Document No: N3171=10-0161
Date: 2010-10-18
Reply to: Michael Spertus
mike_spertus@symantec.com

Proposed resolution for US104: Allocator-aware regular expressions

Rationale

For a comprehensive discussion of the rationale for allocator-aware regular expressions and justification of its status as a defect, see Pablo Halpern's paper D3172, "Allocators for stringstream (US140)." Briefly, the value of allocators depends greatly on their consistent use in the standard library. As stated there, all value-semantic classes that allocate memory appear, such as containers (§23), strings (§21), shared pointers(§20.9.11.2) and function (§20.8.14), make consistent use of allocators, with the (apparently) sole exceptions of regular expressions (§28) and string streams (§27.8). This inconsistency is a defect as described in D3172 and as stated in US104 and US140.

I would merely add two small points. (1) At Symantec, we have found this issue to be more than just theoretical and having considerable practical importance. We have regularly 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. (2) it should be noted that C++0x regular expressions already differ from the non-normative 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, regular expressions can be imbued with a locale. Locales typically use raw pointers internally, so they cannot be, for example, shared using an interprocess allocator. Note that `basic_stringstream` already has the same limitation. The LWG issue entitled "Requirements on internal pointer representations in containers" clarifies wording on this.

   In the default case where the default locale is used, it is desirable to simply encode in the string stream or regular expression that the default locale is being used, so that such types are not gratuitously excluded from, say, being placed in shared memory.

Wording

At the end of 28.3, add a paragraph (pending acceptance of the LWG issue entitled “Requirements on internal pointer representations in containers”):

The requirements of 23.2.1 paragraphs 3 and 8 apply to the regular expression library except that regular expressions may contain pointers to the locale used by the regular expression. If the locale compares equal to the default locale, these pointers must be null.

In all cases, make the following changes

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 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);
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);
```

…

28.8 Class template basic_regex [re.regex]

…

```
namespace std {
  template <class charT,
            class traits = regex_traits<charT>,
            class Allocator >
  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(const Allocator &a = Allocator());
       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());

…
```

28.8.2 basic_regex constructors [re.regex.construct]

basic_regex();

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);
```