Ceng492 Graduation Project

The Bride Project

The Code Standards

Presented by Meda

Ankara, 2005

1. General

The purpose of this document is to define the standards for coding to be used to implement the software for the Bride project. In this document, the term “shall” indicates a mandatory practice, whereas terms such as “should” indicate a recommended guideline or constraint.

2. Conventions

The naming of files and code entities (variables, classes, functions, methods, etc.) should be as descriptive as possible without being cumbersome. For example, a variable named X, although syntactically valid, is not descriptive enough to give a programmer any information about what it contains. A variable name such as iAircraftLatitude is much more descriptive.

2.1. Header Comment Blocks

All files shall have a header comment block that contains the following information:

- Name and address of company
- Name of the Project
- File name
- Revision number of file
- Date of revision
- Author of revision

Examples:

Comment Block Example

```
/*
 *==========================================================================
 *
 * MEDA Software Development
 * METU – Ceng490 Senior Project
 * The Bride
 *
 *==========================================================================
 *
 * $File$
 *
 * $Revision$
 *
 * $Date$
 *
 * $Author$
 *
 *==========================================================================
 */
```

2.2. Coding Style and Format

The code should be written so that it is readable and program flow can be easily identified. The code shall be indented to facilitate the ease of understanding the code.

Code Block
For the nested statements, the left brace "}" that signals the start of a code block should be on the next line with the same indention as the statement that indicate the start of the code block. The right brace "}" that signals the end of a code block should be on a separate line after the last execution statement with the same indention as the statement that indicate the start of the code block. If only 1 line of code exists in the block, the use of braces is optional.

Examples:

```
if ( condition )
{
  statement1;
  statement2;
}
else if
{
  statement3;
  statement4;
}
```

```
while ( condition )
{
  statement;
  statement;
}
```

```
for ( ; condition ; )
{
    statement;
}
```

2.3. Variable Initialization

All variables shall be explicitly initialized before use. Variables are not used for more than one purpose.

Examples:

Unacceptable:	`char szString[30];`
	`      strcpy(zsString, “Bad”);`
	`//Not filling all 30 bytes/characters`

Acceptable:	`char szString[30] = “\0”;`

Acceptable:	`char szString[30];`
	`x2 = f2( b );`

	`zeroMem( szString, 30 );`
Acceptable:	`const char szString[] = “Some text”;`

2.4. Protection against Multiple #includes

Each non-system #include file shall #define a unique token that identifies that header file. The header file shall test for the existence of this token and #define it if it does not exist, and then include the rest of the file.

Example:

```
#ifndef FILENAME_H
#define FILENAME_H
        [contents of file go here …]
#endif // FILENAME_H
```

2.4.1. #include Conventions for Header Files

- Compiler-supplied #include files shall always be specified with angle brackets.

  ```
  #include <string.h>
  ```

- There shall not be path names in #include statements, with the exception of the traditional system defined #includes like:

  ```
  #include <sys\stat.h>
  ```

- User-defined headers shall be included after system headers included.

2.4.2. #include Conventions for Source Files

- Compiler-supplied #include files shall always be specified with angle brackets:

  ```
  #include <stdlib.h>
  ```

- All other #include files shall be specified with quotes:

  ```
  #include "myStuff.h"
  ```

- There shall not be path names in #include statements, with the exception of the traditional system defined #includes:

  ```
  #include <sys\stat.h>
  ```

- User-defined headers shall be included after system headers included.
- Header files shall not be dependent on order of inclusion in source file.

2.5. Function/Method Comment Block in Source Files

Functions/methods shall have a comment block as follows with the following information:

- Name
- Description
- Example of this is:

Example:
```
/*
 * getType( void )
 *
 *  Gets object type
 *
 */
[Function/Method definition goes here ...]
```

2.6. Data and Control Coupling

Function/Method parameters and variables that represent quantities (e.g. feet) should have a comment specifying the units of measure. This will help in data and control coupling analysis activities.

2.7. Naming Conventions

This table contains the preliminary naming conventions.

Type	**Prefix**
Global Variables	g_
Member Variables	m_
Pointers – Double Pointers – Triple Pointers etc.	p – pp - ppp
Arrays – Double Arrays – Triple arrays etc.	a – aa - aaa
Character String (char *)	sz
GLenum, enum	e
Glboolean, bool	b
Glbitfield	bf
Glbyte	b
Glshort, short	h
Glint, int	i
Glsizei	s
Glubyte	ub
Glushort, unsigned short	uh
Gluint, unsigned int	ui
Glfloat, float	f
Glclampf	cf
Gldouble, double	d
Glclampd	cd
Glvoid, void	v

Note: GL types should be used only when necessary. Use standard primitive types where applicable.

Examples:

```
Global GLbitfield Double Array:
GLbitfield g_aabfTheArray[X][X];

Member GLclampf Pointer:
GLclampf ***m_pcfThePointer;

Global char array:
char acTheArray[3] = {‘A’, ‘X’, ‘P’};

Global Character String:
char szTheString[6] = “Trial”;
```