Clause 24 (Iterators) Issues List

David Dodgson
dsd@tr.unisys.com
UNISYS

The following list contains the issues for Clause 24 on Iterators. The list is divided based upon the status of the issues. The status is either *active* - under discussion, *resolved* - resolution accepted but not yet in the working paper, *closed* - working paper updated, or *withdrawn* - issue withdrawn or rejected. They are numbered chronologically as entered in the list. Only the active and resolved issues are presented here. Those wishing a complete list may request one.

The proposed resolutions are my understanding of the consensus on the reflector.

1. *Active Issues*

```
---------------------------------------------------------------------------
 Work Group:     Library Clause 24
 Issue Number:   24-012
 Title:          Addition operators added to iterators
 Section:        24.1
 Status:         active
 Description:
         24.1.3-24.1.5 p24-3 to 24-6:
   Add addition and subtraction operators to forward, bidirectional and
   reverse iterators.

 Alex Stepanov in lib-3611:
   And if you reconsider the iterator requirements, you might as well
   reconsider the exclusion of + (and related operators) for non-random
   iterator categories. I really hate advance and distance
   templates. They are such a pain to use and they are really ugly. (To
   see what I mean, take a look at what we now need to do to implement,
   say, lower_bound algorithm. It is in algo.h in our implementation.)

 Later discussions show that this should not include output iterators,
 and at most only - operations for input iterators.

 Discussion at Monterey meeting:
    The library subgroup was in favor of this change.  It was felt
    that the added convenience of this is worthwhile.  The cost
    is in making it unclear that '+' can be a linear operation.  If
    operator+ is added to the base forward iterator in terms of using
    the advance template it should add little cost to existing
    implementations.

 Proposed Resolution:
    See paper N0739R1 in the post-Monterey mailing.

 Requestor:      Alex Stepanov
 Owner:          David Dodgson (Iterators)
 Emails: lib 3611-3613
 Papers: Operator+ and Operator- for Iterators, 95-0139/N0739R1,
            David Dodgson, post-Monterey


---------------------------------------------------------------------------
```

Work Group: Library Clause 24
Issue Number: 24-015
Title: Char-oriented stream iterators
Section: 24.4.3 [lib.istreambuf.iterator]
Status: active
Description:

24.4.3 p24-23: [Box 118]

The istream_iterator and ostream_iterator are defined only for the char-oriented, but not the wchar_t-oriented or parameterized streams.

Resolution:
Requestor: Editorial Box
Owner: David Dodgson (Iterators)
Emails:
Papers:

Work Group: Library Clause 24
Issue Number: 24-017
Title: Exceptions in ostreambuf_iterator
Section: 24.4.4 [lib.ostreambuf.iterator]
Status: active
Description:

24.4.4.1 and 24.4.4.3:

Nathan Myers in message lib-3812:

As Plauger noted in some previous mail relating to locale, the ostreambuf_iterator used to decouple iostream from the locale facet interface provides no mechanism for reporting output errors. Changing the interface to allow the iterator to be compared against an "end" iterator doesn't help; again (as Plauger points out) the Output Iterator abstraction doesn't support comparison, and standard algorithms don't assume it.

** Discussion

Failures of abstraction in C++ are handled by throwing an exception. Output Iterators are, in general, allowed to throw if they cannot perform an operation; it is necessary only to specify that this is what ostream_iterator does.

ostream and locale need to have specified what happens in the event of an output error, so the failure can be handled according to ostream's policy without imposing knowledge of it upon all locale facets.

Proposed Resolution

1. Specify that operator=(charT c) throws an exception catchable as type runtime_error if sbuf_->sputc(c) returns traits::eof().

2. Eliminate ostreambuf_iterator members equal() and global operators == and !=. No function that takes an iterator can use them anyway, so they only add clutter. (This does not imply any corresponding changes to istreambuf_iterator.)

Requestor: Nathan Myers
Owner: David Dodgson (Iterators)
Emails: lib-3809,3812
Papers:

Work Group: Library Clause 24
Issue Number: 24-018
Title: Cleanups in [io]streambuf_iterator
Section: 24.4.3 and 24.4.4
Status: active
Description:

24.4.3 [lib.istreambuf.iterator] and
24.4.4 [lib.ostreambuf.iterator]:

1. The typedefs declared in the streambuf iterators can lead to confusion because the have the same names as global typedefs. While this does not confuse compilers, it confuses readers, and is easily fixed.

