Document No.:   J16/97-0060
                  WG21/N1098
Date: 17 July 1997
Project: Programming Language C++
Reply-To: Steve Rumsby
                  steve@maths.warwick.ac.uk

Library WP changes for London
=============================

```
diff -rc copyof~1/lib-algorithms librar~1/lib-algorithms
*** copyof~1/lib-algorithms   Tue Jul 15 17:24:18 1997
--- librar~1/lib-algorithms   Wed Jul 16 20:10:06 1997
***************
*** 555,561 ****
  .CW "if (\f6pred\fP(*\f6first\fP)){...}" .
  The function object
  .CW pred
! is assumed not to apply any non-constant
  function through the dereferenced iterator.
  This function object may be a pointer to function,
  or an object of a type with an appropriate function call operator.
--- 555,564 ----
  .CW "if (\f6pred\fP(*\f6first\fP)){...}" .
  The function object
  .CW pred
! .\" 25 [lib.algorithms]
! .\" CD2 comment UK 687
! .\" Steve Rumsby
! shall not to apply any non-constant
  function through the dereferenced iterator.
  This function object may be a pointer to function,
  or an object of a type with an appropriate function call operator.
***************
*** 575,588 ****
  as its argument and \f6first1\fP and \f6first2\fP as
  its iterator arguments, it should work correctly in
  the construct
! .CW "if (\f6pred\fP(*first, *\f6first2\fP)){...}" .
  .CW BinaryPredicate
  always takes the first iterator
  type as its first argument, that is, in those cases when
  .CW "T \f6value\fP"
  is part of the signature, it should work
  correctly in the context of
! .CW "if (\f6pred\fP(*first, \f6value\fP)){...}" .
  \f6binary_pred\fP shall not
  apply any non-constant function through the dereferenced iterators.
  .P
--- 578,594 ----
  as its argument and \f6first1\fP and \f6first2\fP as
  its iterator arguments, it should work correctly in
  the construct
! .\" 25 [lib.algorithms]
! .\" CD2 comment UK 688
! .\" Steve Rumsby
! .CW "if (\f6binary_pred\fP(*first1, *\f6first2\fP)){...}" .
  .CW BinaryPredicate
  always takes the first iterator
  type as its first argument, that is, in those cases when
  .CW "T \f6value\fP"
  is part of the signature, it should work
  correctly in the context of
! .CW "if (\f6binary_pred\fP(*first1, \f6value\fP)){...}" .
```

```
  \f6binary_pred\fP shall not
  apply any non-constant function through the dereferenced iterators.
  .P
***************
*** 594,600 ****
  they do not have to be defined.
  In these cases the semantics of
  .CW a+n
! is the same is that of
  .Cb
    { X tmp = a;
      advance(tmp, n);
--- 600,609 ----
  they do not have to be defined.
  In these cases the semantics of
  .CW a+n
! .\" 25 [lib.algorithms]
! .\" CD2 comment UK 689
! .\" Steve Rumsby
! is the same as that of
  .Cb
    { X tmp = a;
      advance(tmp, n);
***************
*** 605,614 ****
  .CW a-b
  is the same as of
  .Cb
!   { Distance n;
!     distance(a, b, n);
!     return n;
!   }
  .Ce
  .\"----------------------------------------------------------------------
----------
  .H2 "Non-modifying sequence operations" lib.alg.nonmodifying
--- 614,627 ----
  .CW a-b
  is the same as of
  .Cb
! .\" 25 [lib.algorithms]
! .\" CD2 comment UK 702
! .\" Steve Rumsby
! .\"  { Distance n;
! .\"    distance(a, b, n);
! .\"    return n;
! .\"  }
!   return distance(a, b)
  .Ce
  .\"----------------------------------------------------------------------
----------
  .H2 "Non-modifying sequence operations" lib.alg.nonmodifying
***************
*** 1491,1505 ****
  such that for each non-negative integer
  .CW "i < (\f6last\fP - \f6first\fP)"
  the following assignment takes place:
! .Cb
!     *(\f6first\fP + i) =  *(\f6result\fP + (i + (\f6middle\fP -
\f6first\fP)) % (\f6last\fP - \f6first\fP))
! .Ce
! .eN
! Should this be:
  .Cb
```

```
      *(\f4result\fP + i) =  *(\f4first\fP + (i + (\f6middle\fP -
\f6first\fP)) % (\f6last\fP - \f6first\fP))
  .Ce
! .nE
  .La Returns:
  .CW "result + (\f6last\fP - \f6first\fP)" .
  .La Requires
--- 1504,1521 ----
  such that for each non-negative integer
  .CW "i < (\f6last\fP - \f6first\fP)"
  the following assignment takes place:
! .\" [lib.alg.rotate] 25.2.10
! .\" Library issue 25-001
! .\" Steve Rumsby
! .\".Cb
! .\" *(\f6first\fP + i) =  *(\f6result\fP + (i + (\f6middle\fP -
\f6first\fP)) % (\f6last\fP - \f6first\fP))
! .\" .Ce
! .\" .eN
! .\" Should this be:
  .Cb
      *(\f4result\fP + i) =  *(\f4first\fP + (i + (\f6middle\fP -
\f6first\fP)) % (\f6last\fP - \f6first\fP))
  .Ce
! .\".nE
  .La Returns:
  .CW "result + (\f6last\fP - \f6first\fP)" .
  .La Requires
***************
*** 1542,1557 ****
  (where \f6n\fP is a positive argument of type
  .CW iterator_traits<RandomAccessIterator>::difference_type )
  returns a randomly chosen value
! of type
  .CW iterator_traits<RandomAccessIterator>::difference_type )
  in the interval
  .CW "[0, \f6n\fP)" .
