File Naming

- The file name is the same as the class name.
- The implementation file can be split into multiple files.

ex:

Util.h

Util.cpp

Util_win32.cpp

File

- The end line is set to LF.
- Encoding is UTF-8 (without BOM).
- Always put a period(.) at the end of the comment.
- One line should not exceed a maximum of 160 columns.
- The header file should not exceed a maximum of 500 lines.
- The implementation file should not exceed a maximum of 1000 lines.

Class Naming

- Namespaces and classes start with an uppercase letter.
- All words are written together and the first letter of each word is capitalized.
- Make it into a noun form whenever possible.

ex:

```
namespace BnD;
namespace Util;
Class AlarmManager;
Class PushManager;
Class DeviceRepository;
```

Variable Naming

- Variables start with a lowercase letter, and member variables are prefixed with '_', static variables are prefixed with 's_', and global variables are prefixed with 'g_'.

- All words are written together, and the first letter from the second word is capitalized.

ex:

```
int count, maxCount;

string _name;

char* s_buffer;

char* g_sharedBuffer;
```

Function Naming 1

- Functions start with a lowercase letter.

- All words are written together and the first letter from the second word is capitalized.

- Make it into a verb form whenever possible.

ex:

```
void initialize();

bool createAgent();

int findPath();
```

Function Naming 2

- For a member variable getter, uses the member variable name as the function name (get is not added. get is added only when receiving a value as an argument).

- For a member variable setter, capitalize the first letter of the member variable name and add set.

ex:

```
int size() const;

void getSize(int* size) const;

void setSize(int size);
```

Constant Naming

- All constants are capitalized.
- All words are connected with ‘_’.

ex:

```
const int VISIBLE = 0x04;
const int MAX_SIZE = 10;
enum { ON, OFF, NOT_SET };
const string DEFAULT_NAME = ”John”;
```

#include order: Header file

- Include in the following order: B1 library, D1 library, general header, and system library.
- Include in alphabetical order.

ex:

```
#include <B1/B1BaseServer.h>

#include <D1/D1BaseServer.h>

#include “PushManager.h”

#include <string>
#include <vector>
```

#include order: Impl. file

- Include in the following order: general header, B1 library, D1 library, and system library.
- Include in alphabetical order.

ex(If it is an implementation file belonging to the B1Base library):

```
#include “B1Condition.h”
#include “B1Thread.h”

#include <B1Network/B1BaseServer.h>

#include <string>
#include <vector>
```

Forward declaration order

- Declare in alphabetical order.

ex:

```
class AlarmManager;
class PushManager;
```

Access Specifier order

- Declared in the order of private, protected, public (except constructor public)

- Declare member variables and functions in order.

- Basically, member variables are not declared public.

ex:

```
private:
    mutable B1Lock _lock;
protected:
    int _value;
private:
    void doSomethingPrivate();
protected:
    void doSomethingProtected();
public:
    void doSomething();
```

#define Naming

- #define is in all uppercase letters and starts with '_'.
- All words are connected with '_'.

ex:

```
#define _DEBUG
#define _ALARM_MANAGER_H
```

Bracket and indent

- Everything inside curly brackets {} is indented.
- The opening bracket { is placed at the end of the line.
- Indentation should be 4 spaces (tabs are not used).

ex:

```
namespace BnD {
    class AlarmManager {
            …
    };
}
```

Bracket and indent exception

- Class access specifiers are not indented.
- The opening bracket { of the function is placed at the beginning of the next line.

ex:

```
class AlarmManager {
    public:    …
};

void function()
{
    …
}
```

Indent: if-else

```
if (condition) {
    …
}
else if (condition) {
    …
}
else {
    …
}
```

Indent: switch

```
switch (value) {
    case condition1:
        …
        break;
    case condition2:
    {   // exception for case.
        …
    }
         break;
    default:
        break;
}
```

Indent: for

```
for (init; condition; step) {
    …
}
for (longlong_initialization;  // 160 column limit
     longlong_condition;
     longlong_step) {
    …
}
```

Indent: while

```
while (condition) {
    …
}
do {
    …
} while (condition);
```

Indent: try-catch

```
try {
    …
}
catch (exception) {
    …
}
catch (…) {
    …
}
```

template declaration

- Template declarations are separated on separate lines.

ex:

```
template<typename T>
class AlarmManager{
    …
};
template<typename T>
void AlarmManager<T>::add(const T& alarm)
{
    …
}
```

Bracket in control statements

- If the control statement is one line long, bracket can be omitted.

ex:

```
if (condition)
    return;

void function(int value, bool flag)
{
    if (value < 0) return;
    if (flag != true) return;
    …
}
```

Spacing: control statements

- Control statements are enclosed in brackets and spaces.
- The brackets and the contents within them are attached.

ex:

```
if (condition) {
    …
}
if (   condition   ) { // prohibition.
```

Spacing: function

- Functions are written with brakcets.
- The brackets and the contents within them are attached.
- Use spaces between function arguments.

ex:

```
func(arg1, arg2);

func (arg1, arg2);       // prohibition.
func(  arg1,arg2  );     // prohibition.
func(arg1,arg2);         // prohibition.
```

Spacing: operator

- Operators and operands, except single operators, are separated.

ex:

```
a * (b + c)

(a < b) ? a : b

(0 <= a) && (a > 10)

a++  // except single operator.
```

Spacing: pointer, reference

- Pointer and reference symbols are attached to variable types.

ex:

```
int* pointer;
string& reference;
```

Line break

- When dividing long lines, place commas or operators at the end of the line.

ex:

```
result = value 1 +
         value2;
if (condition1 &&
    condition2) {
```

Function argument format

- If the variable passed as an argument to the function is mutable, it is passed as a pointer. Otherwise, it is passed as a const reference.

ex:

```
void getData(Data* data) const;

void setData(const Data& data);
```

Etc

- C++ is based on the C++20 version.
- Windows is distinguished by whether '_WIN32' is declared.
- Linux is distinguished by whether '__linux__' is declared.