WG21/N0891
X3J16/96-0073
1996-03-13
Thomas Plum

To: C++ libraries mailing list
Message c++std-lib-4502

I've had difficulty transmitting the entire locales issue list. Besides, when we work our way through the open issues in Santa Cruz, I believe we'll make progress better if we take them in groups. So, this first batch of open issues are relatively non-controversial, but require real decisions between alternative choices. I'll propose that we tackle them first.

"Decisions Needed" Issues

Issues: 22-054, 22-056, 22-057, 22-060, 22-062

---

** Work Group: Library: Localization Clause 22
** Issue Number: 22-054
** Title: should num_put<>::get(..., unsigned long) accept sign?
** Sections: [lib.locale.num.get]
** Status: closed, already in WP

** Description:

The WP isn't clear whether the sequence "-1" is a valid specification of an unsigned integer, or what the resultant unsigned value should be. I believe scanf() accepts that sequence, but I don't know whether all traditional iostream implementations do.

** Discussion:

We should match either the consensus of existing implementations, if such exists, or scanf(), or what makes more sense, collectively.

** Proposed Resolution:

One of:
1. signs are recognized on unsigned integer formats.
2. signs are not recognized on unsigned integer formats.

| Specified by '%u'
| Same as C; signs are recognized.

** Requestor:
** Owner: Plum

---

** Work Group: Library: Localization Clause 22
** Issue Number: 22-056
** Title: What is the default monetary format?
** Sections: [lib.locale.moneypunct]
** Status: closed, already in WP

** Description:

The WP doesn't say what formats are specified by the base class numpunct<>::do_pos_format and do_neg_format().

** Discussion:

The choice is rather arbitrary, but should probably match the
format for international monetary values.

>From Plum:
The subgroup minutes from Tokyo say "present the proposed
resolution as written", but there was no such motion.
I'm not sure what the "proposed resolution" specifies?

** Proposed Resolution:

Specify that the default return value of numpunct<>::do_[pos|neg]_format()
is { symbol, sign, value, none }.

** Requestor:      Ramesh S. <ramesh@modena.uunet.in>
** Owner:          Plum

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-057
** Title:          money_put<> and _get<> facets missing from table
** Sections:       [lib.locale.category], [lib.locale.money.get],
                   [lib.locale.money.put]
** Status:         active

** Description:

In the table of locale facet categories, and the facets that
are required to be provided as part of each, the facets
money_[get|put]<[char|wchar_t],true> are not mentioned.
They either need to be added, or the second template parameter
          [They are now in WP.]
eliminated in favor of an argument on the put() and get()
members.
[this is an Editorial Proposal.]

** Discussion:
It is easy to imagine code in which the choice of local
or international monetary format is made at runtime, so
a template parameter to num_put and num_get would make
such code unnecessarily clumsy.

** Proposed Resolution:
In both money_put<> and money_get<> (but not in moneypunct<>):

1. Eliminate the second template parameter "bool Intl = false",
   and the static const member "bool intl".

2. Add to the members money_get<>::get() and money_put<>::put()
   another argument, immediately after the "const locale&" argument,
   of type bool:

      ...const locale&, const bool intl, ...

   Specify that if true, the function uses the moneypunct<charT,true>
   facet, else it uses the moneypunct<charT,false> facet, of the locale
   argument to control [parsing|formatting].

** Requestor:
** Owner:          Plum

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-060
** Title:          time_get::do_get_year spec more vague than necessary.

** Sections:       [lib.locale.time.get.virtuals]
** Status:         closed, no change to WP

** Description:

The description of the response of time_get::do_get_year to a two-digit year says it is "unspecified" whether or how two-digit year values are interpreted.  This leaves implementations a little too much latitude. [this is an Editorial Proposal.]

** Discussion:

It should be "implementation-defined".

** Proposed Resolution:

Change the description for do_get_year to say "It is implementation-defined whether two-digit year numbers are accepted, and if so what century they are assumed to lie in."