2. The description of semantics of istreambuf_iterator member operators is too vague: "Advances the iterator" and "Extract one character" are subject to interpretation.

Proposed Resolution

1. In both 24.4.3 and 24.4.4, change the typedefs "streambuf", "istream", and "ostream" to "streambuf_type", "istream_type", and "ostream_type", respectively, to prevent confusion. (The declarations of the constructors and the private members "sbuf_" should be changed to match.)

2. In 24.4.3.2 and 24.4.3.3, the descriptions of istreambuf_iterator operators have become unfortunately vague:

   - operator*() should be documented to return (specifically) the result of calling (the equivalent of) sbuf_->sgetc().

   - operator++() should be documented to perform (the equivalent of) sbuf_->snextc().

   - operator++(int) should be documented to return a proxy object constructed from (the equivalent of) the expression proxy(sbuf_->sbumpc(),sbuf_).

Requestor: Nathan Myers
Owner: David Dodgson (Iterators)
Emails: lib-3812
Papers:

Work Group: Library Clause 24
Issue Number: 24-021
Title: Separate Header for Stream Iterators
Section: 24.4
Status: active
Description:

24.4:

From public review:
Drawing iostream into an implementation that just needs iterators is most unfortunate.

The current iterator header includes headers <ios> and <streambuf> to handle the stream iterators in 24.4. This requires all of I/O to be included in the iterators header. Yet I/O only needs this if the iterators are used.

```
If a new header is used should it be in clause 24 or in clause 27?
Is <iositer> a good name for the new header?

Proposed Resolution:
   Move the stream iterators into a separate header.

 Update Table 55 (pg 24-1) to include header <iositer> in 24.4.

 Remove #include's for iosfwd, ios, and streambuf from 24.1.6
 [lib.iterator.tags] Header <iterator> synopsis and tags for
 subclause 24.4.  Create new header <iositer> in section 24.4 with
 #include's for iterator, iosfwd, ios, and streambuf and tags for
 section 24.4 from the <iterator> header.

Requestor:      Public Review & Library WG
Owner:          David Dodgson (Iterators)
Emails:
Papers:


--------------------------------------------------------------------------
Work Group:     Library Clause 24
Issue Number:   24-022
Title:          Input Iterator Semantics
Section:        24.1.1
Status:         active
Description:
        24.1.1 p24-2:
What are the semantics of input iterators in the following:
   input_iterator i;
   cout << *i;       // Object "a"
   cout << *i;       // Continues to return object "a"?
    /* This seems to be implied by requirement a == b implies *a == *b.
       Therefore *a == *a should be true.
       This implies the input object is 'saved' in some fashion. */

   input_iterator j = i;
   cout << *j;       // Object "a"
   ++i;
   cout << *i;       // Object "b"
   cout << *j;       // Object "a", "b", or undefined?
    /* Returning "b" implies that all input iterators remain in
       lockstep and all point to the same item.  This is not how
       other iterators work.

       Undefined implies that changing a different iterator can
       affect the value of this iterator, even though no change
       has been made to this iterator.

       Returning "a" is how other iterators work.  It implies that
       the 'saved' object is not destroyed when an input occurs.
       Bill Plauger states that several STL algorithms depend on
       this behaviour.
    */
   ++j;              // What is the effect on j after i has been
                     // incremented; object "b", "c", or undefined?
Nathan Myers proposes a change to semantics of copying in 3943:
   Copying an input iterator should invalidate the previous version
   of the iterator.  This ensures that there is only one current
   version of the iterator usable for ++.  It would prevent
   dereferencing of the copied version of the iterator. [See 3943
   for a complete description].
```

Andy Koenig proposes a simplified scheme in 4114:
Copying an input iterator does not invalidate the previous version. Both may be dereferenced until one is incremented. It is undefined to dereference any copy other than the one incremented.

Summary
Various other messages discuss the relative merits of different semantics. There are three proposed methods.

-- Local Copy
This method requires the iterator to retain a separate copy of the object. Any copy of the iterator has a distinct copy of the object.

After a copy (b) is made of an iterator (a) then both a == b and *a == *b return true.

If a copy is incremented, the value returned by dereferencing a previous copy is well-defined and unchanged.

*a++ is valid and probably implemented by returning a temporary copy of the iterator for operator++.

