Doc. No.:	X3J16 / 95-0098
	WG21 / N00698
Date:	22 May 1995
Project:	C++ Standard Library
Reply to:	Nathan Myers
	myersn@roguewave.com

Clause 22 (Localization Library) Issues (V.1)

Revision History

Version 1 - 22 May 1995

Introduction

This document is a summary of issues identified for the Clause 22, identifying resolutions as they are voted on and recommendationsfor unsolved problems in the Draft.

[Maintainer's note: I apologize for the lack of detail in this list. I had a complete list written and lost it to a disk failure, so this is reconstructed from all-too-human memory.]

Work Group: Library: Localization Clause 22
Issue Number: 22-001
Title: locale usage syntax `loc.template use<>()` too clumsy
Sections: 22.1.1.3
Status: active
Description:

The resolution, in Austin, of syntax for calling explicitly qualified member template functions is too clumsy for the primaryinterface to locales, if any alternative is possible. With no change, calls look like:

```
loc.template use<Facet>().member()
```

Discussion:

The language offers another alternative: a non-member friend template function. Using it, the call above looks like:

```
use_facet<Facet>(loc).member()
```

more closely resembling a cast.

Proposed Resolution:

In place of the members `std::locale::use<>()` and `std::locale::has<>()`, provide global templates, with the same semantics. These must be friends of `locale`.

```
template <class Facet> const Facet& use(const locale&);
template <class Facet> bool         has(const locale&)
throw();
```

Also, change all examples that mention the old form to the new form.

Requestor: Nathan Myers
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-002

Title: locale member constant `all` overconstrained.
Sections: 22.1.1.1.1
Status: active
Description:

During editorial work the member "`all`" was changed to require that

```
(collate | ctype | monetary | numeric | time
|messages)==all
```

be true.

Discussion:

This overconstrains implementors by preventing them from adding categories of their own.

Proposed Resolution:

Specify instead that:

```
(collate | ctype | monetary | numeric | time | messages |
all) == all
```

is true, as originally documented.

Requestor: Nathan Myers
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-003
Title: Effect of `operator|()` and `operator()&` on categories is unspecified.
Sections: 22.1.1.1.1
Status: active
Description:

In the same section as above, on applying bitwise operators to categories:

> Further, the result of applying operators & and | to any two valid values is itself valid.

It's valid, but what does it mean?

Discussion:

Clearly we want set union and intersection behavior.

Proposed Resolution:

Add to the above:

> ", and results in the setwise union or intersection, respectively, of the argument categories."

Requestor: P.J.Plauger
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-004
Title: Description of `byname` facets too vague
Sections: 22.1.1.1.2
Status: active
Description:

Paragraph 4, where `byname<>` classes are described, leaves some issues unresolved.

Discussion:
Proposed Resolution:

Add to paragraph 4:

"If the `const char*` argument to a `byname` facet constructor does not identify a valid locale name, the constructor throws an exception of type `std::runtime_error`."

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-005
Title: Locale operators shouldn't throw exceptions
Sections: 22.1.1.2
Status: active
Description:

The class locale is intended to be stored in user data structures and copied freely. For safe system design it is necessary to be assured that such operations will not throw any exceptions, because that would corrupt those data structures.

Discussion:

Adding empty throw specifications to the declarations provides this guarantee and also allows more efficient operation on some architectures.

Proposed Resolution:

Declare the following locale members as:

```
locale() throw();
locale(const locale& other) throw();
~locale() throw();  // non-virtual
const locale& operator=(const locale& other) throw();
template <class Facet> bool has() throw() const;
```

and document that they do not throw any exceptions.
Note: If the recommendation for issue 22-001 is accepted, the last declaration above would become, instead:

```
template <class Facet>
riend bool has(const locale&) throw();
```

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-006
Title: locale constructors should say they throw `runtime_error`
Sections: 22.1.1.2
Status: active
Description:

The descriptions of the constructors:

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

don't say what happens if the implementation cannot provide a locale of the requested name.

Discussion:

These constructors can also throw `bad_alloc` if the various parts of the locale can't be created, so a throw specification seems inappropriate.

Proposed Resolution:

Add text to the descriptions:

If the `std_name` argument is not a valid locale name, throws `runtime_error`. May also throw other exceptions, if resources necessary to construct the locale are unavailable.