! .eN
! Can it accept an argument that yields a result of a type
! that, although different from
! .CW RandomAccessIterator::difference_type ,
! can be converted to it?
! .nE
  .\"===
  .H3 "Partitions" lib.alg.partitions
  .\"----
--- 1558,1576 ----
  (where \f6n\fP is a positive argument of type
  .CW iterator_traits<RandomAccessIterator>::difference_type )
  returns a randomly chosen value
! .\" [lib.alg.random.shuffle] 25.2.11
! .\" Library issue 25-002
! .\" Steve Rumsby
! of a type convertible to
  .CW iterator_traits<RandomAccessIterator>::difference_type )
  in the interval
  .CW "[0, \f6n\fP)" .
! .\" .eN
! .\" Can it accept an argument that yields a result of a type
! .\" that, although different from
! .\" .CW RandomAccessIterator::difference_type ,
! .\" can be converted to it?
! .\" .nE
  .\"===
  .H3 "Partitions" lib.alg.partitions
  .\"----
***************
*** 1899,1905 ****
  The furthermost iterator
  .CW i
  in the range
! .CW "[\f6first\fP, \f6last\fP)"
  such that for any iterator
  .CW j
  in the range
--- 1918,1927 ----
  The furthermost iterator
  .CW i
  in the range
! .\" 25.3.3.1  lower_bound [lib.lower.bound]
! .\" CD2 comment Germany _25331/
! .\" Steve Rumsby
! .CW "[\f6first\fP, \f6last\fP]"
  such that for any iterator
  .CW j
  in the range
***************
*** 1908,1919 ****
  .CW "*j < \f6value\fP"
  or
  .CW "\f6comp\fP(*j, \f6value\fP) != false"
! .eN
! Should the range of
! .CW i
! be changed to:
! .CW "[\f6first\fP, \f6last\fP\f3]\fP" ?
! .nE
  .La Complexity:
  At most
  .CW "log(\f6last\fP - \f6first\fP) + 1"
--- 1930,1944 ----
  .CW "*j < \f6value\fP"
  or
  .CW "\f6comp\fP(*j, \f6value\fP) != false"
! .\" 25.3.3.1  lower_bound [lib.lower.bound]
! .\" CD2 comment Germany _25331/
! .\" Steve Rumsby
! .\" .eN
! .\" Should the range of
! .\" .CW i
! .\" be changed to:
! .\" .CW "[\f6first\fP, \f6last\fP\f3]\fP" ?
! .\" .nE
  .La Complexity:
  At most
  .CW "log(\f6last\fP - \f6first\fP) + 1"
diff -rc copyof~1/lib-containers librar~1/lib-containers
*** copyof~1/lib-containers   Tue Jul 15 17:24:22 1997
--- librar~1/lib-containers   Wed Jul 16 22:33:14 1997
***************
*** 113,118 ****
--- 113,124 ----
  X::iterator     iterator type pointing to \&\f5T\fP\&     T{
  any iterator category
  except output iterator.
+ .\" 23.1  Container requirements [lib.container.requirements]
+ .\" CD2 comment 23-014
+ .\" Steve Rumsby
+ .br
+ convertible to
+ .CW X::const_iterator .
  T}  compile time
  _
  X::const_iterator     T{
***************
*** 148,154 ****
  (&a)->~X();     \&\f5void\fP\&    T{
  note: the destructor is applied
  to every element of \&\f5a\fP\&;
! all the memory is returned.
  T}  linear
  _
  a.begin();      \&\f5iterator\fP\&;           constant
--- 154,163 ----
  (&a)->~X();     \&\f5void\fP\&    T{
  note: the destructor is applied
  to every element of \&\f5a\fP\&;
! .\" 23.1 Container requirements [lib.container.requirements]
! .\" CD2 comment UK 654
! .\" Steve Rumsby
! all the memory is deallocated.
  T}  linear
  _
  a.begin();      \&\f5iterator\fP\&;           constant
***************
*** 369,375 ****
  _
  a.insert(p,n,t) \&\f5void\fP\&    inserts n copies of \&\f5t\fP\& before
\&\f5p\fP\&.
  _
! a.insert(p,i,j) \&\f5void\fP\&    inserts copies of elements in
\&\f5[i,j)\fP\& before \&\f5p\fP\&.
  _
  a.erase(q)      \&\f5iterator\fP\&      erases the element pointed to by
\&\f5q\fP\&.
  _
--- 378,391 ----
  _
  a.insert(p,n,t) \&\f5void\fP\&    inserts n copies of \&\f5t\fP\& before
\&\f5p\fP\&.
  _
! .\" 23.1.1 [lib.sequence.reqmts]
! .\" CD2 comment 23-009
! .\" Steve Rumsby
! a.insert(p,i,j) \&\f5void\fP\&    T{
! pre: \f5i\fP,\f5j\fP are not iterators into \f5a\fP.
! .br
! inserts copies of elements in \&\f5[i,j)\fP\& before \&\f5p\fP\&.
! T}
  _
  a.erase(q)      \&\f5iterator\fP\&      erases the element pointed to by
\&\f5q\fP\&.
  _
***************
*** 710,715 ****
--- 726,736 ----
  T}
  _
  a.insert(i,j)   \&\f5void\fP\&    T{
+ .\" 23.1.3 [lib.associative.reqmts]
+ .\" CD2 comment 23-009
+ .\" Steve Rumsby
+ pre: \f5i\fP,\f5j\fP are not iterators into \f5a\fP.
+ .br
  inserts the elements from the
  range \&\f5[i,\fP\& \&\f5j)\fP\& into the container.
