Doc. No.: 97-0058=N1096
Date: 1997-07-17
Project: Programming Language C++
Reply to: Nathan Myers <ncm@cantrip.org>
Dietmar Kuehl <dk@asgard.in-berlin.de>

Clarifications of Locale Money and Time Formats

This document addresses issues 22-021, 22-030, 22-031, and 22-034 as found in 97-0036=N1074 (Library Issues List for CD2 - Version 6).

[Replace 22.2.6.3 (lib.locale.moneypunct) paragraph 1 with:]

The moneypunct<> facet defines monetary formatting parameters used by money_get<> and money_put<>. A monetary format is a sequence of four components, specified by a _pattern_ value _p_, such that the _part_ value _static_cast<part>(p.field[i])_ determines the _i_th component of the format. [footnote: an array of _char_ rather than of _part_ is specified for _pattern::field_ purely for efficiency.] In the _field_ member of a _pattern_ object, each value _symbol_, _sign_, _value_, and either _space_ or _none_ appear exactly once. The value _none_, if present, is not first; the value _space_, if present, is neither first nor last.

Where _none_ or _space_ appears, whitespace is permitted in the format, except where _none_ appears at the end, in which case no whitespace is permitted. The value _space_ indicates that at least one space is required at that position. Where _symbol_ appears, the sequence of characters returned by _curr_symbol()_ is permitted, and may be required. Where _sign_ appears, the first (if any) of the sequence of characters returned by _positive_sign()_ or _negative_sign()_ (respectively as the monetary value is non-negative or negative) is required. Any remaining characters of the sign sequence are required after all other format components. Where _value_ appears, the absolute numeric monetary value is required.

The format of the numeric monetary value is a decimal number

```
  value ::= units [ decimal-point [ digits ]] |
            decimal-point digits
```

if _frac_digits()_ returns a positive value, or

```
  value ::= units
```

otherwise. The symbol _decimal-point_ indicates the character returned by _decimal_point()_. The other symbols are defined as follows:

```
  units  ::= digits [ thousands-sep units ]
  digits ::= adigit [ digits ]
```

In the syntax specification, the symbol _adigit_ is any of the values _ct.widen(c)_ for _c_ in the range ['0','9']. _ct_ is a reference of type _const ctype<charT>&_ obtained as described in the definitions of _money_get<>_ and _money_put<>_. The symbol _thousands-sep_ is the character returned by _thousands_sep()_. The space character used is the value _ct.widen(' ')_. Whitespace characters are those characters _c_ for which _ct.is(ct.space,c)_ returns _true_. The number of digits required after the decimal point (if any) is exactly the value returned by _frac_digits()_.

The placement of thousands separator characters (if any) is determined by the value returned by _grouping()_, defined identically as the member _numpunct<>::do_grouping()_.

[Eliminate the first four sentences (up to "in any order.") of the Returns: clause of members do_pos_format and do_neg_format (22.2.6.3.2, lib.locale.moneypunct.virtuals, paragraph 7).]

[Eliminate the last sentence of the Returns: clause of the definition of do_positive_sign() and do_negative_sign() (paragraph 5 of the same section).]

[Replace the Effects: clause of money_put<>::do_put (22.2.6.2.2, lib.locale.money.put.virtuals, paragraph 1) with:]

Effects: Writes characters to _s_ according to the format specified by a _moneypunct<charT,intl>_ facet reference _mp_ and the character mapping specified by a _ctype<charT>_ facet reference _ct_ obtained from the locale returned by _str.getlocale(), and _str.flags()_. The argument _units_ is transformed into a sequence of wide characters as if by

```
  ct.widen(buf1,buf1+sprintf(buf1,"%.0lf",units),buf2)
```

for character buffers _buf1_ and _buf2_. If the first character in _digits_ or _buf2_ is equal to _ct.widen('-')_, then the pattern used for formatting is the result of _mp.neg_format()_; else _mp.pos_format()_. Digit characters are written, interspersed with any thousands separators and decimal point specified by the format, in the order they appear (after the optional leading minus sign) in _digits_ or _buf2_. In _digits_, only the optional leading minus sign and the immediately subsequent digit characters (as classified according to _ct_) are used; any trailing characters (including digits appearing after a non-digit character) are ignored. Calls _str.width(0)_.

[Replace the Notes: clause of money_put<>::do_put (22.2.6.2.2, lib.locale.money.put.virtuals, paragraph 2) with:]

Notes: The currency symbol is generated if and only if _(str.flags() & str.showbase)_ is true. If the number of characters generated for the specified format is less than the value returned by _str.width()_ on entry to the function, then copies of _fill_ are inserted as necessary to pad to the specified width. For the value _af_ equal to _(str.flags() & str.adjustfield)_, if _(af_ ==_str.internal)_ is true, the fill characters are placed where _none_ or _space_ appears in the formatting pattern; otherwise, if (_af_ == str.left) is true, they are placed after the other characters; otherwise, they are placed before the other characters. [Note: it is possible, with some combinations of format patterns and flag values, to produce output which cannot be parsed back in using _num_get<>::get_.]