** Requestor:
** Owner:          Myers

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-062
** Title:          ctype<char> member table() size overspecified
** Sections:       [lib.locale.ctype.char.members]
** Status:         Santa Cruz motiion passed with changes; see N0886

** Description:

The size of the table used by ctype<char> is specified in the WP to be of size UCHAR_MAX+1.  This may be an arbitrarily large value, too large for practicality in some implementations.

** Discussion:

[original proposal, from 1995]
Change the specification to say that the table has an implementation-defined size, less than or equal to numeric_limits<unsigned char>::max() + 1, sufficient to map all valid character values.

[From Myers, 1996]
There is no reason all implementations need to have a table this large, if it is large.  For very large values of UCHAR_MAX, the set of actual character codes used is always much smaller. For values that don't encode a character, is() should always return false, and therefore don't require table entries.  The only problem is to know how large the table must be.

** Proposed Resolution:
>From Myers:
In [lib.facet.ctype.special] 22.2.1.3

  Add a public member:

    static const size_t table_size = IMPLEMENTATION_DEFINED;

  Add a paragraph:

    The value of table_size is at least 256, and greater than the maximum permitted source character set literal.

In [lib.facet.ctype.char.members] 22.2.1.3.2
add a paragraph before the member descriptions:

  For character values v greater than or equal to table_size,
  the value of table()[(unsigned char)v] is assumed to be zero
  without performing the array lookup.

In [lib.facet.ctype.char.statics] 22.1.3.3

  Change the table size to "table_size".

In [lib.facet.ctype.char.members] 22.2.1.3.2, in the constructor, add:

  Precondition: tab either 0 or an array of at least table_size
   elements.

---

Thomas Plum  Plum Hall Inc, PO Box 44610, Kamuela HI 96743 USA
plum@plumhall.com [old]  TEL +1-808-882-1255  FAX +1-808-882-1556
plum@plumtest.com [better]

To: C++ libraries mailing list
Message c++std-lib-4501

"Clarity" Issues

Issues: 22-004, 22-016, 22-030, 22-038

"Clarity" Issues - Generally just need to clarify draft words

---

** Work Group:  Library: Localization Clause 22
** Issue Number:  22-004
** Title:  Description of _byname facets too vague
** Sections:  [lib.locale.facet]
** Status:  active

** Description:
Paragraph 4, where _byname<> classes are described, leaves some
(unspecified) issues unresolved.

** Discussion:
The following was added 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."

[Plauger has remarked that other matters remain unspecified.
TODO: Plauger: suggest other clarifications.]

** Proposed Resolution:

** Requestor: P.J. Plauger
** Owner:

---

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

** Status:  editorial

** 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, the way in which they reference C Library semantics have been alleged to be incompatible with the C++ Library environment.

[TODO: Can anyone provide details regarding the alleged incompatibility?]

** Discussion:
>From Plauger:
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?

** Proposed Resolution:
Encourage editoral aggressiveness in consolidating the descriptions of parsing and formatting, and in collecting issues that arise.

See also issue 22-050.

** Requestor: Several
** Owner:

---

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

** Description:
When a facet member get 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, istream must check the error state itself and throw; if locale throws, iostream probably needs to catch and rethrow.  Thus, we have a choice:

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

(1) implies that it is sufficient for the facet to call setstate(failbit), which is the status quo.  This implies that when istream calls these members, it must turn off the exception control bits, and restore them after the member returns.  At the same time, it must catch any exceptions thrown.

** Proposed Resolution:
No change.  This implies that throwing behavior is controlled by ios_base::exception() state.

** Requestor:
** Owner:

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-038
** Title:          lifetime semantics unclear
** Sections:       22.1.1.3
** Status:         closed, no change

** Description:
>From Plauger:
  Description of locale::use Notes uses the term ‘‘value semantics’’
  and the verb ‘‘to last.’’ Are either of these terms defined within
  the Standard? The sentence should be reworded, or struck since it’s
  non-normative anyway.

