Coding Standards for JEOD 2.x

Requirements are mandatory; some requirements may be waived, but only under unusual circumstances. Violations must be noted in the code comments, and recorded.

1 Requirements

1.1 JEOD top-level

1.1.1.1 There are four types of model - dynamics, environment, interaction, utilities.

Every model shall fit into one of these categories.

1.1.1.2 There shall be one model per directory, and one directory per model.

A model is a cohesive collection of C++ classes.

1.1.1.3 Model directories shall contain only subdirectories.

Possible exception: makefile_overrides file

1.1.1.4 Sub-models are to be located in sub-directories of a model.

Other than location, they shall follow the same rules as for a model.

1.1.1.5 Filenames are to be suffixed correctly, and files located in the appropriate directory.

1.1.1.6 Memory shall be allocated / deallocated using the JEOD macros.

1.1.1.7 Allocated memory from any class must be released in the allocating function prior to returning, or in the class destructor using the appropriate macro:

1. Primitive (JEOD_ALLOC_PRIM) use JEOD_FREE
2. Class (JEOD_ALLOC_CLASS) use JEOD_DELETE
3. Class pointer (JEOD_ALLOC_TYPE_POINTER) use JEOD_FREE

1.2 Style (All source and header files)

1.2.1.1 Style shall be consistent within a file

1.2.1.2 Lines shall contain a maximum of 80 characters

1.2.1.3 Lines end with a non-whitespace character

1.2.1.4 Use UNIX newline character

1.2.1.5 No TAB characters

1.2.1.6 Identifiers (names) shall be meaningful

1.3 File Content

1.3.1 General

1.3.1.1 Defined items (e.g. methods) within a file should all be related somehow.

1.3.1.2 Naming conventions:

1. Preprocessor names in ALL_CAPS, starting with JEOD_
2. Type names in MixedCase
3. Class member names in lower_case_with_underscores
4. Enumeration values in MixedCase with underscores separating an optional group name from the item name.

1.3.1.3 Use of inline functions is preferred over that of macros.

1.3.1.4 The following words are forbidden:

1. __attribute__
2. #pragma
3. asm
4. goto
5. Flag
6. mutable
7. register
8. struct

9. typedef (exception for when defining a function pointer)
10. union
11. using
12. volatile

1.3.1.5 FIXME marker

1. Use during development
2. Remove before release. Must not pass code review.

1.3.1.6 Comments:

1. All code should be commented
2. Comments should be clarifying, meaningful, and informative, but not pedantic, condescending, or otherwise insulting.
3. Do not use comments to identify modifications.
4. Do not use all caps
5. Do not use pretty boxes around comments.

1.3.1.7 Method argument lists:

All methods shall have their argument list ordered such that all input variables come before input/output, which come before output.

1.3.2 Source files

1.3.2.1 Every Source File shall have a header.

See section 2.2.1 for details.

1.3.2.2 Every method contained in a file with other methods shall have its own sub-header.

See section 2.2.2 for details.

1.3.2.3 Source files should be less than 1000 lines.

1.3.2.4 Forbidden syntax

1. Ternary operator (?:)
2. Comma operator
3. One-line "if" statements (use statement blocks)
4. Implicit conversions (use, e,g, static_cast)
5. Default visibility (use public/private/protected)

6. Implicit reliance on freebie methods (use comment to make explicit)

1.3.2.5 #include statements are to be located in a single block at the head of the file.

1. Block should be divided into, and ordered by, the following categorization:
2. System headers (C, C++ or JEOD-approved extension, e.g. POSIX). Use angle brackets
3. Trick headers where absolutely necessary
4. JEOD headers, not a part of this model
   (a) specify path relative to JEOD_HOME/models, do not use relative paths.
5. Model headers (i.e. from this model directory)
   (a) Use ../include to reach header files.

1.3.2.6 Constructors:

1. must initialize all non-static member data
2. must be declared explicit if can take one argument
3. shall not share resources across instances

1.3.2.7 Destructors must free all class-allocated resources.

1.3.2.8 Converting Constructors: implicit conversion discouraged.

1.3.2.9 Copy Constructors:

1. If a copy constructor is necessary (see 1.3.3.15), but it makes no sense to have a copy of a class, make the copy constructor and assignment operator private, and do not provide an implementation.
2. Otherwise, declare public and provide a safe implementation.
3. Never make a shallow copy of an allocated resource.

1.3.2.10Methods shall have an Extended Cyclomatic Complexity of 15 or less.

1.3.2.11Methods shall not exceed 200 lines, including blanks and comments.

1.3.2.12Every local variable declaration shall be commented to indicate its purpose.

1.3.2.13Indentation and braces:

1. All code in a single model must follow a consistent style.
2. Code is to be indented 2-4 spaces per level.
3. Case statements to be indented at the same level as the switch.
4. All control statements (i.e., if, while, for, etc.) shall be followed by a compound statement (i.e., the braces are not optional).

5. The following two coding styles are allowed within JEOD:
   (a) One True Brace
      i. Except for function definitions, do not put open brace on a new line
      ii. Except for data initializations and empty bodies, the open brace should be the last thing on a line.
      iii. Except for extremely short if/else blocks, the close brace should be on a separate line from the else.
   (b) Allman
      i. The opening and closing braces are always on lines by themselves.
      ii. Braces are indented at the same level as the control statement.