The same should be said about `byname<>` facet constructors.

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-007
Title: locale template `use()` throw behavior needs clarification
Sections: 22.1.1.3 [lib.locale.members]
Status: active
Description:

The template:

```
template <class Facet> const Facet& locale::use() const;
```

(or depending on resolution of issue 22-001

```
template <class Facet> friend const Facet& use(const
locale&);
```

) is described as throwing `bad_cast` if the locale does not implement the specified facet. Other exceptions are possible, as `use()` does things "behind the scenes" that consume resources.

Proposed Resolution:

Document that `use()` may throw other unspecified exceptions as well.

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-008
Title: locale member `op()` needs more template parameters
Sections: 22.1.1.4
Status: active
Description:

The locale member template operator:

```
template <class charT>
  bool operator()(const basic_string<charT>& s1,
                  const basic_string<charT>& s2) const;
```

does not accommodate the full generality of strings users may need to compare.

Discussion:

`basic_string<>` has undergone evolution, and we need to track it.

Proposed Resolution:

Replace the above declaration with:

```
template <class charT, class Traits, class Alloc>
 bool operator()(const basic_string<charT,Traits,Alloc>&
s1,
                const basic_string<charT,Traits,Alloc>&
s2)
                   const;
```

Requestor: Takanori Adachi
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-009
Title: Global locale effect on C Lib functions unspecified
Sections: 22.1.1.5
Status: active
Description:

The global locale `locale()`, as set by `locale::global(...)`, is described as affecting the C library functions, but the Draft doesn't say what facets and members are used.

Discussion:

The mapping is quite straightforward, in most cases, but should be spelled out. In particular, it is not obvious how some of the `lconv` members returned by the C function `localeconv` may be derived from `numpunct<>` and `moneypunct<>` members. (I have solved this, but need to write it up.)

Proposed Resolution:

The details proposed will be in a separate paper. [I planned to write this paper for the mailing, but the aforementioned disk crash intervened.]

Requestor: P. J. Plauger
Owner: Nathan Myers

Work Group: Library: Localization Clause 22
Issue Number: 22-010
Title: Convenience functions is???(c, const locale&) are slow
Sections: 22.1.2.1
Status: active
Description:

The C functions corresponding to these functions are usually implemented as macros; these functions cannot be as fast.

Discussion:

The functions are provided only as a convenience for converting old code.

Proposed Resolution:

Add a footnote indicating that if the test is to be applied in a loop there are faster ways to do the same thing.

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-011
Title: `ctype_base` member `ctype_mask` name is too long
Sections: 22.2.1
Status: active
Description:

The type `ctype_base::ctype_mask` is named badly.

Discussion:

In use it is always qualified with `ctype_base`, so the "ctype_" prefix is unnecessary. (This name has a messy history.)

Proposed Resolution:

Change the name to `ctype_base::mask`, and the corresponding function parameter types to match.

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-012
Title: `ctype<>::is(...)` result inconsistent with other members
Sections: 22.2.1.1.2 and 22.2.1.3.2
Status: active
Description:

The ctype<> and ctype<char> members

```
       const charT* [do_]is(const charT* low, const charT* high,
                            ctype_mask* vec) const;
```

are documented to return `low`, unlike the other members of `ctype<>`. This inconsistency was accidental.

Discussion:

Returning `high` is consistent not only with other members but with Container members and Algorithms.

Proposed Resolution:

Change the descriptions *in both places* to indicate it returns `high`.

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-013
Title: `ctype<char>` derivation interface overconstrained.
Sections: 22.2.1.3
Status: active
Description:

From the Draft, box 19:

> Members `table`, `classic_table`, and `delete_it` should be clearly described in terms of their (lack of) constraints on the details of the implementation. In particular, it must be made clear whether these members must appear with these particular names, who can get to them, and so on.

Discussion:

As Plauger points out, `ctype<char>`'s derivation interface is "grossly overconstrained".

Proposed Resolution:

Eliminate mention of member `delete_it` describe destructor semantics in terms of the argument value to the constructor. Replace members `table` and `classic_table` with:

```
       protected:
         static const ctype_mask* classic_table();
         const ctype_mask* table() const;
```

Requestor:
Owner:

Work Group: Library: Localization Clause 22

Issue Number: 22-014
Title: `ctype_byname<char>` specialization not described.
Sections: 22.2.1.3
Status: active
Description:

In the front matter (22.1) the specialization `ctype_byname<char>` is mentioned, but it has no section.

