Clause 22 diffs

--- lib-locales.new.txt Thu Jul 17 08:28:30 1997
***************
*** 6,9 ****
--- 6,17 ----
    support for character classification and string collation, numeric,
    monetary, and date/time formatting and parsing, and message retrieval.
+ ." Germany 22/ lib.localization, actually a change to lib.iterator.requirement
+ .eN
+ Change clause [lib.iterator.requirement] so that is explicitly says
+ that _all_ functions in the library, that take ranges, have to define
+ a reasonable behavior for valid ranges and may exhibit undefined
+ behavior for invalid ranges. So far, the standard states this only
+ for algorithms.
+ .nE
 .P
The following subclauses describe components for
***************
*** 38,56 ****
    template <class Facet> bool has_facet(const locale&) throw();
 .Ce
 .Cb
   // subclause __lib.locale.convenience_, convenience interfaces:
   template <class charT> bool isspace (charT c, const locale& loc) const
   template <class charT> bool isprint (charT c, const locale& loc) const
   template <class charT> bool iscntrl (charT c, const locale& loc) const
   template <class charT> bool isupper (charT c, const locale& loc) const
   template <class charT> bool islower (charT c, const locale& loc) const
   template <class charT> bool isdigit (charT c, const locale& loc) const
   template <class charT> bool ispunct (charT c, const locale& loc) const
   template <class charT> bool isxdigit(charT c, const locale& loc) const
   template <class charT> bool isalpha (charT c, const locale& loc) const
   template <class charT> bool isgraph (charT c, const locale& loc) const
   template <class charT> charT toupper(charT c, const locale& loc) const
   template <class charT> charT tolower(charT c, const locale& loc) const
 .Ce
 .Cb
*** 46,65 ****
    template <class Facet> bool has_facet(const locale&) throw();
 .Ce
 .Cb
   // subclause __lib.locale.convenience_, convenience interfaces:
   template <class charT> bool isspace (charT c, const locale& loc) const
   template <class charT> bool isprint (charT c, const locale& loc) const
   template <class charT> bool iscntrl (charT c, const locale& loc) const
   template <class charT> bool isupper (charT c, const locale& loc) const
   template <class charT> bool islower (charT c, const locale& loc) const
   template <class charT> bool isdigit (charT c, const locale& loc) const
   template <class charT> bool ispunct (charT c, const locale& loc) const
   template <class charT> bool isxdigit(charT c, const locale& loc) const
   template <class charT> bool isalpha (charT c, const locale& loc) const
   template <class charT> bool isgraph (charT c, const locale& loc) const
   template <class charT> charT toupper(charT c, const locale& loc) const
   template <class charT> charT tolower(charT c, const locale& loc) const
 .Ce
 .Cb
--- 46,65 ----
    template <class Facet> bool has_facet(const locale&) throw();
 .Ce
 .Cb
   // subclause __lib.locale.convenience_, convenience interfaces:
   template <class charT> bool isspace (charT c, const locale& loc) const
   template <class charT> bool isprint (charT c, const locale& loc) const
   template <class charT> bool iscntrl (charT c, const locale& loc) const
   template <class charT> bool isupper (charT c, const locale& loc) const
   template <class charT> bool islower (charT c, const locale& loc) const
   template <class charT> bool isdigit (charT c, const locale& loc) const
   template <class charT> bool ispunct (charT c, const locale& loc) const
   template <class charT> bool isxdigit(charT c, const locale& loc) const
   template <class charT> bool isalpha (charT c, const locale& loc) const
   template <class charT> bool isgraph (charT c, const locale& loc) const
   template <class charT> charT toupper(charT c, const locale& loc) const
   template <class charT> charT tolower(charT c, const locale& loc) const
 .Ce
 .Cb
--- 46,65 ----
    template <class Facet> bool has_facet(const locale&) throw();
template <class charT> bool iscntrl (charT c, const locale& loc);

template <class charT> bool isupper (charT c, const locale& loc);

template <class charT> bool islower (charT c, const locale& loc);

template <class charT> bool isalpha (charT c, const locale& loc);

template <class charT> bool isdigit (charT c, const locale& loc);

template <class charT> bool ispunct (charT c, const locale& loc);

template <class charT> bool isxdigit(charT c, const locale& loc);

template <class charT> bool isalnum (charT c, const locale& loc);

template <class charT> bool isgraph (charT c, const locale& loc);

template <class charT> charT toupper(charT c, const locale& loc);

template <class charT> charT tolower(charT c, const locale& loc);

locale(const locale& other, const char* std_name, category);

template <class Facet> locale(const locale& other, Facet* f);

locale(const locale& other, const locale& one, category);

~locale() throw();  // non-virtual

const locale& operator=(const locale& other) throw();

---

locale(const locale& other, const char* std_name, category);

template <class Facet> locale(const locale& other, Facet* f);

locale(const locale& other, const locale& one, category);

~locale() throw();  // non-virtual

const locale& operator=(const locale& other) throw();

---

locale combine(const locale& other);

****sense, it's just a class interface; at the same time, it's an index into a locale's set of facets.

Access to the facets of a CW locale

is via two member function templates,

and

sense, it's just a class interface; at the same time, it's an index into a locale's set of facets.

