Proposed Coding Standards Changes in Winter 2012 Review

The TCT met 2011-10-05 to dicsuss this set of proposed changes to the coding and documentation standards.

Attendees:

- K-T (in for Gregory)
- Ray
- Robert
- Russell (also voting for Robyn)
- Jim Bosch (not voting)

Opening remarks:

- K-T pointed out that these standards exist to make our code more uniform, so others can read and understand it, and to improve the quality of the code.
- Russell reminded folks that the long-term plan is to enforce the required standards. The question came up of how much code is in compliance; nobody knew the answer.

We then voted on the proposals. The TCT decisions are listed below, after each proposal.

Finally, the action items are:

- Russell will update the relevant standards
- Jim Bosch will update the emacs config file for the new function argument format, and will see if he can do the same for the vi config file

---

- Namespaces in source files: we should use namespace blocks in source files, and prefer unqualified (or less-qualified) names within those blocks over global-namespace aliases (from JFB).
    - PAP agrees
    - Russell Owen: I disagree. I find it much harder to figure out where symbols come from in such code (which symbols are defined locally vs. imported from some other file. However, it is clear that many programmer prefer this style so I'm willing to consider allowing it. In my opinion making it the recommended style is going too far.

      The TCT voted to allow namespace blocks in source code (.cc) files. There was enough momentum for changing this to a "should" that we may wish to revisit this at some point, but meanwhile all existing code that uses the old standard is still compliant.

- Pure-Python modules should use relative imports to access other modules within the same package (after upgrade to 2.7.x is complete). This is the `from . import xxx` syntax (from JFB).
    - PAP disagrees: Modules are difficult to `reload` from the Python command-line when that form is used.
    - Russell Owen: I agree with the proposal, though reluctantly. Relative import is the modern recommended way to import package-local modules to avoid name collision. However, I find the syntax ugly.

- RayPlante: I tend to disagree, but this rule needs some justification/motivation (perhaps with explanation as to the importance of 2.7.x). I have found that Python can get confused if a sub-module name matches a top-level package name; maybe this is behavior has changed in 2.7.*

  The TCT accepted this proposal (as a "should"). Robert would like to revisit the topic of importing names in init.py files at a future date.

- Functions should only take arguments by shared_ptr when a reference or const reference argument cannot be used. Examples of when a shared_ptr may be used include when the pointer itself may be reset, when a null pointer is considered a valid input, and when the pointer (not the pointee) will be copied and held after after the function returns (as in a constructor or member function setter). In all other cases, reference or const reference arguments should be used. Motivation: it is difficult and sometimes expensive to create a shared_ptr from a reference or plain value, so a shared_ptr should not be required to call functions unless necessary.

  The TCT accepted this proposal, though Russell plans to go back to the TCT for clarification as to whether this applies to all smart pointers. This is apparently a new standard. Robert would like a future discussion on use of shared_ptr.

- Russell Owen: 6-6 I wish to change the alternate form for long argument lists or argument lists containing Doxygen annotation as follows (note: in Python the last argument should have a trailing comma, to avoid editing mistakes when changing the argument list).

```
returnType foo(
    type1 argument1,   ///< Doxygen documentation comment (possibly long)
    type2 argument2,   ///< more Do
    ....
    type3 lastArgument
);
```

  This changes two of the current rules for the alternate form:

- 4 spaces indent instead of 8. This gives more room for the Doxygen comment and more unity with other standards (an indent of 4 is used everywhere else).
- The closing parenthesis is aligned with the start of the function, closing a block. Advantages:
  - It is easier to see the closing parenthesis (it is just below the white space of the last argument, instead of buried in some odd level).
  - It is easier to type in most editors
  - It is more consistent with other closures

  I'll even propose that this be the preferred form, but I'll be happy with its being the alternate form as long as I can use it.

- Russell Owen: I hate the capitalization of acronyms (Wcs instead of WCS). I'm not proposing we change it (I sure don't want to fix all the existing code), I just wanted to get that off my chest.
  - JFB: I hate it too! But I also admit it seems too trivial to bother fixing at this point.

