**

Bazinga Coding Standard

Version 1.0

February 9, 2012

PAAC Demonstration System



1. **Introduction**
   1. **Overview**

This document outlines general coding standards for Bazinga software.

* 1. **Overview**

The coding standards outlined in this document are general practices that should be followed when writing software code for Bazinga software. This document does not cover all possible scenarios that may be encountered when writing software. In cases where a standard is not outlined in this document, the software author is responsible for using best judgment and using the same style consistently.

1. **Coding Standards**
   1. **Naming Conventions**
      1. **Variable Names**
         1. The first letter of local variables should be lower case
         2. Variables with multiple words should use “Camel Case”, i.e. the first letter of each word is capitalized
         3. Module-level private variables should be preceded with a lower case “m\_” to designate it as a module variable.
         4. The names “i”, “j”, and “k” should be reserved for iteration variables, e.g. in a For loop.

**Examples:**

int variable

private int m\_myVariable;

* + 1. **Class Names**
       1. The first letter of class names should be capitalized
       2. Classes with multiple words should use “Camel Case”, i.e. the first letter of each word is capitalized

**Examples:**

public class MyClass

* + 1. **Method Names**
       1. The first letter of private and public names should be lower case
       2. Methods with multiple words should use “Camel Case”, i.e. the first letter of each word is capitalized

**Examples:**

public int getValue( )

* + 1. **Properties**
       1. The first letter of public properties should be capitalized
       2. Properties with multiple words should use “Camel Case”, i.e. the first letter of each word is capitalized

**Examples:**

public int Value { get; set;}

* + 1. **Constants**
       1. Public and module-level constants should be in all capital letters
       2. Constants with multiple words should use an underscore to separate words

**Examples:**

public const int CONSTANT\_VARIABLE

* + 1. **Interfaces**
       1. Interfaces should follow the naming convention of classes, but should begin with a capital I

**Examples:**

public interface IMyInterface

* 1. **Commenting**
     1. **Block Comments**
        1. Block comments should begin each line with 3 slashes
        2. HTML-style tags (auto-generated by Visual Studio) should be used where applicable in block comments
        3. Each file should have a block comment at the top of the file stating the file name, author, and company name (Bazinga)
        4. Each method should have a block comment before it that outlines the purpose of the method, the parameters it requires, and the returned object.
        5. Each property and enum type should have a block comment before it that outlines the purpose of the property or enum

**Examples:**

/// <summary>

/// This class is used for doing work.

/// </summary>

* + 1. **Line Comments**
       1. Line comments should be added wherever a the function of the code is not immediately obvious to the reader
       2. Variables only need to be commented when the name and/or purpose is not immediately obvious to the reader
       3. Line comments should be added immediately above the line of code it applies to
       4. Short line comments may be added to the right side of the line of code it applies to, provided that the comment does not make the line exceed 80 characters

**Examples:**

//Calculate the value of x

x = y + z;

if (x > 0) //Check if x is valid

* 1. **White Space**
     1. **Spaces**
        1. A space should be added between the slashes of a comment and the comment text
        2. Spaces should be added between arithmetic/logical operators and the operands

**Examples:**

// Comment

if (x == y)

x = (y + z) / 2

* + 1. **Tabs and Indentation**
       1. Tabs should be used instead of spaces to indent lines
       2. Each statement inside a block of code should be indented one tab from the opening brace

**Examples:**

{

x++;

}

* 1. **Miscellaneous Coding Practices**
     1. **Braces**
        1. The opening and closing brace of a statement block should be on a line by itself
        2. Braces should always be present, even if the functional block is only one line

**Examples:**

if (x == y)

{

x++;

}

* + 1. **Statements**
       1. There should be only one code statement per line, with the exception of the heading of a for loop
       2. Only one variable should be declared per line

**Examples:**

for (int i = 0; i < count; i++)

int x;

int y;

* + 1. **Modifiers**
       1. Classes, methods, interfaces, properties, and module variables should explicitly be declared as public or private in all cases
       2. Classes, methods, and module variables should be private or protected whenever possible and practical. Only classes, methods, and module variables that can be practically used by external classes shall be declared public.
    2. **Exception Handling**
       1. Operations that have a significant likelihood of generating exceptions should be surrounded by a try/catch block. The catch block should, at a minimum, log the exception.
    3. **General**
       1. Properties should be used in place of public variables whenever practical
       2. Switch-case statements should always define a default block, even if the code is unreachable or the default case does nothing
       3. The default block of a switch statement should contain a break statement
       4. “== true” and “== false” should not be used as they are unnecessary for the evaluation of Boolean statements

**Examples:**

switch (x)

{

case 1:

…

case 2:

…

default:

break; //Do nothing

}

if (flag)

if (!flag)