Proposed resolution for US104: Allocator-aware regular expressions (rev 4)

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 construct s 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: Reports an error by throwing an 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
96), 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)
  noexcept(noexcept(e1.swap(e2)));
...

// 28.11.2, function template regex_match:
template <class BidirectionalIterator, class Allocator = Allocator, class charT, class traits, class RA>
bool regex_match(BidirectionalIterator first, BidirectionalIterator last,
match_results<BidirectionalIterator, Allocator, RA>& m,
  const basic_regex<charT, traits, RA>& e,
  regex_constants::match_flag_type flags =

regex_constants::match_default);

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

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

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

// 28.11.3, function template regex_search:
template <class BidirectionalIterator, class AllocatorRA, class charT, class traits, class RA>
bool regex_search(BidirectionalIterator first, BidirectionalIterator last,
match_results<BidirectionalIterator, AllocatorRA>& m,
const basic_regex<charT, traits, RA>& e,
regex_constants::match_flag_type flags =
regex_constants::match_default);

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

template <class ST, class SA, class AllocatorRA, class charT, class traits, class RA>
bool regex_search(const basic_string<charT, ST, SA>& s,
match_results<
typename basic_string<charT, ST, SA>::const_iterator,
AllocatorRA>& m,
const basic_regex<charT, traits, RA>& 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 RA>
OutputIterator
regex_replace(OutputIterator out,
    BidirectionalIterator first, BidirectionalIterator last,
    const basic_regex<charT, traits, RA>& 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 RA>
OutputIterator
regex_replace(OutputIterator out,
    BidirectionalIterator first, BidirectionalIterator last,
    const basic_regex<charT, traits, RA>& 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 RA>
basic_string<charT, ST, SA>
regex_replace(const basic_string<charT, ST, SA>& s,
    const basic_regex<charT, traits, RA>& 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 RA>
basic_string<charT, ST, SA>
regex_replace(const charT* s,
    const basic_regex<charT, traits, RA>& e,
    const basic_string<charT, ST, SA>& fmt,
    regex_constants::match_flag_type flags =
    regex_constants::match_default);

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

...

28.5.3 Implementation-defined error_type [re.err]
namespace std::regex_constants {
    typedef T3 error_type;
    constexpr error_type error_collate = unspecified;
    constexpr error_type error_ctype = unspecified;
    constexpr error_type error_escape = unspecified;
    constexpr error_type error_backref = unspecified;
    constexpr error_type error_brack = unspecified;
    constexpr error_type error_paren = unspecified;
    constexpr error_type error_brace = unspecified;
    constexpr error_type error_badbrace = unspecified;
    constexpr error_type error_range = unspecified;
    constexpr error_type error_space = unspecified;
    constexpr error_type error_badrepeat = unspecified;
    constexpr error_type error_complexity = unspecified;
    constexpr error_type error_stack = unspecified;
    constexpr error_type error_locale = unspecified;
}

The type error_type is an implementation-defined enumerated type (17.5.2.1.2). Values of type error_type represent the error conditions described in Table 139:

<table>
<thead>
<tr>
<th>Value</th>
<th>Error condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_collate</td>
<td>The expression contained an invalid collating element name.</td>
</tr>
<tr>
<td>error_ctype</td>
<td>The expression contained an invalid character class name.</td>
</tr>
<tr>
<td>error_escape</td>
<td>The expression contained an invalid escaped character, or a trailing escape.</td>
</tr>
<tr>
<td>error_backref</td>
<td>The expression contained an invalid back reference.</td>
</tr>
<tr>
<td>error_brack</td>
<td>The expression contained mismatched [ and ].</td>
</tr>
<tr>
<td>error_paren</td>
<td>The expression contained mismatched ( and ).</td>
</tr>
<tr>
<td>error_brace</td>
<td>The expression contained mismatched ( and ).</td>
</tr>
<tr>
<td>error_badbrace</td>
<td>The expression contained an invalid range in a () expression</td>
</tr>
<tr>
<td>error_range</td>
<td>The expression contained an invalid character range, such as [b-a] in most encodings.</td>
</tr>
<tr>
<td>error_space</td>
<td>There was insufficient memory to convert the expression into a finite state machine.</td>
</tr>
<tr>
<td>error_badrepeat</td>
<td>One of *?+{ was not preceded by a valid regular expression.</td>
</tr>
<tr>
<td>error_complexity</td>
<td>The complexity of an attempted match against a regular expression exceeded a pre-set level.</td>
</tr>
<tr>
<td>error_stack</td>
<td>There was insufficient memory to determine whether the regular expression could match the specified character sequence.</td>
</tr>
<tr>
<td>error_locale</td>
<td>Unable to imbue with a locale</td>
</tr>
</tbody>
</table>

28.8 Class template basic_regex [re.regex]
28.8.2 basic_regex constructors [re.regex.construct]

`basic_regex();` noexcept(noexcept(Allocator())); 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& e);
basic_regex(const basic_regex&, const Allocator&);
basic_regex(const Allocator&);
basic_regex(const basic_regex&&, const Allocator&);
basic_regex(const Allocator&)

Constructs an object of class `basic_regex` as a copy of the object `e`.

Effects: Constructs and object of class `basic_regex` as a copy of the object `e`.