Discussion:

This matters because `ctype_byname<>` is used polymorphically, and so must be described as inheriting from `ctype<char>` for the facet to work correctly.

Proposed Resolution:

Add a section specifying that `ctype_byname<char>` is derived publicly from `ctype<char>`.

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-015
Title: `codecvt<>` usage could be better described
Sections: 22.2.1.4 [lib.locale.codecvt]
Status: active
Description:

Several people, including Plauger, have asked for clarification of the role of the stateT template parameter to `codecvt<>`.

Discussion:

`codecvt<>` is an open-ended set of conversion facilities. Implementors are only required to provide instantiations of `codecvt<char,wchar_t,mbstate_t>` and `codecvt<wchar_t,char,mbstate_t>`, most probably by specialization. These are used by filebuf to serialize wide characters, and by the C functions to convert between multibyte and `wchar_t` encodings. By specializing with other types in place of `mbstate_t`, users can specify conversions for codesets unknown to the implementor. mbstate_t is an opaque type from C, Amendment 1; implementors can put anything in it as needed for translation.

Proposed Resolution:

Add to paragraph 3:

> Instantiations on `mbstate_t` perform conversion between any encodings known to the library implementor. Other encodings can be converted by specializing on a user-defined `stateT` type. The `stateT` object can contain any state that is useful to communicate to or from the specialized `convert()` member. The base class implementations convert the implementation-defined native execution codeset.

And add a footnote: the type `mbstate_t` is an opaque type inherited from the C Library.

Requestor: Plauger and others
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-016
Title: Numeric parsing & formatting description is poorly organized
Sections: 22 (many)

Status: active
Description:
As several people have pointed out, the descriptions of parsing and formatting semantics for iostreams, and facet members put* and get*, are scattered in both Clauses 22 and 27. Further, they reference C Library semantics in ways that are incompatible with the C++ Library environment.
Discussion:
Since iostreams delegates all its formatting and parsing to locale, the descriptions of such semantics might best be in Clause 22. Also, the more general semantics of locale facilities raises some questions about parsing: e.g. what is the effect if a digit group separator is specified to be a digit value, or equal to the decimal separator?
(Plauger)
Proposed Resolution:
We should encourage editoral aggressiveness in consolidating the descriptions of parsing and formatting, and in collecting issues that arise.
Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-017
Title: facet members `put*()` have no way to detect output
errors.
Sections: 22
Status: active
Description:
Facet members take a single `OutputIterator` and assign characters through it. This interface offers no indication of failure, and no way to limit the number of characters produced.
Discussion:
Part of the semantics of the `put*()` members is to set flags in the `basic_ios<>` argument if an error occurs. To do this they must be able to detect errors.
Proposed Resolution:
As I see it now we have two choices:

1. Specify that `put*()` members do not detect output errors. Iostream functions must check the state of the streambuf after return from the put function and set error state themselves.

2. Add to each put and do_put member another `OutputIterator` argument, end, and require the put members to compare each successive iterator position to end, and report an error if they match. `ostreambuf_iterator` must then be specified so that comparison of a "null iterator" with a blocked iterator yields true.

I don't know yet which alternative to prefer.
Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-018

Title:
Sections: 22
Status: active
Description:

From Box 24:

Is support for syntax like "0xFF" required for iostreams support? If so, we need to add language describing it.

Discussion:

AT&T iostreams did not support it on input, and generated it on output if showbase was set. Other implementations more closely matched C printf/scanf conventions. This may be an iostreams issue.

Proposed Resolution:
Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-019
Title: `numpunct<>::do_grouping` not like C
Sections: 22.2.3.1.2
Status: active
Description:

The result of `numpunct<>::do_grouping()` is a vector with semantics somewhat similar to those of the C Library `lconv::grouping char*` member. It has been suggested they should be identical.

Discussion:

C++ vector<>s are not null-terminated like C strings. The specified semantics is appropriate for vector<>.

Proposed Resolution:

No change. I'd like to state clearly that the C++ Library is not intended as wallpaper over the C Library facilities; and that any similarity between features provided is a result of our intention to provide no less functionality than the C Library, and not because it is meant to be implemented using C Library facilities.

