Steem Source Code Reorganization

For Steem source tree reorganization I have utilized some basic principles briefly explained below.

Steem was violating a lot of the normal practices. The most unusual techniques used were:

- Mixture of declaration **and** definitions in either .h **or** .cpp
- The inclusion of .cpp files from other .cpp files
- The effect of including some header files **was dependent upon the context** in which it is included. For example the usage of IN_MAIN and IN_EMU defines was completely changing the result of the inclusion of most of the .h files.

Exemple

```
#ifdef IN_MAIN
int foo = 3;
void baz() { do_something(); }
#else
extern int foo;
void baz();
#endif
```

Some Useful References

- http://wiki.hsr.ch/Prog3/files/overload72-FINAL_DesigningHeaderFiles.pdf
- http://www.gamedev.net/page/resources/_/technical/general-programming/organizing-code-files-in-c-and-c-
- http://geosoft.no/development/cppstyle.html#Source Files
- http://www.cs.utexas.edu/~lavender/courses/EE360C/lectures/lecture-02.pdf
- http://www.gamedev.net/page/resources/_/technical/general-programming/organizing-code-files-in-c-and-c-r1798

Tree file Organization

Steem is now organized as a collection of “header” files (.h) and implementation files (.cpp).

Good programming practice is to pair the name of a .h file with a .cpp file, for example:

- fdc.h defines fdc constants, types, class declaration, and inline methods
- fdc.cpp defines the implementation of what has been declared in the .h file.

The user of a class should include the corresponding .h file at compile-time, and links to the already compiled .cpp file

The .h files

Header files contain mostly declarations, to be used in the rest of the program. The skeleton of a class is usually provided in a header file, while an accompanying implementation file provides the definitions to put the meat on the bones of it. Header files are not compiled, but rather provided to other parts of the program through the use of #include.

A typical header file looks like the following:

```
// Inside sample.h
#ifndef SAMPLE_H
#define SAMPLE_H

// Contents of the header file are placed here.

#endif /* SAMPLE_H */
```

Since header files are included in other files, problems can occur if they are included more than once. This often results in the use of "header guards" using the preprocessor directives (#ifndef, #define, and #endif).

Header files contain:

- #include of other .h files
- preprocessor macros (#define)
- constant declarations (consts)
- type definitions (typedefs)
- enumerated types (eums)
- class declaration
- inline function/method definitions
- procedure declarations (prototypes)

Considerations for Designing C++ Header Files

- Create Self Contained Headers that do not rely on the presence of macros, declarations or definitions being available at the point of inclusion. Another way to look at this is that they should compile on their own. One common way to ensure headers are self-contained is to make them the first include in the corresponding implementation file
- The effect of including a header file should be deterministic (and not be dependent upon the context in which it is included).
- Including a header file should be idempotent (including it several times should have the same effect as including it once).
- A header file should have a coherent purpose (and not have unnecessary or surprising effects).
- A header file should have low coupling (and not introduce excessive dependencies on other headers).

The .cpp files contain:

An implementation file includes the specific details, which are the definitions, for what is done by the program. While the header file declared what a class could do, the corresponding .cpp file defines how the class acts.

- #include of .h files
- global/static data and object definitions
- global/static functions and procedures
- class “method” implementations

Exceptions

Draw_c.cpp makes use of a technique referred to as “File Iteration” (repeatedly #including a file under the control of preprocessor macros to generate families of macro expansions). Although this technique is undeniably useful, the circumstances where it is appropriate are rare.