Gameloft Coding Guideline

Version 1.0.7

Last update : 2007.10.03

1 General :

1.1 Language

- All the code, comments, variable names, resources
  EVERYTHING must be written in ENGLISH only!

1.2 Code formatting

- Only one instruction is allowed per line
  that is, you cannot have more than one ';' per lines.

2 Data Type (class, struct, typedef)

- always start with a capital letter (to differentiate with variable name)

```
//Bad
class myType
{

//Good
class MyType
{
```

- [c++] the following type must be defined and used instead of the intrinsic ones: s8, u8, s16, u16, s32, u32, etc.

3 Variables

<scope> + <underscore> + <nameOfTheVariable>

Where <scope> is there for every non-function variables.
k -> constant
m -> member

Where <nameOfTheVariable>:

- Starts with a lower case.
- Every other word starts with a capital letter.
- Is always clear enough to avoid reading a comment explaining it's function.
- Is not a negation for booleans.
- Is a question if it contains a bool. The value of the bool is the answer asked by it's name.

```
//Bad
m_paint;
m_isNoError;
m_isNotFound;

//Good
m_isPainting;
m_isError;
m_isFound;
```

- [java] Always use integers instead of bytes and shorts. They are all aligned on 4 bytes and some phones have nasty bugs handling shorts and bytes. (except for arrays)

3.1 Suggestions

- [c++] Favor ‘int’ over ‘unsigned int’.
  Only use ‘unsigned int’ if you’re really going to use all those bits, for example if you’re using it to store bit flags.
  Unsigned int can cause nasty bugs like: for (unsigned int i = 10; i >= 0; --i);
  (use s32 or u32)
- [c++] If you need char (for instance for strings, letters, etc), always specify signed or unsigned. (use s8 or u8)
- [c++] Always assign pointers to 0 after deleting them.
- [c++] If a class member can be either a pointer or a reference, prefer a reference.
- Variable should be declared in the smallest scope possible; you shouldn’t use global or static variable in a loop.
- [c++] For member variables Prefer calling constructors in the initialisation list from the assignation with operator=.

```
//Not that good
Object::Object()
{
   m_member = 0;
}

//Awesome
Object::Object()
       :   m_member(0)
{
}
```

- Variable name shouldn’t be longer than 30 characters.

4 Functions

4.1 function declaration

<Header>
<scope> <return type> <NameOfTheFunction> <space> <argument list>

Where < Header > :
- Is always there
- Always starts with /// to allow documentation by doxygen.
- Never contains /* and */ .
- Is preceded by at least 1 empty line.

```
...previous function declaration...

///------------------------------------------------------
/// This function contains blablabla...
///------------------------------------------------------
```

Where <scope> is one of the language valid scope

```
public
private
protected
static public
...etc...
```

Where < NameOfTheFunction >

- Every word starts with a capital letter.
- Is always followed by one space (before the parenthesis) for the definition and is never followed by a space (before the parenthesis) for the call (makes search easier).
- Is always clear enough to avoid reading a comment explaining it's function.
- Is a question if the function returns a bool.

Where <Space>
- always follow each function name
  this makes search of function definition within code easier
  if you search for the definition of oneFunction(...) you just have to search for the string "oneFunc ("

```
//Bad
static int FirstFunc()
{
}

int secondfunc(int param1, long param2)
{
}


//Good
//-----------------------------------------------------
/// This function is the first function
//-----------------------------------------------------
static int FirstFunc ()
{
}

//-----------------------------------------------------
/// this function is the second function
//-----------------------------------------------------
int Secondfunc (int param1, int param2)
{
}
```

4.2 function invocation

<NameOfTheFunction><argument list>

no space between function name and argument list

```
//Bad
...some code...
onefunctionCall (param1, param2);
...some code


//Good
...some code...
onefunctionCall(param1, param2);
...some code
```

4.3 Function Parameters

<type> <nameOfTheVariable>

Where <type> is one of the language valid type

```
        int
        long
        byte
        ...etc...
```

Where <nameOfTheVariable>:

- Starts with a lower case.
- Every other word starts with a capital letter.
- Is always clear enough to avoid reading a comment explaining it's function.

