

N4255

Revision of: N3254

Date: 2014-11-03

Authors :

Michael Spertus - [mike\\_spertus@symantec.com](mailto:mike_spertus@symantec.com)

John Maddock - [boost.regex@virgin.net](mailto:boost.regex@virgin.net)

## Proposed resolution for US104: Allocator-aware regular expressions (rev 3)

### Rationale

The standard library contains many data structures that hold user data, such as containers (23), strings (21), string streams (27.8), and string buffers (27.8). These data structures are allocator-aware, using storage allocators (20.2.5) to manage their storage requirements and internal pointer representations. However, regular expressions (28) are not allocator-aware.

Such inconsistent treatment creates complexity by giving C++ programmers another inconsistency to remember. More importantly, the reasons for making the above data structures allocator-aware apply equally well to regular expressions. At Symantec, we have had to engineer around the lack of allocator-support in regular expressions because `tr1::regex` objects cannot be placed in shared memory as they cannot be assigned a shared memory allocator. While the overall notion of allocators has generated some controversy within the C++ community, it seems clear that as long as other C++ containers are allocator-aware, regular expressions should be also.

Finally, it should be noted that C++0x regular expressions already differ from TR1 regular expressions, so it is possible to rectify this situation for C++0x, but once normatively standardized, it will be extremely difficult if not impossible to make such a breaking change.

### Approach

The proposed wording below was arrived at *mutatis mutandis* from the corresponding wording from `string`. However, a few comments are in order.

1. Class `match_results` (28.10) currently uses an allocator. This allocator has no relationship to the allocator used internally in the regular expression, as has always been the case (regarding the current `regex` as regular expression using the standard allocator). Similar comments apply to string allocators.
2. Although most C++ containers consistently use pointer traits internally, `regex_traits` use locales, so they cannot be, say, shared using an interprocess allocator. Note that `basic_stringstream` and `basic_stringbuf` already use both allocators and locales, so supporting allocators in regular expressions do not introduce any new problems.

Although these locale considerations prevent regular expressions using `std::regex_traits` from

being shared between processes, there is no reason to prevent users from defining their own allocator-aware regex traits. To facilitate this, based on `uses_allocator<traits, Allocator>`, `std::basic_regex` either default constructs `regex_traits` (e.g. `std::regex_traits`) or constructs from an allocator. We allow `imbue` to throw a `regex_error` (although `std::regex_traits` doesn't) as user-defined `regex_traits` may not be able to handle all possible locales (e.g., some custom locales).

## Status

A working implementation is available. In addition, an exemplary user-defined `regex_traits` is available that allows locales to be passed between process based on their locale names.

## Wording

In 17.6.3.5p1, change the final sentence to

All of the string types (Clause 21), containers (Clause 23) (except array (Clause 23)), string buffers and string streams (Clause 27), **regular expressions (clause 28)**, and `match_results` (Clause 28) are parameterized in terms of allocators.

Apply the following changes to clause 28. Note that if this is applied to a TS, include wording to the effect that the text from clause 28 should be included with the following changes.

### 28.3 Requirements [re.req]

In the entry for `u.imbue` in table 137 add: **Report error by throwing exception of type `regex_error`**

At the end of 28.3, add the following paragraph.

**Class template `basic_regex` satisfies the requirements for an Allocator-aware container (Table 99), except that `basic_regex` does not construct or destroy any objects using `allocator_traits<Alloc>::construct` and `allocator_traits<Alloc>::destroy`**

### 28.4 Header <regex> synopsis

...

*// 28.8, class template `basic_regex`:*

```
template <class charT, class traits = regex_traits<charT>,  
         class Allocator = allocator<charT> > class basic_regex;
```

...

*// 28.8.6, `basic_regex` swap:*

```
template <class charT, class traits, class Allocator >  
void swap(basic_regex<charT, traits, Allocator>& e1,  
         basic_regex<charT, traits, Allocator>& e2);
```

...

