BES III Offline C++ Coding Specification

09/03/2004
Version 1.0

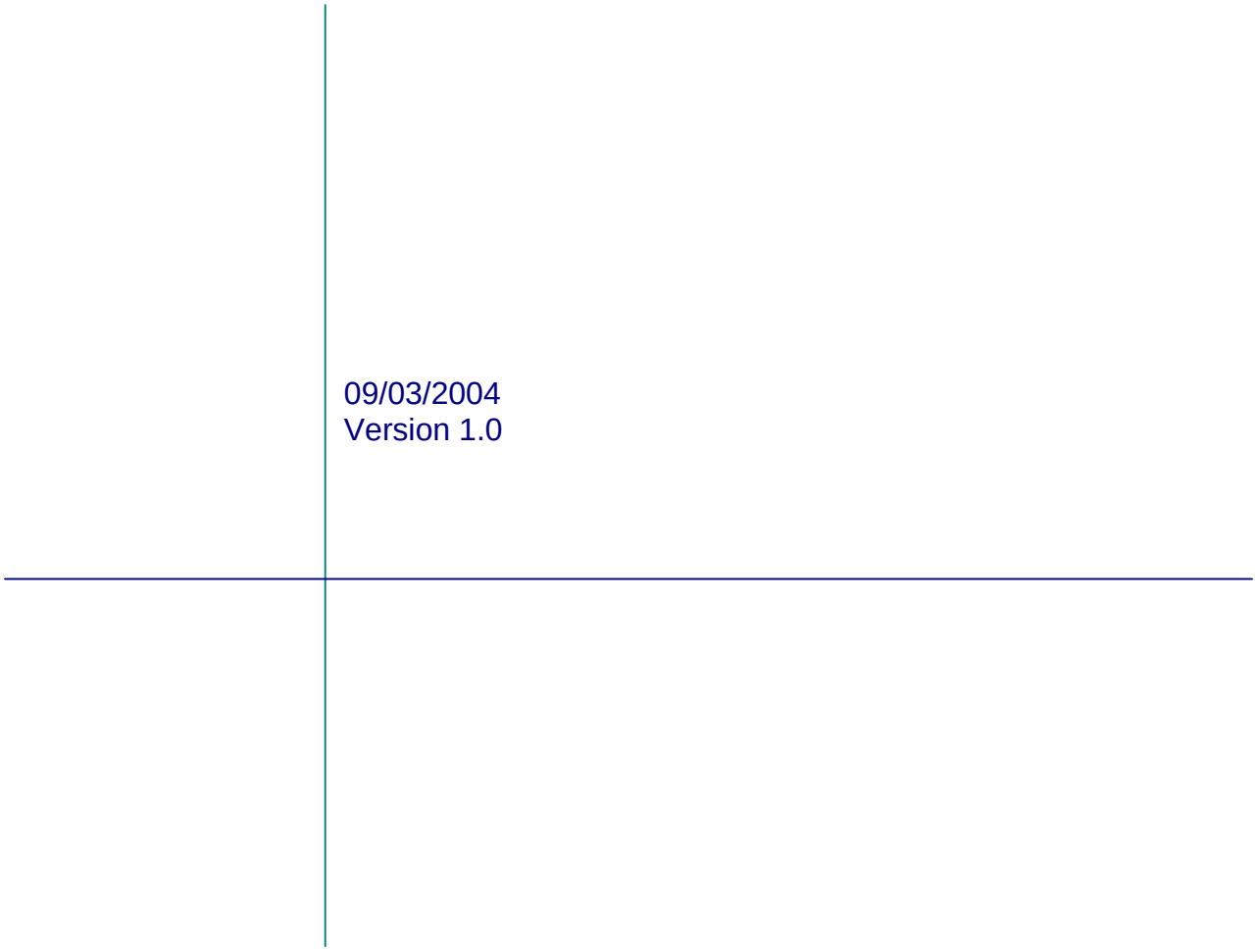

Table of contents

1. Naming

1.1. Naming of Files (NF)

NF1 The name of the header file must be the same as the name of the class it defines, with a suffix ".h" appended. (REQUIRED)

NF2 The name of the implementation file must be the same as the name of the class it implements, with a suffix ".cxx" appended. (REQUIRED)

1.2. Illegal No-recommended Naming (NI)

NI1 Do not create very similar names. (RECOMMENDED)

NI2 Do not use identifiers that begin with an underscore. (REQUIRED)

1.3. Naming Conversions (NC)

NC1 Use prefix "m_" for private attributes (i.e. data members) in each class. (REQUIRED)

NC2 Use prefix "s_" for private static attributes (i.e. data members) in each class. (RECOMMENDED)

NC3 Use namespace to avoid name conflicts between classes. (REQUIRED)

NC4 Start class names, typedefs and enum types with an uppercase letter. (REQUIRED)

NC5 Start names of variables and functions with a lowercase letter. (REQUIRED)

NC6 In names that consist of more than one word with the words together, and start each word that follows the first one with an upper case letter. (RECOMMENDED)

NC7 All package names in the release must be unique, independent of the package's location. (REQUIRED)

2. Coding

2.1. Organizing the Code (CO)

CO1 Header files must begin and end with multiple-inclusion protection. (REQUIRED)

CO2 Use forward declaration instead of including a header file, if this is sufficient. (RECOMMENDED)

CO3 Each header file must contain one class (or embedded or very tightly coupled classes) declaration only. (REQUIRED)