4.4 Suggestions

- always try to minimize the number of parameter to pass to a function
- Use terms "Get/Set" when accessing an attribute directly.
- Use term "Compute" when something is calculated.
- Use term "Find" when the function search for something.
- Use term "Initialize" when an object or a concept is established.
- Functions name shouldn't be longer than 30 characters.
- Complement names must be used for complement operations:
    - Set/Get
    - Add/Remove
    - Create/Destroy
    - Start/Stop
    - Insert/Delete
    - Increment/Decrement
    - Old/New
    - Begin/End
    - First/Last
    - Up/Down
    - Min/Max
    - Next/Previous
    - Open/Close
    - Show/Hide
    - Suspend/Resume
    - …
- [c++] If a function takes a parameter that can be either a pointer or a reference, prefer a reference.

For instance, the address cannot be 0.

5 Constants

- [c++] Use "const" as much as possible for constant variables, class member, function arguments and constant functions.
  In fact, if anything can be consted, then const it!
- [java] Use "final static int" scope as much as possible for constant declaration

6 Constructors

6.1 Suggestions

- [c++] Always overload the operator ‘=’. If you don’t need it, declare it as private and don’t implement it.
- [c++] Always declare constructors with explicit.

7 Namespaces

- [c++] Don’t use (create) them! Some ARM compiler are having problems with them

8 Header Files

- [c++] Avoid using #include as much as possible in header files; prefer forward declaration; it reduces dependencies.

```
// Bad
#include “b.h”
class A
{
   B* b;
}

// Good
class B;
class A
{
   B* b;
}
```

- [c++] Only specify 1 include path for your project in the “additional include directories” in Ms VisualStudio. For all include of header in subfolders, use a relative path to the base path you included. So if you want to include the file AI.h which in on the folder src/AI, use src as your base path and:
  #include “AI/AI.h”

- The order of inclusion has to be:
  - The global header file if there’s one.
  - Empty line.
  - The header file of the current cpp file.
  - Empty line.
  - System header files. (STL, standard libs) -> Usually between ‘<’ and ‘>’
  - Empty line.

9 Preprocessor commands

- **Are never used to define constants** (see 5.Constants for constants declaration).
- Always contains a comment at the end of a #else / #endif to remind the #if.
- Use #define with 0/1 as value to disable/enable it. In the code, use "#if defined" instead of "#ifdef"

```
#define USE_SYSTEM_FONT 0

#if defined USE_SYSTEM_FONT
       ...some code...
#else // #if defined USE_SYSTEM_FONT
       ...some code...
#endif // #if defined USE_SYSTEM_FONT
```

- preprocessor conditional command must be indented within the source code as a regular if..else code branch

```
// Bad
...some code before...
#if defined SOME_DEFINE
       ...some code in define 1...
#else
...some code in define 2...
#endif
...some code after...


// Good
...some code before...
#if defined SOME_DEFINE
       ...some code in define 1...
#else //#if defined SOME_DEFINE
       ...some code in define 2...
#endif //#if defined SOME_DEFINE
...some code after...
```

  the historical reason for preprocessor command to start at beginning of a line is now a thing of the past
  Modern preprocessor are now able to parse correctly preprocessor command which are not at the beginning of a line
  if you happen to meet an old preprocessor which doesn't accept this syntax, use cpp.exe from cygwin instead

9.1 Suggestions

- avoid as much as possible the use of preprocessor command
  almost all of the preprocessor command can be replaced by the use of constants

```
// this code
#define SOME_SWITCH

...some code before...
#if defined SOME_SWITCH
       ...some code in switch 1...
#else //#if defined SOME_SWITCH
       ...some code in switch 2...
#endif //#if defined SOME_SWITCH
...some code after...

// can be replaced in java by
final static boolean SOME_SWITCH = true;

...some code before...
if (SOME_SWITCH)
{
       ...some code in switch 1...
}
else
{
       ...some code in switch 2...
}
...some code after...
```

10 Comments

- always use single line comments "//"

– the use of /* ... */ comments is forbidden
  /*...*/ comments should only be used for debugging purpose to comment large chunk of code
  once the debugging is over, remove those comment (and the line in comment if you don't need those)
  (SVN will always allow you to recover those line if you find out you erased those by mistake)
– Are always on a new line.
– Are always up-to-date !
  always update your comment as soon as you made a change

10.1 Documenting code for Doxygen

Note that the goal of using Doxygen is to describe the interface and the usage of each class, function, etc. Other detailed comments addressing algorithms and specific code will still be found into the definition itself (using "//").

