# naming and aliases

Document #: P2065R0
Date: 2020-01-13
Project: Programming Language C++
Library Evolution Group
Reply-to: Kirk Shoop
<kirk.shoop@gmail.com>

## Contents



## 1 Introduction

Naming is hard. Thus there is a lot of discussion about naming. One guideline that has been making naming discussions more difficult and the resulting names less usable is described in [P0816R0] 'No More Nested Namespaces in Library Design'. I have read the paper and agree with the world it describes. I have watched the effects it has on the naming of new features and would like to refine the guidance in this paper.

## 2 Effects

I find comments like the following to result in names that are not desireable.

— " 'task' is being reserved for something, so we can't use 'task' " (names won't match the functionality).
— "the views `... | join()` and `concat(...)` cannot share the same name even though they are the same algorithm" (names won't match the functionality).
— "Drop nested namespace fmt and prefix some names with 'format..' to disambiguate" (c-style namespacing).
— "add `pmr`, because `polymorphic_memory_resources` is too long to type" (names won't be descriptive)
— "add an alias for `filesystem` to `fs`, because `std::filesystem` is too long to type" *see* [P1005R1] (names won't be descriptive)

— "don't put the concepts in a namespace, because `std::concepts::` is too long to type" (names won't be descriptive)

# 3 Discussion

[P0816R0] *"No More Nested Namespaces in Library Design"* outlines how nested namespaces affect name lookup and how that has been an issue that required effort to clean up in existing codebases.

> The guidance for users should be single-level broad namespaces - namespaces are best used to disambiguate between the local project and imported projects. Because of the promiscuity of name lookup rules, nested namespaces should not be used as an attempt to introduce partitions in the current project - it doesn't work that way.

This paper proposes that this guidance be changed, despite the issues with name lookup. Until the language offers a different solution, namespaces need to be used to disambiguate names within projects, not just across projects.

It should already be best practice to use fully specified names everywhere. Following these new guidelines should motivate more education and tools to enforce usage of fully specified names in a way that helps code improve, for instance as part of the Core Guidelines initiative.

> example core guideline T.69: Inside a template, don't make an unqualified nonmember function call unless you intend it to be a customization point

# 4 naming guidance proposal

## 4.1 Prefer nested namespaces to naming prefixes

This is existing practice. `ranges` and `filesystem` did this to disambiguate functions and algorithms that conflicted with existing meanings in the STL (eg. copy).

A new naming complexity was introduced with concepts. A concept and a vocabulary type will naturally have the same name. for instance `class task` and `concept task`. When renaming the concepts a conflict between `concept view` and `namespace view` was resolved by renaming to `namespace views`. This was made easier because a namespace could be made plural and still make sense. Had there been a vocabulary `class view` disambiguating the type, the concept and the namespace would have been more challenging.

Having a `concepts` namespace that is structured like the `literals` namespaces would make this disambiguation easier to manage as new concept + type sets are added to the library.

## 4.2 Prefer descriptive names

Acronyms like `pmr` are not descriptive.

# 5 Accommodating opposition to long names

Attempts to use descriptive names and nested namespaces have already been blocked by cries of 'But what about my poor poor fingers!'. Some accommodation is needed for those that do not want to type.

### 5.1 the std::aliases namespace

Adding a new nested namespace with a descriptive name will allow standard aliases to be specified for all the namespaces.

This namespace will also provide a scoped means of pulling in the aliases without also pulling in types or function names.

```
namespace std {
namespace aliases {

namespace cr = std::chrono;
namespace ex = std::execution;
namespace exp = std::experimental;
namespace fs = std::filesystem;
namespace num = std::numbers;
namespace ph = std::placeholders;
namespace pmr = std::pmr;
namespace ro = std::rel_ops;
namespace rgc = std::regex_constants;
namespace rng = std::ranges;
namespace rvw = std::ranges::views;
namespace tt = std::this_thread;

namespace lit = std::literals;
namespace crlit = std::literals::chrono_literals;
namespace cxlit = std::literals::complex_literals;
namespace stlit = std::literals::string_literals;
namespace svlit = std::literals::string_view_literals;
} // namespace aliases
} // namespace std
```

see live code here https://godbolt.org/z/uHWeN3

Table 1: compare usage with full names to aliased names

| Before | After |
|---|---|
| `#include <chrono>`<br>`#include <thread>`<br><br>`using namespace`<br>`  std::literals::chrono_literals;`<br><br>`int main() {`<br>`    std::this_thread::sleep_for(10ms);`<br>`    return 0;`<br>`}` | `#include <chrono>`<br>`#include <thread>`<br><br>`using namespace std::aliases;`<br>`using namespace crlit;`<br><br>`int main() {`<br>`    tt::sleep_for(10ms);`<br>`    return 0;`<br>`}` |

**Before**

```cpp
#include <chrono>
#include <thread>



using namespace
  std::literals::chrono_literals;

int main() {
    std::this_thread::sleep_for(10ms);
    return 0;
}
```

**After**

```cpp
#include <chrono>
#include <thread>

// users that want to protect against name
// conflicts with current and future aliases
namespace sa = std::aliases;
using namespace sa::crlit;

int main() {
    sa::tt::sleep_for(10ms);
    return 0;
}
```

# 6 Appendices

## 6.1 collecting namespace list from a std library implementation

List of namespaces:

```
grep -Roh "^[ ]*[a-z]*[ ]*namespace[ ]\+[^_{][^: {]*" gcc/install/include/c++/9.0.0/* | sed
"s/^[ ]*[a-z]*[ ]*namespace//" | sort | uniq
```

```
 abi
 chrono
 chrono_literals
 complex_literals
 decimal
 detail
 experimental
 filesystem
 fundamentals_v1
 fundamentals_v2
 literals
 placeholders
 pmr
 regex_constants
 rel_ops
 std
 string_literals
 string_view_literals
 this_thread
 tr1
 tr2
 typelist
 v1
```

# 7 References

[P0816R0] Titus Winters. 2017. No More Nested Namespaces in Library Design.
https://wg21.link/p0816r0

[P1005R1] Bryce Adelstein Lelbach, Davis Herring. 2018. namespace std { namespace fs = filesystem; }.
https://wg21.link/p1005r1