ES 036 Coding Style

Purpose

1. Easy to figure out someone else's code
2. New members of the team can get up to speed quickly
3. People new to a programming language are spared the need to develop a personal style and defend it to death
4. People make fewer mistakes in a consistent environment

Naming Conventions

Names are the heart of programming. Make the name indicate the action. E.g. `DumpDataToFile()` instead of `Dump()` or `Data()`. Never use names such as `Thing`, `DoIt()`, `param_1`

- Suffixes and prefixes add a lot of meaning. E.g. `Max, Key, Is, Get, Set`
- No all upper case abbreviations. E.g. `CourseSe250a` instead of `CourseSE250a`
- Classes, Global variables and global functions begin with an upper case letter and use upper case letters as word separators. Everything else is lowercase. No underbars in the name. E.g. `DumpDataToFile()`
- Class methods and attributes are prefixed with 'm' and follow the naming convention of globals. E.g. `mDumpData()` and `mRetryMax`
- All local variables and function parameters are all lowercase with underbars as word separators. E.g. `file_name, cell_max`
- Constants and macros should be all uppercase with underbars as word separators. E.g. `LENGTH_MAX, CELL_COUNT`
- Typedef names should follow the same convention as globals with '`Type`' appended. E.g. `CellType`.

Layout

- Braces should be placed under and in line with keywords or function names.
- Indent using 4 spaces for each level
- Indent as much as needed, but no more. If you have more than 5-6 levels, you may want to think about factoring out the code.
- Do not put parentheses next to keywords. Do put them next to function names.
- Always put the constant in the left hand side of a comparison.
- Falling through a case statement is permitted as long as a comment is included
- The default case should always be present and trigger an error if it should not be reached, yet is reached.
- Block declarations should be aligned
- There should be only one statement per line unless the statements are very closely related.
- Comments may consist of multiple lines, but the first line should be a self-containing meaningful summary.
- Make gotchas explicit by using embedded keywords (E.g. `:TODO:, :BUG:, :KLUDGE:, :TRICKEY:, :WARNING:, :COMPILER:`), in comments. This also allows easy auto-generation of reports. Embedding date and author allows someone else to quickly find when it happened and who to ask for more information.
- Add a comment to the ending brace of large blocks to indicate its purpose of the block
- Use blank lines to separate code blocks meaningfully.

Code examples

```
// :AUTHOR: Jagath Samarabandu
// :TODO: 2/3/00 JS: Make better examples
//    Only if there was more time
//
// :TODO: 15/9/05 JS: Make it better suited for C++
// :WARNING: This code will not compile


enum DoodleType {THIS, THAT, OTHER};

void MakeSample(int section, int sample_max) // global and local naming
{
    // Aligned blocks
    int           sum, cell_max;
    float         variance;
    DoodleType    doodle;
    static int    CellCount;
    const int     SAMPLE_LENGTH(20); // constants are all uppercase

    if (30 == cell_max) // braces formatting and comparisons
    {
        while (twiddling_thumbs)
        {
            sum = sum + 1;
        }

        switch (doodle)
        {
            case THIS: DoThis(); break; // Same line is OK
            case THAT: // fall through!!
            case OTHER: // Same line is not OK
                DoThat();
                DoOther();
                break;
            default:
                cerr << This shouldn't be happening";
                break;
        }
    } // if (30 == cell_max)

    MoreCode();

} // void MakeSample
```