Transformation Trait **uncvref**

Document #: WG21 P0550R0
Date: 2017-02-01
Project: JTC1.22.32 Programming Language C++
Audience: LEWG ⇒ LWG
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

This paper proposes **uncvref**, a new TransformationTrait for the **<type_traits>** header. Like **decay**, **uncvref** removes any *cv* and reference qualifiers, but unlike **decay**, it does not mimic any array-to-pointer or function-to-pointer conversion.

I love criticism just so long as it's unqualified praise.
— NOEL COWARD

I'm unqualified for anything else. I'm barely qualified for this.
— CATHERINE KEENER

Knowledge unqualified is knowledge simply of something learned.
— PLATO

1 Introduction

The **decay** trait[1] sees significant use in the Standard Library, as it plays an important role in the specification of several Library components. Alas, a number of these uses are more elaborate than is strictly necessary. In particular, **decay** is used in several places where it would suffice simply to strip *cv* and reference qualifiers, with no need for the decaying conversions (array-to-pointer and function-to-pointer) that gave rise to **decay**'s name.

To address this situation, this paper proposes **uncvref**, a new TransformationTrait[2] for the **<type_traits>** header. Like the **decay** trait, **uncvref** removes any *cv* and reference qualifiers; unlike **decay**, it does not mimic any array-to-pointer[3] or function-to-pointer[4] conversion. The next section will provide several examples in which this proposed trait would serve as a more accurate and slightly cheaper alternative to the current use of **decay**.

[1] See [N4618], [meta.trans.other].

[2] See [N4618], [meta.rqmts]/3.

[3] See [N4618], [conv.array].

[4] See [N4618], [conv.func].

2 Discussion

To a first approximation, the full **`decay`** treatment seems typically needed when forwarding arguments. In contrast, merely comparing types seems typically to require only **`uncvref`**. In the absence of the latter (proposed) trait, **`decay`** has frequently served as a convenient substitute for it. Here are several examples (from [N4618]) of such overly enthusiastic use of **`decay`**:

- The following excerpt occurs twice in [tuple.apply]:[5]

  ```
  ··· make_index_sequence<tuple_size_v<decay_t<Tuple>>>{});
  ```

  In this context, the **`decay_t`** metafunction call is unrelated to any possible array or function type. Instead, it is only the unqualified **`Tuple`** type that is wanted, so a call to the proposed **`uncvref`** would suffice:[6]

  ```
  ··· make_index_sequence<tuple_size_v<uncvref_t<Tuple>>>{});
  ```

- As a more interesting example, consider [optional.ctor]/16, where we find the metafunction call:

  ```
  is_same_v<optional<T>,decay_t<U>>
  ```

  As before, there is no role here for any decay conversion; only *cv* and reference qualifiers need be removed from type **`U`**. This therefore seems a perfect candidate for the proposed **`uncvref`** trait, leading to:

  ```
  is_same_v<optional<T>,uncvref_t<U>>
  ```

  instead. (A similar example is found in [variant.ctor]/16.)

- Finally, in [func.require], we find several conditions asking about the relationship of **`decay_t<decltype(t1)>`** to other types (e.g., to **`reference_wrapper`**). Applying **`uncvref`** instead of **`decay`** would suffice for these, too.[7]

In all, the Library clauses directly apply **`decay_t`** circa forty times; we recommend that each be audited along the lines we have begun above. If the present proposal is accepted, we are prepared to undertake these audits and report their result, with recommendations, in a future paper. (Clause 30 also uses the macro-like ***`DECAY_COPY`*** almost twenty additional times; these, too, should be similarly audited, although Lavavej opines[8] "that every use of ***`DECAY_COPY`*** is necessary.")

Finally, we note that several vendors have already implemented the proposed trait, under various private names. The next section will discuss the proposed trait's name.

3 Naming

From a number of private conversations regarding the proposed new trait's name, the following candidates have emerged:

- **`remove_const_volatile_reference`**: There was widespread agreement that this is the most descriptive name. If it were shorter, it would be the obvious choice, but no one wanted this much to type.

[5] [N4618] actually contains a third instance of just such an excerpt, but the Example (in [intseq.make]) of which it was a part has been editorially removed since [N4618] was published.

[6] Even that much is technically unnecessary; since **`tuple_size`** is defined for *cv*-qualified types, it would here suffice to remove reference qualification, leaving any *cv*-qualification.

[7] Similarly, applying **`uncvref`** instead of **`decay`** would suffice in the [futures.task.members]/3 specification that currently reads "... if **`decay_t<F>`** is the same type as **`packaged_task<R(ArgTypes...)>`**."

[8] Stephan T. Lavavej: "Re: remove_cv_ref." Personal correspondence, 2017–01–05.

- **`remove_cv_reference`**: Even this was seen as too long a name.
- **`remove_cv_ref`**: This seemed acceptable to all parties, but rather grudgingly so. No one wanted to lobby for it very strongly. One individual did rate the variant spelling **`remove_cvref`** as slightly more preferable for reasons of consistency with existing **`remove_*`** traits.
- **`strip`**: This name is in private use for the trait; it had been chosen by that implementer because the trait "strips qualifiers" from the given type. The name was generally seen as acceptable, but without significant enthusiasm due mostly to a lack of specificity.
- **`uncvref`**: Not only is this name in private use for the trait, a capitalized version (**`UNCVREF`**) is in use within the Ranges TS [N4620], as well. All consulted parties considered it sufficiently clear and acceptably short. For all these reasons, this seemed the best compromise name, and so we propose it here.

4 Proposed wording[9]

4.1 Insert into [meta.type.synop] (20.10.2) as shown:

```
namespace std {
  ...
  template <size_t Len, class... Types> struct aligned_union;
  template <class T> struct uncvref;
  template <class T> struct decay;
  ...
  template <size_t Len, class... Types>
    using aligned_union_t = typename aligned_union<Len, Types...>::type;
  template <class T>
    using uncvref_t = typename uncvref<T>::type;
  template <class T>
    using decay_t = typename decay<T>::type;
  ...
}
```

4.2 Between the rows specifying **`aligned_union`** and **`decay`**, insert the following new row into Table 50 — Other transformations:

`template <class T>` `struct uncvref;`	The member typedef **`type`** shall name the same type as `remove_cv_t<remove_reference_t<T>>`.

5 Acknowledgments

Many thanks for their thoughtful comments to Andrey Semashev and the other readers of pre-publication drafts of this paper.

[9] All proposed additions (there are no ~~deletions~~) are relative to the post-Issaquah Working Draft [N4618]. Editorial notes are displayed against a gray background.

6 Bibliography

[N4618] Richard Smith: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N4618 (post-Issaquah mailing), 2016–11–28. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/n4618.pdf.

[N4620] Eric Niebler and Casey Carter: "Working Draft, C++ Extensions for Ranges." ISO/IEC JTC1/SC22/WG21 document N4620 (post-Issaquah mailing), 2016–11–27. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/n4620.pdf.

7 Document history

Version	Date	Changes
0	2017-02-01	• Published as P0550R0.