**Method and Function Names**

- Suffixes are sometimes useful:
  - `Max` to mean the maximum value something can have.
  - `Cnt` the current count of a running count variable.
  - `Key` key value.

For example: `RetryMax` to mean the maximum number of retries, `RetryCnt` to mean the current retry count.

- Prefixes are sometimes useful:
  - `Is` to ask a question about something. Whenever someone sees `Is` they will know it's a question.
  - `Get` get a value.
  - `Set` set a value.

For example: `IsHitRetryLimit`.

---

**Include Units in Names**

If a variable represents time, weight, or some other unit then include the unit in the name so developers can more easily spot problems. For example:

```cpp
uint32 mTimeoutMsecs;
uint32 mMyWeightLbs;
```

Better yet is to make a variable into a class so bad conversions can be caught.

---

**No All Upper Case Abbreviations**

When confronted with a situation where you could use an all upper case abbreviation instead use an initial upper case letter followed by all lower case letters. No matter what.

**Example**

```cpp
class FluidOz // NOT FluidOZ
class NetworkAbcKey // NOT NetworkABCKey
```

---

**Class Names**

- Use upper case letters as word separators, lower case for the rest of a word
- First character in a name is upper case
- No underbars ('_')

**Example**

```cpp
class NameOneTwo
class Name
```

---

**Method Names**

Use the same rule as for class names.

**Example**

```cpp
class NameOneTwo
{
    public:
        int DoIt();
        void HandleError();
}
```

---

**Class Attribute Names**

- Attribute names should be prepended with the character 'm'.
- After the 'm' use the same rules as for class names.
- 'm' always precedes other name modifiers like 'p' for pointer.
Example

class NameOneTwo
{
public:
    int VarAbc();
    int ErrorNumber();
private:
    int mVarAbc;
    int mErrorNumber;
    String* mpName;
}

Method Argument Names

- The first character should be lower case.
- All word beginnings after the first letter should be upper case as with class names.

Example

class NameOneTwo
{
public:
    StartYourEngines(Engine& rSomeEngine,
                      Engine& rAnotherEngine);
}

Variable Names on the Stack (used only inside function)

- use all lower case letters
- use '_' as the word separator.

Example

int NameOneTwo::HandleError(int errorNumber)
{
    int error= OsErr();
    Time time_of_error;
    ErrorProcessor error_processor;
}

Reference Variables and Functions Returning References

References should be prepended with 'r'.

Example

class Test
{
public:
    void DoSomething(StatusInfo& rStatus);
    StatusInfo& rStatus();
    const StatusInfo& Status() const;
private:
    StatusInfo& mrStatus;
}

Global Variables

Global variables should be prepended with a 'g'.
Example

Logger  gLog;
Logger* gpLog;

Global Constants

Global constants should be all caps with '_' separators.

Example

const int A_GLOBAL_CONSTANT= 5;

Static Variables

Static variables may be prepended with 's'.

Example

class Test
{
  public:
  private:
    static StatusInfo msStatus;
}

Enum Names

Labels All Upper Case with '_' Word Separators

This is the standard rule for enum labels.

Example

class PinStateType
{
  PIN_OFF,
  PIN_ON

Required Methods for a Class

Default Constructor

Copy Constructor

If your class objects should not be copied, make the copy constructor and assignment operator private and don't define bodies for them.

Assignment Operator

If your objects should not be assigned, make the assignment operator private and don't define bodies for them.

The Law of The Big Three

A class with any of (destructor, assignment operator, copy constructor) generally needs all 3. An example using default values:

class Planet
{
public:
  // The following is the default constructor if
  // no arguments are supplied:
  //
  Planet(int radius= 5);

  // Use compiler-generated copy constructor, assignment, and destructor.
  // Planet(const Planet&);
Braces / Policy

Brace Placement

- Place brace under and inline with keywords:
  
  ```c
  if (condition) while (condition)
  {                {
    ...            ...
  }                }
  ```

When Braces are Needed

All if, while and do statements must either have braces or be on a single line.

Indentation/Tabs/Space Policy

- Indent using 4 spaces for each level.
- Do not use tabs, use spaces. Most editors can substitute spaces for tabs.

Example

```c
void func()
{
  if (something bad)
  {
    if (another thing bad)
    {
      while (more input)
      {
      }
    }
  }
}
```

Parens () with Key Words and Functions Policy

- Do not put parens next to keywords. Put a space between.
- Do put parens next to function names.
- Do not use parens in return statements when it's not necessary.

Example

```c
if (condition)
{
}
```
```c
while (condition)
{
}
```
```c
strcpy (s, s1);
return 1;
```