insps = (selectedInspector1Id > -1 ? "TopList" : "") + "/";

where += (where.Length > 0 ? " or " : "") + Constants.SCHID\_FK\_COL;

This very short format can be difficult to understand if not used carefully.

### Form 6

ViewState["SearchString\_BPL"] = Request.QueryString["SearchString\_BPL"] ?? "";

The null-coalescing operator is a quick way to check for null values.

## Long Strings and SQL Inside Code

When SQL statements are included in the code, they should be divided into logical lines. Web sites such as <http://www.dpriver.com/pp/sqlformat.htm> or <https://www.freeformatter.com/sql-formatter.html> make it easy to convert long SQL statements into nicely formatted code.

Most of the code which accesses the database using SQL belongs in the DAL folder.

There are several ways to include long and multi-line strings, such as SQL statements, inside the code.

### Style 1

string sql = "SELECT COLS.ORDINAL\_POSITION, COLS.COLUMN\_DEFAULT, TT.User\_Label,\n";

sql += "FROM INFORMATION\_SCHEMA.COLUMNS COLS\n";

sql += " INNER JOIN UFS\_Tool\_Tip TT\n";

sql += " ON TT.Column\_Name = COLS.COLUMN\_NAME\n";

sql += "WHERE COLS.TABLE\_NAME = '" + TableName + "'\n";

sql += " AND TT.Table\_Name = '" + TableName + "'\n";

This style uses a separate statement for each line or segment of the string. It is not recommended since it is slower and harder to read.

### Style 2

string sql = "SELECT COLS.ORDINAL\_POSITION, COLS.COLUMN\_DEFAULT, TT.User\_Label,\n"

+ "FROM INFORMATION\_SCHEMA.COLUMNS COLS\n"

+ " INNER JOIN UFS\_Tool\_Tip TT\n"

+ " ON TT.Column\_Name = COLS.COLUMN\_NAME\n"

+ "WHERE COLS.TABLE\_NAME = '" + TableName + "'\n"

+ " AND TT.Table\_Name = '" + TableName + "'\n";

In this style, we use separate strings for each line concatenated together.

### Style 3

string sql = @"UPDATE [dbo].[WBS]

SET [FY] = @FY

,[Division\_ID] = @Division\_ID

,[WBS\_Name] = @WBS\_Name

,[BPL\_ID] = @BPL\_ID

,[Functional\_Area\_ID] = @Functional\_Area\_ID

,[Time\_Stamp] = getdate()

,[Modified\_By] = @Modified\_By ";

This style may be used for multi-line strings. It uses the “@” symbol before the string to tell the compiler to interpret the string literally. Therefore, it will not be necessary to include the newline character “\n” at the end of each line.

### Style 4

StringBuilder sql = new StringBuilder();

sql.Append("SELECT Baseline.Baseline\_ID, Functional\_Area.Functional\_Area\_Name, Amount\n"

+ "FROM dbo.Baseline\n"

+ "WHERE 1=1\n");

if (str\_SelectedFYs.Length > 0)

{

sql.Append(" AND Baseline.FY IN (" + str\_SelectedFYs + ")\n");

}

if (str\_SelectedDivisionIDs.Length > 0)

{

sql.Append(" AND Baseline.Division\_ID IN (" + str\_SelectedDivisionIDs + ")\n");

}

sql.Append("ORDER BY Functional\_Area\_Name, UFS\_Divisions.Division\_Name ASC;\n");

In some cases, the StringBuilder class offers better performance than the String class. The Microsoft online documentation has information about when it makes sense to use StringBuilder.

## ASP.NET

Avoid passing data in the URL. Use session variables instead.

## Terminology

Addr

(in USA) street address, city, state, and zip code, generally separated by CRLFs

DateTime

a date and a time of day, such as 2019-03-20 14:22

DayOfMonth

a number from 1 to 31 representing the day of the month.

DayOfWeek

This is the day of the week, which may be expressed in several forms

* Spelled out, such as “Sunday”, “Monday”, “Tuesday”, etc.
* Abbreviated, such as "Sun", "Mon", "Tue", etc.
* A number from 1 to 7

Duration

length of time, such as 24 minutes

StreetAddr

street address WITHOUT the city, state, and zip code

Time

time of day (not a length of time), such as 15:21

Xable

enable or disable, as appropriate

PhoneNum

(in USA) telephone number, generally in the format ###-###-####

EmailAddr

email address