Structured naming for function object and CPO values

Draft Proposal

Document #:	D0000R0
Date:	2021-11-01
Project:	Programming Language C++
Audience:	LEWG Library Evolution

Reply-to:	Kirk Shoop <kirk.shoop@gmail.com>

Contents

1 Changes

1.1 R0

— first revision

2 Introduction

There was a discussion of naming for the set of `fold` algorithms in [P2322R5]. Naming conversations are particularly fraught because names veer into the subjective and the technical aspects of naming get obscured by the subjective aspects of naming.

I submitted a novel way to name sets of function object values (like CPO sets and other function object sets such as `fold`). This is novel only for function naming, it is established practice for naming of other entities such as: namespace, class, struct, union, etc..

I submitted this very late. I agree that this proposal was not something that should block [P2322R5] from being forwarded. I am under the impression that this name change proposal can be applied after it arrives in LWG. I was asked to write a paper describing the motivation and reasoning behind this naming proposal. This is that paper.

3 Motivation

Statement: Naming sets of things is made much easier by adding nested structure.

structured name	unstructured name
`std::vector<T>`	`std_vector<T>`
`std::vector<T>::value_type`	`std_vector_value_type<T>`
`std::list<T>`	`std_list<T>`
`std::list<T>::const_iterator`	`std_list_const_iterator<T>`
`std::chrono`	`std_chrono`
`std::chrono::steady_clock`	`std_chrono_steady_clock`
`std::chrono::steady_clock::time_point`	`std_chrono_steady_clock_time_point`

The examples of unstructured naming in this table are probably perceived as an absurd straw-man. I would suggest that this reaction is due to an expectation of naming structure that we have unconsciously assumed was inviolate and obvious. I suggest that much of the committee agrees that structured naming, where it is possible, is necessary and any unstructured alternative is generally considered absurd.

[P2322R5] was forwarded to electronic polling with this poll:

POLL: Forward P2322R5: ranges::fold with the name set in option D to LWG for C++23 (to be confirmed with electronic poll) priority 2

Strongly Favor	Weakly Favor	Neutral	Weakly Against	Strongly Against
8	4	2	1	0

The names in [P2322R5] option D:

name
`fold_left()`
`fold_left_first()`
`fold_right()`
`fold_right_last()`

These names are roots. The suffix `_with_iter` is applied to expand to the full set of names selected by option D in [P2322R5]

name
`fold_left()`
`fold_left_first()`
`fold_right()`
`fold_right_last()`
`fold_left_with_iter()`

name
`fold_left_first_with_iter()`

We are used to function names being unstructured. When these were plain functions, unstructured naming was very hard to avoid. [P2322R5] as I understand it, is proposing that the names in the above table, be implemented as function objects. There are also other proposals that are creating names for function objects and CPOs, that could be structured, but are not.

An excerpt from [P2322R5] indicates that more `fold` functions are forthcoming:

> The problem going past that is that we end up with this combinatorial explosion of algorithms based on a lot of orthogonal choices:
>
> iterator pair or range
> left or right fold
> initial value or no initial value
> short-circuit or not short-circuit
> return T or (iterator, T)
>
> Which would be… 32 distinct functions (under 16 different names) if we go all out. And these really are basically orthogonal choices. Indeed, a short-circuiting fold seems even more likely to want the iterator that the algorithm stopped at! Do we need to provide all of them? Maybe we do!

4 Proposals

Given that many new functions are being specified as namespaced function objects and CPOs, there is a structured naming option available.

Observation: function objects and CPOs can have nested function objects and CPOs as members.

This observation provides the option for using structured names for sets of nested functions that would historically use unstructured naming.

4.1 Fold

An example of structured names as an alternative for the names in [P2322R5] option D:

structured name	unstructured name
`fold.left()`	`fold_left()`
`fold.left.with_iter()`	`fold_left_with_iter()`
`fold.left.first()`	`fold_left_first()`
`fold.left.first.with_iter()`	`fold_left_first_with_iter()`
`fold.right()`	`fold_right()`
`fold.right.last()`	`fold_right_last()`

The concrete proposal for this paper is that structured names be applied to `fold` for C++23.