Germany 22 22. (or lib.locale) ["pure virtual" - decision: no change]

Netherlands _2211/a lib.locale [use_facet and has_facet aren't members]

USA CD2-22-008 22.1.1 [duplicate issue]

Access to the facets of a CW locale

is via two function templates,

and

****sense, it's just a class interface; at the same time, it's an index into a locale's set of facets.

Netherlands _2211/b 22.1.1 lib.locale [cerberos needs ctor argument]

USA CD2-22-009 22.1.1 lib.locale [duplicate issue]
***************
*** 193,197 ****
   operator<< (basic_ostream<charT,traits>& s, Date d)
{
    typename basic_ostream<charT,traits>::sentry cerberos;
    if (cerberos) {
      ios_base::iostate err = 0;
--- 207,211 ----
   operator<< (basic_ostream<charT,traits>& s, Date d)
{
    typename basic_ostream<charT,traits>::sentry cerberos(s);
    if (cerberos) {
      ios_base::iostate err = 0;
***************
*** 352,356 ****
  num_put<char>, num_put<wchar_t>
  time time_get<char>, time_put<wchar_t>,
  time_put<char>, time_put<wchar_t>
--- 366,370 ----
  num_put<char>, num_put<wchar_t>
--- 358,362 ****
  .TE
  .Te
! time time_get<char>, time_put<wchar_t>,
  time_put<char>, time_put<wchar_t>
--- 372,377 ----
  num_put<char>, num_put<wchar_t>
  time time_get<char>, time_get<wchar_t>,
  time_put<char>, time_put<wchar_t>
--- 388,395 ****
  monetary moneypunct_byname<char,International>,
  moneypunct_byname<wchar_t,International>,
  money_get<char,InputIterator>,
  money_get<wchar_t,InputIterator>,
  money_put<char,OutputIterator>,
  money_put<wchar_t,OutputIterator>
--- 404,409 ----
  monetary moneypunct_byname<char,International>,
  moneypunct_byname<wchar_t,International>,
  money_get<C,InputIterator>,
  money_put<C,OutputIterator>
--- 404,420 ****
For the facets

the implementation provided must depend only on the corresponding facets
and
instantiated on the same character type. Other facets are allowed to
depend on any other facet that is part of a standard category.

In declarations of facets, a template formal parameter with name

+ messages messages_byname<char>, messages_byname<wchar_t>

+ \" Germany 221111/ lib.locale.category [add messages_byname to T53]
+ \" Germany CD2-22-004 22.1.1.1.1 lib.locale.category
+ \" USA CD2-22-004 22.1.1.1.1 [duplicate issue]

The provided implementation of members of facets
calls
only for facet
of types
and
and for locale
the value obtained
by calling member
on the
argument to these functions.

\" Germany _221111/ 22.1.1.1.1 lib.locale.facet [num_get, num_put, message_byname]
\" TODO

In declarations of facets, a template formal parameter with name
***************
*** 458,462 ****
derived from
locale::facet
and containing a declaration as follows:

static ::std::locale::id id;

**** 489,501 ****
derived from
locale::facet
and containing a publicly-accessible declaration as follows:

\" USA CD2-22-010 22.1.1.1.2 [mention "public"]
and containing a publicly-accessible declaration as follows:
.\" Germany CD2-22-019 22 actually 22.1.1.1.2 lib.locale.facet
This is a complete list of requirements; there are no other requirements.
Thus, a facet class need not have a public copy constructor, assignment, default constructor, destructor, etc.

```
static ::std::locale::id id;
```

--- 511,531 ---
argument to the constructor is used for lifetime management.

```
If commanders
```

--- 655,660 ----
Constructors of all

Notes:
The resulting locale has no name.
locale(const locale& other, const locale& one, category cats);

***************
*** 644,647 ****
--- 677,702 ----
.H4 "\f7locale\fP\& members" lib.locale.members
.
+." USA CD2-22-005 [also affecting 22.1.1.2 lib.locale.cons; delete one ctor]
+ .Pb
.locale(const locale& \f6other\fP, const locale& \f6one\fP, category \f6cats\fP);
+ .Pe
***************
*** 652,655 ****
--- 707,719 ----
.CW *this ,
if it has one; otherwise, the string \f5"*"\fP.
+." USA CD2-22-012 22.1.1.2 lib.locale.cons
+ If
+ .CW *this
+ has a name, then
+ .CW locale(name())
+ is equivalent to
+ .CW *this.
+ Details of
+ the contents of the resulting string are otherwise implementation-defined.
."----
.H4 "\f7locale\fP\& operators" lib.locale.operators
***************
*** 752,759 ****
 .La Returns:
 a reference to the corresponding facet of \&\f6loc\fP, if present.
 .La Throws:
 .CW bad_cast
 if
 ! .CW "has_facet<Facet>(*this)"
 is
 .CW false .
--- 816,824 ----
 .La Returns:
a reference to the corresponding facet of \&\f6loc\fP, if present.
+." USA CD2-22-013 22.1.2 lib.locale.global.templates [change *this to loc]
 .La Throws:
 .CW bad_cast
if
! .CW "has_facet<Facet>(loc)"
is
! .CW false.
**************
*** 842,846 ****
and
! respectively. Each such member function takes an
! .CW ios_base&
.ix "[ios_base] [flags]"
--- 907,914 ----
and
! respectively,
! ." Germany CD2-22-029 22 lib.localization [no change]
! ." Explain why the put & get functions take non-const ios_base& ?
! Each such member function takes an
! .CW ios_base&
.iix "[ios_base] [flags]"
**************
*** 901,907 ****
explicit ctype(size_t \f6refs\fP = 0);
!
bool         is(mask \f6m\fP, charT \f6c\fP) const;
!     const charT* is(const charT* \f6low\fP, const charT* \f6high\fP, mask* vec) const;
const charT* scan_is(mask \f6m\fP,
    const charT* \f6low\fP, const charT* \f6high\fP) const;
--- 969,977 ----
explicit ctype(size_t \f6refs\fP = 0);
!
+ ." editorial font correction on vec below
+ ." USA CD2-22-014 22.2.1.1 lib.locale.ctype [add arg types to toupper prototype (above)]
!
bool         is(mask \f6m\fP, charT \f6c\fP) const;
!     const charT* is(const charT* \f6low\fP, const charT* \f6high\fP, mask* \f6vec\fP)
) const;
const charT* scan_is(mask \f6m\fP,            
    const charT* \f6low\fP, const charT* \f6high\fP) const;
**************
*** 908,912 ****
const charT* scan_not(mask \f6m\fP,              
    const charT* \f6low\fP, const charT* \f6high\fP) const;
! charT    toupper(charT) const;
const charT* toupper(charT* \f6low\fP, const charT* \f6high\fP) const;
! charT    tolower(charT \f6c\fP) const;
--- 978,982 ----
const charT* scan_not(mask \f6m\fP,              
    const charT* \f6low\fP, const charT* \f6high\fP) const;
! charT    toupper(charT \f6c\fP) const;
const charT* toupper(charT* \f6low\fP, const charT* \f6high\fP) const;
! charT    tolower(charT \f6c\fP) const;
**************
*** 956,961 ****
for character classing during input parsing.
!
The base class implementation implements character classing appropriate
! to the implementation’s native character set.
.\H5 "\x7fctype\fP\& members" lib.locale.ctype.members
--- 1026,1038 ----
for character classing during input parsing.
!
The instantiations required in Table 52, namely
C
ctype<char>
and
C
ctype<wchar_t>
, implement character classing appropriate
to the implementation’s native character set.
+ .nE
+ Check the table number for "Table 52" hereafter.
+ .nE