*// 28.11.2, function template `regex_match`:*

```
template <class BidirectionalIterator, class Allocator AllocMatch,  
         class charT, class traits, class Allocator >
```

```
bool regex_match(BidirectionalIterator first, BidirectionalIterator last,  
                match_results<BidirectionalIterator, Allocator AllocMatch>& m,
```

```

        const basic_regex<charT, traits, Allocator>& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);
template <class BidirectionalIterator, class charT, class traits, class Allocator>
bool regex_match(BidirectionalIterator first, BidirectionalIterator last,
        const basic_regex<charT, traits, Allocator>& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);
template <class charT, class AllocatorAllocMatch, class traits, class Allocator>
bool regex_match(const charT* str,
        match_results<const charT*, AllocatorAllocMatch >& m,
        const basic_regex<charT, traits, Allocator>& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);

template <class ST, class SA, class AllocatorAllocMatch, class charT,
        class traits, class Allocator>

bool regex_match(const basic_string<charT, ST, SA>& s,
        match_results<
        typename basic_string<charT, ST, SA>::const_iterator,
        AllocatorAllocMatch >& m,
        const basic_regex<charT, traits, Allocator>& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);
template <class charT, class traits, class Allocator >
bool regex_match(const charT* str,
        const basic_regex<charT, traits, Allocator >& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);
template <class ST, class SA, class charT, class traits, class Allocator >
bool regex_match(const basic_string<charT, ST, SA>& s,
        const basic_regex<charT, traits, Allocator>& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);

//28.11.3, function template regex_search:
template <class BidirectionalIterator, class AllocatorAllocMatch,
        class charT, class traits, class Allocator >

bool regex_search(BidirectionalIterator first, BidirectionalIterator last,
        match_results<BidirectionalIterator, AllocatorAllocMatch >& m,
        const basic_regex<charT, traits, Allocator >& e,

        regex_constants::match_flag_type flags =
        regex_constants::match_default);
template <class BidirectionalIterator, class charT, class traits, class Allocator >
bool regex_search(BidirectionalIterator first, BidirectionalIterator last,
        const basic_regex<charT, traits, Allocator >& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);
template <class charT, class Allocator, class traits, class Allocator >
bool regex_search(const charT* str,
        match_results<const charT*, Allocator>& m,
        const basic_regex<charT, traits, Allocator >& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);
template <class charT, class traits, class Allocator >
bool regex_search(const charT* str,
        const basic_regex<charT, traits, Allocator >& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);
template <class ST, class SA, class charT, class traits, class Allocator >
bool regex_search(const basic_string<charT, ST, SA>& s,
        const basic_regex<charT, traits, Allocator >& e,
        regex_constants::match_flag_type flags =
        regex_constants::match_default);

```

```

template <class ST, class SA, class AllocatorAllocMatch, class charT,
         class traits, class Allocator >
bool regex_search(const basic_string<charT, ST, SA>& s,
                 match_results<
                 typename basic_string<charT, ST, SA>::const_iterator,
                 AllocatorAllocMatch >& m,
                 const basic_regex<charT, traits, Allocator >& e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);

