Integrating C++20 Features into the Art Framework

Zachary Evans, ORISE MSIIP Intern with the Art framework group at FNAL

HSF Frameworks Group

25 October 2023

Why would we change what works?

- Faster and/or uses less memory
- More compact code that's better at representing complex abstractions
- Improves ease of use, user-visible diagnostics
- Easier to maintain codebase

```
//=============================================================
// Check if sequence type
//
template <typename Container>
struct is_sequence_type : std::false_type {};

template <typename ARG, std::size_t SZ>
struct is_sequence_type<std::array<ARG, SZ>> : std::true_type {};
template <typename... ARGS>
struct is_sequence_type<std::tuple<ARGS...>> : std::true_type {};
template <typename... ARGS>
struct is_sequence_type<std::vector<ARGS...>> : std::true_type {};

template <typename T>
inline constexpr bool is_sequence_type_v{is_sequence_type<T>::value};
```

```
template <typename T>
concept is_fhicl_sequence =
  requires { typename T::fhicl_sequence_tag; };
```

```
//=====================================================================
// floating-point
template <class T>
std::enable_if_t<std::is_floating_point_v<T>>
fhicl::detail::decode(std::any const& a, T& result)
{
  ldbl via;
  decode(a, via);
  result = via; // boost::numeric_cast<T>(via);
}
```

```
//=====================================================================
// floating-point
template <std::floating_point T>
void
fhicl::detail::decode(std::any const& a, T& result)
{
  ldbl via;
  decode(a, via);
  result = via; // boost::numeric_cast<T>(via);
}
```

Concepts & Constraints

- Constraints are logical operations that specify requirements for correct usage of templates
- Concepts are reusable constraints with higher flexibility in terms of compounding constraints ("subsummation")
- More concise and specific diagnostic
- Simpler code: Much *much* less metaprogramming
- Straightforward template accessibility for users
- Constrain template argument type, member function availability, more

```
namespace cet::detail {
  template <class T>
  concept cet_exception =
    std::is_base_of_v<cet::exception, std::remove_reference_t<T>>;

  template <typename T>
  concept Streamable = requires(std::ostream os, T value) { os << value; };
} // cet::detail
```

Concept definitions

```
template <detail::cet_exception E, class T>
E&&
operator<<(E&& e, T const& t)
{
  e.append(t);
  return std::forward<E>(e);
}
```