."

."
ctype<fP>`members" lib.locale.ctype.members
***************
*** 1043,1052 ****
of type
C
ctype_base::mask
. The second form places \f6M\fP for all
C
*\f6p\fP
where
C
"(\f6low\fP<=\f6p\fP && \f6p\fP<\f6high\fP)"
, into
C
\f6vec\fP[\f6p\fP-\f6low\fP]".
La Returns:
The first form returns the result of the expression
--- 1120,1132 ----
of type
C
ctype_base::mask
. The second form identifies a value \f6M\fP of type
C
C
ctype_base::mask
! for each
C
*\f6p\fP
where
C
"(\f6low\fP<=\f6p\fP && \f6p\fP<\f6high\fP)"
, ! and places it into
C
\f6vec\fP[\f6p\fP-\f6low\fP]".
+ ." Germany CD2-22-025 22.2.1.1.2 lib.locale.ctype.virtuals [clarify do_is]
La Returns:
The first form returns the result of the expression
***************
*** 1300,1309 ****
static const mask* classic_table() throw();
.Ce
.
Cc
~ctype(); \f6// virtual\fP
! virtual char do_toupper(char) const;
! virtual const char* do_toupper(char* \f6low\fP, const char* \f6high\fP) const;
! virtual char do_tolower(char) const;
! virtual const char* do_tolower(char* \f6low\fP, const char* \f6high\fP) const;
};
--- 1380,1400 ----
static const mask* classic_table() throw();
.Ce
.
Cc
~ctype(); \f6// virtual\fP
! virtual char do_toupper(char \f6c\fP) const;
! virtual const char* do_toupper(char* \f6low\fP, const char* \f6high\fP) const;
! virtual char do_tolower(char \f6c\fP) const;
! virtual const char* do_tolower(char* \f6low\fP, const char* \f6high\fP) const;
+ 
+ virtual char do_widen(char \f6c\fP) const;
+ virtual const char* do_widen(const char* \f6low\fP,
const char* \f6high\fP, char \f6default\fP) const;
+ virtual char do_narrow(char \f6c\fP, char \f6default\fP) const;
+ virtual const char* do_narrow(const char* \f6low\fP,
const char* \f6high\fP,
char \f6default\fP) const;
};

***************
*** 1444,1454 ****
    char* _f6to_fP) const;
    .Pe
    ! .La Effects:
    ! .CW "Memcpy(_f6to_fP, _f6low_fP, _f6high_fP-_f6low_fP)"
    ! .ix "[memcpy]"
    ! .La Returns:
    ! .CW c
    ! or
    ! .CW hi
    ."----
    .ix "[ctype<char>] [narrow]"
--- 1535,1542 ----
    char* _f6to_fP) const;
    .Pe
    ! ". USA CD2-22-001 22.2.1.3.2 lib.facet.ctype.char.members
    ! .La Returns:
    ! .CW "do_widen(_f6low_fP, _f6high_fP, _f6to_fP)"
    ! .ix "[do_widen]"
    ."----
    ! .ix "[ctype<char>] [narrow]"
***************
*** 1458,1480 ****
    char /*dfault*/, char* _f6to_fP) const;
    .Pe
    - .La Effects:
    - .CW "Memcpy(_f6to_fP, _f6low_fP, _f6high_fP-_f6low_fP)"
    - .ix "[memcpy]"
    - .La Returns:
    - ! _f6c_fP or _f6high_fP.
    ! .eN
    ! The library WG feels that the members
    ! .CW widen " and"
    ! .CW narrow
    ! should delegate to virtual members
    ! .CW do_widen
    ! and
    ! .CW do_narrow ,
    ! as in the
    ! .CW ctype<
    ! template, to permit
    ! .CW char
    ! encodings that differ from the basic execution character encoding.
    ! .nE
    ."----
    ! .ix "[ctype<char>] [table]"
--- 1546,1552 ----
    char /*dfault*/, char* _f6to_fP) const;
    .Pe
    .La Returns:
    ! .CW "do_narrow(_f6low_fP, _f6high_fP, _f6to_fP)"
    ! .ix "[do_narrow]"
    ."----
    ! .ix "[ctype<char>] [table]"
***************
*** 1498,1501 ****
--- 1570,1574 ----
.\"==
.\H5 "\f7ctype<char>\fP\& virtual functions" lib.facet.ctype.char.virtuals
+ .\" USA CD2-22-001 22.2.1.3.4 lib.facet.ctype.char.virtuals
.Pb
    char do_toupper(char) const;
***************
virtual char        do_widen(char) const;
virtual const char* do_widen(char* low, const char* high) const;