  T}  T{
***************
*** 1050,1057 ****
  .\" 94-0171/N0558
      template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
!     template <class Size, class T>
!       void assign(Size n, const T& t = T());
      allocator_type get_allocator() const;
  .Ce
  .Cb
--- 1071,1080 ----
  .\" 94-0171/N0558
      template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
! .\" CD2 comment Sweden _23/b
! .\" CD2 comment Sweden _23/c
! .\" Steve Rumsby
!     void assign(size_type n, const T& t);
      allocator_type get_allocator() const;
  .Ce
  .Cb
***************
*** 1091,1097 ****
      void push_back(const T& x);
  .Ce
  .Cb
!     iterator insert(iterator position, const T& x = T());
      void     insert(iterator position, size_type n, const T& x);
      template <class InputIterator>
        void insert (iterator position, InputIterator first, InputIterator
last);
--- 1114,1122 ----
      void push_back(const T& x);
  .Ce
  .Cb
! .\" CD2 comment Sweden _23/b
! .\" Steve Rumsby
!     iterator insert(iterator position, const T& x);
      void     insert(iterator position, size_type n, const T& x);
      template <class InputIterator>
        void insert (iterator position, InputIterator first, InputIterator
last);
***************
*** 1189,1195 ****
    insert(begin(), first, last);
  .Ce
  .Pb
! template <class Size, class T> void assign(Size n, const T& t = T());
  .Pe
  .La Effects:
  .Cb
--- 1214,1223 ----
    insert(begin(), first, last);
  .Ce
  .Pb
! .\" CD2 comment Sweden _23/b
! .\" CD2 comment Sweden _23/c
! .\" Steve Rumsby
! void assign(size_type n, const T& t);
```

```
  .Pe
  .La Effects:
  .Cb
***************
*** 1217,1223 ****
  .\"
  .ix "[deque] [insert]"
  .Pb
! iterator insert(iterator position, const T& x = T());
  void     insert(iterator position, size_type n, const T& x);
  template <class InputIterator>
    void insert(iterator position,
--- 1245,1253 ----
  .\"
  .ix "[deque] [insert]"
  .Pb
! .\" CD2 comment Sweden _23/b
! .\" Steve Rumsby
! iterator insert(iterator position, const T& x);
  void     insert(iterator position, size_type n, const T& x);
  template <class InputIterator>
    void insert(iterator position,
***************
*** 1324,1331 ****
  .\" 94-0171/N0558
      template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
!     template <class Size, class T>
!       void assign(Size n, const T& t = T());
      allocator_type get_allocator() const;
  .Ce
  .Cb
--- 1354,1363 ----
  .\" 94-0171/N0558
      template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
! .\" CD2 comment Sweden _23/b
! .\" CD2 comment Sweden _23/c
! .\" Steve Rumsby
!     void assign(size_type n, const T& t);
      allocator_type get_allocator() const;
  .Ce
  .Cb
***************
*** 1362,1368 ****
      void pop_back();
  .Ce
  .Cb
!     iterator insert(iterator \&\fP\f6position\&\fP\f5, const T&
\&\fP\f6x\&\fP\f5 = T());
      void     insert(iterator \&\fP\f6position\&\fP\f5, size_type
\&\fP\f6n\&\fP\f5, const T& \&\fP\f6x\&\fP\f5);
      template <class InputIterator>
        void insert(iterator \&\fP\f6position\&\fP\f5, InputIterator
\&\fP\f6first\&\fP\f5,
--- 1394,1402 ----
      void pop_back();
  .Ce
  .Cb
! .\" CD2 comment Sweden _23/b
! .\" Steve Rumsby
!     iterator insert(iterator \&\fP\f6position\&\fP\f5, const T&
\&\fP\f6x\&\fP\f5);
      void     insert(iterator \&\fP\f6position\&\fP\f5, size_type
\&\fP\f6n\&\fP\f5, const T& \&\fP\f6x\&\fP\f5);
      template <class InputIterator>
        void insert(iterator \&\fP\f6position\&\fP\f5, InputIterator
\&\fP\f6first\&\fP\f5,
***************
*** 1472,1478 ****
    insert(begin(), first, last);
  .Ce
  .Pb
! template <class Size, class T> void assign(Size n, const T& t = T());
  .Pe
  .La Effects:
  .Cb
--- 1506,1515 ----
    insert(begin(), first, last);
  .Ce
  .Pb
! .\" CD2 comment Sweden _23/b
! .\" CD2 comment Sweden _23/c
! .\" Steve Rumsby
! void assign(size_type n, const T& t);
  .Pe
  .La Effects:
  .Cb
***************
*** 1498,1504 ****
  .H4 "\&\f7list\fP\& modifiers" lib.list.modifiers
  .ix "[list] [insert]"
  .Pb
! iterator insert(iterator \&\fP\f6position\&\fP\f5, const T&
\&\fP\f6x\&\fP\f5 = T());
  void     insert(iterator \&\fP\f6position\&\fP\f5, size_type
\&\fP\f6n\&\fP\f5, const T& \&\fP\f6x\&\fP\f5);
  template <class InputIterator>
    void insert(iterator \&\fP\f6position\&\fP\f5, InputIterator
\&\fP\f6first\&\fP\f5,
--- 1535,1543 ----
  .H4 "\&\f7list\fP\& modifiers" lib.list.modifiers
  .ix "[list] [insert]"
  .Pb
! .\" CD2 comment Sweden _23/b
! .\" Steve Rumsby
! iterator insert(iterator \&\fP\f6position\&\fP\f5, const T&
\&\fP\f6x\&\fP\f5);
  void     insert(iterator \&\fP\f6position\&\fP\f5, size_type
\&\fP\f6n\&\fP\f5, const T& \&\fP\f6x\&\fP\f5);
  template <class InputIterator>
    void insert(iterator \&\fP\f6position\&\fP\f5, InputIterator
\&\fP\f6first\&\fP\f5,
***************
*** 1552,1557 ****
--- 1591,1602 ----
  and
  .CW x
  becomes empty.