-- Global Copy
This method allows the iterator to point to one particular copy of the object. Any copy of the iterator may point to the same copy of the object.

After a copy (b) is made of an iterator (a) then both a == b and *a == *b return true.

If a copy is incremented, it is undefined behaviour to dereference a previous copy.

*a++ is valid and probably implemented by returning a temporary instance of a proxy class (although an iterator implemented with local copy semantics will conform to global copy requirements).

-- Unique Copy
This method allows only one valid copy of an iterator to be dereferenced at a time.

After a copy (b) is made of an iterator (a) then a == b returns true but *a == *b is undefined, because *a is undefined.

It is undefined behaviour to increment any iterator other than the one which may be dereferenced.

*a++ is valid is probably implemented by returning a temporary instance of a proxy class.

Resolution:
Local copy is the status quo. Global copy has been proposed by Andy Koenig, unique copy by Nathan Myers. One specific method must be chosen. Consensus seems to be towards global copy but the issue is controversial.

Requestor: Library WG
Owner: David Dodgson (Iterators)
Emails: lib-3938,3941-3950,3956-3959,4013-4050,4055-4059, 4068-4070,4074,4081,4084,4086-4088,4114-4118, 4122-4127,4132-4138,4141

```
Papers:


----------------------------------------------------------------------------
Work Group:     Library Clause 24
Issue Number:   24-023
Title:          Bad description for istreambuf_iterator constructors
Section:        24.4.3.2 [lib.istreambuf.iterator.cons]
Status:         active
Description:
        24.4.3:
 The header for istreambuf_iterator contains the constructor
    istreambuf_iterator( streambuf* s );
 but there is no description for this constructor in section
 24.4.3.2 [lib.istreambuf.iterator.cons].

 The description for constructor istreambuf_iterator(istream) states
 the effect as constructing 'istream_iterator' not
 'istreambuf_iterator'.

Proposed Resolution:
 Add the description for constructor istreambuf_iterator(streambuf* s)
 in section 24.4.3.2 [lib.istreambuf.iterator.cons] with effects of
 constructs the istreambuf_iterator pointing to s.

 Change 'istream_iterator' on line 4 to 'istreambuf_iterator'.

Requestor:      Library WG
Owner:          David Dodgson (Iterators)
Emails:
Papers:


----------------------------------------------------------------------------
Work Group:     Library Clause 24
Issue Number:   24-024
Title:          Operator -> Issues for Iterators
Section:        24.1.3, 24.1.1
Status:         active
Description:
        24.1.1, 24.1.3 p24-2,4:
  Should operator->* be added for iterators?
     Section 14.3.3 [temp.opref] specifically allows operator-> to
     appear in a template where its return type cannot be dereferenced
     if it is not used.  No such guarantee is made for operator->*.
     If operator->* is desired, the same guarantee should be made.

  Does operator-> work correctly for input iterators? (*a can
  return an rvalue).

Resolution:
Requestor:      Library WG
Owner:          David Dodgson (Iterators)
Emails:
Papers:


----------------------------------------------------------------------------

Work Group:    Library Clause 24
Issue Number:  24-025
Title:         Input Iterator Assignment
Sections:      [lib.stream.iterators], [lib.istreambuf.iterator],
```

```
 Status:        Active
 Description:
        24.1.1 p24-2 [lib.input.iterators]


From Nathan Myers:

This is what I want to say:
  template <class T> void f(const T&);

  // with assignment:
  template <class InputIterator, class Pred>
  void grab(InputIterator begin, InputIterator end, Pred const& pred)
  {
    while ((begin = find(begin, end, pred)) != end)
      f(*begin++);
  }

But without assignment, I can't.

 Proposed Resolution:
   Add the following to table 57-'Input Iterator requirements' and
   table 58-'Output iterator requirements' in sections 24.1.1
   [lib.input.iterators] and 24.1.2 [lib.output.iterators]:
   u = a      X&            post: u is a copy of a

 Requestor:   Nathan Myers
 Owner:       David Dodgson
 Emails:      lib-3936,3939-3940,3942-3943,4114,4116-4118
 Papers:


--------------------------------------------------------------------------
 Work Group:     Library Clause 24
 Issue Number:   24-026
 Title:          Istream Iterator Interactive Input
 Section:        24.4.1
 Status:         active
 Description:
         24.4.1 p24-22:

Bernd Eggink in lib-4007
> No, it does not. The code in HP implementation is:
>
> class istream_iterator : public input_iterator<T, Distance> {
> friend bool operator==(const istream_iterator<T, Distance>& x,
>          const istream_iterator<T, Distance>& y);
> protected:
>     istream* stream;
>     T value;
>     bool end_marker;
>     void read() {
>  end_marker = (*stream) ? true : false;
>  if (end_marker) *stream >> value;
>  end_marker = (*stream) ? true : false;
>     }
> public:
>     istream_iterator() : stream(&cin), end_marker(false) {}
>     istream_iterator(istream& s) : stream(&s) { read(); }
>     const T& operator*() const { return value; }
>     istream_iterator<T, Distance>& operator++() {
>  read();
>  return *this;
```

```
>     }
>     istream_iterator<T, Distance> operator++(int)  {
>  istream_iterator<T, Distance> tmp = *this;
>  read();
>  return tmp;
>     }
> };
>
```