>From Myers:
The intent is clear; we need normative language to express it.

** Proposed Resolution:
Editorial.

** Requestor:      Plauger
** Owner:

---

---
Thomas Plum              Plum Hall Inc, PO Box 44610, Kamuela HI 96743 USA
plum@plumhall.com [old]  TEL +1-808-882-1255  FAX +1-808-882-1556
plum@plumtest.com [better]


To: C++ libraries mailing list
Message c++std-lib-4503

Issues: 22-017, 22-046, 22-050, 22-067

"Ratify Existing Words" Issues - unless there are specific proposals in
the Santa Cruz mailing, I propose that all these issues appear in one
motion to "close the issue without any further action".  Other Library
groups have documented their decision process by this method, and I’m
convinced that it’s useful to have the proper paper trail.

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-017
** Title:          facet members put*() have no way to detect output errors.
** Sections:       [lib.ostream.formatted.reqmts], paragraph 3
** Status:         active (part)?  mostly passed at Tokyo, motion 21
                   [remainder is editorial]

** Description:
Facet members take a single Output Iterator and assign characters
through it.  This interface offers no indication of failure, and no
way to limit the number of characters produced.

** Discussion:
ostream operators << are required to report output errors, but
the put() members give no indication of errors because the Output
Iterator interface used by the put() members offers no indication
of failure.  However, an iterator that fails can record the fact.

The ostreambuf_iterator<> that ostream passes to the put() members
can be extended to provide the necessary semantics.

```
** Proposed Resolution:
Add to ostreambuf_iterator the following member:

  bool failed() const throw();

  Returns: false if no previous call to streambuf::sputc reported
   a failure, true otherwise.

  [ Already in WP ]

Add to [lib.ostream.formatted.reqmts], paragraph 3:

  Checks for a failure by calling ostreambuf_iterator<>::failed(),
  and calls setstate(failbit) if true.
                    ^^^^^^^
                    should say  badbit  - JSS, NM

                   [ see issue 27-403 and 27-914 ]

The example in the same section should be replaced with either:

  { sentry cerberos(*this); if (!cerberos) return;
    iostate save = exceptions(); exceptions(0);
    try {
      if (use_facet< num_put<charT,ostreambuf_iterator<charT,traits> >(
            getloc()).put(*this,*this,fill(),getloc(),val).failed())
        setstate(failbit); // won't throw
    } catch (...) { exceptions(save); setstate(badbit); throw; }
                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                    setstate(badbit); exceptions(save);
                    [reverse these two statements - JSS, NM ]
    exceptions(save);
    setstate(rdstate());  // might throw
  }

if the ostream::sentry proposal passes, or
[it passed, so delete the rest of the proposed resolution ...]

//   if (opfx()) {
//     iostate save = exceptions(); exceptions(0);
//     try {
//       if (use_facet< num_put<charT,ostreambuf_iterator<charT,traits> >(
//             getloc()).put(*this,*this,fill(),getloc(),val).failed())
//         setstate(failbit);  // won't throw
//     } catch (...) {
//       setstate(badbit);  // won't throw
//       exceptions(save);
//       osfx();
//       setstate(rdstate());  // might throw failure
//       throw;
//     }
//     exceptions(save);
//     osfx();
//     setstate(rdstate());  // might throw failure
//   }
//
// if it doesn't.
//
// ** Note:
//
// The complexity of these code snippets reveals the complexity of the
// specification on how iostreams handle exceptions.  Probably we should
// simplify it.  I recommend that a throw of ios_base::failure be propagated
// without affecting the iostate.  Then the example above looks like:
//
```

```
//   { sentry cerberos(*this); if (!cerberos) return;
//     try {
//       if (use_facet< num_put<charT,ostreambuf_iterator<charT,traits> >(
//             getloc()).put(*this,*this,fill(),getloc(),val).failed())
//         setstate(failbit);  // might throw
//     }
//     catch (const ios_base::failure&) { throw; }
//     catch (...)                      { setstate(badbit); throw; }
//   }
```