CO4 Implementation files must hold the member function definitions for one class (or embedded or very tightly coupled classes) as defined in the corresponding header file. (REQUIRED)

2.2. Control Flow (CF)

CF1 Do not change a loop variable inside a for loop block. (REQUIRED)

CF2 All switch statements must have a default clause. (REQUIRED)

CF3 Each clause of a switch statement must end with a break. (REQUIRED)

CF4 An “if statement” which does not fit in one line must have brackets around the conditional statement. (REQUIRED)

CF5 Do not use goto. (REQUIRED)

2.3. Object Life Cycle (CL)

2.3.1. Initialization of Variable and Constants

CL1 Declare each variable with the smallest possible scope and initialize it at the same time. (RECOMMENDED)

CL2 Declare each type of variable in a separate declaration statement, and don’t declare different types (e.g.

init and int point) in one declaration statement. (REQUIRED)

CL3 Do not use the same variable name in outer and inner scope. (REQUIRED)

2.3.2. Constructor Initializer Lists

CL4 Let the order in the initialization list be the same as the order of declarations in the head file: first the base classes then the data members. (RECOMMENDED)

2.4. Conversions (CC)

CC1 Use explicit rather than implicit type conversion. (REQUIRED)

CC2 Use the new cast operators (dynamic_cast or static_cast) instead of C-style casts. (REQUIRED)

CC3 Do not convert const objects to non-const. (REQUIRED)

2.5. The Class Interface (CI)

CI1 Head files must contain no implementation except for small functions to be inlined. (REQUIRED)

CI2 Pass an un-modifiable argument by value only if it is of built-in type or small. Otherwise, pass the argument by const reference or by pointer to const. (RECOMMENDED)

CI3 Have operator = return a reference to *this. (REQUIRED)

CI4 Declare a pointer or reference argument, passed to a function, as const if the function does not change the object bound to it. (RECOMMENDED)

CI5 The argument to a copy constructor and to an assignment operator must be a const reference. (REQUIRED)

CI6 In a class method, do not return pointers or non-const references to private data members. (REQUIRED)

CI7 Declare as const a member function that does not affect the state of the object. (REQUIRED)

CI8 Do not let const member functions change the state of the

program. (REQUIRED)

CI9 Use function overloading only when methods differ in their argument list, but the task performed is the same. (REQUIRED)

2.6. new and delete (CN)

CN1 Match every invocation of new with one invocation of delete in all possible control flows from new. (REQUIRED)

CN2 A function must not use the delete operator on any pointer passed to it as an argument. (REQUIRED)

CN3 Do not access a pointer or reference to a deleted object. (REQUIRED)

CN4 After deleting a pointer, assign it to zero. (REQUIRED)

2.7. Static and Global Objects (CS)

CS1 Do not declare global variables. (REQUIRED)

2.8. Object-Oriented Programming (CP)

CP1 Do not declare data members to be public. (REQUIRED)

CP2 If a class has at least one virtual method then it must have a public virtual destructor. (REQUIRED)

CP3 Always re-declare virtual functions as virtual in derived classes. (REQUIRED)

CP4 Avoid multiple inheritance except for abstract interfaces. (RECOMMENDED)

2.9. Assertions and Error Handling (CE)

CE1 Make sure assertions are not compiled in the production releases. (REQUIRED)

CE2 Use the standard error printing facility for informational messages. Do not use cerr and cout. (RECOMMENDED)

2.10. Parts of C++ to Avoid (CA)

CA1 Do not use malloc, acalloc, realloc and free. Use new and delete instead. (REQUIRED)

CA2 Do not use functions defined in stdio. Use the iostream functions in their places. (RECOMMENDED)

CA3 Do not use NULL to indicate a null pointer, use the integer constant 0. (REQUIRED)

CA4 Do not use const char* or built-in array "[]", use std::string instead. (RECOMMENDED)

CA5 Do not use union types. (REQUIRED)

CA6 Do not use the keyword struct. (RECOMMENDED)

CA7 Do not declare your own typedef for booleans. Use the bool type of C++ for booleans. (REQUIRED)

CA8 Avoid pointer arithmetic. (REQUIRED)

2.11. Readability and Maintainability (CR)

CR1 Avoid duplicated code. (RECOMMENDED)

CR2 If you use a Typedef, it should be declared private or protected. (REQUIRED)

CR3 Code should be written to use standard BES III units for time, distance and energy, etc. (RECOMMENDED)

3. Style

3.1. General Aspects of Style (SG)

SG1 The public, protected and private sections of a class must be declared in the order. Within each section, nested types (e.g. enum or class) must appear at the top. (REQUIRED)

SG2 Keep the ordering of methods in the header file and in the source files identical. (REQUIRED)

SG3 Limit line length to 120 character positions (including white spaces). (REQUIRED)

SG4 Include meaningful dummy argument names in function declarations. Any dummy argument names used in function declaration must be the same as in the definition. (REQUIRED)

SG5 The code should be properly indented for readability reason. (RECOMMENDED)

SG6 Do not use spaces in front of [], (), and to either side of . and ->, in references to functions or arrays. (REQUIRED)

3.2. Comments (SC)

SC1 Use "//" before class/method/field declarations and comments in method bodies. (RECOMMENDED)

4. Edit History

Revision	Date	By	Description
1.0	09/03/2004	Li Weidong	First import