`mdspan::size_type` should be `index_type`

Document number: P2599R1
Date: 2022-06-14
Project: Programming Language C++, Library Evolution Working Group
Reply-to: Nevin “☺” Liber, nliber@anl.gov

Table of Contents

Revisions

R1

In order to strengthen consensus, LEWG requested that in addition to the changes requested in P2599R0 (change all current references of `size_type` to `index_type`), we also add a new `size_type` typedef mapped to the unsigned type corresponding to what would now be `index_type`.

Polls

__POLL: Send P2599R0 (`mdspan::size_type` should be `index_type`) to Library for C++23 classified as an improvement (B2) to be confirmed with a Library Evolution electronic poll.__

Strongly Favor	Weakly Favor	Neutral	Weakly Against	Strongly Against
2	7	2	2	1

__Attendance:__ 21

__# of Authors:__ 1

__Author Position:__ SF

__Outcome:__ Weak consensus in favor.

SA: It's already a conscious choice by the user to use a signed type. So I don't think it will be surprising. The consistency of having it be called `size_type` is more important.

__POLL: Rename `mdspan` and friend's `size_type` member to `index_type` and have a `size_type` member be present only if `index_type` is unsigned.__

Author note: not polled, as the poll below had consensus.

__POLL: `mdspan`, `extents`, and layouts should have both an `index_type` (which is whatever the user provides for the first template parameter to `extents`) and a `size_type` (which is `make_unsigned_t<index_type>`).__

Strongly Favor	Weakly Favor	Neutral	Weakly Against	Strongly Against
3	9	1	1	0

__Attendance:__ 19

__# of Authors:__ 1

__Author Position:__ SF

__Outcome:__ Consensus in favor, and stronger consensus that the paper as written.

WA: It's additional complexity.

Introduction

With the adoption of P2553R1, `mdspan::size_type` may now be a signed type. `size_type` is no longer an appropriate name for this type and it should be changed to `index_type`.

Motivation and Scope

Throughout the C++ standard, `size_type` stands for an unsigned type. `mdspan` and its related class templates should be consistent with this.

When P2553R0 was proposed, `extents::size_type` was going to be constrained to `unsigned_integral`. At the request of LEWG, that constraint was removed in P2553R1 and adopted via electronic polling.

Now that it can be a signed type, `size_type` is no longer the correct name for this. It should revert back to `index_type`, which was used in `mdspan` until P0009R11 when the following change was made:

> Change all the sizes
> from `ptrdiff_t` to `size_t` and `index_type` to `size_type`, for consistency with `span` and the rest of the standard library

In addition to `extents`, there are other class templates which take `Extents` as a template parameter and adopt the `size_type` typedef from `Extents` into their interface. Those class templates should also have their `size_type` typedefs changed to `index_type`.

LEWG requested that a new `size_type` that corresponds to the unsigned version of `index_type` also be added to these class templates.

Specifically, the following class templates should replace their usage of `size_type` with `index_type` and then add a new `size_type`:

- `extents`
- `layout_left::mapping`
- `layout_right::mapping`
- `layout_stride::mapping`
- `mdspan`

Impact On the Standard

Given that `mdspan` and its related classes are new class templates for C++23, the impact should be minimal. Also, no feature test macro should be necessary.

Technical Specifications

The changes proposed here are:

- Normatively change the spelling of `size_type` to `index_type`
- Editorially change the spelling of template parameter `SizeT` / `SizeType` to `IndexType`
- Editorially change the spelling of template parameter `OtherSizeT` / `OtherSizeType` to `OtherIndexType`
- Editorially change the spelling of template parameter `SizeTypes` to `IndexTypes`
-

Then apply the following additions:

- To `extents`, normatively add the public definition `using size_type = make_unsigned_t<index_type>;`
- To `layout_left::mapping`, `layout_right::mapping`, `layout::stride::mapping` and `mdspan`, add the public definition `using size_type = typename extents_type::size_type;`

Note: P0009 and P2553 are currently undergoing revisions as requested by LWG. If this proposal is approved, the author will rebase these changes on the result of those changes.

Acknowledgements

This was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative. Additionally, this research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

P0009 mdspan, Christian Trott *et al.*
P2553 Make mdspan size_type controllable, Christian Trott *et al.*

Appendix

Rendered diffs against the 9e0e246 source commit of P0009 and P2553 (found at commit 9980c74).

```
template<size_t ... Extents>
using extents = basic_extents<size_t, Extents...>;
```

LEWG would need to decide whether to make `dextents` have a `size_type` template parameter or not.

2.3 Why we can't fix this later

In principle we could add a second extents type later, though it may break code such as the one shown before (in the sense that it wouldn't generally work for every instance of `mdspan` anymore):

```
template<class T, class L, class A, size_t ... Extents>
void foo(mdspan<T,extents<Extents...>,L,A> a) {...{
```

3 Editing Notes

The proposed changes are relative to the working draft of the standard as of P0009 R16.

4 Wording

4.1 In 22.7.X [mdspan.syn]

Replace:

```
template<size_t... Extents>
  class extents;

template<size_t Rank>
  using dextents = see below;
```

with:

```
template<class IndexType, size_t... Extents>
  class extents;

template<class IndexType, size_t Rank>
  using dextents = see below;
```

4.2 In 22.7.X.1 [mdspan.extents.overview]:

— In the synopsis replace:

```
template<size_t... Extents>
class extents {
public:
  using size_type = size_t;
  using rank_type = size_t;
```

with

```
template<class IndexType, size_t... Extents>
class extents {
public:
  using index_type = IndexType;
  using size_type = make_unsigned_t<index_type>;
  using rank_type = size_t;
```