Additionally, I propose that any other set of function object and CPO names that are using unstructured naming and that are being added in C++23 should adopt structured names for C++23.

4.2 Concepts and CPOs

The core observation of this paper can also be applied to structure other names being proposed for C++23:

structured name	unstructured name
`std::execution::receiver.set_value()`	`std::execution::set_value()`
`std::execution::receiver.set_error()`	`std::execution::set_error()`
`std::execution::receiver.set_done()`	`std::execution::set_done()`
`std::execution::sender.connect()`	`std::execution::connect()`

NOTE: `receiver` and `sender` are already defined as concepts today which cannot have members.

The challenge to this naming approach for concepts and structured CPOs today is that without some structured naming solution for concept definitions, we will waste a huge amount of time in LEWG trying to name concepts and their structured CPOs in a way that does not conflict.

NOTE: If I might speculate wildly here. It would be exceptionally useful for the `receiver` and `sender` objects to be able to define `template<class... TN> operator concept<TN...>() = ..;` that would operate as a concept when `receiver<T>` and `sender<T>` were used. This could be an avenue for resolving the naming discussions in LEWG between concepts and their structured CPO names. I expect that some disambiguator would be needed - like `o::template m<>` is today - for objects that have template arguments and an `operator concept<>` definition. Perhaps by attempting to follow existing practice `o<>::concept <>` would be an option. Obviously, I do not expect that a change like this could be applied in C++23.

5 Q&A

5.1 Q: does this propose replacing every _ in every name of every new function object with a .?

Nope. I think this is subjective. I do not consider `with` in `with_iter` to represent a space that would contain `iter`. I see `with_iter` as a compound name. In fact, I think that this proposal allows those distinctions to be expressed by choosing to use `.` or `_` while unstructured naming does not allow those distinctions to be expressed in the name.

I do not think that structured naming should be blindly applied to every function that is not part of a set. I do think that there will be cases where a new function is known to be the first of a set that will be proposed over time and may preemptively be given a structured name even if it is initially the only nested object.

5.2 Q: if an object for a name (like fold) is claimed for one purpose, must an unrelated object with the prefix `fold_` become a member?

Nope. Even [P2322R5] option D can be an example of this question. Is `with_iter` a valid member of the `fold` set, or is it a different algorithm with its own distinct set of structured names? I think that the following expression of structured names is just as valid as the one derived from Option D in Proposals:

structured name	unstructured name
`fold.left()`	`fold_left()`
`fold.left.first()`	`fold_left_first()`
`fold.right()`	`fold_right()`
`fold.right.last()`	`fold_right_last()`
`fold_with_iter.left()`	`fold_left_with_iter()`
`fold_with_iter.left.first()`	`fold_left_first_with_iter()`

The difference between these two is somewhat subjective IMO and I would expect this to be part of the naming discussion.

5.3 Q: should all namespaces become objects (eg. should `views` have been an object instead of a namespace, given this proposal)?

Nope. I think that `views` intends to prevent naming conflicts by providing a common prefix name to a set of functions. The alternative was unstructured names for each function object and CPO that were prefixed with `view_` or suffixed with `_view`. I think that `views` would be valid as an object, with nested objects and CPOs, if its intent was to create a semantic nesting rather than it being intended to use nesting to prevent name conflicts.

The `views` namespace is an example of the common desire and accepted utility of structured names in C++.

Ultimately intent is subjective, so in practice `views` might have gone the other way under different circumstances, but I see the selection of the namespace for `views` as valid and compatible with this proposal.

6 Conclusion

We have an opportunity to apply the naming structure we love and expect for many entities in C++ to functions specified as objects and CPOs. We know that structured names have concrete benefits. We prefer structured names for other things. We should take this opportunity to adopt structured names for function objects and CPOs going forward.

7 References

[P2322R5] Barry Revzin. 2021-10-18. ranges::fold.
https://wg21.link/p2322r5