  This TCT accepted this proposal

Mike Jarvis sent email 2010-09-13 that asks for a number of other changes. I've transcribed most of it here (typing from a fax):

- 3-16 (use "compute"): heavy-handed. Suggest only requiring "compute" for methods that return a single value, i.e. things that act like a get method but which need to do a calculation rather than just return a value that is already stored.
    - ♦ Russell Owen: I agree.

      The TCT voted to delete rule 3-16

- 3-23: I think iterators should always be called it, possibly with a suffix to indicate which iterator (e.g. id1, itStarList). i, j, k should always be integers imo.
    - ♦ Russell Owen: I agree, though I propose "iter" as clearer and less ambiguous than "it". I personally use <name>Iter all over the place. I propose that we suggest iterators be called iter or <name>Iter or iter<name> -- leave the order up to the programmer. I'll also note that it is often impractical to put variables in loop scope (unfortunately, and auto may help).
    - ♦ JFB: Disagree. Iterators can often have more descriptive meanings, too, and I don't think we should have a rule about what to call them at all. It's almost always clear from context. Any rules on iterators will involve changing a lot of deeply-buried code to bring it up to conformance.
    - ♦ Russell Owen: fine. Let's vote on dropping the rule. An alternative is to keep the part about making loop variables local, but I'm not convinced we need a rule for such minutiae.

- 3-24: I would add to this the option of having a noun before the verb to indicate a component of the object. E.g. my Matrix class in TVM has the method divIsSet()...if m.divIsSet() seems clearer than...(RO's version) m.isDivSet()

      The TCT voted to handle this by changing 3-4 to: Names representing methods or functions should naturally describe the action of the method and will usually start with a verb.

- 3-30: I find the Error suffix to be usually more appropriate than Exception. It's more like the English language version of what you mean, rather than the code version of how you are implementing what you mean. To me ending with Exception is like naming the Image class "ImageClass?". You wouldn't do that because Class is just how you are implementing the concept in C++, not what the concept is. Likewise exceptions are (usually) some kind of error.
    - ♦ Russell Owen: I agree and this matches the standard exceptions in C++ and Python
    - ♦ JFB: Agree.

      The TCT voted for the following proposal: The name of exceptions that indicate an error condition must ends with Error. (Note that this applies to all LSST standard exceptions.)

- 3-33: I don't like decl for declination; it looks like an abbreviation for...declare... But...it only applies to values exposed to a database, so it doesn't really affect me.

      The TCT voted to delete all section 3.3 rules except for rule 3-34, because they deal with databases, not C++ code, and SQL limitations should not leak into C++ code. Rule 3-34 is useful for C++ names.

- 4-6: I'd rather 80 columns.

- Russell Owen: I hope we don't change this. Making existing wide code shorter is not fun unless you want to write an automatic script. There are plenty of us (including me) who find 110 unpleasantly short. I can't imagine we'd revisit this.
- JFB: I think we're at 120 now, and I like it there.

  The TCT showed great restraint in not revisiting this issue.

- 4-11: While generally fine there are some cases where it makes sense to have an include file at the end. E.g. my scheme for doing template instantiations with a list of types that can be specified in single place.

  The TCT voted to change this from a MUST to a SHOULD.

- 5-10 const: Hate hate hate this one. Number 1 gripe. I find the const much harder to see when it comes after the type. And the argument about reading from right to left is not convincing. For a pointer, the const should either go before the type (in which case the thing pointed to is const) or after the * (in which case the pointer is const). The only place that is ambiguous and requires knowing the C++ convention is when the const goes between the type and the *. In other words, "int const* x" is confusing and that is what is mandated by the standard....
  - Russell Owen: I agree it's harder to spot the const, but I find it easier to parse and would rather not change it.
  - JFB: After some time adjusting, I like the current form better and support keeping it.

  The TCT was sympathetic, but chose to retain the current rule.