For comments describing a class, a struct, a function, a member variable or an enum:
– Use "///" instead of "//";
– Always write the comment before the declaration;
– Typically, the description for each function and variable will be into the class definition
– The comment describing a function should contain the following information (and respects the same ordering):
  – Function description
  – Parameter description using "@param"
  – Return description using "@return"
  – Exception description using "@exception"
  – Risk description using "@warning" (for CPU and memory consuming functions)

```
//-----------------------------------------------------
/// calculate the integer square root of a value
/// @param value to calculate
/// @return integer square root of the value
/// @exception NaN if value < 0
/// @warning this function uses a lof of CPU
//-----------------------------------------------------
static int sqrt (int value)
{
      ...
```

10.2 Suggestions

To clarify the code, any space and empty lines can be added between special comments.
It is encouraged to use other features of Doxygen to extend the potential of the documentation.
– Using indented lists :

```
/// This is a list
/// - Item 1
///    - SubItem 1
///    - SubItem 2
/// - Item 2
///    - Etc...
```

– Using "@see" to link to any another function or class

```
/// @see ClassNameOrFunctionName
```

– Using "@todo" in class description to mention any unfinished work

```
/// @todo Optimize that function
```

See Doxygen documentation for more information.

11 if / for / while / do

- Always have curly braces, even for single line statements

```
//Bad
if (isTestOk)
       doSomething();

//Good
if (isTestOk)
{
       doSomething();
}
```

- Executable statements in conditionals are forbidden

```
//Bad
if (!(fileHandle = Open (fileName, "w")))
{
}

//Good
fileHandle = Open(fileName, “w”);
if (!fileHandle)
{
}
```

- each conditionnal statement must have its own parenthesis
  conditionnal statements must be indented on the same level
  when spreaded across many lines, boolean operator are always at the begining of the line (indented with boolean operator at the same level)

```
if (    (isTest1Ok)
    &&  (     isTest2Ok
         ||   (isTest3Ok && (myVar1 == 3))
        )
    ||  (myVar != 5)
    )
{
       doSomething();
}
```

- avoid infinite loop
  loop ending condition must be clearly specified

```
//Bad
for (;true;)
{
}
//Bad
while(true)
{
       ...
       if (someCondition)
              break;
}
```

12 switch...case

- Always have curly braces between each case
- Always have the default statement
- "case" statement are indented within the "switch" statement

```
//Bad
switch(value)
{
case 0:
	...some code for value=0...
break;

case 1:
case 2:
	...some code for value=1 or value=2...
break;
}

//Good
switch(value)
{
	case 0:
	{
		...some code for value=0...
	}
	break;

	case 1:
	case 2:
	{
		...some code for value=1 or value=2...
	}
	break;

	default:
		ERROR("unexpected value");
}
```

– only for this following case, it is accepted to not use the curly brace after the "case" statement

```
case 1:
	...some code for value=1...
case 2:
	...some code for value=1 or value=2...
break;
```

13 Operations

– Are always surrounded by a set of parenthesis if there's more than one operation in the same line.

```
// Bad
int i = 10 + 30 / 2;

// Good
int i = 10 + (30 / 2);
```

14 Operators

<space><operator><space>

– Where the space is always there except for unitary operators ('!', '~', '++', '--', '&', '*', '.', '->')

```
// Bad
int i = 10 + (3/2);
int i = 10+(3 / 2);
int i=10;

// Good
int i = 10 + (3 / 2);
int i = 10;
```

15 Numbers

- Utilisation of magic numbers in the code has to be avoided. Numbers other than 0 and 1 should be declared as named constants instead.
- [c++] Floating point constants should be avoided (especially on critic code) and should always be written with decimal point and at least one decimal.

```
//Bad
double total = 0;
double speed = 3e8;

//Good
double total = 0.0;
double speed = 3.0e8
```

16 Casting

- [c++] Use C style casting for built-in/intrinsic type.

17 Templates, STL

- [c++] **NEVER use Template**. They can easily make the code size explode, or hard to understand.
- [c++] If you have to use them (which you shouldn't), never use default parameter in a template: they are not supported by some versions of ARMcc.
- [c++] **Don't use STL** or any outside libs. They're only compiling with GCC.