Requestor: P.J. Plauger
Owner: Nathan Myers

Work Group: Library: Localization Clause 22
Issue Number: 22-020
Title: collate virtuals description need editing
Sections: 22.2.4.1.2
Status: active
Description:

1. The names of the virtuals are documented as "hash" and "transform", not "do_hash" and "do_transform" as in the class definition. This is purely editorial.
2. The definition of `do_hash()` is too vague to be normative.
3. The definition of `do_transform` can be misinterpreted to refer to the global `std::compare` rather than the member.
4. Base class semantics is not defined.

Discussion:

For (2):

> The probability that the result equals that for another string which does not compare equal should be very small, approaching (2.0/`numeric_limits<long>::max()`) or less for longer strings.

I don't know any way to describe a hash function more normatively without overconstraining implementors. Let's just consider it non-normative. (I believe "should" signals that already.)

Proposed Resolution:

Fix the member names. Specify `do_tranform` and `do_hash` in terms of `do_compare` so that no confusion is possible. Describe the base class semantics of `do_compare` as performing a lexicographic ordering.

Requestor:

Owner:

Work Group: Library: Localization Clause 22

Issue Number: 22-021

Title: `time_get<>` members need clarification

Sections: 22.2.5.1.2

Status: active

Description:

The descriptions of `time_get<>` members `do_date_order`, `do_get_date`, and `do_get_time` mention a format character 'X' or 'x', but don't say in what context.

Discussion:

`time_get<>` is described as parsing the formats produced by `time_put<>`.

Proposed Resolution:

Specify that the 'X' or 'x' format character is as interpreted by `time_put<>::do_put`.

Requestor:

Owner:

Work Group: Library: Localization Clause 22

Issue Number: 22-022

Title: `time_get<>::get_*` error semantics incomplete

Sections: 22.2.5.1.2

Status: active

Description:

The descriptions of `time_get<>::do_get_date` and `do_get_time` don't say how many characters are consumed if a recognizable date format is not available.

Discussion:

It was intended that these functions not be as rigorously defined as the monetary and numeric parsers, to allow implementors more latitude in recognizing the many variations in notation. However, we should not allow these functions to consume an infinite number of characters just because of an error.

Proposed Resolution:

Specify, in the event of a bad input format, one of:

1. the functions consume no control characters that are not found in the output format.

2. the functions consume no end-of-line characters, as defined by `ctype<>::widen('\n')`.

Note that `get_monthname` and `get_weekday` are already completely specified.

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-023
Title: `time_put<>::put(... const char* ...)` multibyte mistake
Sections: 22.2.5.3.1
Status: active
Description:

From the Draft:

> The first form interprets the characters between `pattern` and `pat_end` identically as `strftime()`, (though not treating the null character as a terminator).

This, unfortunately and unintentionally, implies that put identifies multibyte characters in the argument string and treats them as units according to the current global locale.

Discussion:

A key design criterion in internationalizing the C++ Library was to keep multibyte character representations off in the margins of a system; wherever characters are treated in memory, large character sets are represented using `wchar_t` or user character types.

`time_put<>::put` would be the only exception to that rule, which introduces a variety of issues we have been careful not to need to address.

Proposed Resolution:

Replace the above text:

> The first form interprets characters immediately following a '%' in the sequence between `pattern` and `pat_end` as format specifiers, in according to the mapping used by the `<ctime>` function `strftime()`. Characters are converted using `ctype<>::narrow()` to identify format specifiers. [Note: this implies that if `narrow()` has no mapping for the character '%', no format specifiers are identified.]

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-024
Title: `money_get<>` needs static const member `intl`
Sections: 22.2.6.1
Status: active
Description:

`money_put<>` and `moneypunct<>` both have a public member:

```
static const boolean intl = Intl;
```

to mirror their template argument. This was inadvertently omitted from `money_get<>`.

Discussion:

All the library components mirror their template parameters, or should. This allows access to the parameter in the case another template is instantiated on the component type; the argument is otherwise unavailable.

Proposed Resolution:

Add the member to `money_get<>`.

Requestor:

Owner:

Work Group: Library: Localization Clause 22

Issue Number: 22-025

Title: Facet members `string` and `ios` are confusing

Sections: 22 (many)

Status: active

Description:

Many of the facets declare public typedefs

```
typedef basic_string<charT> string;
typedef basic_ios<charT> ios;
```

for convenience in declaring member arguments and return types. In some cases, "string" and "basic_string<char>" are both used in a declaration. This is confusing, because the global "string" is identical with "basic_string<char>".