Postconditions: flags() and mark_count() return e.flags() and e.mark_count(), respectively.
basic_regex(const basic_regex& e, const Allocator& a)  
Effects: Constructs and object of class basic_regex as a copy of the object e.  
Postconditions: flags() and mark_count() return the values that e.flags() and e.mark_count(), respectively, had before construction. e is in a valid state with unspecified value.

template <class ST, class SA>  
basic_regex(const basic_string<CharT, ST, SA>& s,  
    flag_type f = regex_constants::ECMAScript,  
    const Allocator& a = Allocator());  
Throws: regex_error if s is not a valid regular expression.  
Effects: Constructs an object of class basic_regex; the object’s internal finite state machine is constructed from the regular expression contained in the string s, and interpreted according to the flags specified in f.  
Postconditions: flags() returns f.mark_count() returns the number of marked sub-expressions within the expression.

template <class ForwardIterator>  
basic_regex(ForwardIterator first, ForwardIterator last, flag_type f =  
    regex_constants::ECMAScript, const Allocator& a = Allocator());  
Throws: regex_error if the sequence [first, last) is not a valid regular expression.  
Effects: Constructs an object of class basic_regex; the object’s internal finite state machine is constructed from the regular expression contained in the sequence [first, last), and interpreted according to the flags specified in f.  
Postconditions: flags() returns f.mark_count() returns the number of marked sub-expressions within the expression.

basic_regex(initializer_list<CharT> il,  
    flag_type f = regex_constants::ECMAScript,  
    const Allocator& a = Allocator());  
Effects: Same as basic_regex(il.begin(), il.end(), f, a).

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.

28.8.6 basic_regex swap [re.regex.swap]

void swap(basic_regex& e)  
noexcept(algorithm_traits<Allocator>::propagate_on_container_move_assignment::value ||  
algorithm_traits<Allocator>::is_always_equal::value);  
Effects: Swaps the contents of the two regular expressions.  
Postcondition: *this contains the regular expression that was in e, e contains the regular expression that was in *this.  
Complexity: Constant time.

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.
28.11.2 regex_match [re.alg.match]

```cpp
template<class BidirectionalIterator, class Allocator, class chart, class Traits, class RA>
bool regex_match(BidirectionalIterator first, BidirectionalIterator last,
match_results<BidirectionalIterator, Allocator> &m,
const basic_regex<chart, traits, RA>& e,
regex_constants::match_flag_type flags =
regex_constants::match_default);
```

(.until after paragraph 3)

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

**Effects:** Behaves “as if” by constructing an instance of `match_results<BidirectionalIterator>` `what`, and then returning the result of `regex_match(first, last, what, e, flags).

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

**Returns:** `regex_match(str, str + char_traits<char>::length(str), m, e, flags)`.

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

**Returns:** `regex_match(s.begin(), s.end(), m, e, flags)`.

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

**Returns:** `regex_match(str, str + char_traits<chart>::length(str), e, flags)`.

28.11.3 regex_search [re.alg.search]

```cpp
template<class BidirectionalIterator, class Allocator, class chart, class traits, class RA>
bool regex_search(BidirectionalIterator first, BidirectionalIterator last,
match_results<BidirectionalIterator, Allocator> &m,
const basic_regex<charT, traits, RA>& e,
regex_constants::match_flag_type flags =
regex_constants::match_default);
```
template <class charT, class Alloc, class traits, class RA>
bool regex_search(const charT* str,
    match_results<const charT*, Alloc>& m,
    const basic_regex<charT, traits, RA>& e,
    regex_constants::match_flag_type flags =
    regex_constants::match_default);

Returns: The result of regex_search(str, str + char_traits<char>::length(str), m, e, flags).

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

Returns: The result of regex_search(s.begin(), s.end(), m, e, flags).

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

Effects: Behaves "as if" by constructing an object what of type
match_results<BidirectionalIterator>, and then returning the result of regex_search(first, last, what, e, flags).

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

Returns: regex_search(str, str + char_traits<char>::length(str), e, flags).

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

Returns: regex_search(s.begin(), s.end(), e, flags).

28.11.4 regex_replace [re.alg.replace]

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

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

(template <class traits, class charT, class ST, class SA,
   class FST, class FSA, class RA>
   basic_string<charT, ST, SA>
   regex_replace(const basic_string<charT, ST, SA>& s,
                  const basic_regex<charT, traits, RA>& 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 RA>
   basic_string<charT, ST, SA>
   regex_replace(const basic_string<charT, ST, SA>& s,
                  const basic_regex<charT, traits, RA>& e, const
                  charT* fmt, regex_constants::match_flag_type
                  flags =
                  regex_constants::match_default);
   Effects: Constructs an empty string result of type basic_string<charT, ST, SA> and calls
   regex_replace(back_inserter(result), s.begin(), s.end(), e, fmt, flags).

   Returns: result.

   template <class traits, class charT, class ST, class SA, class RA>
   basic_string<charT>
   regex_replace(const charT* s,
                 const basic_regex<charT, traits, RA>& e, const
                 basic_string<charT, ST, SA>& fmt,
                 regex_constants::match_flag_type flags =
                 regex_constants::match_default);
   template <class traits, class charT, class RA>
   basic_string<charT>
   regex_replace(const charT* s,
                 const basic_regex<charT, traits, RA>& e, const
                 charT* fmt, regex_constants::match_flag_type
                 flags =
                 regex_constants::match_default);
   Effects: Constructs an empty string result of type basic_string<charT> and calls
   regex_replace(back_inserter(result), s, s + char_traits<charT>::length(s), e, fmt, flags).

   Returns: result.

28.13 Modified ECMAScript regular expression grammar [re.grammar]

The regular expression grammar recognized by basic_regex objects constructed with the ECMAScript flag is that
specified by ECMA-262, except as specified below.

Objects of type specialization of basic_regex store within themselves an 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 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.
```