** Requestor:  Plauger
** Owner:      Myers

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-046
** Title:          Locale facets for money/time/messages need semantics
** Sections:       22.2.4
** Status:         active

** Description:
>From Plauger:
  Template classes collate, time_get, time_put, money_get, money_put,
  money_punct, messages, and their support classes still have only
  sketchy semantics -- over a year after they were originally
  accepted into the draft. They are based on little or no prior
  art, and they present specification problems that can be addressed
  properly only with detailed descriptions, which do not seem to be
  forthcoming. Even if adequate wording were to magically appear
  on short notice, the public still deserves the courtesy of a
  proper review. For all these reasons, and more, the remainder
  of clause 22 from this point on should be struck.

** Discussion:
>From Plum:
  I believe the status quo is that these classes essentially define
  the syntax of an interface, with relatively little detail regarding
  semantics.  At this point, any proposal to add further
  semantic restrictions would be a substantive change to the CD,
  requiring an explicit proposal.

  The text of this issue itself contains a proposal to strike the
  classes itemized above.  In order to reach closure on the issue,
  that proposal should be addressed by the Library group next meeting.
  My opinion is that removing library classes from the CD will be
  considered to be a more drastic course than providing an interface
  with no required semantics behind it, and that the status quo will
  therefore remain in place.

** Proposed Resolution:

** Requestor: Plauger
** Owner:     Plum

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-050
** Title:          case of whitespace thousands separator needs clarification
** Sections:       [lib.locale.num.get], [lib.locale.money.get]
** Status:         closed, no action


** Description:

In some locales, the preferred thousands separator is a space.
We need to clarify how spaces in and around numeric sequences
are handled.

** Discussion:

Where the thousands separator is a whitespace character, two adjacent
whitespace characters must represent a normal end of the number; while
two commas, if comma is the separator, would be an error.  (The first
would be absorbed, leaving the second in the input sequence.)

Examples:  a) "123,456"  b) "123,456, "  c) "123 456"  d) "123 456  "

(b) is ill-formed, because the grouping constraint is not met, and the
final space is not consumed.  However, (d) is good, though again the
final space is not consumed.

I don't think any other special cases ("What if it's a digit?")
are necessary -- if we don't say, the effect is unspecified.

** Proposed Resolution:
Specify for both num_get<> and money_get<> members get():

  Note: if [numpunct|moneypunct]<>::thousands_sep() returns a
    whitespace character, and grouping() is non-empty, then
    that character following a digit, and followed by a non-digit,
    is consumed but not treated as a thousands separator.

Note: Tokyo subgroup decision was to close the issue without action,
but that recommendation has not yet been ratified by WG vote.

** Requestor:
** Owner:          Myers

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-067
** Title:          ios_traits template arg needs approval
** Sections:       lib.locales
** Status:         closed, no action

** Description:

22.1:
template operator<<(basic_ostream, const locale&) as well as
template operator>>(basic_ostream, const locale&) now have a second
template argument (for ios traits) added without approval. While
this change may be a good idea, it should be applied uniformly (which
has not happened), and only after committee approval.

** Proposed Resolution:

This was once issue number 22-031, which somehow got marked as
"closed" before any committee decision was obtained.  If a proposal
for change is received, it will be reviewed on its merits.
Then, the subgroup recommendation will be voted upon by full WG vote.

** Requestor:
** Owner:

---

--------------------------------------------------------------------------
Thomas Plum              Plum Hall Inc, PO Box 44610, Kamuela HI 96743 USA
plum@plumhall.com [old]  TEL +1-808-882-1255  FAX +1-808-882-1556

plum@plumtest.com [better]

To: C++ libraries mailing list
Message c++std-lib-4504

"Transparent Locale" Issues

Issues: 22-034, 22-037, 22-039, 22-058, 22-068, 22-069

These are all related to the controversy over "transparent locales".

