Coding standards for MIBitCoinANSIC

1. Comments

Please put comments on

- a broad overview at the beginning of a module
- data structure definitions
- global variable definition
- at the beginning of a function
- tricky steps within a function

2. Source File Organization

Use the following organization for source files:

- includes of system headers
- includes of local headers
- type and constant definitions
- global variables
- functions

Notes: define functions before their use and avoid having nested includes

3. Declarations and Types

- When declaring a global function or variable in a header file, use an explicit extern. For functions, provide a full ANSI C prototype:
  - extern int errno;
  - extern void free(void *);
- Do not use parameter names in function prototypes - you are increasing the risk of a name collision with a previously-defined macro, e.g.:
  - #define fileptr stdin
  - ...
  - extern int foo(FILE *fileptr);
- Instead, document parameter names only as necessary using comments:
  - extern void veccopy(double * /*dst*/, double * /*src*/, size_t)
- Don't rely on C's implicit int typing;
  - i.e. don't say: extern foo; say: extern int foo;

4. Use of the Preprocessor

- Macros should avoid side effects. If possible, mention each argument exactly once. Fully parenthesize all arguments. When the macro is an expression, parenthesize the whole macro body. If the macro is the inline expansion of some function, the name of the macro should be the same as that of the function, except fully capitalized.
- When continuing a macro across multiple lines with backslashes, line up the backslashes way over on the right edge of the screen to keep them from cluttering up the code. Example:

```
#define OBNOXIOUS(X)                               \
(save = (X),                                       \
dosomethingwith(X),                          \
(X) = save)
```

5. Naming Conventions

- Names should be meaningful in the **application** domain, not the **implementation** domain. This makes your code clearer to a reader who is familiar with the problem you're trying to solve, but is not familiar with your particular way of solving it
- Names should be chosen to make sense when your program is read
  - Variables should be noun clauses
  - Boolean variables should be named for the meaning of their "true" value
  - Function names should reflect what they return
  - boolean-valued functions of an object should be named for the property their true value implies about the object
  - lowercase Function Names, Function Parameters, variables
  - Capitalize Enumeration Constants, Typedef Names and Capitalize the tag name to match the typedef name

6. Indentation and Layout

- Try to stay inside the mythical 79 column limit. If you can't, look for a tasteful place to break the line
- Use real tab characters for indenting. Tabs are always 8 spaces
- Avoid unnecessary curly braces, but if one branch of an if is braced, then the other should be too, even if it is only a single line
- In switch statements, be sure every case ends with either a break, continue, return, or /* fall through */ comment
- Multiple declarations can go on one line, but if the line gets too long don't try to continue it in some fancy way, just start a new declaration on the next line
- Avoid declarations in all but the most complex inner blocks
- Avoid initializations of automatic variable in declarations, since they can be mildly disconcerting when stepping through code with a debugger
- Use spaces around keywords
- Use spaces around binary operators, except . and ->, for they are morally equivalent to array subscripts, and the ``punctuation'' operator ','
- Don't use spaces around unary operators, except sizeof and casts

7. Expressions and Statements

- Write infinite loops as:

```
for (;;)
        ...
not

while (1)
        ...
```

- Never return from the function main(), explicitly use exit()