Standard Library Modules

Gabriel Dos Reis Billy O'Neal Stephan T. Lavavej

Jonathan Wakely

Abstract

With recent progress made on the 'module' language feature, it is long overdue that we start a conversation around uses of modules in the standard library. More specifically, how should the standard library be logically presented as set of modules to C++ programmers? This is a proposal to answer that question.

Design Principles

The presentation of the standard library as set of modules follows a few design principles:

- Leave "C headers" alone
- Present modules as logical set of functionalities
- Every standard facility is provided by exactly one module

We strongly recommend to leave "C headers" alone because only confusion and chaos has ever resulted from past attempts at legislating what C standard headers should contain. The reality is that in practice, the contents of those headers are outside the reach of C++ implementers for the most part. One could wish that wasn't the case, but this is the world we live in.

In the past, we've structured standard headers to minimize their contents in the hope of aiding compiler throughput. The module system is designed to attain the ideal of zero abstraction overhead. That is, a module does not impose any measurable overhead if none of the names it provides is referenced in a translation unit. Consequently, we propose to structure standard library modules around logical set of functionalities, shifting the focus to uses as opposed to implementations.

Proposed Structures

Naming

All standard library modules should be named according to the pattern '`std.`*xyz*'. However, we do not recommend a reflexive one-to-one mapping of headers <*xzy*> to module '`std.`*xzy*'.

Modules

As a first approximation, we suggest the following:

1. Module `std.fundamental` provides the declarations of the following facilities:
   - Content of `<cstddef>`
   - Content of `<limits>`
   - Content of `<cfloat>`
   - Content of `<cstdint>`

- Content of <new>
- Content of <typeinfo>
- Content of <exception>
- Content of <initializer_list>
- Content of <csignal>
- Content of <csetjump>
- Content of <cstdlib>
- Content of <stdexcept>
- Content of <system_error>
- Content of <utility>
- Content of <tuple>
- Content of <optional>
- Content of <variant>
- Content of <any>
- Content of <bitset>
- Content of <type_traits>
- Content of <ratio>
- Content of <chrono>
- Content of <ctime>
- Content of <atomic>

- Content of <typeindex>
- std::unique_ptr and associated classes and functions from <memory>
- std::shared_ptr and associated classes and function from <memory>

2. Module std.core provides the declarations of the following facilities:
    - Content of <array>
    - Content of <list>
    - Content of <forward_list>
    - Content of <vector>
    - Content of <deque>
    - Content of <queue>
    - Content of <stack>
    - Content of <map>
    - Content of <set>
    - Content of <unordered_map>
    - Content of <unordered_set>
    - Content of <iterator>
    - Content of <algorithm>
    - Content of <execution>
    - Content of <functional>
    - Content of <string_view>
    - Content of <string>
    - Re-export of module std.fundamental

3. Module std.io provides the declarations from the following headers:
    - <cctype>
    - <cwctype>
    - <cwchar>
    - <cstdlib>
    - <cuchar>
    - <locale>
    - <codecvt>
    - <clocale>
    - <iosfwd>
    - <iostream>
    - <ios>
    - <streambuf>
    - <istream>
    - <ostream>
    - <iomanip>
    - <sstream>
    - <fstream>
    - <cstdio>
4. Module std.filesystem provides the declarations from the following header:
    - <filesystem>
5. Module std.regex provides the declarations from the following header:
    - <regex>
6. Module std.threading provides the declarations for the following facilities:
    - Content of <mutex>
    - Content of <thread>
    - Content of <condition_variable>
    - Content of <shared_mutex>
    - Content of <future>
7. Module std.memory provides the declarations for the following facilities:
    - Content of <memory> except std::unique_ptr and associated classes and functions, std::shared_ptr and associated classes and functions
    - Content of <memory_resource>
    - Content of <scoped_allocator>
8. Module std.numeric provides the declarations from the following headers:
    - <complex>
    - <numeric>
    - <valarray>
    - <random>
    - <cmath>

Clearly, this list is incomplete, but it is suggested as a starting point of the conversation.

Another name to consider for std.threading is std.concurrency.

Freestanding implementations are required to provide at least std.fundamental.