Proposal for Changes to the DM C++ Coding Standards

References

- C++ Coding Standards - statement of LSST DM C++ Coding Standards.

Yet Another New Rule 1 (Unused exception variables must be unnamed)

Unused exception variables must be unnamed.

```
    try {
    } catch (ExceptionClass &) {      // OK
    };
```

Although C++ allows omission of the variable name, some compilers generate warnings if a named variable is not accessed. This Rule will reduce unnecessary compiler warnings.

Sponsor

RHL

Justification

I've been compiling afw and meas/algorithms with the intel compiler on my os/x +10.6 laptop. I don't have all the issues resolved (I have one bug report into +intel and will file another when I can localise the problems to a test case).

Anyway, our code generates LOTS of warnings. Some are silly and can be disabled +(e.g. warning about order of evaluation in "cout << foo->getX() << " " << +foo->getY() << endl;" which is true but unhelpful), but many are real. Some of +the real ones are in boost headers, but some are in our code.

I'd like the TCT to declare that some of the doubtful coding practices are not +permitted by LSST.

Action Requested

Add this new Rule to the DM C++ Coding Standards.

Discussion

in lsstdata mail group

Yet Another New Rule 2 (Unused method and function arguments must be unnamed)

Unused method and function arguments must be unnamed.

```
    void MyDerivedClass::foo( double /* scalefactor */ ) { // OK
    };

    void MyDerivedClass::foo( double ) { // OK
    };
```

This is common in template specializations and derived methods, where a variable is needed for some cases but not all. In order to remind the developer of the significance of the missing parameter, an in-line C comment may be used.

Although C++ allows omission of an unused argument's name, some compilers generate warnings if a named argument is not accessed. This Rule will reduce unnecessary compiler warnings.

Sponsor

RHL

Justification

I've been compiling afw and meas/algorithms with the intel compiler on my os/x +10.6 laptop. I don't have all the issues resolved (I have one bug report into +intel and will file another when I can localise the problems to a test case).

Anyway, our code generates LOTS of warnings. Some are silly and can be disabled +(e.g. warning about order of evaluation in "cout << foo->getX() << " " << +foo->getY() << endl;" which is true but unhelpful), but many are real. Some of +the real ones are in boost headers, but some are in our code.

I'd like the TCT to declare that some of the doubtful coding practices are not +permitted by LSST.

Action Requested

Add this new Rule to the DM C++ Coding Standards.

Discussion

in lsstdata mail group

Yet Another New Rule 3 (A class or struct definition must explicitly declare the privacy qualifier of its base classes)

A class or struct definition must explicitly declare the privacy qualifier of its base classes.

```
struct derived : public base {};
class d : private b {};
```

Although C++ provides the above definitions as defaults, some compilers generate warnings if explicit privacy qualifiers are not specified. This Rule will reduce unnecessary compiler warnings.

Sponsor

RHL

Justification

I've been compiling afw and meas/algorithms with the intel compiler on my os/x +10.6 laptop. I don't have all the issues resolved (I have one bug report into +intel and will file another when I can localise the problems to a test case).

Anyway, our code generates LOTS of warnings. Some are silly and can be disabled +(e.g. warning about order of evaluation in "cout << foo->getX() << " " << +foo->getY() << endl;" which is true but unhelpful), but many are real. Some of +the real ones are in boost headers, but some are in our code.

I'd like the TCT to declare that some of the doubtful coding practices are not +permitted by LSST.

Action Requested

Add this new Rule to the DM C++ Coding Standards.

Discussion

in lsstdata mail group

Recommendation Level Restatement

The following is the restatement of the Recommendation Levels as taken as an Action Item from TCT Meeting on 20 January 2010.

1.2 Recommendation Importance

In the guideline sections, the terms 'must', 'should' and 'may', amongst others, have special meaning. We will use the spirit of the IETF organization's RFC 2199 Reference(10) definitions; they are paraphrased below for our purpose:

REQUIRED

- **REQUIRED** means that the Rule is an absolute requirement of the specification. The developer needs to petition the DM TCT to acquire explicit approval to contravene the Rule.
- **PROHIBITED** is the opposite of **REQUIRED**.

MUST or SHALL

- **MUST** and **SHALL** mean that there may exist valid reasons in particular circumstances to ignore a particular Rule, but the full implications must be understood and carefully weighed before choosing a different course. The developer needs to petition the lead developer to acquire explicit approval to contravene the Rule.
- **MUST NOT** and **SHALL NOT** are their opposites.

SHOULD or RECOMMENDED or MAY

- **SHOULD**, **RECOMMENDED** and **MAY** mean that there are valid reasons in particular circumstances to ignore a particular Rule. The developer is expected to personally consider the full implications before choosing a different course.

- **SHOULD NOT**, **NOT RECOMMENDED** and **MAY NOT** are their opposites.