**GreenShift Coding Conventions**

In this document all coding conventions for C# will be documented. This is useful for the following reasons:

* They create a consistent look to the code, so that readers can focus on content, not layout.
* They enable readers to understand the code more quickly by making assumptions based on previous experience.
* They facilitate copying, changing, and maintaining the code.
* They demonstrate best practices.

**C# Coding Conventions**

**Naming Conventions**

There are several naming conventions to consider when writing C# code.

In the following examples, any of the guidance pertaining to elements marked (public) is also applicable when working with (protected) and (protected internal) elements, all of which are intended to be visible to external callers.

**Pascal Case**

Use pascal casing ("PascalCasing") when naming a (class), (record), or (struct).

A picture containing graphical user interface

Description automatically generated

Text

Description automatically generated with low confidence

Graphical user interface

Description automatically generated with medium confidence

When naming an interface, use pascal casing in addition to prefixing the name with an I. This clearly indicates to consumers that it's an interface.

Graphical user interface

Description automatically generated

When naming public members of types, such as fields, properties, events, methods, and local functions, use pascal casing.

Text

Description automatically generated

When writing positional records, use pascal casing for parameters as they're the public properties of the record.

Graphical user interface

Description automatically generated with medium confidence

**Camel Case**

Use camel casing ("camelCasing") when naming private or internal fields, and prefix them with \_ (underscore).

Graphical user interface, text

Description automatically generated

When working with static fields that are private or internal, use the s\_ prefix and for thread static use t\_.

Text

Description automatically generated

When writing method parameters, use camel casing. A screen shot of a computer

Description automatically generated with medium confidence

**Layout conventions**

Good layout uses formatting to emphasize the structure of your code and to make the code easier to read. Microsoft examples and samples conform to the following conventions:

* Use the default Code Editor settings (smart indenting, four-character indents, tabs saved as spaces).
* Write only one statement per line.
* Write only one declaration per line.
* If continuation lines are not indented automatically, indent them one tab stop (four spaces).
* Add at least one blank line between method definitions and property definitions.
* Use parentheses to make clauses in an expression apparent, as shown in the following code.

A screenshot of a computer

Description automatically generated with medium confidence

**Place the using directives outside the namespace declaration**

When a using directive is outside a namespace declaration, that imported namespace is its fully qualified name. That's more clear. When the using directive is inside the namespace, it could be either relative to that namespace or it's fully qualified name. That's ambiguous.

A screenshot of a computer program

Description automatically generated with medium confidence

**Commenting conventions**

* Place the comment on a separate line, not at the end of a line of code.
* Begin comment text with an uppercase letter.
* End comment text with a period.
* Insert one space between the comment delimiter (//) and the comment text, as shown in the following example.

A screenshot of a computer

Description automatically generated with medium confidence

* Don't create formatted blocks of asterisks around comments.
* Ensure all public members have the necessary XML comments providing appropriate descriptions about their behavior.

**Language guidelines**

The following sections describe practices that the C# team follows to prepare code examples and samples.

**Implicitly typed local variables**

Use implicit typing for local variables when the type of the variable is obvious from the right side of the assignment, or when the precise type is not important.

A picture containing text, font, multimedia software, software

Description automatically generated

Don't use var when the type is not apparent from the right side of the assignment. Don't assume the type is clear from a method name. A variable type is considered clear if it's a new operator or an explicit cast.

A screen shot of a computer

Description automatically generated with low confidence

Don't rely on the variable name to specify the type of the variable. It might not be correct. In the following example, the variable name inputInt is misleading. It's a string.

A picture containing text, font, multimedia software, software

Description automatically generated