At the end, I have attached the collected email conversations
that were relevant to these issues.

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-034
** Title:          locale::transparent restrictions gall
** Sections:       22.1.1
** Status:         motion passed at Santa Cruz, N0860 = 96-0042

** Description:
>From Plauger:
  Paragraph 8 says that locale::transparent() has unspecified behavior
  when imbued on a stream or installed as the global locale. There is
  no good reason why this should be so and several reasons why the
  behavior should be clearly defined. The sentence should be struck.

and later:
  Paragraph 9 says that ‘‘caching results from calls to locale facet
  member functions during calls to iostream inserters and extractors,
  and in streambufs between calls to basic_streambuf::imbue, is
  explicitly supported.’’ In the case of inserters and extractors,
  this behavior follows directly from paragraph 8. No need to say it
  again. For basic_streambuf, the draft can (and should) say explicitly
  that the stream buffer fixates on a facet at imbue time and ignores
  any subsequent changes that might occur in the delivered facet
  until the next imbue time (if then). (An adequate lifetime for the
  facet can be assured by having the basic_streambuf object memorize
  a copy of a locale object directly containing the facet,
  as well as a pointer to the facet, for greater lookup speed.)
  In any event, saying something ‘‘is explicitly supported’’ doesn’t
  make the behavior *required.* The paragraph should be struck, and
  words added to the description of basic_streambuf to clarify the
  lifetime of an imbued codecvt facet. (More words are needed here
  anyway, for other reasons.)

** Discussion
Clarification is clearly needed.

>From Myers:
If a streambuf made a copy of an imbued transparent locale, it
would be fixated on nothing -- any change of the global locale
could eliminate the facets it refers to.  If it constructs a new
locale object to keep the facet, it is the same as if the current
global locale was imbued.

** Proposed Resolution:

** Requestor:      Plauger
** Owner:          Plum

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-037
** Title:          locale immutability disputed
** Sections:       22.1.1.3 [lib.locale.cons], [lib.locale.globals]
** Status:         closed by motion, N0860 = 96-0042

** Description:
>From Plauger:
  Description of locale::use() Effects contains a nonsense statement:
  ‘‘Because locale objects are immutable, subsequent calls to use<Facet>()
  return the same object, regardless of changes to the global locale.’’

  If a locale object is immutable, then changes to the global locale
  should *always* shine through, for any facet that is not present
  in the *this locale object. If the intent is to mandate caching
  semantics, as sketched out in the original locales proposal, this
  sentence doesn’t quite succeed. Nor should it. Caching of facets
  found in the global locale leads to horribly unpredictable behavior,
  is unnecessary, and subverts practically any attempt to restoee
  compatibility with past C++ practice and the current C Standard.
  The sentence should be struck.

** Discussion:
>From Myers:
locale objects are immutable in the sense that a call to use_facet<>
or has_facet<> on a given locale object always yields the same result.
If changing the global locale led to varying results from use_facet<>,
the objects could not be treated as immutable, and users could not
cache the result of use_facet<>.

Here is the invariant:  use_facet<facet>(loc) always yields the same
result for any locale value.

Here is what I see as the alternatives that maintain this:
1. Status quo:
  If it has been called successfully before, return the same value.
  Else, if the locale was constructed with such a facet, return it.
  Else, if the global locale has such a facet, return it.
  Else, throw bad_cast.

2. Strict:
  If the locale was constructed with such a facet, return it.
  Else, throw bad_cast.

3. Compromise:
  During construction of a locale (except copy), all facets found in
  the global locale, and not specified otherwise, are incorporated.
  Otherwise, same as (2).

(3) eliminates any possible confusion about the source of
the facet used.  It means that if a facet might not be present, and
you want to use the current global one instead, you can say so:

  const foo& f = use_facet<foo>(has_facet<foo>(loc) ? loc : locale());

This change substantially simplifies the semantics of use_facet,
without reducing its convenience.