- I didn't see anything that says Doxygen comments go in the .cc file. But I could easily have missed it. If so, I don't like that rule either.
  - Russell Owen: the rules are here DocumentationStandards (thanks to K-T for the reference). I will add a link to this in the coding standards. The argument for our current standard was put forth by Robert Lupton. He claimed it is easier to keep the documentation current and consistent with the code if it lives with the code. I was used to having it in the header file, so it came as something of a shock. I find when I write or modify code that I agree with Robert, but when I try to read other people's code I prefer to have the documentation in the header files. Using the Doxygen documentation helps (though we've had some issues keeping it current on Trac). This one would be a mess to change, so I encourage you to think carefully if you really feel it is necessary to change it.
  - RayPlante: I feel quite strongly that the documentation should go in the header for the following reasons:
    - The biggest reason for me is somewhat philosophical: What a function does should be thought of as part of its declaration, just like its signature; it establishes the contract with the caller. Others have argued that if you are changing the implementation in the .cc file, you can change the documentation in the same file. I think this is a bad practice. One should establish the essentials of the documentation when designing the class--i.e. when (or before) you create the .h file. The job of the .cc file is to fulfill the contract of the .h file.
    - This follows on above: if you are browsing a class via its source, you are more likely to read the .h file first to browse the interface; this is where the documentation should be.
    - This is the most practical: there is some documentation that can only appear in the .h, most important being the class summary documentation. On the other hand, there is no important documentation that *must* be in the .cc file. It's better to keep it in one

place.

The TCT voted to accept this proposal. Comments now belong in header files. This is a major change. Robert would like to revisit the use of @param at some point; we ran out of time before we could discuss it in any length.

- Also, in general, I would prefer that the style strictures basically only apply to the .h files. I don't see any downside in letting people code in whatever style they prefer in the .cc files. The vast majority of users should never need to look into the .cc files at all, so it shouldn't matter if they don't conform to the official LSST style. In fact, just adopting this single change would make almost all of my gripes go away.
    - Russell Owen: I disagree. I think our code will be much more maintainable if we have common standards. This code may be in use for many years and the code we write may be rewritten by many others. That said, perhaps we could relax some particular standards for .cc files, though I'm not sure how this would affect standards checking.

      The TCT was generally not in favor of this, but did not formally vote on it.

Jim Bosch sent a reply email 2010-09-13 with this list, in decreasing order of importance to him.

- 3-6: namespace in source files (see above)

- ?: I like Doxygen in the header files instead of source files. I primarily read the comments in the code rather than the Doxygen output and that's a lot easier in the headers. (Discussed above under Mike Jarvis' comments).

- 6-6: I'd like to indent function parameters however it makes sense for the particular function. (I particularly dislike close-parens floating out in the middle of nowhere on their ow line; I"d prefer for them to behave like close-braes in matching the indent of the outer scope).
    - JFB: I like Russell's proposed alternative form above.

- 5-17: "break" and "continue" are necessary and important. Of course, they are generally only present when an algorithm is intrinsically a little hard to follow, but they should not be blamed for that fact = IMHO code that tries to avoid them would be even harder to follow.
    - Russell Owen: I agree. I propose we delete this rule.

      The TCT voted to delete rule 5-17.

- 3-16: "compute" on everything just seems to restrictive. (See also Mike's detailed proposal above).

- 3-20: I like just pluralizing names for containers, but I'm not super upset about it.
    - Russell Owen: I was also used to using "s" and "List" seems too specific for a wide class of collections. But I agree with the rule's motivation: names that differ only by a trailing "s" can be hard to distinguish when reading code. I propose we live with it.

      The TCT did not take up this issue

- I kind of like Mike's "anything goes in .cc files" proposal, at least if the documentation comments are in the headers. I think we'd all end up having pretty close to the standards anyhow, just because we wouldn't make source different from headers unless we really hated something or we copied it from elsewhere and it's a lot of extra work to make it standards conforming.