18 Memory allocation

- Always use macro GLL_NEW, GLL_DELETE and GLL_DELETE_ARRAYS instead of new, delete and delete[].
  It helps integrating a memory manager and port from C to Java.
- Always allocate memory at loading time, never dynamically during the game since some phones are *really* slow to allocate it.

18.1 Suggestions

- [c++] Use a memory pool (allocate a big chunk of memory and assign it yourself) to speed-up things.
- [c++] Use a memory manager to track memory leaks.

19 Code Refactoring

– When removing (merging) a class (for a Java port for instance), change the name of the members of this class (functions and variables) to include the name of the class before its suppression.

```
//Before refactoring
class AnimationData
{
       int m_oneVariable;
       ...
};
class Renderer
{
       ...
};

//After refactoring
class Renderer
{
       // Class Animation data is now regrouped in class
       // Renderer and each instance of the old
       // AnimationData is now an index in the array.
       int m_animationData_oneVariable [];
       ...
}
```

20 Float and fixed point

– **All 2D games MUST NOT used float type**
  Fixed point values must be used instead of float
– **[c++] all 3D gamesMUST NOT use float type**
  Fixed point values must be used instead of float
  For 3D games on "big" platform (PC, Xbox, etc...) this rule can be bend
– [java] only 3D games using jsr184 and linking with CLDC1.1 may use float type

  ARM processor (present on almost all phones/embedded devices) are very bad at float calculation, therefore using fixed point guarantee better performance and portability

21 Tools

As we are coding our games for J2ME phones (java) and Brew Phones (C/C++) mainly
We highly recommend that you code the tools you're going to make with one of these language (java or C/C++), simply because we are all familiar with it.

python is the only accepted scripting language.
The version number of the interpreter should be documented in the Doc directory.
It is a good idea to include the installation package for the interpreter in the Tool directory (in Tool/install for example)

the use of other languages to code our tools is stricly forbidden (C#, ruby, j#, perl, etc…)
but you definitely have the right to study those for your own culture :)

22 SDK version

- As much as possible make sure your project compile with the latest version of the sdk.
  If a new version of the sdk comes out during the development period. Please install it and update your version so that it compiles with it.
- If you are using an old SDK for very specific reasons (tools require this SDK version, some dependencies where removed, some api were deprecated, etc…) --> **add an explicit explanation about the reason why it is was not possible to use the latest SDK version**

23 Build script

- **All game must be able to be compiled from the DOS command line**
- **build script must contain a configuration file called "config.bat" and a "make.bat" batch file**

  Config.bat shall set all the path needed to configure properly the project from one machine to another. If the project was to be given to a new programmer, he should only have to modify this file, set the path to his sdk in order to be able to use make.bat and compile a version of the game

On the command line, calling

- "Make data"
  compile all the data needed for the game
- "Make debug" or "Make release "
  When using pre-processor first call should compile the game in debug mode while second one should compile in release mode
  When not using any pre-processor, both call should compile a version of the game. Debug or release configuration should be found inside the source code in a const variable
- "Make run"
  Launch the game, on an emulator, or whatever
  this command is optionnal but appreciated

in any case, the important point is that any new user must be able to build the data and a version (in debug or release mode) from the command line in less than 5 minutes.

24 Compat.txt File

- **each build must include a file called "compat.txt" which contain the list of phones that this build include**
- **it is also required to fill the SVN tree for those build with a link toward the "parent" build (either an external on the parent source, or a compat.txt file explaining where to find the sources)**

```
if SonyEricsson K800i, SonyEricsson W850i and SonyEricsson W900i are based on the same build, then
SVN structure must be as followed

ProjectRepository/
    |
    |- SonyEricsson
    |    |- K800i
    |    |    |- Tags
    |    |    |- Trunk
    |    |    |    |- compat.txt (contains SEK800i, SEW850i, SEW900i)
    |    |- W850i
    |    |    |- Tags
    |    |    |- Trunk
    |    |    |    |- compat.txt (explain that the sources are located in SonyEricsson/K800i)
    |    |- W900i
    |    |    |- Tags
    |    |    |- Trunk
    |    |    |    |- compat.txt (explain that the sources are located in SonyEricsson/K800i)

or

ProjectRepository/
    |
    |- SonyEricsson
    |    |- K800i
    |    |    |- Tags
    |    |    |- Trunk
    |    |    |    |- compat.txt (contains SEK800i, SEW850i, SEW900i)
    |    |- W850i
    |    |    |- Tags
    |    |    |- Trunk --> external on SonyEricsson/K800i/Trunk
    |    |- W900i
    |    |    |- Tags
    |    |    |- Trunk --> external on SonyEricsson/K800i/Trunk
```