+ .\" 23.2.2.4 [lib.list.ops]
+ .\" CD2 comment Sweden_23224
+ .\" Steve Rumsby
+ .P
+ Invalidates all iterators and references to the list
+ .CW x .
  .La Complexity:
  Constant time.
  .Pb
```

```
***************
*** 1568,1573 ****
--- 1613,1623 ----
  .CW "position == i"
  or
  .CW "position == ++i" .
+ .\" 23.2.2.4 [lib.list.ops]
+ .\" CD2 comment Sweden_23224
+ .\" Steve Rumsby
+ .P
+ Invalidates only the iterators and references to the spliced element.
  .La Requires:
  .CW i
  is a valid dereferenceable iterator of
***************
*** 1593,1598 ****
--- 1643,1653 ----
  .CW position
  is an iterator in the range
  .CW "[first, last)" .
+ .\" 23.2.2.4 [lib.list.ops]
+ .\" CD2 comment Sweden_23224
+ .\" Steve Rumsby
+ .P
+ Invalidates only the iterators and references to the spliced elements.
  .La Complexity:
  Constant time if
  .CW "&\f6x\fP == this" ;
***************
*** 2025,2031 ****
  .\" 94-0171/N0558
      template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
!     template <class Size, class U> void assign(Size n, const U& u = U());
      allocator_type get_allocator() const;
  .Ce
  .Cb
--- 2080,2089 ----
  .\" 94-0171/N0558
      template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
! .\" CD2 comment Sweden _23/b
! .\" CD2 comment Sweden _23/c
! .\" Steve Rumsby
!     void assign(size_type n, const T& u);
      allocator_type get_allocator() const;
  .Ce
  .Cb
***************
*** 2065,2071 ****
    \&\f6// _lib.vector.modifiers_ modifiers:\fP\&
      void push_back(const T& x);
      void pop_back();
!     iterator insert(iterator position, const T& x = T());
      void     insert(iterator position, size_type n, const T& x);
      template <class InputIterator>
        void insert(iterator position, InputIterator first, InputIterator
last);
--- 2123,2131 ----
    \&\f6// _lib.vector.modifiers_ modifiers:\fP\&
      void push_back(const T& x);
      void pop_back();
! .\" CD2 comment Sweden _23/b
! .\" Steve Rumsby
!     iterator insert(iterator position, const T& x);
      void     insert(iterator position, size_type n, const T& x);
      template <class InputIterator>
          void insert(iterator position, InputIterator first, InputIterator
last);
***************
*** 2150,2156 ****
    insert(begin(), first, last);
  .Ce
  .Pb
! template <class Size, class U> void assign(Size n, const U& u = U());
  .Pe
  .La Effects:
  .Cb
--- 2210,2219 ----
    insert(begin(), first, last);
  .Ce
  .Pb
! .\" CD2 comment Sweden _23/b
! .\" CD2 comment Sweden _23/c
! .\" Steve Rumsby
! void assign(Size n, const T& t);
  .Pe
  .La Effects:
  .Cb
***************
*** 2217,2223 ****
  .\"
  .ix "[vector] [insert]"
  .Pb
! iterator insert(iterator position, const T& x = T());
  void     insert(iterator position, size_type n, const T& x);
  template <class InputIterator>
    void insert(iterator position, InputIterator first, InputIterator
last);
--- 2280,2288 ----
  .\"
  .ix "[vector] [insert]"
  .Pb
! .\" CD2 comment Sweden _23/b
! .\" Steve Rumsby
! iterator insert(iterator position, const T& x);
  void     insert(iterator position, size_type n, const T& x);
  template <class InputIterator>
    void insert(iterator position, InputIterator first, InputIterator
last);
***************
*** 2316,2322 ****
  .\" 94-0171/N0558
      template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
!     template <class Size, class U> void assign(Size n, const U& u = U());
      allocator_type get_allocator() const;
  .Ce
  .Cb
--- 2381,2390 ----
  .\" 94-0171/N0558
      template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
! .\" CD2 comment Sweden _23/b
! .\" CD2 comment Sweden _23/c
! .\" Steve Rumsby
!     void assign(Size n, const T& t);
      allocator_type get_allocator() const;
  .Ce
  .Cb
***************
*** 2356,2362 ****
    \&\f6// modifiers:\fP\&
      void push_back(const bool& x);
      void pop_back();
!     iterator insert(iterator position, const bool& x = bool());
      void     insert (iterator position, size_type n, const bool& x);
      template <class InputIterator>
          void insert (iterator position, InputIterator first,
InputIterator last);
--- 2424,2432 ----
    \&\f6// modifiers:\fP\&
      void push_back(const bool& x);
      void pop_back();
! .\" CD2 comment Sweden _23/b
! .\" Steve Rumsby
!     iterator insert(iterator position, const bool& x);
      void     insert (iterator position, size_type n, const bool& x);
      template <class InputIterator>
          void insert (iterator position, InputIterator first,
InputIterator last);
***************
*** 3214,3220 ****
    set(InputIterator \f6first\fP, \f6last\fP,
        const COmpare& \f6comp\fP = Compare(), const Allocator& =
Allocator());
  .Pe
! .La Effects: COnstructs an empty
  .CW set
  using the specified comparison object and allocator,
  and inserts elements from the range
--- 3284,3292 ----
    set(InputIterator \f6first\fP, \f6last\fP,
        const COmpare& \f6comp\fP = Compare(), const Allocator& =
Allocator());
  .Pe
! .\" Editorial
! .\" Steve Rumsby
! .La Effects: Constructs an empty
  .CW set
  using the specified comparison object and allocator,
  and inserts elements from the range
diff -rc copyof~1/lib-diagnostics librar~1/lib-diagnostics
*** copyof~1/lib-diagnostics  Tue Jul 15 17:24:24 1997
--- librar~1/lib-diagnostics  Tue Jul 15 22:17:08 1997
***************
*** 75,81 ****
  namespace std {
    class logic_error : public exception {
    public:
!     logic_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
--- 75,84 ----
  namespace std {
    class logic_error : public exception {
    public:
! .\" 19.1.1  Class logic_error [lib.logic.error]
! .\" CD2 comment Germany _191
! .\" Steve Rumsby
!     explicit logic_error(const string& \&\fP\f6what_arg\&\fP\f5);
```

```
    };
  }
  .Ce
***************
*** 103,109 ****
  namespace std {
    class domain_error : public logic_error {
    public:
!     domain_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
--- 106,115 ----
  namespace std {
    class domain_error : public logic_error {
    public:
! .\" 19.1.2  Class domain_error [lib.domain.error]
! .\" CD2 comment Germany _191
! .\" Steve Rumsby
!     explicit domain_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
***************
*** 129,135 ****
  namespace std {
    class invalid_argument : public logic_error {
    public:
!     invalid_argument(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
--- 135,144 ----
  namespace std {
    class invalid_argument : public logic_error {
    public:
! .\" 19.1.3  Class invalid_argument [lib.invalid.argument]
! .\" CD2 comment Germany _191
! .\" Steve Rumsby
!     explicit invalid_argument(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
***************
*** 154,160 ****
  namespace std {
    class length_error : public logic_error {
    public:
!     length_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
--- 163,172 ----
  namespace std {
    class length_error : public logic_error {
    public:
! .\" 19.1.4  Class length_error [lib.length.error]
! .\" CD2 comment Germany _191
! .\" Steve Rumsby
!     explicit length_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
***************
```

```
*** 181,187 ****
  namespace std {
    class out_of_range : public logic_error {
    public:
!     out_of_range(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
--- 193,202 ----
  namespace std {
    class out_of_range : public logic_error {
    public:
! .\" 19.1.5  Class out_of_range [lib.out.of.range]
! .\" CD2 comment Germany _191
! .\" Steve Rumsby
!     explicit out_of_range(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
***************
*** 208,214 ****
  namespace std {
    class runtime_error : public exception {
    public:
!     runtime_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
--- 223,232 ----
  namespace std {
    class runtime_error : public exception {
    public:
! .\" 19.1.6  Class runtime_error [lib.runtime.error]
! .\" CD2 comment Germany _191
! .\" Steve Rumsby
!     explicit runtime_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
***************
*** 234,240 ****
  namespace std {
    class range_error : public runtime_error {
    public:
!     range_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
--- 252,261 ----
  namespace std {
    class range_error : public runtime_error {
    public:
! .\" 19.1.7  Class range_error [lib.range.error]
! .\" CD2 comment Germany _191
! .\" Steve Rumsby
!     explicit range_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
***************
*** 260,266 ****
  namespace std {
    class overflow_error : public runtime_error {
    public:
```

```
!     overflow_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
--- 281,290 ----
  namespace std {
    class overflow_error : public runtime_error {
    public:
! .\" 19.1.8  Class overflow_error [lib.overflow.error]
! .\" CD2 comment Germany _191
! .\" Steve Rumsby
!     explicit overflow_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
***************
*** 284,290 ****
  namespace std {
    class underflow_error : public runtime_error {
    public:
!     underflow_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
--- 308,317 ----
  namespace std {
    class underflow_error : public runtime_error {
    public:
! .\" 19.1.9  Class underflow_error [lib.underflow.error]
! .\" CD2 comment Germany _191
! .\" Steve Rumsby
!     explicit underflow_error(const string& \&\fP\f6what_arg\&\fP\f5);
    };
  }
  .Ce
diff -rc copyof~1/lib-intro librar~1/lib-intro
*** copyof~1/lib-intro  Tue Jul 15 17:24:26 1997
--- librar~1/lib-intro  Tue Jul 15 20:59:16 1997
***************
*** 1,12 ****
  .H1 "Library introduction" lib.library 17
  .P
  This clause describes the contents of the
! .I "\*C Standard library" ,
! .ix "\*C Standard library"
  how a well-formed \*C program makes use of the library, and
  how a conforming implementation may provide the entities in the library.
  .P
! The \*C Standard library provides an extensible framework, and contains
components for:
  language support,
  diagnostics, general utilities, strings, locales,
  containers, iterators, algorithms, numerics, and input/output.
--- 1,18 ----
  .H1 "Library introduction" lib.library 17
  .P
  This clause describes the contents of the
! .\" 17 Library Introduction [lib.library]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! .I "\*C Standard Library" ,
! .ix "\*C Standard Library"
  how a well-formed \*C program makes use of the library, and
  how a conforming implementation may provide the entities in the library.
  .P
! .\" 17 Library Introduction [lib.library]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! The \*C Standard Library provides an extensible framework, and contains
components for:
  language support,
  diagnostics, general utilities, strings, locales,
  containers, iterators, algorithms, numerics, and input/output.