+ virtual char        do_widen(char\ f6c\fP) const;
+ virtual const char* do_widen(const char* \f6low\fP,
+                const char* \f6high\fP,
+                char* \f6to\fP) const;
+ virtual char        do_narrow(char\ f6c\fP, char \f6default\fP) const;
+ virtual const char* do_narrow(const char* \f6low\fP,
+                const char* \f6high\fP,
+                char \f6default\fP, char* \f6to\fP) const;

These functions are described identically as those members of the

protected:
~ctype_byname(); \f6// virtual\fP
! virtual char        do_toupper(char) const;
virtual const char* do_toupper(char* \f6low\fP, const char* \f6high\fP) const;
! virtual char        do_tolower(char) const;
virtual const char* do_tolower(char* \f6low\fP, const char* \f6high\fP) const;

};

--- 1600,1616 -----
protected:
~ctype_byname(); \f6// virtual\fP
! virtual char        do_toupper(char \f6c\fP) const;
virtual const char* do_toupper(char* \f6low\fP, const char* \f6high\fP) const;
! virtual char        do_tolower(char \f6c\fP) const;
virtual const char* do_tolower(char* \f6low\fP, const char* \f6high\fP) const;
+
+ virtual char        do_widen(char \f6c\fP) const;
+ virtual const char* do_widen(char* \f6low\fP, const char* \f6high\fP) const;

};

***************
*** 1568,1576 ****
const internT* \f6from\fP, const internT* \f6from_end\fP, const internT*& \f6f
rom_next\fP,
externT* \f6to\fP, externT* \f6to_limit\fP, externT*& \f6t
o_next\fP) const;
- virtual result do_unshift(stateT& \f6state\fP,
- externT* \f6to\fP, externT* \f6to_limit\fP, externT*& \f6t
o_next\fP) const;
- virtual result do_in(stateT& \f6state\fP,
- const externT* \f6from\fP, const externT* \f6from_end\fP, const externT*& \f6f
rom_next\fP,
internT* \f6to\fP, internT* \f6to_limit\fP, internT*& \f6t
o_next\fP) const;

virtual int do_encoding() const throw();
virtual bool do_always_noconv() const throw();
--- 1659,1667 -----
const internT* \f6from\fP, const internT* \f6from_end\fP, const internT*& \f6f
rom_next\fP,
externT* \f6to\fP, externT* \f6to_limit\fP, externT*& \f6t
o_next\fP) const;
- virtual result do_in(stateT& \f6state\fP,
- const externT* \f6from\fP, const externT* \f6from_end\fP, const externT*& \f6f
rom_next\fP,
Implementations are required to provide instantiations for .CW <wchar_t,char,mbstate_t> and .CW <char,char,mbstate_t> .

The base class instance of the latter implements a degenerate conversion: ! .CW always_noconv() ! returns ! .CW true ! and ! .CW max_length() ! returns ! .CW 1 . ! "The following sentences are from (N0699,22-015)."

Instantiations on .CW mbstate_t --- 1684,1698 ----
argument selects the pair of codesets being mapped between.

" Germany 22-019(SecondPart) 22 actually 22.2.1.5 lib.locale.codecvt
The instantiations required in Table 52, namely .CW codecvt<wchar_t,char,mbstate_t> and .CW codecvt<char,char,mbstate_t> ,
convert the implementation-defined native character set.
ANTEDENCODING > //C "The following sentences are from (N0699,22-015)."

Instantiations on .CW mbstate_t --- 1684,1698 ----
argument selects the pair of codesets being mapped between.

" Germany 22-019(SecondPart) 22 actually 22.2.1.5 lib.locale.codecvt
The instantiations required in Table 52, namely .CW codecvt<wchar_t,char,mbstate_t> and .CW codecvt<char,char,mbstate_t> ,
convert the implementation-defined native character set.
ANTEDENCODING > //C "The following sentences are from (N0699,22-015)."

Instantiations on .CW mbstate_t --- 1684,1698 ----
argument selects the pair of codesets being mapped between.

" Germany 22-019(SecondPart) 22 actually 22.2.1.5 lib.locale.codecvt
The instantiations required in Table 52, namely .CW codecvt<wchar_t,char,mbstate_t> and .CW codecvt<char,char,mbstate_t> ,
convert the implementation-defined native character set.
ANTEDENCODING > //C "The following sentences are from (N0699,22-015)."

Instantiations on .CW mbstate_t --- 1684,1698 ----
argument selects the pair of codesets being mapped between.
The instantiations required in Table 52, namely

- `codecvt<wchar_t,char,mbstate_t>`
- `codecvt<char,char,mbstate_t>`

store no characters.

- "Germany CD2-22-023 lib.locale.codecvt.virtuals"
- "USA CD2-22-023 [duplicate issue]"

Stores no more than
- `"(f6to_limit\fP-f6to\fP)`
- destination elements.
- It always leaves the `\f6to_next\fP` pointer
- pointing one beyond the last element successfully stored.

La "Returns"
An enumeration value, as summarized in Table \n+(Tn:

***************
*** 1783,1787 ****
 .TE
 .Te
! The base class implementation returns
 .CW noconv .
 .\"---
---- 1881,1886 ----
 .TE
 .Te
! `codecvt<char,char,mbstate_t>` returns
! returns
 .CW noconv .
 .\"---
***************
*** 1813,1823 ****
 returns
 .CW noconv
! for all valid argument values. The base class implementation
! for the instantiation
! `<char,char,mbstate_t>` returns
! `true`; others return
! `false`.
 .\"---
 .\"
---- 1912,1921 ----
 returns
 .CW noconv
! for all valid argument values.
! `codecvt<char,char,mbstate_t>` returns
! `true`.  
 .\"Germany CD2-33-024 22.2.1.5 lib.locale.codecvt"
 .\"USA Cd2-22-024 [duplicate issue]"
 .\"----
 .\"
***************
*** 1845,1849 ****
or fewer valid complete characters of type
 .CW \f6internT\fP
! The base class implementation returns the lesser of
 .CW \f6max\fP and
---- 1943,1951 ----
or fewer valid complete characters of type
 .CW \f6internT\fP
! The instantiations required in Table 52, namely
! `codecvt<wchar_t,char,mbstate_t>`
! and
! `codecvt<char,char,mbstate_t>`
! return the lesser of
and

**************
*** 1864,1867 ****
--- 1966,1971 -----
value

.CW \f6state\fP.
+ .CW codecvt<\char,\char,mbstate_t>
+ returns 1.
."
"
"---

**************
*** 1868,1871 ****
--- 1972,1976 -----
.H4 "Template class \&\f7codecvtbyname\fP\&" lib.locale.codecvtbyname
 .ix "[codecvt__byname]"
+ ." USA CD2-22-006 22.2.1.6 lib.locale.codecvtbyname [add do_unshift]
 .Cb
namespace std {

**************
*** 1882,1885 ****
--- 1987,1992 ------
 const externT* \f6from\fP, const externT* \f6from_end\fP, const externT* & \f6from_next\fP, 
 internT* \f6to\fP, internT* \f6to_limit\fP, internT* & \f6to_next\fP) const;
 + virtual int do_unshift(stateT& \f6state\fP, 
 externT* \f6to\fP, externT* \f6to_limit\fP, externT* & \f6to_next\fP) const;
 virtual int do_encoding() const throw();
 virtual bool do_always_noconv() const throw();

**************
*** 1886,1889 ****
--- 1993,1998 -----
 virtual int do_length(const stateT&,
 const externT* \f6from\fP, const externT* \f6end\fP, 
 size_t \f6max\fP) const;
 + virtual result do_unshift(stateT& \f6state\fP, 
 externT* \f6to\fP, externT* \f6to_limit\fP, externT* & \f6to_next\fP) const;
 virtual int do_max_length() const throw();
};

**************
*** 1892,1895 ****
--- 2001,2005 -----
 ."===
 .H3 "The numeric category" lib.category.numeric
+ ." Germany _2222/ 22.2.2 lib.category.numeric [C and C++ locales - No changes]
 .P
 The classes

**************
*** 1909,1913 ****
 .Fe
 .P
 ! The base class implementation refers to the
 .CW ios_base::
 argument for formatting specifications (_lib.locale.categories_),
--- 2019,2037 -----
 .Fe
 .P
 ! ." Germany CD2-22-019(SecondPart) 22 actually 22.2.2 lib.category.numeric
 ! All specifications of member functions for
 ! num_put
 ! and
 ! num_get
 ! in the subclauses of _lib.category.numeric_ only apply to the
 ! instantiations required in Table 52 and Table 53, namely
 ! .CW num_get<\char>,

Page 13
These instantiations refer to the argument for formatting specifications (_lib.locale.categories_),

```cpp
/* 2042,2050 */
! use_facet< ctype<charT> >(_loc_),
! use_facet< numpunct<charT> >(_loc_),
If an error occurs, _val_ is unchanged; otherwise it is set to the resulting value.
```

The details of this operation occur in two stages.

```
Stage 1: Determine a conversion specifier
--- 2166,2179 -----
! use_facet< ctype<charT> >(_loc_),
! use_facet< numpunct<charT> >(_loc_),
where _loc_ is _str_.getloc() .
If an error occurs, _val_ is unchanged; otherwise it is set to the resulting value.
```

The details of this operation occur in three stages.

```
Stage 1: Determine a conversion specifier
```

```
Stage 1:
The function initializes local variables via
--- 2188,2210 -----
fmtflags basefield = (flags & ios_base::basefield);
fmtflags uppercase = (flags & ios_base::uppercase);
fmtflags boolalpha = (flags & ios_base::boolalpha);
```

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table

```
/* 2066,2077 */
fmtflags basefield = (flags & ios_base::basefield);
fmtflags uppercase = (flags & ios_base::uppercase);
```

Page 14
For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.

For conversion to an integral type, the function determines the integral conversion specifier as indicated in Table \n+(Tn. ! That is, the first line whose condition is true applies.
Digit grouping is checked. That is, the positions of discarded separators is examined for consistency with

```
use_facet<numpunct<charT> > >located>.grouping()
```

Is the treatment of separators here clear?

If they are not consistent then

```
iose_base::failbit
```

accumulated in stage 2 would have caused scanf to report an input failure.

```
iose_base::failbit
```

If a value is being stored into `val`, the value is determined according to the following:

--- 2330,2338 ----

```
\f6str\f7.flags()&&ios_base::boolalpha==0
```

If `str` except that if a value is being stored into `val`, the value is determined according to the following:

--- 2435,2443 ----

```
\f7num_put\fP\& members
```

```
\f7num_put\fP[put]
```

--- 2298,2303 ----

```
locale \f6loc\fP = \f6str\fP.getloc();
```

The base class implementation is described in several stages

LI

Stage 1:

--- 2476,2482 ----

```
locale \f6loc\fP = \f6str\fP.getloc();
```
The details of this operation occur in several stages:

**Stage 1:**

--- 2345,2349 ----

The first action of stage 1 is to determine a conversion specifier.

--- 2372,2377 ----

All tables used in describing stage 1 are ordered.

--- 2435,2438 ----

Type a length modifier is added to the conversion specifier as indicated in Table \n+(Tn.}

--- 2579,2583 ----

<table>
<thead>
<tr>
<th>Type</th>
<th>Length Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>h</td>
</tr>
<tr>
<td>unsigned short</td>
<td>h</td>
</tr>
</tbody>
</table>

--- 2513,2520 ----

All tables used in describing stage 1 are ordered.

--- 2525,2530 ----

That is, the first line whose condition is true applies.

--- 2525,2530 ----

All tables used in describing stage 1 are ordered.

--- 2558,2560 ----

<table>
<thead>
<tr>
<th>Type</th>
<th>Length Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>h</td>
</tr>
<tr>
<td>unsigned short</td>
<td>h</td>
</tr>
</tbody>
</table>

--- 2486,2490 ----

The tables that describe this determination use the following local variables

--- 2513,2520 ----

The first action of stage 1 is to determine a conversion specifier.

--- 2525,2530 ----

The tables that describe this determination use the following local variables

--- 2525,2530 ----

That is, the first line whose condition is true applies.

--- 2525,2530 ----

A line without a condition is the default behavior when none of the earlier lines apply.

--- 2525,2530 ----

A line without a condition is the default behavior when none of the earlier lines apply.

--- 2525,2530 ----

That is, the first line whose condition is true applies.

--- 2525,2530 ----

A line without a condition is the default behavior when none of the earlier lines apply.

--- 2525,2530 ----

That is, the first line whose condition is true applies.

--- 2525,2530 ----

A line without a condition is the default behavior when none of the earlier lines apply.

--- 2525,2530 ----

That is, the first line whose condition is true applies.

--- 2525,2530 ----

A line without a condition is the default behavior when none of the earlier lines apply.

--- 2525,2530 ----

That is, the first line whose condition is true applies.

--- 2525,2530 ----

A line without a condition is the default behavior when none of the earlier lines apply.

--- 2525,2530 ----

That is, the first line whose condition is true applies.

--- 2525,2530 ----

A line without a condition is the default behavior when none of the earlier lines apply.

--- 2525,2530 ----

That is, the first line whose condition is true applies.

--- 2525,2530 ----

A line without a condition is the default behavior when none of the earlier lines apply.
long l
--- 2588,2591 ----
***************
*** 2469,2473 ****
= T{ ! an integral type other than a character type
  T} flags & showpos +
  \^ flags & showbase #
--- 2610,2614 ----
= T{ ! an integral type
  T} flags & showpos +
  \^ flags & showbase #
***************
*** 2478,2488 ****
." Germany _222222/g 22.2.2.2.2 lib.facet.num.put.virtuals [drop "other than char"]
." Germany _222222/e 22.2.2.2.2 lib.facet.num.put.virtuals [use "widen" not "charT"
"
For conversion from a floating-point type, if
! .CW "(flags() & fixed) != 0"
or if
! .CW "precision() > 0",
then
! .CW precision()
is specified in the conversion specification.
." Germany _222222/g 22.2.2.2.2 lib.facet.num.put.virtuals [drop "other than char"]
." Germany _222222/e 22.2.2.2.2 lib.facet.num.put.virtuals [use "widen" not "charT"
"
Any character \f6c\fP other than a decimal point(.) is converted to a
.CW charT
via
! .CW "charT(\&\f6c\fP\&)"
.P
A local variable \f6punct\fP is initialized via
--- 2641,2649 -----
." Germany _222222/g 22.2.2.2.2 lib.facet.num.put.virtuals [drop "other than char"]
." Germany _222222/e 22.2.2.2.2 lib.facet.num.put.virtuals [use "widen" not "charT"
"
Any character \f6c\fP other than a decimal point(.) is converted to a
.CW charT
via
! .CW "use_facet<ctype<charT> >(\f6loc\fP).widen(\&\f6c\fP\&)"
.P
A local variable \f6punct\fP is initialized via
The location of any padding is determined according to Table

The conversion specification
! 
\begin{itemize}
\item \texttt{\#o}
\item \texttt{0}
\item \texttt{width()}
\end{itemize}

If \texttt{str.width()} is nonzero and the number of
\texttt{charT's} in the sequence after stage 2 is less than
\texttt{str.width()}, then enough \texttt{fill} characters are added to the sequence at the position indicated for padding to bring the length of the sequence to
\texttt{width()}. 
\texttt{width(0)} is called.

--- 2702,2723 -----
T}    pad after x or X

--- 2688,2681 -----
T}    pad after x or X
*\f6out\fP++ = c

*\f6out\fP++ = c

! If at any point
! .CW out.failed()
! becomes true, then output is terminated.

iter_type put(iter_type \f6out\fP, ios_base& \f6str\fP, char_type \f6fill\fP,
--- 2729,2733 ----
*\f6out\fP++ = c

! ." USA CD2-22-003 22.2.2.2 lib.facet.num.put.virtuals [out.failed() is undefined]

iter_type put(iter_type \f6out\fP, ios_base& \f6str\fP, char_type \f6fill\fP,
***************
*** 2594,2599 ****
.CW (\f6str\fP.flags()&ios_base::boolalpha)==0
then do

! \f6out\fP = put(\f6out\fP\&, \f6str\fP\&, \f6fill\fP\&, (int)\f6val\fP\&)
.Ce
Otherwise do
--- 2738,2744 ----
.CW (\f6str\fP.flags()&ios_base::boolalpha)==0
then do
+ ." Germany _222222/d 22.2.2.2.2 lib.facet.num.put.virtuals ["put" becomes "do_put"]
.Cb
! \f6out\fP = do_put(\f6out\fP\&, \f6str\fP\&, \f6fill\fP\&, (int)\f6val\fP\&)
.Ce
Otherwise do
***************
*** 2604,2653 ****
.Ce
and then insert the characters of \f6s\fP into \f6out\fP.
! .IR out .
\"==
.H3 "The numeric punctuation facet" lib.facet.numpunct
--- 2749,2753 ----
.Ce
and then insert the characters of \f6s\fP into \f6out\fP.
! .I out .
\"==
.H3 "The numeric punctuation facet" lib.facet.numpunct
***************
*** 2645,2653 ****
.CW numpunct<
specifies numeric punctuation.
! The base class provides classic
.CW C \\
! numeric formats, while the
! .CW _byname \\
! version supports named locale (e.g. POSIX, X/Open) numeric formatting semantics.
.P
The syntax for number formats is as follows, where
--- 2790,2806 ----
.CW numpunct<
specifies numeric punctuation.
! ." Germany CD2-22-019(SecondPart) 22 actually 22.2.3.1 lib.locale.numpunct
! The instantiations required in Table 52, namely
! .CW numpunct<wchar_t>
! and
The syntax for number formats is as follows, where

```
*** 2737,2741 ****

Returns:
A character for use as the decimal radix separator.
The base class implementation returns \f5'.'\fP.
```

```
--- 2890,2896 ----

Returns:
A character for use as the decimal radix separator.
```

```
*** 2745,2749 ****

Returns:
A character for use as the digit group separator.
The base class implementation returns \f5','\fP.
```

```
--- 2900,2904 ----

Returns:
A character for use as the digit group separator.
```

```
*** 2757,2762 ****

represents the number of digits\f
```

```
--- 2912,2921 ----

represents the number of digits\f
```

```
+ ."      Thus, the string \f5"e003"\fP specifies groups of 3 digits each, and
+ \f5"3"\fP probably indicates groups of 51 (!) digits each.
```

```
--- 2929,2934 ----

the size of the digit group is unlimited.
```

```
--- 2934,2938 ----

the size of the digit group is unlimited.
```
The required instantiations return the empty string, indicating no grouping.

---

In the base class implementation these names are \f5"true\fP and \f5"false\fP.

In the base class implementation these names are \f5"true\fP and \f5"false\fP, or \f5L"true\fP and \f5L"false\fP.

uses the collate facet to allow a locale to act directly as the predicate argument for standard algorithms (_lib.algorithms_) and containers operating on strings.

The instantiations required in Table 52, namely .CW collate<char>
and .CW collate<wchar_t> ,
apply lexicographic ordering (_lib.alg.lex.comparison_).

Each function compares a string of characters

if the first string is greater than the second, .CW -1
if less, zero otherwise. The base class implementation implements a lexicographical comparison (_lib.alg.lex.comparison_).

---

if the first string is greater than the second, .CW -1
if less, zero otherwise.

The instantiations required in Table 52, namely .CW collate<char>
and .CW collate<wchar_t> ,
implement a lexicographical comparison (_lib.alg.lex.comparison_).

---

Time_put provide date and time formatting and parsing.

The effect of the ios_base argument to member functions of time_put
and time_get is unspecified. Do they have any effect at all? Which format flags are ignored? Or is it undefined, or
Actually, the same is true for the monetary facets. For instance, do the dec, hex, oct flags have any effect on the formatting of monetary amounts, or is it implementation-dependent whether they have an effect?

Date/time and money formats are implementation-defined.

```
+ Actually, the same is true for the monetary facets. For instance, do
+ the dec, hex, oct flags have any effect on the formatting of monetary
+ amounts, or is it implementation-dependent whether they have an
+ effect?
+  
+ Date/time and money formats are
+ implementation-defined.
+ .NE
+ \" Germany CD2-22-019(SecondPart) 22 actually 22.2.5 lib.category.time
+ All specifications of member functions for
+ time_put
+ and
+ time_get
+ in the subclauses of _lib.category.time_ only apply to the
+ instantiations required in Table 52 and Table 53.
+ Their members use their
+ .CW ios_base&
+ **************
*** 3254,3257 ****
--- 3446,3454 -----
  .CW ctype<>::narrow()
  to identify format specifiers.
+ \" Germany CD2-22-034 2.2.5.3.1 lib.locale.time.put.members
+ \" USA CD2-22-034 [duplicate issue]
+ .eN
+ Add wording as per 22-034 - NCM.
+ .nE
.n[
+ This implies that if
**************
*** 3314,3321 ****
  local or international monetary formats are to be used.
  .P
+ .CW money_get<
+ and
+ .CW money_put<
+ members use their
+ .CW ios_base&
+ .CW ios_base::iostate&
--- 3511,3522 -----
  local or international monetary formats are to be used.
  .P
+ \" Germany CD2-22-019(SecondPart) 22 actually 22.2.2 lib.category.numeric
+ ! All specifications of member functions for
+ ! .CW money_put
+ and
+ ! .CW money_get
+ ! in the subclauses of _lib.category.monetary_ only apply to the
+ ! instantiations required in Table 52 and Table 53.
+ ! Their members use their
+ .CW ios_base&
+ .CW ios_base::iostate&
+  **************
*** 3327,3330 ****
--- 3528,3533 -----
  .CW ctype<>
  facets, to determine formatting details.
+ \" Germany CD2-22-026 22.2.6 lib.category.monetary [decision: no change]
+ \" USA CD2-22-026 [duplicate issue, no change]
+ ---
  .H4 "Template class \&\f7money_get\fP\" lib.locale.money.get
  **************
*** 3351,3356 ****
  string_type& \f6digits\fP) const;
  .Ce
```
static const bool Intl = Intl;
static locale::id id;
--- 3729,3733 ----
  .CW moneypunct<charT,true>
or!
  .CW moneypunct<charT,false>
  facet of \f6loc\fP (depending on whether \f6intl\fP is
  .CW true
***************
*** 3536,3543 ****
  .La Notes:
The currency symbol is generated only if
  ! .CW "((\f6str\fP.flags() & ios_type::showbase)"
is true.
If
  .CW "((\f6str\fP.flags() & ios_type::adjustfield) == ios_type::internal)"
the fill characters are placed where
  .CW none
--- 3743,3751 ----
  .La Notes:
The currency symbol is generated only if
  ! ." Germany _222622/ 22.2.6.2.2. lib.locale.money.put.virtuals
  ! .CW "((\f6str\fP.flags() & ios_base::showbase)"
is true.
If
  .CW "((\f6str\fP.flags() & ios_base::adjustfield) == ios_base::internal)"
the fill characters are placed where
  .CW none
***************
*** 3622,3625 ****
--- 3830,3845 ----
decimal point is exactly the value returned by 
  .CW frac_digits .
  + ." Germany CD2-22-021 22.2.6.2.2.3 lib.locale.moneypunct [more about money_base::part]
  + .eN
  + More description of money_base::part as per 22-021 - NCM.
  + .eN
  + ." Germany CD2-22-022 22.2.6.2.3 lib.locale.moneypunct
  + .eN
  + More about why "pattern" is "char", as per 22-022 - NCM.
  + .eN
  + ." Germany CD2-22-030 22.2.6.3, really 22.2.6.3, lib.locale.moneypunct
  + .eN
  + More about "space" and "none", as per 22-030 - NCM.
  + .eN
  ."----
  .H5 "\f7moneypunct\fP\f6 members" lib.locale.moneypunct.members
***************
*** 3715,3718 ****
--- 3935,3939 ----
  int do_frac_digits() const;
  + .Pe
  + ." Germany _222632/ 22.2.6.3.2 lib.locale.moneypunct.virtuals [no changes]
  .Pe
  Returns:
The number of digits after the decimal radix separator, if any.*f
***************
*** 3746,3754 ****
if present, is neither first nor last.
Otherwise, the elements may appear in any order.
! The base class implementation returns an object of type
  .CW pattern
initialized to
  ! .CW "\{ symbol, sign, none, value \}"
! this value is also returned for all international instantiations.*f
  .Fs
Note that the international symbol returned by
--- 3967,3981 ----
if present, is neither first nor last.
Otherwise, the elements may appear in any order.
The instantiations required in Table 52, namely
CW moneypunct<char>,
CW moneypunct<wchar_t>,
CW moneypunct<char,true>,
and
CW moneypunct<wchar_t,true>,
return an object of type
CW pattern
initialized to
CW "\{ symbol, sign, none, value \}" \*f
Note that the international symbol returned by
***************
--- 3879,3882 -----
is used for character set code conversion when retrieving messages, if needed.
--- 4106,4110 -----
+ CW Germany _222712/a 22.2.7.1.2 lib.locale.messages.virtuals [no changes]
+ CW "[messages] [do__get]"
***************
--- 4054,4057 -----
+ CW Netherlands _2211/c 22.1.2 actually 22.2.8 [cerberos needs ctor argument]
+ CW USA CD2-22-009 22.1.1 lib.locale [duplicate issue]

std::istream& operator>>(std::istream& s, Date& d)
***************
--- 4058,4062 -----
{ using namespace std;
  istream::sentry cerberos;
  if (cerberos) {
    ios_base::iostate err = goodbit;
  --- 4288,4292 -----
  { using namespace std;
    istream::sentry cerberos(s);
    if (cerberos) {
      ios_base::iostate err = goodbit;