```

// 28.11.4, function template `regex_replace`:

```

template <class OutputIterator, class BidirectionalIterator,
         class traits, class charT, class ST, class SA, class Allocator >
OutputIterator
regex_replace(OutputIterator out,
             BidirectionalIterator first, BidirectionalIterator last,
             const basic_regex<charT, traits, Allocator >& e,
             const basic_string<charT, ST, SA>& fmt,
             regex_constants::match_flag_type flags =
             regex_constants::match_default);
template <class OutputIterator, class BidirectionalIterator,
         class traits, class charT, class Allocator >
OutputIterator
regex_replace(OutputIterator out,
             BidirectionalIterator first, BidirectionalIterator last,
             const basic_regex<charT, traits, Allocator >& e,
             const charT* fmt,
             regex_constants::match_flag_type flags =
             regex_constants::match_default);
template <class traits, class charT, class ST, class SA,
         class FST, class FSA, class Allocator >
basic_string<charT, ST, SA>
regex_replace(const basic_string<charT, ST, SA>& s,
             const basic_regex<charT, traits, Allocator >& e,
             const basic_string<charT, FST, FSA>& fmt,
             regex_constants::match_flag_type flags =
             regex_constants::match_default);
template <class traits, class charT, class ST, class SA, class Allocator >
basic_string<charT, ST, SA>
regex_replace(const basic_string<charT, ST, SA>& s,
             const basic_regex<charT, traits, Allocator >& e,
             const charT* fmt,
             regex_constants::match_flag_type flags =
             regex_constants::match_default);
template <class traits, class charT, class ST, class SA, class Allocator >
basic_string<charT>
regex_replace(const charT* s,
             const basic_regex<charT, traits, Allocator >& e,
             const basic_string<charT, ST, SA>& fmt,
             regex_constants::match_flag_type flags =
             regex_constants::match_default);
template <class traits, class charT, class Allocator >
basic_string<charT>
regex_replace(const charT* s,
             const basic_regex<charT, traits, Allocator >& e,
             const charT* fmt,
             regex_constants::match_flag_type flags =
             regex_constants::match_default);

```

...

### 28.5.3 Implementation-defined `error_type` [re.err]

Add an error type to `regex_constants` and table 140 called `error_locale` with Error Condition: Unable to imbue with a locale.

## 28.8 Class template `basic_regex` [re.regex]

```
...
namespace std {
    template <class charT,
              class traits = regex_traits<charT>,
              class Allocator = allocator<charT> >
    class basic_regex {
    public:
        // types:
        typedef charT value_type;
        typedef regex_constants::syntax_option_type flag_type;
        typedef typename traits::locale_type locale_type;
        typedef Allocator allocator_type;

        ...

        // 28.8.2, construct/copy/destroy:
        basic_regex() : basic_regex(Allocator()) {}
        explicit basic_regex(const Allocator &a);
        explicit basic_regex(const charT* p,
                             flag_type f = regex_constants::ECMAScript,
                             const Allocator &a = Allocator());
        basic_regex(const charT* p, size_t len, flag_type f,
                   const Allocator &a = Allocator());
        basic_regex(const basic_regex&);
        basic_regex(basic_regex&&);
        template <class ST, class SA>
            explicit basic_regex(const basic_string<charT, ST, SA>& p,
                                flag_type f = regex_constants::ECMAScript,
                                const Allocator &a = Allocator());
        template <class ForwardIterator>
            basic_regex(ForwardIterator first, ForwardIterator last,
                       flag_type f = regex_constants::ECMAScript,
                       const Allocator &a = Allocator());
        basic_regex(initializer_list<charT>,
                   flag_type = regex_constants::ECMAScript,
                   const Allocator &a = Allocator());

        // 28.8.10, allocator (Section number may be chosen editorially):
        allocator_type get_allocator() const noexcept;
    };
};
```

### 28.8.2 `basic_regex` constructors [re.regex.construct]

```
basic_regex(const Allocator &a = Allocator());
```

*Effects:* Constructs an object of class `basic_regex` that does not match any character sequence.

```
basic_regex(const charT* p, flag_type f = regex_constants::ECMAScript,
            const Allocator &a = Allocator());
```

```
basic_regex(const charT* p, size_t len, flag_type f,
            const Allocator &a = Allocator());
```

```
template <class ST, class SA>
    basic_regex(const basic_string<charT, ST, SA>& s,
               flag_type f = regex_constants::ECMAScript,
               const Allocator &a = Allocator());
template <class ForwardIterator>
```

```

basic_regex(ForwardIterator first, ForwardIterator last,
            flag_type f = regex_constants::ECMAScript
            const Allocator &a = Allocator());
}
basic_regex(initializer_list<charT> il,
            flag_type f = regex_constants::ECMAScript
            const Allocator &a = Allocator());