***************
*** 137,153 ****
  .LI
  .B "reserved function:"
  .ix "reserved function"
! A function, specified as part of the \*C Standard library, that must be
defined by the
  implementation.
  If a \*C program provides a definition for any reserved function, the
results are undefined.
  .ix "undefined
  Clause _intro.defs_ defines additional terms used elsewhere in this
  International Standard.
  .\"-------------------------------------------------------------------
----------
  .H2 "Method of description (Informative)" lib.description
  .P
  Clause _lib.description_ describes the conventions used to describe
! the \*C Standard library.
  It describes the structures of the normative Clauses
_lib.language.support_ through
  _lib.input.output_ (_lib.structure_), and
  other editorial conventions (_lib.conventions_).
--- 143,169 ----
  .LI
  .B "reserved function:"
  .ix "reserved function"
! .\" 17.1 Definitions [lib.definitions]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! A function, specified as part of the \*C Standard Library, that must be
defined by the
  implementation.
  If a \*C program provides a definition for any reserved function, the
results are undefined.
  .ix "undefined
+ .\" 17.1 Definitions [lib.definitions]
+ .\" CD2 comment UK 648
+ .\" Steve Rumsby
+ .P
  Clause _intro.defs_ defines additional terms used elsewhere in this
  International Standard.
  .\"-------------------------------------------------------------------
----------
  .H2 "Method of description (Informative)" lib.description
  .P
  Clause _lib.description_ describes the conventions used to describe
! .\" 17.2 Method of description (Informative) [lib.description]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! the \*C Standard Library.
  It describes the structures of the normative Clauses
_lib.language.support_ through
  _lib.input.output_ (_lib.structure_), and
  other editorial conventions (_lib.conventions_).
```

```
***************
*** 351,357 ****
  .ix "convention"
  .P
  This subclause describes several editorial conventions used to describe
the contents
! of the \*C Standard library.
  These conventions are for describing
  implementation-defined types (_lib.type.descriptions_),
  and member functions (_lib.functions.within.classes_).
--- 367,376 ----
  .ix "convention"
  .P
  This subclause describes several editorial conventions used to describe
the contents
! .\" 17.2.2 Other conventions [lib.conventions]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! of the \*C Standard Library.
  These conventions are for describing
  implementation-defined types (_lib.type.descriptions_),
  and member functions (_lib.functions.within.classes_).
***************
*** 615,621 ****
  .CW "\f6T A\fP[\f6N\fP]" ,
  where \&\f6T\fP\& is type
  .CW wchar_t
! (_basic.fundmental_),
  .ix "[wchar__t]"
  optionally qualified by any combination of
  .CW const
--- 634,643 ----
  .CW "\f6T A\fP[\f6N\fP]" ,
  where \&\f6T\fP\& is type
  .CW wchar_t
! .\" 17.2.2.1.3.3 Wide-character sequences [lib.wide.characters]
! .\" CD2 comment UK Ed-684
! .\" Steve Rumsby
! (_basic.fundamental_),
  .ix "[wchar__t]"
  optionally qualified by any combination of
  .CW const
***************
*** 717,729 ****
  .\"-----------------------------------------------
  .H3 "Library contents and organization" lib.organization
  .P
! This subclause provides a summary of the entities defined in the \*C
Standard library.
  Subclause _lib.contents_ provides an alphabetical listing of entities by
type,
  while subclause _lib.headers_ provides an alphabetical listing of library
headers.
  .\"----
  .H4 "Library contents" lib.contents
  .P
! The \*C Standard library provides definitions for the following types of
entities:
  Macros, Values, Types, Templates, Classes, Functions, Objects.
  .P
  All library entities except macros,
--- 739,757 ----
  .\"-----------------------------------------------
  .H3 "Library contents and organization" lib.organization
  .P
! .\" 17.3.1 Library contents and organization [lib.organization]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! This subclause provides a summary of the entities defined in the \*C
Standard Library.
  Subclause _lib.contents_ provides an alphabetical listing of entities by
type,
  while subclause _lib.headers_ provides an alphabetical listing of library
headers.
  .\"----
  .H4 "Library contents" lib.contents
  .P
! .\" 17.3.1.1 Library contents [lib.contents]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! The \*C Standard Library provides definitions for the following types of
entities:
  Macros, Values, Types, Templates, Classes, Functions, Objects.
  .P
  All library entities except macros,
***************
*** 738,744 ****
  .\"----
  .H4 "Headers" lib.headers
  .P
! The elements of the \*C Standard library are declared or defined (as
appropriate) in a
  .I header .\*f
  .Fs
  A header is not necessarily a source file, nor are the sequences
delimited by
--- 766,775 ----
  .\"----
  .H4 "Headers" lib.headers
  .P
! .\" 17.3.1.2 Headers [lib.headers]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! The elements of the \*C Standard Library are declared or defined (as
appropriate) in a
  .I header .\*f
  .Fs
  A header is not necessarily a source file, nor are the sequences
delimited by
***************
*** 748,754 ****
  in header names necessarily valid source file names (_cpp.include_).
  .Fe
  .P
! The \*C Standard library provides 32
  .I "\*C headers" ,
  .ix "\*C headers"
  as shown in Table \n+(Tn:
--- 779,788 ----
  in header names necessarily valid source file names (_cpp.include_).
  .Fe
  .P
! .\" 17.3.1.2 Headers [lib.headers]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! The \*C Standard Library provides 32
  .I "\*C headers" ,
  .ix "\*C headers"
  as shown in Table \n+(Tn:
***************
*** 789,801 ****
  as specified in ISO/IEC 9899:1990 Programming Languages C (Clause 7),
  or ISO/IEC:1990 Programming Languages\(emC AMENDMENT 1: C Integrity,
  (Clause 7), as appropriate, as if by inclusion. In the \*C Standard
! library, however, the declarations and definitions (except for
  names which are defined as macros in C) are within namespace scope
  (_basic.scope.namespace_) of the namespace
  .CW std.