BTW, this implementation is practically unusable for interactive input because of the read() in the constructor (which could be easily eliminated
by introducing a bool member which tells whether or not the current element
has been read).

Nathan Myers in lib-4010
I agree that the above change should be made. Who will write up the WP changes? (The delta in the WP would be quite small: I believe it would involve two paragraphs.) This would not break code.

Resolution:
Requestor: Bernd Eggink
Owner: David Dodgson (Iterators)
Emails: lib-4007,4010
Papers:

Work Group: Library Clause 24
Issue Number: 24-027
Title: Istream Iterator Semantics
Section: 24.4
Status: active
Description:
24.4.3:

24.4.3.5 equal seems at variance with the standard definition. If istreambuf_iterator i = j; Then i == j should be true even if not at end-of-stream.

In general, the semantics of istream_iterator should conform to the semantics of input iterators in general. See issue 22 for a resolution to input iterator semantics. Once input iterator semantics are resolved, the semantics for istream iterators must be examined.

Also, the level of detail specified for istream iterators in the standard must be determined. To what extent should the details of istream iterator be defined? Should specific iostream calls be mandated? Must further explanation of items already defined by input iterator semantics be given. For example, does the 'proxy' class need to be specified or should that be left up to the individual implementation of input iterator?

Resolution: Dependent on issue 22
Requestor: David Dodgson
Owner: David Dodgson (Iterators)
Emails: lib-4065-4069
Papers:

2. *Resolved Issues*

```
---------------------------------------------------------------------------
 Work Group:     Library Clause 24
 Issue Number:   24-003
 Title:          const operation for iterators
 Section:        24.3
 Status:         resolved
 Description:
         24.3.1 p24-13  Box 116
    Suggest that the operator *() for STL iterators be made
    into a const operation.

    The function
       void fn (const ReverseIterator & x) {
        ...
           y = x*;
        ...
       }
     shows that the operation * is not defined as const in the
     reverse_iterator (DRAFT 20 Sept 1994, 24.2.1.2).  However, the
     body of the function does not modify the iterator object.

     Of course, const Iterator is different from const_iterator and from
     const const_iterator.

 Proposed Resolution:
    Both base() and operator*() should be const.
    Accepted in Monterey - N740
 Requestor:      Bob Fraley <fraley@porter.hpl.hp.com>
 Owner:          David Dodgson (Iterators)
 Emails: c++std-lib-3135
 Papers:  N740 - Small Changes


---------------------------------------------------------------------------
 Work Group:     Library Clause 24
 Issue Number:   24-006
 Title:          Relaxing Requirement on Iterator++ Result
 Section:        24.4.3
 Status:         resolved
 Description:
         24.4.3 p24-23
    The return type of operator++ for istreambuf_iterator is listed
    as 'proxy'.  This suggestion is to make the return type an object
    which is "convertible to const X&" rather than "X&".
 Resolution:     accepted in Austin
 Requestor:      Nathan Myers
 Owner:          David Dodgson (Iterators)
 Emails:
 Papers: 95-0021/N0621 (Pre-Austin mailing)


---------------------------------------------------------------------------
 Work Group:     Library Clause 24
 Issue Number:   24-007
 Title:          Fixing istreambuf_iterator
 Section:        24.4.3
```

```
Status:         resolved
Description:
        24.4.3 p24-23:
   Proposes the addition to istreambuf_iterator of
      inline istreambuf::proxy::operator istreambuf_iterator()
             { return sbuf_; }
   to better conform to the Forward Iterator specification.
Resolution:     accepted in Austin
Requestor:      Nathan Myers
Owner:          David Dodgson (Iterators)
Emails:
Papers: 95-0022/N0622 (Pre-Austin mailing)


------------------------------------------------------------------------
Work Group:     Library Clause 24
Issue Number:   24-008
Title:          Iterator Requirements
Section:        24.1.3 and 24.1.4
Status:         resolved
Description:
        24.1.3 Table 59 and 24.1.4 Table 60
  The requirement r == s and r is dereferenceable implies ++r == ++r
  should read ++r == ++s in table 59.  Similarly in table 60,
  --r == --r implies r == s should read --r == --s.
Resolution:
   Table 59 for forward iterators was updated.
   Table 60 for bidirectional iterators is not updated.
      It should read: --r == --s implies r == s.
Requestor:      Nathan Myers
Owner:          David Dodgson (Iterators)
Emails: c++std-lib-3543
Papers: N740 - Small Changes


------------------------------------------------------------------------
Work Group:     Library Clause 24
Issue Number:   24-010
Title:          Operator-> in Iterators
Section:        24.
Status:         resolved
Description:
        Throughout clause 24:

The suggestion is for inclusion of operator-> in iterators.

Sean Corfield asks in c++std-lib-3596:

Each iterator has operator*() defined to return T& (or const T& as
appropriate).  Builtin pointer types also have this.

However,builtin pointer types also have operator->() when the underlying
type is a struct/class/union. Is there any reason why iterators don't
have T* operator->() defined? Did we ever decide to delay checking of
the return type of -> to the point of use? I remember we discussed it...
Without this, we have the slightly unpalatable:

  StructThing* p1 = &v1[0];
  StructThing* e1 = &v1[SIZE];
  while (p1 != e1) { process(p1->member); ++p1; }

  vector<StructThing>::iterator p2 = v2.begin();
  vector<StructThing>::iterator e2 = v2.end();
```

```
  while (p2 != e2) { process((*p2).member); ++p2; } // ugh!

Bob Fraley and Richard Minner offer agreement,
stating that it is an obvious need and would be extremely confusing
otherwise.

Nathan Myers and Jerry Schwarz dissent,
stating that there are objects for which -> may be meaningless
and that the current interface for iterators is minimal.

John Max Skaller in message c++std-lib-3602 points out that

So I think the question is whether the Standard Library
iterators should, or should not, mandate operator->().
This is not the same question as whether STL should require
operator->().

John Bruns and Fergus Henderson argue in favor of adding operator ->.

Alex Stepanov (and others) argues that operator-> should be provided
for all iterators or none.  Anything else would be too confusing.
Note that this would apply only to iterators over class type.

Unresolved questions:
   Given an output iterator o what are the semantics of o->member?
   Since insert iterators and ostream_iterator derive from output
   iterator, should they define operator->?

   These questions are discussed in lib-3817 and 3818.

 Proposed Resolution:
  A.
    Add the following row in Table 59-Forward iterator requirements in
    lib.forward.iterators [24.1.3] after the row describing *a:
       Expression:  a->m
       Semantics:   (a->m == (*a).m)