```

### 28.8.5 basic\_regex locale [re.regex.locale]

locale\_type imbue(locale\_type loc);

*Effects:* Returns the result of traits\_inst.imbue(loc) where traits\_inst is a (default initialized if uses\_allocator<traits, Allocator> (20.9.2.2) has a base characteristic (20.7.1) of false and initialized from get\_allocator() if uses\_allocator<traits, Allocator> has a base characteristic of true)

instance of the template type argument traits stored within the object. After a call to imbue the basic\_regex object does not match any character sequence.

locale\_type getloc() const;

*Effects:* Returns the result of traits\_inst.getloc() where traits\_inst is a (default initialized if uses\_allocator<traits, Allocator> has a base characteristic (20.7.1) of false and initialized from get\_allocator() if uses\_allocator<traits, Allocator> has a base characteristic of true)

instance of the template parameter traits stored within the object.

### 28.8.8 basic\_regex allocator [re.regex.allocator]

allocator\_type get\_allocator() const noexcept;

*Returns:* a copy of the Allocator object used to construct the basic\_regex or, if that allocator has been replaced, a copy of the most recent replacement.

**Note:** The next few sections simply update signatures as in the header

### 28.11.2 regex\_match [re.alg.match]

```

template <class BidirectionalIterator, class AllocatorAllocMatch,
         class charT, class traits, class Allocator>

```

```

bool regex_match(BidirectionalIterator first, BidirectionalIterator last,
                match_results<BidirectionalIterator, AllocatorAllocMatch>& m,
                const basic_regex<charT, traits, Allocator>& e,

                regex_constants::match_flag_type flags =
                regex_constants::match_default);

```

➤ (until after paragraph 3)

```

template <class BidirectionalIterator, class charT, class traits, class Allocator>
bool regex_match(BidirectionalIterator first, BidirectionalIterator last,
                const basic_regex<charT, traits, Allocator>& e,
                regex_constants::match_flag_type flags =
                regex_constants::match_default);

```

➤ (until after paragraph 4)

```

template <class charT, class AllocatorAllocMatch, class traits, class Allocator>

```

```

bool regex_match(const charT* str,
                 match_results<const charT*, AllocatorAllocMatch >& m,
                 const basic_regex<charT, traits, Allocator>& e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);

```

› (until after paragraph 5)

```

template <class ST, class SA, class AllocatorAllocMatch, class charT,
         class traits, class Allocator>

bool regex_match(const basic_string<charT, ST, SA>& s,
                 match_results<
                 typename basic_string<charT, ST, SA>::const_iterator,
                 AllocatorAllocMatch >& m,
                 const basic_regex<charT, traits, Allocator>& e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);

```

› (until after paragraph 6)

```

template <class charT, class traits, class Allocator>
bool regex_match(const charT* str,
                 const basic_regex<charT, traits, Allocator>& e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);

```

› (until after paragraph 7)

```

template <class ST, class SA, class charT, class traits, class Allocator>
bool regex_match(const basic_string<charT, ST, SA>& s,
                 const basic_regex<charT, traits, Allocator>& e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);

```

### 28.11.3 regex\_search [re.alg.search]

```

template <class BidirectionalIterator, class AllocatorAllocMatch,
         class charT, class traits, class Allocator>

bool regex_search(BidirectionalIterator first, BidirectionalIterator last,
                 match_results<BidirectionalIterator, AllocatorAllocMatch >& m,
                 const basic_regex<charT, traits, Allocator>& e,

                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);

```

› (until after paragraph 3)

```

template <class charT, class Allocator, class traits, class Allocator>
bool regex_search(const charT* str,
                 match_results<const charT*, Allocator>& m,
                 const basic_regex<charT, traits, Allocator>& e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);

