User-defined Literals for std::complex
part 2 of UDL for Standard Library Types (version 4)
(part 1 is N3642)

Peter Sommerlad

2013-04-19

Document Number:	N3660 (part 2 of update of N3531/N3468/N3402)
Date:	2013-04-19
Project:	Programming Language C++

1 History

1.1 Changes from N3531

The following changes (or non-changes) result from the discussion in Bristol.

- Split the proposal to enable voting on imaginary literal suffixes for `std::complex` separately. This is the `std::complex` only part. All other parts (including the demo implementation) are given in N3642.
- Discussion about whether the "Imaginary Literal Suffixes for std::complex" should be `noexcept` as well as `constexpr`. `complex<float>` converts from `double`, and hence can have undefined behavior, i.e., a narrow contract, and hence maybe should not be `noexcept`. Alisdairs suggestion is to advance this paper now and open an issue to look at what should be `noexcept`.

  I conclude no change for `noexcept`. IMHO it is a non-issue for `constexpr` UDL operators, since they are applied by the compiler. Exceptions might only be thrown if called at run-time explicitly, i.e., `operator"" i_f(3.14);` never by using the suffix.
- Change "Effects: creates a complex literal as XXX" to "Returns XXX".
- "[Note: The keyword if is not available as a suffix.]" was criticized by STL for being too helpful, and not consistent with the rest of the standard.

1.2 Changes from N3468

The following changes were made based on input from BSI.

- move implementation code to appendix.
- add discussion on a suffix for `std::string_ref`.
- refer to SI units abbreviations definition.

1.3 Changes from N3402

The following changes result from discussions in Portland.

- drop binary literals and ask core/evolution first if it would be done by core. it should be done in code via a prefix "`0b`".
- drop real-part `std::complex` literals `operator"" r()` and make the imaginary-part operators `i`, `il`, and `i_f`.
- drop mechanics for type deduction of integers from standard. They should be part of type traits anyway and should be also an integral_constant. This paper still provides their updated implementation as an example for potential implementors.
- make the floating point representation type of `chrono::duration` floating point literals unspecified.

2 Introduction

see N3642 for other proposed UDL operators.

Based on the discussion this paper proposes to include UDL operators for the following library component.

- `std::complex`, suffixes `i, il, i_f` in inline namespace `std::literals::complex_-literals`

2.1 Rationale

see N3642.

2.2 Open Issues Discussed

see N3642.

2.3 Acknowledgements

Acknowledgements go to the original authors of the sequence of papers the lead to inclusion of UDL in the standard and to the participants of the discussion on UDL on the reflector. Special thanks to Daniel Krügler for tremendous feedback on all drafts and to Jonathan Wakely for guidelines on GCC command line options. Thanks to Alberto Ganesh Barbati for feedback on duration representation overflow and suggestion for also providing the number parsing as a standardized library component. Thanks to Bjarne Stroustrup for suggesting to add more rationale to the proposal.

Thanks to all participants in the discussions in groups "library" and "evolution" in Portland.

Thanks to the BSI reviewers of N3468 and Roger Orr as their spokesperson.

Thanks to the library group participants in Bristol discussing the paper N3531 and draft versions of this paper for their feedback and making me aware of it in time.

3 Proposed Library Additions

The addition proposed in the next section that are not specific to user-defined literals for imaginary numbers are identical to those proposed in N3642. They are included here only to allow voting separately on the two papers. I propose that N3642 is voted first and if successful the proposed change in the next section is skipped by the editor.

It must be decided in which section to actually put the proposed changes. I suggest we add them to the corresponding library parts, where appropriate. The following change should only be applied if N3642 is not voted for inclusion but this paper is.

3.1 namespace literals for collecting standard UDLs

As a common schema this paper proposes to put all suffixes for user defined literals in separate inline namespaces that are below the inline namespace `std::literals`. [*Note:* This allows a user either to do a `using namespace std::literals;` to import all literal operators from the standard available through header file inclusion, or to use `using namespace std::complex_literals;` to just obtain the literals operators for a specific type. — *end note*]

3.2 Imaginary Literal Suffixes for std::complex

Make the following additions and changes to library subclause 26.4 [complex.numbers] to accommodate user-defined literal suffixes for complex number literals.

Insert in subclause 26.4.1 [complex.syn] in the synopsis at the appropriate place the namespace std::literals::complex_literals

```
namespace std{
inline namespace literals{
```

```
inline namespace complex_literals{
constexpr complex<long double> operator"" il(long double);
constexpr complex<long double> operator"" il(unsigned long long);
constexpr complex<double> operator"" i(long double);
constexpr complex<double> operator"" i(unsigned long long);
constexpr complex<float> operator"" i_f(long double);
constexpr complex<float> operator"" i_f(unsigned long long);
}}}
```

Insert a new subclause before subclause 26.4.9 [ccmplx] as follows

3.3 Suffix for complex number literals [complex.literals]

1 This section describes literal suffixes for constructing complex number literals. The suffixes `i`, `il`, `i_f` create complex numbers with their imaginary part denoted by the given literal number and the real part being zero of the types `complex<double>`, `complex<long double>`, and `complex<float>` respectively.

```
constexpr complex<long double> operator"" il(long double d);
constexpr complex<long double> operator"" il(unsigned long long d);
```

2 *Returns:* `complex<long double>{0.0L, static_cast<long double>(d)}`.

```
constexpr complex<double> operator"" i(long double d);
constexpr complex<double> operator"" i(unsigned long long d);
```

3 *Returns:* `complex<double>{0.0, static_cast<double>(d)}`.

```
constexpr complex<float> operator"" i_f(long double d);
constexpr complex<float> operator"" i_f(unsigned long long d);
```

4 *Returns:* `complex<float>{0.0f, static_cast<float>(d)}`.