25 SVN

– **All Project must use the main versioning server (SVN)**
  Contact World-SVN (World-SVN@gameloft.com) to create repository, and manage access right

  in case of slow connection speed, a mirroring solution is also available -> please contact World-SVN
– read "Gamelfot SVN Usage" document to get more information regarding our usage of SVN

26 Project Golded

When your project reach gold stateYou must
– on SVN
  While approaching the gold version, you must convert all the external in your version to branches to make sure a modification made outside of your project won't interfere with your source;

  then regularly create tags in the tags directory for each important step (gold candidate, etc…)

– When final gold is achieved
  pack all the sources, data, build script and send the package to your data manager for storage.
  This is extremely important. Do not consider the SVN as a storage device.

The gold package must be sent to the data manager for Storage.

27 Document history

27.1 Version 0.0.1

Tuesday, May 24th, 2005:
- Document creation.
- Branch from Gameloft Montréal C++ Coding Guideline.

27.2 Version 0.0.2

Monday, May 30th, 2005 :
- Integrating France and China's comments.

27.3 Version 0.0.3

Tuesday, June 14th, 2005:
- Integrating France's comments regarding:
- Package submission (batch file).
- Refactoring section.
- Integrating Romania's comments regarding:
  - Data type to use in C++.
  - Precisions about variables scopes.
- Integrating China's comments regarding:
  - Using numbers in variables names.
  - Variable and function name length.
- Header file inclusion.
- Braces alignments.
- Spaces for unitary operators.
- Comments for classes and structs.
- Integrating Canada's comments regarding:
  - Changing GL for GLL for memory manager (conflict with OpenGL).
  - Default parameters on templates.
  - Usage of unsigned int.

27.4 Version 0.0.4

Monday, July 25th, 2005:
- Integrating NY comments regarding:
  - STL
  - Floating point
- Changing the requirements for constructors as suggestions.

27.5 Version 0.0.6

Wednesday, October 12th, 2005:
- Modified "Tools" subdirectory in section "20.1 Project Architecture"
- Removed section "10.1 Suggestions"
- Added section "10.1 Documenting code for Doxygen"
- Added section "10.2 Suggestions"
- Modified DATA TYPE section according to China's comments concerning types definition (it's now only a suggestion).
- Modified CONSTRUCTORS section according to China's comments: removed the medatory declaration of the copy constructor.

27.6 Version 1.0.0

Thursday, October 13th, 2005:

- Removed the static prefix from the variable declaration because it shouldn't be used at all to help Brew => JAVA ports.
- Removed C++ style casting to help portability with other technologies.

27.7 Version 1.0.1

Monday, October 17th, 2005:
- Changed Project Architecture section according to Deployment-publishing needs: added a trunk and branch sub-folder to the phoneModel folder, added a PhoneConstructor folder parent of the phoneModel folder.

27.8 Version 1.0.2

Monday, December 19th, 2005:
- Changed Project Architecture section: ressources got their own directory outside of the project, use of external to use the correct resources per project

27.9 Version 1.0.3

Monday, September 26th, 2006:
- Added SVN architecture, and explanation

27.10 Version 1.0.4

Thursday, September 28th, 2006:
- Added a rule in variable section, asking to use only ints in java (no shorts and bytes).
- Cleaning different obselete rules.
- Specifying the language between [] for language specific rules.
- Added a rules stating that we shouldn't use templates.
- Added a section about SVN usage.

27.11 Version 1.0.5

Thursday, June 6th, 2007:
- Added rule : all 2D games must not used float type, but fixed point
  float type are only authorized on 3D java games

27.12 Version 1.0.6

Monday, August 8th, 2007:
- reformat the documentation to open office
- added more strict rules
- seperated detailed SVN usage into a seperated document

27.13 Version 1.0.7

Wednesday, October 3, 2007:
- add compat.txt requirement