1.3.2.14Initialization of data

1. If any element of an array is initialized, it should all be initialized.
2. Typically, use brace rules per 1.3.2.12, but one line is acceptable for short initializations.

1.3.2.15Switch statements:

1. Cover all cases over an enumerated value
2. Comment fall-throughs (except where multiple cases are started together)
3. End each case with a break
4. Use default to cover all impossible cases.

1.3.3 Header files

1.3.3.1 Every Header File shall contain a header.

See section 2.2.3 for details

1.3.3.2 Every class contained in a file with other classes shall have its own sub-header.

See section 2.2.4 for details.

1.3.3.3 Every Header file shall contain the following protection against problems caused by multiple inclusions:

1. <comments and blank lines only>
2. #ifndef JEOD_<UNIQUE IDENTIFIER>_HH
3. #define JEOD_<UNIQUE IDENTIFIER>_HH
4. <file body>
5. #endif

1.3.3.4 Inclusion of a Header file must not adversely affect source file compilation, including memory and resources.

1.3.3.5 #include statements are to be located in a single block at the head of the file

Block should be divided into, and ordered by, the following categorization:

1. System headers (C, C++ or JEOD-approved extension, e.g. POSIX). Use angle brackets.
2. Trick headers where absolutely necessary
3. JEOD headers, not a part of this model
    1. specify path relative to JEOD_HOME/models, do not use relative paths.
4. Model headers (i.e. from this model directory)
    1. do not use ../include to reach files in this directory.

Exceptions made for

1. Trick: Where forward-declaration necessitates early definition, in which case #include should be encapsulated in a #ifdef TRICK_VER block.
2. Inline functions: Where inline functions are defined in a separate header file, that file should be #include 'd at the END of the class-defining header file.

1.3.3.6 Class outline:

1. JEOD_MAKE_SIM_INTERFACES (ClassName)
    (a) required if there are private data members, otherwise optional
2. Members - order each member group in public/protected/private order
    (a) Static data
    (b) Static methods
    (c) Instance data*
    (d) Instance methods.*

* Deliberate omission of these requires a comment to that effect.

1.3.3.7 All member data need a Trick comment, including units.

1.3.3.8 All regular member functions are to be declared, but not defined in this file.

1.3.3.9 All inline functions are to be defined in either the class-defining file, or in a separate inline header file.

1.3.3.10 Unimplemented pure virtual functions need comments.

1.3.3.11 Never overload a virtual method

1.3.3.12 Never override a non-virtual method

1.3.3.13 Never overload an operator (except operator =)

1.3.3.14 Visibility of virtual methods:

1. If intent is to provide generic interface, make public/protected
2. If intent is to limit use to non-virtual methods defined by base class, make private.

1.3.3.15 If defining a destructor, copy constructor, or assignment operator, provide all 3 (see 1.3.2.9).

1.3.4 Data (Default data) files

1.3.4.1 All default data files shall contain a header.

See section 2.2.5 for details.

1.3.4.2 Data are to be presented in Trick default data format, not Modified data format.

1.4 Compilation

1.4.1.1 Must compile clean with the following flags:

1. -Wall
2. -Wold-style-cast
3. -Woverloaded-virtual

2 Trick header formats

2.1 Entries

This section provides the format for each of the entries in a file header.

Purpose:

(This is a description of the thing being described.)

Programmers:

(((name) (organization) (date) (ticket #) (explanation)))

For releases, a brief history. One entry (max) per release. No change = no entry

In development, more detail allowed, but must be compressed for release

Format: A list of 5 item lists.

Library dependencies:

((file_name1.o) (file_name2.o))

Lists the files needed to make a simulation compile.

Use the Unix 'nm' tool to determine dependencies.

Format: A list of single items.

Assumptions and limitations:

((Assumption 1: words))

Assumptions are things that are assumed to be true; exceptions will result in unpredictable consequences.

Limitations are things are assuredly true; the code will fail on exceptions.

Format: A list of single items

Reference:

(((author) (title) (publisher) (date) (pages) (ISBN #)))

Specifying references in source code is not necessary; references must be specified in documentation.

Exception: References for data files are a good idea

Format: A list of 6 item lists.

Class:

(Trick job class name)

This is the Trick class, not the C++ class and pertains only to methods

Do not specify if the class is N/A

2.2 Specific Instances

2.2.1 Source Code File Header

The main file header, at the top of the .cc file:

1. Purpose (of the file, not of the methods)
2. Programmers
3. Library Dependencies
4. Reference (if appropriate)
5. Assumptions and Limitations (applied to the file, not the methods)

2.2.2 Source Code Method Header

A sub-header preceding each method in a .cc file:

1. Purpose (of the method)
2. Assumptions and Limitations that are specific to this method.
3. Reference (if appropriate)
4. Class (if appropriate)

2.2.3 Header File Header

The main file header, at the top of the .hh file:

1. Purpose (of the file, not of the classes)
2. Programmers
3. Library Dependencies

2.2.4 Class Header

A sub-header preceding each declared class:

1. Purpose (of the class).

2.2.5 Data File Header

1. Copyright (if appropriate)
2. Purpose (of the file, not of the methods)

3. Programmers
4. Reference (if appropriate)

2.2.6 S_define Header

The S_define file does not require a trick-readable header, so does not require exactly the same format. Nevertheless, it should contain the following:

1. Purpose
2. A list of defined objects to be found in the S_define
3. Programmers (aka Authors)