Discussion:

Other typedefs used look like "char_type". We should be consistent.

Proposed Resolution:

Change the member typedefs in facets from "string" and "ios" to "string_type" and "ios_type", and change the member function declarations to match.

Requestor: John Dlugosz <jdlugosz@objectspace.com>

Owner:

Work Group: Library: Localization Clause 22

Issue Number: 22-026

Title: `money_get<>` and `money_put<>` need control for currency symbol

Sections: 22.2.6.1.2

Status: active

Description:

It is very common to format monetary values both with and without a currency symbol in the same application. Therefore a runtime control is needed on whether it is required, particularly for formats in which it appears after the value.

Discussion:

The ios flag showbase is otherwise unused for monetary formats.

Proposed Resolution:

For money_get<>::get():

If `showbase` is off, the currency symbol is optional; if it appears after all other required elements, it is not consumed. [See issue 27.] If `showbase` is on, the currency symbol is required, and always consumed. Example: if `showbase` is off, then in "(100 L)" (when the sign is "()") the "L" is consumed; in "-100 L" (when the sign is "-") it is not.

For money_put<>::put():

The currency symbol is emitted only if showbase is on.

[The draft already says this.]

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-027
Title: `do_positive_sign` and `do_negative_sign` are not right yet
Sections: 22.2.6.3.2
Status: active
Description:

The description of `do_negative_sign()`:

> Returns:
> The string to use to indicate a negative monetary value.
> Notes:
> If it is a one-character string containing '(', it is paired with a matching ')'.

is both vague and limiting. We should be able to do much better.

Discussion:

The intention was to support notations in which negative values are represented in parentheses: ($100.00). We could use a special value in the format, and give users no choice of bracketing; but I think we can do better.

Proposed Resolution:

Begin by merging the descriptions of members `do_positive_sign` and `do_negative_sign` -- no special case for negative. Then: if it returns a string containing more than one character, the first appears in the position specified by the format and the remaining characters appear after all other format elements. When parsing, if the first character of a sign is recognized, any subsequent characters are required. (E.g. "($100.00" would not be a valid monetary value. Also, in "(100 L)" the "L" is consumed even if showbase is false.

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-028
Title: messages catalog identifier underspecified
Sections: 22.2.7.1
Status: active
Description:

From the Draft:

> We should clarify the meaning of
> THE_POSIX_CATALOG_IDENTIFIER_TYPE above.

Discussion:

Each execution environment that provides message catalogs has its own identifiers for them.

Proposed Resolution:

State that the message catalog member typedef is implementation-defined. The requirements on it are that it needs a default value, `catalog()`, and copy operators, that do not throw exceptions. User programs cannot safely copy a catalog value after it has been closed. (Thus, it may be a pointer.)

Requestor:
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-029
Title: `codecvt<>::convert` boundary condition imprecise
Sections: 22.2.1.4.2
Status: active

Description:

The description of `codecvt<>::convert` has a note:

> Does not write into `*to_limit`.

As Plauger points out, this doesn't say what it does instead -- is it allowed to skip over `*to_limit` and keep writing?

Discussion:

Obviously not.

Proposed Resolution:

Remove the offending sentence. Add after the first sentence in the preceding paragraph:

> "It produces no more than (`to - to_limit`) characters."

Requestor: P.J. Plauger
Owner:

Work Group: Library: Localization Clause 22
Issue Number: 22-030
Title: Do facet gets / puts throw on error?
Sections: 22 (many)
Status: active

Description:

When a facet member get or put identifies an error, it is documented as setting a bit in its "ios_type" argument's iostate. In iostream, when this happens an exception is thrown if the corresponding bit is set in the exception state. Does an exception get thrown under the same circumstances in locale functions? The Draft is inconsistent.

Discussion:

If the locale doesn't throw, iostreams must check the error state itself and throw; if locale throws, iostream probably needs to catch and rethrow.

Proposed Resolution:

A choice:

1. Locale members throw if the exception bit says so.
2. Locale members don't throw, the only set iostate.

I don't know which is better, but I lean toward 1.

Requestor:
Owner:
