**Proposal for C2x**
**WG14 N2930**

**WG21 P2541R0**

# Consider renaming `remove_quals`

Reply-to: Corentin Jabot (corentinjabot@gmail.com)
Document No: N2930 / P2541R0
Date: 2022-02-06

## Abstract

N2927 Introduces `typeof` and `remove_quals` to the C standard. This is a great addition, however the name `remove_quals` is somewhat inconsistent and impedes efforts of forward compatibility of C with C++. We would like WG14 to consider renaming `remove_quals`.

## Inconsistency

Both `typeof` and remove_quals accept either a type or an expression and produce a type. While it is evident what `typeof` does from the label, it is not immediately clear that `remove_quals(<expression>)` produces a type, or indeed that `typeof` and `remove_quals` are related facilities. Symmetry in the names would improve the ergonomy of the feature.

## Compatibility with C++

N2927 argues that decltype was not suitable for C.

Indeed in C++,

- `decltype(x)` denotes the type of the variable x (let's assume it's an lvalue)
- `decltype((x))` denotes the type of the expression `(x)`, which is an lvalue reference.

Lacking references, C would produce the same type for both expressions, which according to N2927 is an issue, as parentheses may be used to tame the preprocessor in C, and this is arguably, an important use case.

`typeof` then is added to C, with the intent that if C++ ever adopts it, it would be consistent with C: remove references and keep qualifiers. (Otherwise we would have the same problem with `typeof` that are the motivation for not simply adopting `decltype`).

The same motivation exists for remove_quals: for that feature to be consistent in both languages, such that C libraries compile consistently in either C or C++ (should C++ ever adopt it) C++ would have to strip references.

But references are not qualifiers, and the name `remove_quals` would be confusing to a C++ developer.

And so either

- `remove_quals` is inconsistent across languages
- `remove_quals` remove references in C++, despite its name implying it doesn't.

Neither solution is great, which means that, under the current naming scheme, I would be strongly opposed to a paper which would add these facilities to C++.

## Name suggestions

But the solution is simple: We can pick a name for `remove_quals` which works well for both C and C++.

Originally, `unqual_typeof` was suggested, and this works: we know that typeof would remove references, and unqual_typeof would also remove qualifiers.

I'd argue it is a better name for C independently of C++ compatibility concerns: it conveys that both `typeof` and `unqual_typeof` are related facilities.

N2927 argues:

> The only reason `_Unqual_typeof` would exist is to... well, remove qualifiers. It only makes sense to just name it appropriately by using `remove_quals` as a keyword.

It is true that the only thing `unqual_typeof` does that `typeof` doesn't is to remove qualifiers. It does disservice to try to hide that fact by use of inconsistent names.

Any variation (`unqual_typeof`, `unqualified_typeof`, `typeof_qualified`, `typeof_unqual`) would be equally suitable, and I have no strong preference.

## Should C++ adopt this feature?

I don't know what WG21 will elect to do in this space. However, should they decide to adopt it (or even if compilers decide to support this feature in C++ mode as an extension), having a name that is suitable for both languages and (familiars to users of either) would avoid unnecessary pain points.

## Wording

Replace all instances of `remove_quals` in the wording added by N2927 with `unqual_typeof`.

## Acknowledgment

I'd like to thank JeanHeyd Meneide for their great work on N2927.

## References

N2927 Not-so-magic - typeof for C - JeanHeyd Meneide

http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2927.html