```

› (until after paragraph 4)

```

template <class ST, class SA, class charT, class traits, class Allocator>

```

```
bool regex_search(const basic_string<charT, ST, SA>& s,
                 const basic_regex<charT, traits, Allocator> & e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);
```

⌋ (until after paragraph 5)

```
template <class BidirectionalIterator, class charT, class traits, class Allocator >
bool regex_search(BidirectionalIterator first, BidirectionalIterator last,
                 const basic_regex<charT, traits, Allocator> & e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);
```

⌋ (until after paragraph 6)

```
template <class charT, class traits, class Allocator >
bool regex_search(const charT* str,
                 const basic_regex<charT, traits, Allocator> & e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);
```

⌋ (until after paragraph 7)

```
template <class ST, class SA, class charT, class traits, class Allocator >
bool regex_search(const basic_string<charT, ST, SA>& s,
                 const basic_regex<charT, traits, Allocator> & e,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);
```

#### 28.11.4 regex\_replace [re.alg.replace]

```
template <class OutputIterator, class BidirectionalIterator,
         class traits, class charT, class ST, class SA, class Allocator >
OutputIterator
regex_replace(OutputIterator out,
             BidirectionalIterator first, BidirectionalIterator last,
             const basic_regex<charT, traits, Allocator> & e,
             const basic_string<charT, ST, SA>& fmt,
             regex_constants::match_flag_type flags =
             regex_constants::match_default);
template <class OutputIterator, class BidirectionalIterator,
         class traits, class charT, class Allocator >
OutputIterator
regex_replace(OutputIterator out,
             BidirectionalIterator first, BidirectionalIterator last,
             const basic_regex<charT, traits, Allocator> & e,
             const charT* fmt,
             regex_constants::match_flag_type flags =
             regex_constants::match_default);
```

⌋ (until after paragraph 2)

```
template <class traits, class charT, class ST, class SA,
         class FST, class FSA, class Allocator >
basic_string<charT, ST, SA>
regex_replace(const basic_string<charT, ST, SA>& s,
             const basic_regex<charT, traits, Allocator> & e,
             const basic_string<charT, FST, FSA>& fmt,
             regex_constants::match_flag_type flags =
             regex_constants::match_default);
template <class traits, class charT, class ST, class SA, class Allocator >
basic_string<charT, ST, SA>
```



```

regex_replace(const basic_string<charT, ST, SA>& s,
              const basic_regex<charT, traits, Allocator >& e,
              const charT* fmt,
              regex_constants::match_flag_type flags =
                regex_constants::match_default);

```

› (until after paragraph 4)

```

template <class traits, class charT, class ST, class SA, class Allocator >
  basic_string<charT>
  regex_replace(const charT* s,
               const basic_regex<charT, traits, Allocator >& e,
               const basic_string<charT, ST, SA>& fmt,
               regex_constants::match_flag_type flags =
                 regex_constants::match_default);
template <class traits, class charT, class Allocator >
  basic_string<charT>
  regex_replace(const charT* s,
               const basic_regex<charT, traits, Allocator >& e,
               const charT* fmt,
               regex_constants::match_flag_type flags =
                 regex_constants::match_default);

```

## 28.13 Modified ECMAScript regular expression grammar [re.grammar]

›

Objects of type specialization of `basic_regex` store within themselves a `default-constructed` instance of their traits template parameter, henceforth referred to as `traits_inst`. It is default constructed if `uses_allocator<traits, Allocator>` has a base characteristic (20.7.1) of false and constructed from `get_allocator()` if `uses_allocator<traits, Allocator>` has a base characteristic of true. This `traits_inst` object is used to support localization of the regular expression; `basic_regex` object member functions shall not call any locale dependent C or C++ API, including the formatted string input functions. Instead they shall call the appropriate traits member function to achieve the required effect.