       Conditions:  pre: (*a) refers to a class object and m is a
                         member of that class
  B.
    Update the predefined iterators to include operator->. Specifically:
       lib.reverse.bidir.iter [24.3.1.1]
       include operator-> after lib.reverse.bidir.iter.op.star
          [24.3.1.2.3]
       lib.reverse.iterator [24.3.1.3]
       after lib.reverse.iter.op.star [24.3.1.4.3]
 Resolution:  Accepted in Monterey - N738
 Requestor:      Sean Corfield
 Owner:          David Dodgson (Iterators)
 Emails: lib 3596-3603,lib 3607-3620,3624,3636-3629,3817-3818
 Papers: Iterators and operator->(), 95-0119/N0719, Sean Corfield
         operator-> for iterators, 95-0138/N0738, David Dodgson


--------------------------------------------------------------------------
 Work Group:     Library Clause 24
 Issue Number:   24-011
 Title:          Small Issues in Austin
 Section:        24.
 Status:         resolved
 Description:
         Throughout clause 24
    Numerous small issues as specified in N0614/95-0014 in pre-Austin
    mailing.
```

```
Resolution:     Accepted in Austin
   Sections 2.4.6 and 2.4.13 of N0614 regarding the inclusion of friend
   declarations are not included in the April 95 WP (intentional?)

   Sections 2.4.9 and 2.4.10 of N0614 regarding the return type of
   operator++(int) being a reference are not included in the April
   95 WP (intentional?)

Requestor:      Larry Podmolik
Owner:          David Dodgson (Iterators)
Emails: none
Papers: N0614/95-0014 in pre-Austin mailing


------------------------------------------------------------------------
Work Group:     Library Clause 24
Issue Number:   24-013
Title:          Const declaration of operator[]
Section:        24.3.1.3 [lib.reverse.iterator]
Status:         resolved
Description:
        24..3.1.3 p24-15.16: [Box 117]
   Should operator[] of reverse_iterator be specified as const?
Proposed Resolution:
   Same resolution as issue 3 (Box 116 in lib.reverse.bidir.iter
   section 24.3.1.1 for reverse_bidirectional_iterator)
Resolution:  specified as const - See N740
Requestor:      Editorial box
Owner:          David Dodgson (Iterators)
Emails:
Papers:  Small Changes, 95-0140/N0740, David Dodgson, post-Monterey


------------------------------------------------------------------------
Work Group:     Library Clause 24
Issue Number:   24-014
Title:          Typo
Section:        24.4.3 [lib.istreambuf.iterator]
Status:         resolved
Description:
        24.4.3 p24-23
   The closing braces for class istreambuf_iterator are in italic
   bold.  They should be in normal font.
Resolution:     Use normal font
Requestor:      David Dodgson
Owner:          David Dodgson (Iterators)
Emails:
Papers:  Small Changes, 95-0140/N0740, David Dodgson, post-Monterey

------------------------------------------------------------------------
Work Group:     Library Clause 24
Issue Number:   24-016
Title:          Typo
Section:        24.2 [lib.iterator.primitives]
Status:         resolved
Description:
        24.2 p24-11:
   The word definable is spelled as 'def inable'
Resolution:
Requestor:      David Dodgson
Owner:          David Dodgson (Iterators)
Emails:
Papers:  Small Changes, 95-0140/N0740, David Dodgson, post-Monterey
```

```
---------------------------------------------------------------------------
 Work Group:     Library Clause 24
 Issue Number:   24-019
 Title:          Comments from German WG member Carsten Bormann
 Section:        24.
 Status:         resolved
 Description:

54.  24.1.4-Table 60

Explain ``--r == --r implies r == s''.

[Note: this is included as issue 24-008]

55.  24.1.6-2

``can be defined'', i.e., it is the user's responsibility?  Explain that
this is part of <iterator>.

56.  24.1.6-5

``may define''?  Repeat language from 20, ``for all memory models, ...''

57.  24.1.6-11

Header <iterator>: Drawing iostream into an implementation that just
needs iterators is most unfortunate.  The contents of the header <itera-
tor> should be confined to those operations that do not need iostream;
the rest should be put into a separate header.

58.  24.4

This should be part of the iostream library clause.  In this context, it
should be decided whether this subclause needs to be templatized
together with the rest of iostream.

59.  24.3.1.2.5

Returns: *this

60.  24.3.1.2.6

Returns: x.base() == y.base(); (There is no conversion from a reverse
iterator to its base.)

61.  24.3.1.3-1

The note seems misplaced, but does also apply here analogously.

62.  24.3.1.4.5

Returns: *this

63.  24.4.3.5

Change ``iterator over'' to ``iterate over''.

 Resolution:
    Comments 54,59,61-63 accepted
    Comment 60 accepted as "x.current == y.current;"
    Comments 57-58 opened as issue 21
```

```
   Items accepted in N0740
Requestor:      Roland Hartinger
Owner:          David Dodgson (Iterators)
Emails: lib-3829
Papers:  Small Changes, 95-0140/N0740, David Dodgson, post-Monterey


--------------------------------------------------------------------------
Work Group:     Library Clause 24
Issue Number:   24-020
Title:          Clause 24.3.1 Effects and Returns
Section:        24.3.1.2 & 24.3.1.4
Status:         resolved
Description:
        24.3.1.2 & 24.3.1.4:
        Review the Effects and Returns clauses for correctness
Resolution:     Updates in N0740
Requestor:      Library WG
Owner:          David Dodgson (Iterators)
Emails:
Papers:  Small Changes, 95-0140/N0740, David Dodgson, post-Monterey
```