SOFTWARE DOCUMENTATION GUIDELINES (draft)

FUNCTIONAL UNITS:

A specification for each functional unit will be written before the unit is coded. The document, as will be outlined below, specifies the functional unit down to the level of the precise procedural interface. This interface may change as clarifications to the design become apparent during the coding process, but a concerted effort should be made to think through the prospective interface before coding.

A component is a -.c and -.h file pair realizing a given computational objective. In many cases we expect a component to realize a data abstraction(s) and thus consist of data types and routines for manipulating instances of the type. On the other hand, some components are expected to be primarily procedural and realize some complex algorithmic processing task carried out on a set of underlying objects.

A functional unit will be presumed to consist of a collection of components, and either an API header file, or an API component pair. The API header file should contain the definitions for the functional unit's interface and no others. In the case of an API component, its -.h file is the API header file, and the -.c file is expected to contain the top-level code for every API procedure. The scope of a functional unit will remain fluid but is expected to realize some computational objective of size appropriate for an individual to complete.

Each functional unit specification will consist of the sections below. The first four sections must be written before coding begins. The remainder should be added as coding proceeds and thereafter remain up-to-date through the life cycle of the unit.

1. **Overview:**
   An overview describing the purpose and overall function of the unit.
2. **Memory Usage:**
   A paragraph or more describing the memory allocation policy for the unit, e.g. who is responsible for allocating structures (unit or its clients), who is responsible for freeing structures (unit or its clients), and what is the extent restriction (if any) on each allocated object.
3. **Interface:**
   A listing of each data type, constant, and externally visible procedure in the API header file with a description following each. For procedures, the description must clearly state the nature and purpose of each parameter and return value (if a function), any exceptions to the memory usage policy above, and the set of assumptions being made about the input to the procedure (i.e. exactly what things must be true in order for the procedure to operate correctly). Each parameter should further be classified as to its data flow usage (I, O, or IO), as described later below.

4. **Design:**
   A brief section on how the code is structured and how it achieves its function. If heavily algorithmic then the algorithm(s) must be outlined or there must be references to a relevant document containing the algorithmic description.
5. **Limitations:**
   Any restrictions or limitations on the use of the functional unit should be described.
6. **Status:**
   Current development state of the unit. If future improvements are anticipated then they should be outlined and the time when they need to be addressed should be given (either as a date or in relation to the start or completion of another task).
7. **Component Architecture and Unit Dependencies:**
   A list of the components constituting the unit should be given with a concise description of what the component does and how it relates to the other components. In addition, all other functional units upon which this unit depends should be listed and notes about any special or pivotal requirements placed on the referred unit should be included.

Functional unit specifications should further have the header used in this document as well as the trailer for author, creation date, and last modification date as at the end of this document.

Dataflow Usages:

Interface descriptions should categories their parameters and output result (if a function) as one of (I) input only, (O) output only, or (IO) input & output. The state of a parameter or result, is the value of the item and if it is a pointer, then also the value of the object pointed to, and recursively the values of anything that object points to. For example, if one passes a pointer to a linked-list then the state of the parameter is the entire list. There are then basic data flow events for a procedure parameter/result:

a) A parameter state has an input effect if and only if the state affects the computation performed by the procedure.
b) A parameter state has an output effect if and only if the state is altered by the procedure.

The classification on parameters follows.

COMPONENTS:

A component will be documented at the code level, i.e., all documentation will be within comments within the code. We assume commenting standards compliant with those set

by Software Systems. In addition, each -.h file shall be commented to include items 1. through 6. for functional unit documents. Key differences are that this documentation does not need to be written before coding, and in general, one can use an economy of words. The idea here is to leave a trail sufficient to ease the life of the next person to look at the code.
In addition, we expect each function to contain a comment header using the following template:

```
/* Function function
   Description:
        ...
   Return Value:

   Inputs:
        arg1
   Outputs:
        arg2
   Input/Outputs:
        arg3

*/
```

C-CODING STANDARDS:

The assembly team will program in ANSI C save for that one may use //-style comments. The use of object-based programs entailing small accessor functions that the compiler will automatically in-line is strongly recommended.

The assembly team will follow the naming conventions set by Software Systems with two exceptions. First, our conventions for the use of capital letters will be as follows:

a) "ALL_CAPS": enum constants and preprocessor items such as defined constants and macros.
b) "CapitalizedNames": All names that have a global extent, specifically, procedure names, global variables, and statically declared local variables.
c) "all_lower_case": Everything else.

One may use underbars (_) as desired save that they should not be the first letter of a name. This convention complies with the Software Systems standard save for category (b) names.

The second exception, is that the assembly team will not use the identifying prefix name for APIs as all assembler interfaces beyond the team are data-based. On the otherhand, should a procedural interface to some component of the assembler be requested later in the life-cycle of the assembler, we are obligated to augment our code to the required spec.

Coding and name conventions should otherwise be fully compliant with those set by

Software Systems.

AUTHORS

Gene Myers

`Created: December 3, ’98`

`Last revised: December 3, ‘98`