Proposed source code formatting

Kyle J. Knoepfel
LArSoft coordination meeting
3 July 2018

Code development among many people…

- *…is good:*
  - Many hands make for light work.
  - Better if more eyes are on the code.

Code development among many people…

- *…is good:*
  - Many hands make for light work.
  - Better if more eyes are on the code.

- *…can be frustrating:*
  - Different knowledge levels
  - Different coding styles
  - Different formatting conventions

Code development among many people…

- *…is good:*
  - Many hands make for light work.
  - Better if more eyes are on the code.

- *…can be frustrating:*
  - Different knowledge levels
  - Different coding styles
  - Different formatting conventions

```
Person(const std::string &name,
       const unsigned age) :
  name_{l}, age_{age}
{}

Person(std::string const& name,
       unsigned const age)
  : name_{l}
  , age_{age}
{}
```

Code development among many people…

- *…is good:*
  - Many hands make for light work.
  - Better if more eyes are on the code.

- *…can be frustrating:*
  - Different knowledge levels
  - Different coding styles
  - Different formatting conventions

```
Person const* p{nullptr};
Person const *p{nullptr};
Person const * p{nullptr};
const Person *p{nullptr};
const Person* p{nullptr};
```

Code development among many people…

- *…is good:*
  - Many hands make for light work.
  - Better if more eyes are on the code.

- *…can be frustrating:*
  - Different knowledge levels
  - Different coding styles
  - Different formatting conventions

```
Person const* p{nullptr};
Person const *p{nullptr};
Person const * p{nullptr};
const Person *p{nullptr};
const Person* p{nullptr};
```

Shared software projects benefit from having a common code format.

Should you adopt a common format?

- **Pro**:
  - It's what professional C++ libraries do
  - It gives a polished look to code
    - Boosts confidence in your software product
  - It makes code diffs much easier to understand

- **Con**:
  - You may not like the format
  - Your text editor may have trouble with some formatting decisions

- I argue the benefits outweigh the drawbacks.

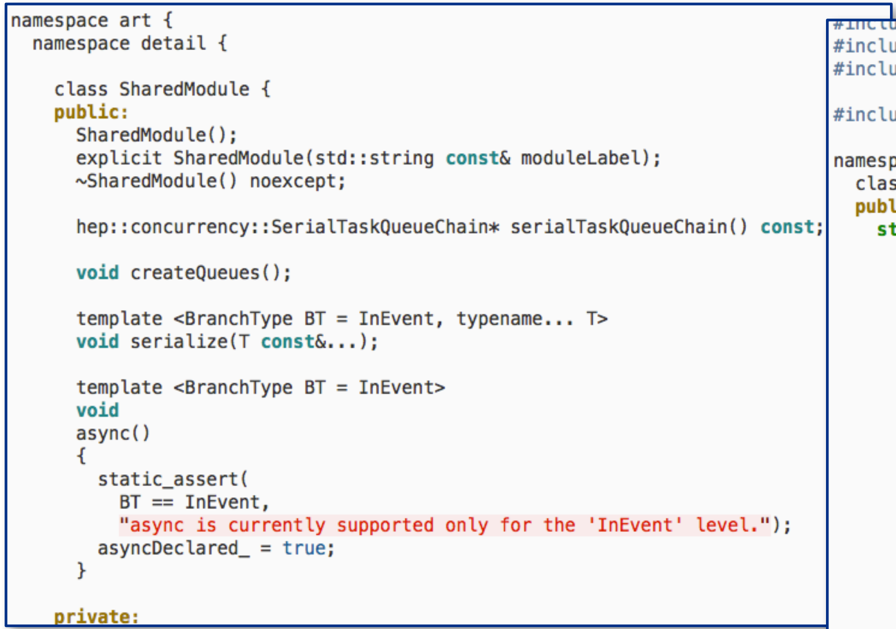
art team (c. Oct. 2017)

- The *art* team decided to adopt a common C++ code format last year:
  - Proposed formatting tool is clang-format (primarily whitespace reorganization)
    - http://clang.llvm.org/docs/ClangFormat.html
  - Goal was to find something that is usable, not necessarily something we all love
  - Chosen style:
    - Concise but clear code
    - Support C++ idioms
    - Demonstrate our knowledge and use of modern C++
    - Should not be constrained by limitations of text editors—they will get better

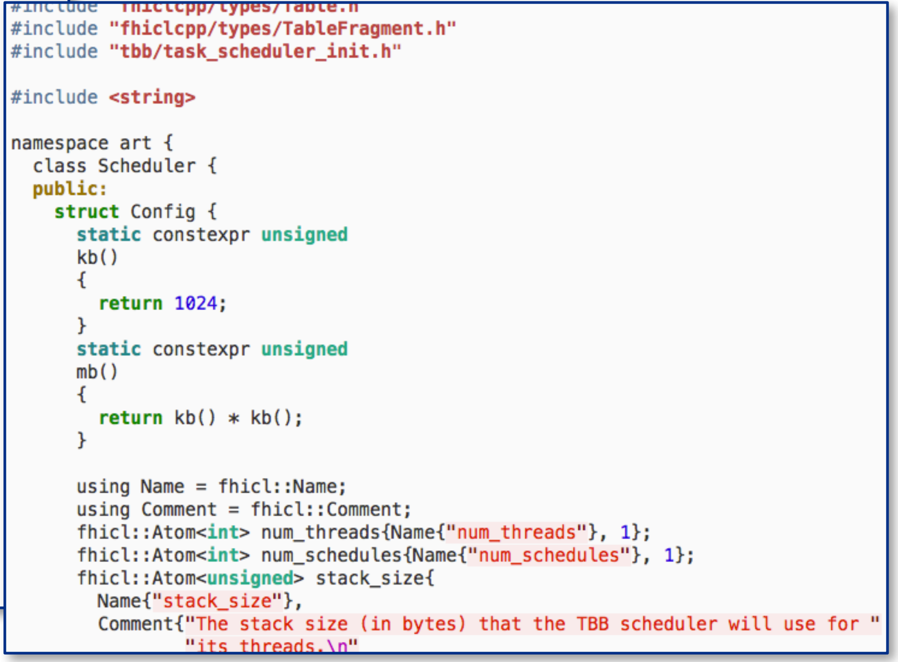
- It has been successful. We do not uniformly like all of the formatting, but the commonality in whitespace usage has been positive.

Examples

- See any of the code in the *art* redmine repository:
  - https://cdcvs.fnal.gov/redmine/projects/art/repository/show/art?rev=ART_SUITE_v3_00_00



```
namespace art {
  namespace detail {

    class SharedModule {
    public:
      SharedModule();
      explicit SharedModule(std::string const& moduleLabel);
      ~SharedModule() noexcept;

      hep::concurrency::SerialTaskQueueChain* serialTaskQueueChain() const;

      void createQueues();

      template <BranchType BT = InEvent, typename... T>
      void serialize(T const&...);

      template <BranchType BT = InEvent>
      void
      async()
      {
        static_assert(
          BT == InEvent,
          "async is currently supported only for the 'InEvent' level.");
        asyncDeclared_ = true;
      }

    private:
```



```
#include "fhiclcpp/types/Table.h"
#include "fhiclcpp/types/TableFragment.h"
#include "tbb/task_scheduler_init.h"

#include <string>

namespace art {
  class Scheduler {
  public:
    struct Config {
      static constexpr unsigned
      kb()
      {
        return 1024;
      }
      static constexpr unsigned
      mb()
      {
        return kb() * kb();
      }

      using Name = fhicl::Name;
      using Comment = fhicl::Comment;
      fhicl::Atom<int> num_threads{Name{"num_threads"}, 1};
      fhicl::Atom<int> num_schedules{Name{"num_schedules"}, 1};
      fhicl::Atom<unsigned> stack_size{
        Name{"stack_size"},
        Comment{"The stack size (in bytes) that the TBB scheduler will use for "
                "its threads.\n"
```

How would this be done in LArSoft?

- Each LArSoft repository with C++ code would have a .clang-format file.
  - The format is highly customizable.
  - I suggest using *art*'s as a starting point
- Automatic ways of applying the formatting are non-trivial.
  - Within *art*, we frequently have commits that say “Apply clang-format.”
- Applying the formatting is done via:

```
setup clang v5_0_1
cd <repo>
for_all_cpp_files | while read cppfile
do
  clang-format -i -style=file $cppfile
done
```

- Best way to do this is to integrate it with your editor.

General recommendations

- Even if LArSoft does not adopt a common format, please:
  - Configure your editor to replace tabs with whitespaces
  - Remove trailing whitespace from lines.
  - Make sure each file ends with a newline character.

Thanks