LHB Project
Coding policy

Version 0.3

Revision History

Date	Version	Description	Author
11.11.2012	0.1	Initial draft	Robert Pofuk
11.11.2012	0.2	revised some parts	Niklas Gillström
11.11.2012	0.3	Added descriptions	Gonçalo Silva

Table of Contents

1. Introduction

This document is meant to define the coding conventions to be followed by the Let's Help Bo project. To minimize code management problems between the teams involved, this project will follow C# coding conventions and the official document **.NET Framework 4.0 Design Guidelines**.

This last document refers to almost all naming conventions, casing rules, etc. However, unlike the Design Guidelines document, you should treat this as a set of suggested guidelines. These generally do not affect the customer view so they are not required.

For additional information regarding the standards of the C# language you can refer to these two documents:

C# Coding Conventions (C# programming guide) http://msdn.microsoft.com/en-us/library/vstudio/ff926074(v=vs.100).aspx

.NET Framework 4.0 Design Guidelines http://msdn.microsoft.com/en-us/library/ms184412(v=vs.100).aspx

2. Coding

2.1 Tabs & Indenting

Use the default Code Editor settings from Visual Studio. This implies four-character indentations with tabs saved as spaces.

2.2 Bracing

Braces should be in the same line or the line after the statement that begins the block. Contents of the brace should be indented by tab. For example:

```
if (someExpression) {
	DoSomething();
}
else {
	DoSomethingElse();
}
```

or

```
if (someExpression)
{
	DoSomething();
}
else
{
	DoSomethingElse();
}
```

“case” statements should be indented from the switch statement like this:

```
switch (someExpression) {
	case 0:
		DoSomething();
		break;
	case 1:
		DoSomethingElse();
		break;
	case 2: {
		int n = 1;
```

```
            DoAnotherThing(n);
            }
        break;
}
```

Braces should never be considered optional. Even for single statement blocks, you should always use braces. This increases code readability and maintainability.

```
for (int i=0; i<100; i++) { DoSomething(i); }
```

In Visual Studio, formatting can be defined on:
Tools -> Options -> Text Editors -> C# ->Formating -> New Lines

2.3 Single line statements

Single line (short) statements can have braces that begin and end on the same line.

```
public class Foo {
    int bar;
    public int Bar {
        get { return bar; }
        set { bar = value; }
    }
}
```

All control structures (if, while, for, etc.) are required to use braces.

2.4 Commenting

Comments should be used to describe intention, algorithmic overview, and/or logical flow. Do not use comments on lines that are self-explanatory. Here's an example of what you should not do:

```
ArrayList list = new ArrayList();
…
// sort the list
list.sort();
```

The purpose of the comments is to make code behaviour understandable for someone other than the developer by just reading the comments alone.

All methods should have XML comments (no exceptions to this) to allow documentation processing. Comments are not needed only on methods that implement interface, and are following requirements specified in interface method documentation.

2.4.1 Comment Style

In order to maintain compatibility for instance generate documentation for the project XML comments should be used. XML comment skeleton can be generated automatically by Visual Studio by doing a triple slashes (///) on the line above a method. Each part of the XML skeleton should be filled correctly. By doing that all other commenting is deprecated when the method is finished. But during development of each method the developer can use comments in code to remember where the programming were left off the last time. In that case the following guidelines should be followed:

The // (two slashes) style of comment tags should be used in most situations. Wherever possible, place comments above the code instead of beside it. Here are some examples:

```
// This is required for WebClient to work through the proxy
GlobalProxySelection.Select = new WebProxy("http://itgproxy");

// Create object to access Internet resources
//
WebClient myClient = new WebClient();
```

Comments can be placed at the end of a line when space allows:

```
public class SomethingUseful {
   private int itemHash;           // instance member
   private static bool hasDoneSomething;    // static member
}
```

2.5 Spacing

Spaces improve readability by decreasing code density. Here are some guidelines for the use of space characters within code:

- Do use a single space after a comma between function arguments.
  - Right: Console.In.Read(myChar, 0, 1);
  - Wrong: Console.In.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)

2.6 Naming

Choose easily readable identifier names, and prefer readability over brevity. A property named *CanScrollHorizontally* is far more understandable than *ScrollableX*, and consequently makes the code more understandable.

Follow all .NET Framework Design Guidelines for both internal and external members. Highlights of these include:

- Do not use Hungarian notation. This is the practice of including information about the parameters in the function name, and is unnecessary in C#.
- Do not use a prefix for member variables (_, m_, s_, etc.). If you want to distinguish between local and member variables you should use “this.” keyword.
- Use camelCasing for member variables, parameters and local variables.
- Use PascalCasing for function names, properties, events, and class names.
- Prefix interfaces names with “I”.
- Do not prefix enums, classes, or delegates with any letter.