[Replace the Effects: clause of money_get<>::do_get (22.2.6.1.2, lib.locale.money.get.virtuals, paragraph 1) with:]

Effects: Reads characters from _s_ to parse and construct a monetary value according to the format specified by a _moneypunct<charT,intl>_ facet reference _mp_ and the character mapping specified by a _ctype<charT>_ facet reference _ct_ obtained from the locale returned by _str.getloc()_, and _str.flags()_. If a valid sequence is not recognized, sets the argument _err_ to _(err|str.failbit)_ and does not change _units_ or _digits_; otherwise, it does not change _err_. Uses the pattern returned by _mp.neg_format()_ to parse all values. The result is returned as an integral value stored in _units_, or as a sequence of digits possibly preceded by a minus sign (as produced by _ct.widen(c)_ where c is '-' or in the range ['0','9']) stored in _digits_. [Example: the sequence $1,056.23 in a common U.S. locale would yield, for _units_, 105623, or for _digits_, "105623".]

If _mp.grouping() indicates no thousands separators are permitted, any such characters are not read, and parsing is terminated at the point where the first appears. Otherwise, thousands separators are optional; if present, they are checked for correct placement only after all format components have been read.

Where _space_ or _none_ appear in the format pattern, except at the end, optional whitespace (as recognized by _ct.is_) is consumed after any required space. If _(str.flags() & str.showbase)_ is false, the currency symbol is optional and is consumed only if other characters are needed to complete the format; otherwise, the currency symbol is required.

If the first character (if any) in the string _pos_ returned by _mp.positive_sign()_ or the string _neg_ returned by _mp.negative_sign()_ is recognized in the position indicated by _sign_ in the format pattern, it is consumed and any remaining characters in the string are required after all other format components. [Example: If showbase is off, then for a _neg_ value of "()" and a currency symbol of "L", in "(100 L)" the "L" is consumed; but for _neg_ "-", the "L" in "-100 L" is not consumed. --end example] If _pos_ or _neg_ is empty, the sign component is optional, and if no sign is detected the result is given the sign corresponding to the source of the empty string. Otherwise, the character in the indicated position must match the first character of _pos_ or _neg_, and the result is given the corresponding sign. If the first character of each of _pos_ and _neg_ are equal, or if both strings are empty, the result is given a positive sign.

Digits in the numeric monetary component are extracted and placed in _digits_, or into a character buffer _buf1_ for conversion to produce a value for _units_, in the order they appear, preceded by a minus sign if and only if the result is negative. The value _units_ is produced as if by
[footnote: the semantics here are different from _ct.narrow_]

```
  for (int i=0; i<n; ++i)
    buf2[i] = src[find(atoms, atoms+sizeof(src), buf1[i]) - atoms];
  buf2[n] = 0;
  sscanf(buf2, "%Lf", &units);
```

where _n_ is the number of characters placed in _buf1_, _buf2_ is a character buffer, and values _src_ and _atoms_ are defined as if by:

```
  static const char src[] = "0123456789-";
  charT atoms[sizeof(src)];
  ct.widen(src, src+sizeof(src)-1, atoms)
```

[In the prototype declarations in the definition of time_put::put (22.2.5.3.1, lib.locale.time.put.members), add the formal parameter name str for the arguments of type ios_base&.]

[Replace the Effects: clause of the definition of time_put::put (22.2.5.3.1, lib.locale.time.put.members, paragraph 1) as follows:]

Effects: The first form steps through the sequence from _pattern_ to _pat_end_, identifying characters that are part of a format sequence. Each character not part of a format sequence is written to _s_ immediately, and each format sequence, as it is identified, results in a call to do_put; thus, format elements and other characters are interleaved in the output in the order in which they appear in the pattern.
Format sequences are identified by converting each character _c_

to a char value as if by _ct.narrow(c,0)_, where _ct_ is a reference
to _ctype<charT>_ obtained from _str.getloc()_. The first character
of each sequence is equal to '%', followed by an optional modifier
character _mod_
[footnote: Although Standard C defines no modifiers, most vendors do.]
and a format specifier character _spec_ as defined for the function
strftime. If no modifier character is present, _mod_ is 0. For
each valid format sequence identified, calls _do_put(s,str,fill,t,
spec,mod)_.
The second form calls _do_put(s,str,fill,t,format,modifier)_.

[Add to the end of the Effects: clause of the definition of
time_put<>::do_put (22.2.5.3.2, lib.locale.time.put):]

... except that the sequence of characters produced for those
specifiers described as depending on the C locale are instead
implementation-defined. [footnote: implementors are encouraged
to refer to other Standards (such as POSIX) for these definitions.]