  .P
  Names which are defined as macros in C shall be defined as macros in
! the \*C Standard library, even if license is granted in C for
  implementation as functions.
  .N[
  the names defined as macros
--- 823,841 ----
  as specified in ISO/IEC 9899:1990 Programming Languages C (Clause 7),
  or ISO/IEC:1990 Programming Languages\(emC AMENDMENT 1: C Integrity,
  (Clause 7), as appropriate, as if by inclusion. In the \*C Standard
! .\" 17.3.1.2 Headers [lib.headers]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! Library, however, the declarations and definitions (except for
  names which are defined as macros in C) are within namespace scope
  (_basic.scope.namespace_) of the namespace
  .CW std.
  .P
  Names which are defined as macros in C shall be defined as macros in
! .\" 17.3.1.2 Headers [lib.headers]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! the \*C Standard Library, even if license is granted in C for
  implementation as functions.
  .N[
  the names defined as macros
***************
*** 811,817 ****
  .N]
  .P
  Names that are defined as functions in C shall be defined as
! functions in the \*C Standard library.\*f
  .Fs
  This disallows the practice, allowed in C, of providing a
  "masking macro" in addition to the function prototype.  The only
--- 851,860 ----
  .N]
  .P
  Names that are defined as functions in C shall be defined as
! .\" 17.3.1.2 Headers [lib.headers]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! functions in the \*C Standard Library.\*f
  .Fs
  This disallows the practice, allowed in C, of providing a
  "masking macro" in addition to the function prototype.  The only
***************
*** 841,847 ****
  .ix "hosted implementation"
  describes the set of available headers.
  .P
! A freestanding implementation has
  .ix "freestanding implementation
  has an implementation-defined set of headers.
  .ix "implementation-defined"
--- 884,893 ----
  .ix "hosted implementation"
  describes the set of available headers.
  .P
! .\" 17.3.1.3 Freestanding implementations [lib.compliance]
! .\" CD2 comment UK 649
! .\" Steve Rumsby
! A freestanding implementation
  .ix "freestanding implementation
  has an implementation-defined set of headers.
  .ix "implementation-defined"
***************
*** 886,898 ****
  .H3 "Using the library" lib.using
  .P
  This subclause describes how a \*C program gains access to the facilities
of
! the \*C Standard library.
  Subclause _lib.using.headers_ describes effects during translation phase
4,
  while subclause _lib.using.linkage_ describes effects during phase 8
(_lex.phases_).
  .\"----
  .H4 "Headers" lib.using.headers
  .P
! The entities in the \*C Standard library are defined in headers,
  whose contents are made available to a translation unit when it contains
the appropriate
  .ix "translation unit"
  .CW #include
--- 932,950 ----
  .H3 "Using the library" lib.using
  .P
  This subclause describes how a \*C program gains access to the facilities
of
! .\" 17.3.2 Using the library [lib.using]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! the \*C Standard Library.
  Subclause _lib.using.headers_ describes effects during translation phase
4,
  while subclause _lib.using.linkage_ describes effects during phase 8
(_lex.phases_).
  .\"----
  .H4 "Headers" lib.using.headers
  .P
! .\" 17.3.2.1 Headers [lib.using.headers]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! The entities in the \*C Standard Library are defined in headers,
  whose contents are made available to a translation unit when it contains
the appropriate
  .ix "translation unit"
  .CW #include
***************
*** 925,931 ****
  .\"----
  .H4 "Linkage" lib.using.linkage
  .P
! Entities in the \*C Standard library have external linkage
(_basic.link_).
  Unless otherwise specified, objects and functions have the default
  \f5extern "C++"\fP
  linkage (_dcl.link_).
--- 977,986 ----
  .\"----
  .H4 "Linkage" lib.using.linkage
  .P
! .\" 17.3.2.2 Linkage [lib.using.linkage]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! Entities in the \*C Standard Library have external linkage
(_basic.link_).
  Unless otherwise specified, objects and functions have the default
  \f5extern "C++"\fP
  linkage (_dcl.link_).
***************
*** 959,965 ****
  .H3 "Constraints on programs" lib.constraints
  .P
  This subclause describes restrictions on \*C programs that use the
facilities
! of the \*C Standard library.
  The following subclauses specify constraints on the program's
  namespace (_lib.reserved.names_),
  its use of headers (_lib.alt.headers_),
--- 1014,1023 ----
  .H3 "Constraints on programs" lib.constraints
  .P
  This subclause describes restrictions on \*C programs that use the
facilities
! .\" 17.3.3 Constraints on programs
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! of the \*C Standard Library.
  The following subclauses specify constraints on the program's
  namespace (_lib.reserved.names_),
  its use of headers (_lib.alt.headers_),
***************
*** 989,995 ****
  that meets the minimum requirements of the Standard.
  .Fe
  .P
! The \*C Standard library reserves the following kinds of names:
  .LI
  Macros
  .LI
--- 1047,1056 ----
  that meets the minimum requirements of the Standard.
  .Fe
  .P
! .\" 17.3.3.1 Reserved names [lib.reserved.names]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! The \*C Standard Library reserves the following kinds of names:
  .LI
  Macros
  .LI
***************
*** 1125,1131 ****
  .P
  If a file with a name
  .ix "implementation-defined"
! equivalent to the derived file name for one of the \*C Standard library
headers
  is not provided as part of the implementation, and a file with that name
  is placed in any of the standard places for a source file to be included
```

```
(_cpp.include_),
  the behavior is undefined.
