`index_type` & `size_type` in `mdspan`

Document number: P2599R2
Date: 2022-06-23
Project: Programming Language C++, Library Evolution Working Group
Reply-to: Nevin “☺” Liber, nliber@anl.gov

Table of Contents

Revisions

R2

- `mdspan::size()` now returns `size_type` (it was returning `size_t` as of P0009R17, and it returned the old `size_type` in P0009R16)
- Rename the paper, as it covers a bit more than just renaming `index_type` to `size_type`
- Rename “Technical Specification” section to “Wording Changes” in this paper.
- Remove `SizeT` and `OtherSizeT` from this paper, as they are no longer in P0009R17
- Replaced `SizeTypes` -> `IndexTypes` spelling change with `OtherSizeTypes` -> `OtherIndexTypes` to match changes in P0009

There was some discussion around changing the return type for `mapping::required_span_size()` & `mapping::operator()`, but no changes are being proposed. Similar discussions were made around accessors, but no changes are being proposed.

Polls

__POLL: Modify P2599R1 (`mdspan::size_type` should be `index_type`) such that `mdspan::size`s return type is `size_type`, and send the modified paper to Library for C++23 classified as B2 - Improvement, to be confirmed with a Library Evolution electronic poll.__

Strongly Favor	Weakly Favor	Neutral	Weakly Against	Strongly Against
5	8	0	1	0

__Attendance:__ 29

__# of Authors:__ 1

__Author Position:__ SF

__Outcome:__ Strong consensus

WA: This is a late change.

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`

As part of P2553R1, `mdspan::size()` was changed to return `size_t` to continue to return an unsigned type. During the review of P2599R1, LEWG requested that it returns the `size_type` proposed here instead.

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.

Wording Changes

The renaming changes proposed here are:

- Normatively change the spelling of `size_type` to `index_type`
- Editorially change the spelling of template parameter `SizeType` to `IndexType`

- Editorially change the spelling of template parameter `OtherSizeType` to `OtherIndexType`
- Editorially change the spelling of template parameter pack `OtherSizeTypes` to `OtherIndexTypes`

Then apply the following additions / changes (summarized here, followed by diffs against P0009R17):

- 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;`
- To `extents`, normatively add the public definition `using size_type = make_unsigned_t<index_type>;`
- To `mdspan`, change the return type of `size()` to `size_type`

Specifically, the new additions / changes relative to P0009R17 *after applying the renaming changes* are:

In 24.7.X.1 [mdspan.extents.overview], in the synopsis change:

```
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;
```

In 24.7.X.5.1 [mdspan.layoutleft.overview], in the synopsis change:

```
template<class Extents>
class layout_left::mapping {
  public:
    using extents type = Extents;
    using index_type = typename extents_type::index_type;
    using size_type = typename extents_type::size_type;
    using rank_type = typename extents_type::rank_type;
```

In 24.7.X.6.1 [mdspan.layoutright.overview], in the synopsis change:

```
template<class Extents>
class layout_left::mapping {
  public:
    using extents_type = Extents;
    using index_type = typename extents_type::index_type;
    using size_type = typename extents_type::size_type;
    using rank_type = typename extents_type::rank_type;
```

In 24.7.X.7.1 [mdspan.layoutstride.overview], in the synopsis change:

```
template<class Extents>
class layout_left::mapping {
  public:
    using extents_type = Extents;
```

```
    using index_type = typename extents_type::index_type;
    using size_type = typename extents_type::size_type;
    using rank_type = typename extents_type::rank_type;
```

In 24.7.X.1 [mdspan.mdspan.overview], in the synopsis change:

```
template<class ElementType, class Extents, class LayoutPolicy, class AccessorPolicy>
class mdspan {
public:
  using extents_type = Extents;
  using layout_type = LayoutPolicy;
  using accessor_type = AccessorPolicy;
  using mapping_type = typename layout_type::template mapping<extents_type>;
  using element_type = ElementType;
  using value_type = remove_cv_t<element_type>;
  using index_type = typename extents_type::index_type;
  using size_type = typename extents_type::size_type;
  using rank_type = typename extents_type::rank_type;
  using pointer = typename accessor_type::pointer;
  using reference = typename accessor_type::reference;
```

and

```
  template<class... OtherSizeTypes>
    constexpr reference operator[](OtherSizeTypes... indices) const;
  template<class OtherSizeType>
    constexpr reference operator[](span<OtherSizeType, rank()> indices) const;
  template<class OtherSizeType>
    constexpr reference operator[](const array<OtherSizeType, rank()>& indices) const;

  constexpr size_type size() const noexcept;

  friend constexpr void swap(mdspan& x, mdspan& y) noexcept;
```

In 24.7.X.3 [mdspan.mdspan.members], change:

```
constexpr size_type size() const noexcept;
```

Precondition: The size of the multidimensional index space `extents()` is representable as a value of type `size_type` ([basic.fundamental]).

Returns: `extents().`*`fwd-prod-of-extents`*`(rank()).`

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*.