** Proposed Resolution
Adopt (3), by adding a paragraph at the beginning of [lib.locale.cons]:

  For all constructors except locale(const locale&), any facet found
  in the current global locale at the time of construction, and not
  otherwise specified by arguments, becomes a part of the constructed
  locale.

And change the descriptions of use_facet<> and has_facet<> as follows:

  template <class Facet> const Facet& use_facet(const locale& loc);

  Get a reference to a facet of a locale.
  Returns: a reference to the corresponding facet of loc, if present.
  Throws: bad_cast if the facet is not present.
  Notes: if loc is (a copy of) locale::transparent(), returns a
   reference to the corresponding facet in locale().  The reference
   returned remains valid as long as any copy of the locale it came
   from exists.

And

  template <class Facet> bool has_facet(const locale& loc) throw();

  Returns: true if loc was constructed with the specified facet.
  Notes: Returns false when applied to (a copy of) locale::transparent().

** Requestor:      Plauger
** Owner:          Plum

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-039
** Title:          transparent locale wanted in iostream
** Sections:       22.1.1.5
** Status:         closed by motion, N0860 = 96-0042

** Description:
>From Plauger:
  locale::transparent() Notes says ‘‘The effect of imbuing this locale
  into an iostreams component is unspecified.’’ If this is a normative
  statement, it doesn’t belong in a Notes clause. And if it’s intended
  to be normative, it should be struck. Imbuing a stream with
  locale::transparent() is the *only* way to restore the behavior
  of iostreams to that in effect for *every* C++ programming running
  today. It is also essential in providing compatible behavior with
  the C Standard. The sentence should be struck.

>From Myers:
  It was specifically intended and stated in the enabling proposal
  that imbuing a transparent locale should be forbidden to provide
  operators << and >> stable semantics from locale objects,
  incompatible with "transparent" behavior.

** Proposed resolution:

** Requestor:      Plauger
** Owner:          Plum

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-058
** Title:          locale constructors with null arguments unspecified.
** Sections:       [lib.locale.cons]
** Status:         closed by motion at Santa Cruz, see N0887 = 96-0071

** Description:

For those locale constructors that take a pointer, it is not
documented what happens if the pointer is 0.  Also, for those
constructors that take another locale and copy facets from it,
it is not specified what happens if it is the transparent

locale.

[this is an Editorial Proposal.]

** Discussion:

They should throw something.

** Proposed Resolution:

Add to each such constructor that takes a pointer argument: if the
pointer argument is 0, throws runtime_error.

Add to each constructor (except the copy constructor) that takes a
locale argument: if the locale argument is equal to (a copy of)
locale::transparent(), throws runtime_error.

** Requestor:
** Owner:          Plum

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-068
** Title:          locale(const locale& other, category)  got dropped
** Sections:       lib.locale
** Status:         close, no change

** Description:

22.1.1:
Class locale is missing the constructor:
 locale(const locale& other, category) (Added Nov. ’94).
 It should be added.

** Discussion:
>From Plum:
I believe that this is conceptually related to the status of
transparent locales.

** Proposed Resolution

This was once issue number 22-033 part 1, which somehow got marked as
"closed" before any committee decision was obtained.  If a proposal
for change is received, it will be reviewed on its merits.
Then, the subgroup recommendation will be voted upon by full WG vote.

** Requestor:
** Owner:

---

** Work Group:     Library: Localization Clause 22
** Issue Number:   22-069
** Title:          locale(const locale& other, const locale& one, category) added
** Sections:       lib.locale
** Status:         close, no change

** Description:

 22.1.1:
 Class locale has the constructor locale::locale(const locale& other,
 const locale& one, category). I can find no resolution that calls
 for this constructor to be added.

** Proposed Resolution

This was once issue number 22-033 part 2, which somehow got marked as
"closed" before any committee decision was obtained.  If a proposal
for change is received, it will be reviewed on its merits.
Then, the subgroup recommendation will be voted upon by full WG vote.

** Requestor:
** Owner:

---