Using Concepts as Constraints

Comparison of Diagnostic

```
/home/greenc/work/cet-is/build/mrb-art-3.13.02/srcs/cetlib/cetlib/test/bad_hypot_t.cc: In function ‘int main()’:
/home/greenc/work/cet-is/build/mrb-art-3.13.02/srcs/cetlib/cetlib/test/bad_hypot_t.cc:11:13: error: no matching function for call to ‘hypot(std::__cxx11::basic_string<char>, std::__cxx11::basic_string<char>)’
   11 |   cet::hypot("lorem"s, "ipsum"s);
      |   ~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~
In file included from /home/greenc/work/cet-is/build/mrb-art-3.13.02/srcs/cetlib/cetlib/test/bad_hypot_t.cc:1:
/home/greenc/work/cet-is/build/mrb-art-3.13.02/srcs/cetlib/cetlib/hypot.h:60:1: note: candidate: ‘template<class T> std::enable_if_t<is_arithmetic_v<T>, T> cet::hypot(T, T)’
   60 | cet::hypot(T x, T y)
      | ^~~
/home/greenc/work/cet-is/build/mrb-art-3.13.02/srcs/cetlib/cetlib/hypot.h:60:1: note:   template argument deduction/substitution failed:
In file included from /scratch/products/gcc/v13_1_0/Linux64bit+3.10-2.17/include/c++/13.1.0/bits/stl_pair.h:60,
                 from /scratch/products/gcc/v13_1_0/Linux64bit+3.10-2.17/include/c++/13.1.0/bits/stl_algobase.h:64,
                 from /scratch/products/gcc/v13_1_0/Linux64bit+3.10-2.17/include/c++/13.1.0/bits/specfun.h:43,
                 from /scratch/products/gcc/v13_1_0/Linux64bit+3.10-2.17/include/c++/13.1.0/cmath:3716,
                 from /home/greenc/work/cet-is/build/mrb-art-3.13.02/srcs/cetlib/cetlib/hypot.h:10:
/scratch/products/gcc/v13_1_0/Linux64bit+3.10-2.17/include/c++/13.1.0/type_traits: In substitution of ‘template<bool _Cond, class _Tp> using std::enable_if_t = typename std::enable_if::type [with bool _Cond = false; _Tp = std::__cxx11::basic_string<char>]’:
/home/greenc/work/cet-is/build/mrb-art-3.13.02/srcs/cetlib/cetlib/hypot.h:60:1:   required by substitution of ‘template<class T> std::enable_if_t<is_arithmetic_v<T>, T> cet::hypot(T, T) [with T = std::__cxx11::basic_string<char>]’
/home/greenc/work/cet-is/build/mrb-art-3.13.02/srcs/cetlib/cetlib/test/bad_hypot_t.cc:11:13:   required from here
/scratch/products/gcc/v13_1_0/Linux64bit+3.10-2.17/include/c++/13.1.0/type_traits:2610:11: error: no type named ‘type’ in ‘struct std::enable_if<false, std::__cxx11::basic_string<char> >’
 2610 |     using enable_if_t = typename enable_if<_Cond, _Tp>::type;
      |           ^~~~~~~~~~
```

```
/home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/test/bad_hypot_t.cc: In function ‘int main()’:
/home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/test/bad_hypot_t.cc:11:13: error: no matching function for call to ‘hypot(std::__cxx11::basic_string<char>, std::__cxx11::basic_string<char>)’
   11 |   cet::hypot("lorem"s, "ipsum"s);
      |   ~~~~~~~~~~^~~~~~~~~~~~~~~~~~~
In file included from /home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/test/bad_hypot_t.cc:1:
/home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/hypot.h:65:10: note: candidate: ‘template<class T>  requires  is_arithmetic<T> T cet::hypot(T, T)’
   65 | inline T cet::hypot(T x, T y)
      |          ^~
/home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/hypot.h:65:10: note:   template argument deduction/substitution failed:
/home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/hypot.h:65:10: note: constraints not satisfied
In file included from /home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/hypot.h:15:
/home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/detail/cetlib_concepts.h: In substitution of ‘template<class T>  requires  is_arithmetic<T> T cet::hypot(T, T) [with T = std::__cxx11::basic_string<char>]’:
/home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/test/bad_hypot_t.cc:11:13:   required from here
/home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/detail/cetlib_concepts.h:17:11:   required for the satisfaction of ‘is_arithmetic<T>’ [with T = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >]
/home/greenc/work/cet-is/build/mrb-art-summer/srcs/cetlib/cetlib/detail/cetlib_concepts.h:17:32: note: the expression ‘is_arithmetic_v<T> [with T = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >]’ evaluated to ‘false’
   17 |   concept is_arithmetic = std::is_arithmetic_v<T>;
      |                           ~~~~~^~~~~~~~~~~~~~~~~~
```

Invalid substitution set vs Constraint failure

Fermilab

Practicalities

```
#include <catch2/catch_test_macros.hpp>

#include "hep_concurrency/SerialTaskQueueChain.h"

namespace {
  int
  num(int mun)
  {
    return mun;
  }
  void
  goodFunc()
  {}

  class GoodTask {
  public:
    GoodTask(int num) noexcept : num_(num) {}
    void
    operator()()
    {}

  private:
    int num_{};
  };

  template <typename T>
  concept can_push_to_chained_queues =
    requires(hep::concurrency::SerialTaskQueueChain& chain, T t) {
      chain.push(t);
    };

  auto
  verify_push_to_chained_queues(auto t)
  {
    return can_push_to_chained_queues<decltype(t)>;
  }
}

TEST_CASE("Enforce task constraints")
{
  CHECK(verify_push_to_chained_queues(GoodTask{3}));
  CHECK(verify_push_to_chained_queues(goodFunc));
  CHECK_FALSE(verify_push_to_chained_queues(num));
}
```

- Must be careful to ensure implemented constraints matches desired semantics
- Compile-only testing vs runtime testing
- Catch2 for runtime testing of concepts. `require()` statement and nested concepts are great tools for testing constraint enforcement.
- Testing can be difficult with constraints because of syntactic evaluation vs actual evaluation.
- You only “need” constraints at the lowest level, but for user-visible diagnostic the constraint will sometimes need to be applied at higher levels (cf rethrow vs wrap and throw)
- Ran into conflict with ROOT when we implemented constraints into something interfaced with a ROOT dictionary (Legacy vs PCM)

Progress has been made

Selick, Henry. *Coraline*. Focus Features, 2009.

- Applied Concepts & Constraints to low level utility libraries. Have moved on to the “meat” of the framework.
- Testing of concepts and constraints needs more understanding
  - Efficient compile-only tests have framework implications

Ranges

Range primitives

Defined in header <ranges>

ranges::iterator_t	(C++20)	obtains associated types of a range (alias template)
ranges::const_iterator_t	(C++23)	
ranges::sentinel_t	(C++20)	
ranges::const_sentinel_t	(C++23)	
ranges::range_difference_t	(C++20)	
ranges::range_size_t	(C++20)	
ranges::range_value_t	(C++20)	
ranges::range_reference_t	(C++20)	
ranges::range_const_reference_t	(C++23)	
ranges::range_rvalue_reference_t	(C++20)	
ranges::range_common_reference_t	(C++20)	

Dangling iterator handling

Defined in header <ranges>

ranges::dangling (C++20)	a placeholder type indicating that an iterator or a subrange should not be returned since it would be dangling (class)
ranges::borrowed_iterator_t (C++20) ranges::borrowed_subrange_t	obtains iterator type or subrange type of a borrowed_range (alias template)

Range concepts

Defined in header <ranges>

ranges::range (C++20)	specifies that a type is a range, that is, it provides a begin iterator and an end sentinel (concept)
ranges::borrowed_range (C++20)	specifies that a type is a range and iterators obtained from an expression of it can be safely returned without danger of dangling (concept)
ranges::sized_range (C++20)	specifies that a range knows its size in constant time (concept)
ranges::view (C++20)	specifies that a range is a view, that is, it has constant time copy/move/assignment (concept)
ranges::input_range (C++20)	specifies a range whose iterator type satisfies input_iterator (concept)
ranges::output_range (C++20)	specifies a range whose iterator type satisfies output_iterator (concept)
ranges::forward_range (C++20)	specifies a range whose iterator type satisfies forward_iterator (concept)
ranges::bidirectional_range (C++20)	specifies a range whose iterator type satisfies bidirectional_iterator (concept)
ranges::random_access_range (C++20)	specifies a range whose iterator type satisfies random_access_iterator (concept)
ranges::contiguous_range (C++20)	specifies a range whose iterator type satisfies contiguous_iterator (concept)
ranges::common_range (C++20)	specifies that a range has identical iterator and sentinel types (concept)
ranges::viewable_range (C++20)	specifies the requirements for a range to be safely convertible to a view (concept)
ranges::constant_range (C++23)	specifies that a range has read-only elements (concept)

Range conversions

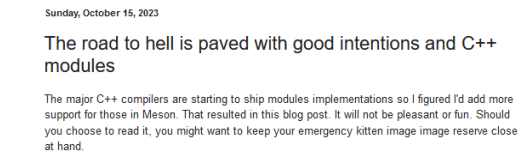
Defined in header <ranges>

ranges::to (C++23)	constructs a new non-view object from an input range (function template)

https://en.cppreference.com/w/cpp/ranges

- Art is already using Range-v3 in specific scenarios
  - https://github.com/ericniebler/range-v3
- C++23 is necessary to fully adopt std::ranges
  - C++20 is missing ranges::to() and useful treatment of ranges over const values
- Until C++23 is available, we'll continue using Range-v3

Modules

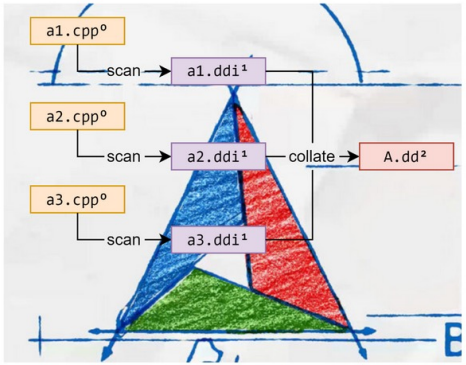

Sunday, October 15, 2023

The road to hell is paved with good intentions and C++ modules

The major C++ compilers are starting to ship modules implementations so I figured I'd add more support for those in Meson. That resulted in this blog post. It will not be pleasant or fun. Should you choose to read it, you might want to keep your emergency kitten image image reserve close at hand.


import CMake; the Experiment is Over!

October 18, 2023 Bill Hoffman, Brad King and Ben Boeckel



- Scaling!
  - Making and using an interlinked set of modules across packages causes issues
- Build-system/compiler interactions
  - https://nibblestew.blogspot.com/2023/10/the-road-to-hell-is-paved-with-good.html
  - Kind of wants you to do the compilers job for it by knowing where the code goes and how before invoking the compiler
- The not-yet-released CMake 3.28 supports modules for Clang 16 (Sep 2023) and GCC 14 (May 2024)
  - https://www.kitware.com/import-cmake-the-experiment-is-over/
- Will revisit this when there's more community support available