2.8 File Organization

Never declare more than one namespace inside each file. Always leave one line empty after the namespace declaration.

Avoid putting multiple classes in a single file.

The name of the public class declared in a single file should match its filename.

Using statements should be placed on the top of the file, before the namespace declaration.

Class members should be alphabetized and grouped into the following sections:

- Member variables/fields
- Constructors and finalizers
- Nested types (structs, classes, enums)
- Properties
- Methods
- Events
- Private interface implementations

To facilitate this you can use the #region directive. Put one empty line before #region statement and one empty line before #endregion statement, and two empty lines between #endregion and #region statements.

```
using System;

namespace MyNamespace  {

        public class MyClass : IFoo  {

            #region fields

                    int foo;

            #endregion


            #region constructors

                public MyClass() { … }

                #endregion;
```

```
        #region properties

            public int Foo { get { … } set { … } }

        #endregion


        #region events

            public event EventHandler FooChanged { add {} remove {} }

            #endregion

            #region methods

                void DoSomething() { … }
                void FindSomethind() { … }

            #endregion


            #region private interface implementations

                void IFoo.DoSomething() { DoSomething(); }

            #endregion


            #region nested types

                class NestedType { … }

            #endregion

    }
}
```

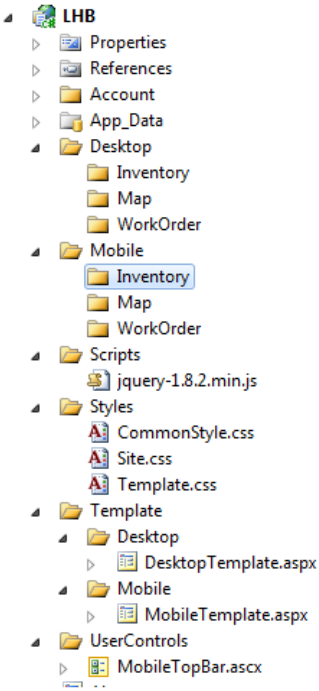
2.9 Properties

When defining fields either initialize them on the definition or initiate them as null. **Do not leave them uninitialized.**

```
#region fields

    SomeClass foo = null;
    OtherClass bar = new OtherClass();

    // Don't do this one
    SomeClass foo

#endregion
```

3 Presentation Layer Code policies

3.1 Presentation layer project folders



In the folder **Desktop** there will be all the web pages concerning the desktop version of the web site.

In the folder **Mobile** there will be all the web pages concerning the mobile version of the web site.

In each of these folders there are three sub-folders:

- Inventory
- Map
- WorkOrder

Each of them contains the web pages of the relative component

In the folder **Scripts** there are all the javascript files that will be used in the web pages.

In the folder **Picture** there are all the images files that will be used in the web pages. There is a folder for the pictures used for the desktop version and one for the pictures use for the mobile version of the web pages. Inside each of this folder there are other two folders, Icons and Images. As the name suggest the first one will contain the icons and the second ones general images.

In the folder **Styles** there are all the css files that will be used in the web pages. In this folder there is the file CommonStyle.css that will be included in all the pages of the web site, and it will contain all the rules for the colors, font etc. to use.

In the folder **Template** there are the template pages for the mobile and desktop vesions. If a new page for desktop or for mobile ,these templatet **must** be considered, copying the code that is in these pages in the new one created.

In the folder **UserControls** there will be all the user controls used in the web pages.

3.2 File names

The web page names have to be named as the component as they are referred and adding the operation that they contain. For example if in the page there is the functionality to create a work order the page have to be named ad WorkOrderCreate.aspx. If in the page there are the adding, removing and search functionality the page can be named [PageName]Manger.aspx (e.g.

WorkOrderManager.aspx).

The css files have to be named as the name of the page. For example if the name of a page is CreateWorkOrder.aspx the relative css file name have to be CreateWorkOrder.css

The javascript files have to be named as the name of the page. For example if the name of a page is CreateWorkOrder.aspx the relative css file name have to be CreateWorkOrder.js

3.2 Templates

In the desktop web page template there is an header that will be present in all the pages of the website. The same is for the mobile web page template.
In the mobile web page template a 300 pixels width div will be used as container for all the elements of the page to simulate the size of the smartphone screen.

Concerning the mobile web pages, the width of almost all the elements must be expressed in percentage and not in pixels. In this way the size of the elements will adapt to the size of the screen.

3.3 Conclusion

This standard can be updated during the project.
This standard could be not perfect but it is a standard and can help us during the development. So let's use it and it will make easier our lives.