--- 1186,1195 ----
  .P
  If a file with a name
  .ix "implementation-defined"
! .\" 17.3.3.2 Headers [lib.alt.headers]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! equivalent to the derived file name for one of the \*C Standard Library
headers
  is not provided as part of the implementation, and a file with that name
  is placed in any of the standard places for a source file to be included
(_cpp.include_),
  the behavior is undefined.
***************
*** 1146,1153 ****
  .ix "altermate definition"
  .P
  Clauses _lib.language.support_ through _lib.input.output_ describe the
behavior of numerous functions defined by
! the \*C Standard library.  Under some circumstances,
! .ix "\*C Standard library"
  however, certain of these function descriptions also apply to replacement
functions defined
  in the program (_lib.definitions_).
  .P
--- 1210,1220 ----
  .ix "altermate definition"
  .P
  Clauses _lib.language.support_ through _lib.input.output_ describe the
behavior of numerous functions defined by
! .\" 17.3.3.4 Replacement functions [lib.replacement.functions]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! the \*C Standard Library.  Under some circumstances,
! .ix "\*C Standard Library"
  however, certain of these function descriptions also apply to replacement
functions defined
  in the program (_lib.definitions_).
  .P
***************
*** 1184,1190 ****
  .\"----
  .H4 "Handler functions" lib.handler.functions
  .P
! The \*C Standard library provides default versions of the three handler
  functions (_lib.language.support_):
  .LI
  .CW new_handler
--- 1251,1260 ----
  .\"----
  .H4 "Handler functions" lib.handler.functions
  .P
! .\" 17.3.3.5 Handler functions [lib.handler.functions]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! The \*C Standard Library provides default versions of the three handler
  functions (_lib.language.support_):
  .LI
  .CW new_handler
***************
*** 1216,1222 ****
  .\" 95-0023/N0623, as amended in Austin:
```

```
  .P
  In certain cases (replacement functions, handler functions, operations on
types used to
! instantiate standard library template components), the \*C Standard
library depends on
  components supplied by a \*C program.
  If these components do not meet their requirements, the Standard places
no requirements
  on the implementation.
--- 1286,1295 ----
  .\" 95-0023/N0623, as amended in Austin:
  .P
  In certain cases (replacement functions, handler functions, operations on
types used to
! .\" 17.3.3.6 Other functions [lib.res.on.functions]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! instantiate standard library template components), the \*C Standard
Library depends on
  components supplied by a \*C program.
  If these components do not meet their requirements, the Standard places
no requirements
  on the implementation.
***************
*** 1250,1257 ****
  .P
  Each of the following statements applies to all arguments
  .ix "argument"
! to functions defined in the \*C Standard library,
! .ix "\*C Standard library"
  unless explicitly stated otherwise.
  .LI
  If an argument to a function has an invalid value (such
--- 1323,1333 ----
  .P
  Each of the following statements applies to all arguments
  .ix "argument"
! .\" 17.3.3.7 Functions arguments [lib.res.on.arguments]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! to functions defined in the \*C Standard Library,
! .ix "\*C Standard Library"
  unless explicitly stated otherwise.
  .LI
  If an argument to a function has an invalid value (such
***************
*** 1327,1333 ****
  .\"----
  .H4 "Global functions" lib.global.functions
  .P
! It is unspecified whether any global functions in the \*C Standard
library are defined as
  .CW inline
  (_dcl.fct.spec_).
  .P
--- 1403,1412 ----
  .\"----
  .H4 "Global functions" lib.global.functions
  .P
! .\" 17.3.4.3 Glocal functions [lib.global.functions]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! It is unspecified whether any global functions in the \*C Standard
Library are defined as
```

```
  .CW inline
  (_dcl.fct.spec_).
  .P
***************
*** 1347,1353 ****
  .\"----
  .H4 "Member functions" lib.member.functions
  .P
! It is unspecified whether any member functions in the \*C Standard
library are defined as
  .CW inline
  (_dcl.fct.spec_).
  .P
--- 1426,1435 ----
  .\"----
  .H4 "Member functions" lib.member.functions
  .P
! .\" 17.3.4.4 Member functions [lib.member.functions]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! It is unspecified whether any member functions in the \*C Standard
Library are defined as
  .CW inline
  (_dcl.fct.spec_).
  .P
***************
*** 1435,1442 ****
  classes (with names reserved to the implementation).
  .P
  Certain classes defined in the \*C Standard Library are derived from
other classes
! in the \*C Standard library:
! .ix "\*C Standard library"
  .LI
  It is unspecified whether a class described in the \*C Standard Library
as
  derived from another class is derived from that class directly, or
through other
--- 1517,1527 ----
  classes (with names reserved to the implementation).
  .P
  Certain classes defined in the \*C Standard Library are derived from
other classes
! .\" 17.3.4.7 Derived classes [lib.derivation]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! in the \*C Standard Library:
! .ix "\*C Standard Library"
  .LI
  It is unspecified whether a class described in the \*C Standard Library
as
  derived from another class is derived from that class directly, or
through other
***************
*** 1470,1477 ****
  .ix "restriction"
  .ix "exception handler"
  .P
! Any of the functions defined in the \*C Standard library
! .ix "\*C Standard library"
  can report a failure by throwing an exception of the type(s) described in
their
  .B Throws:
  paragraph and/or their
```

```
--- 1555,1565 ----
  .ix "restriction"
  .ix "exception handler"
  .P
! .\" 17.3.4.8 Restrictions on exception handling [lib.res.on.exceptions]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! Any of the functions defined in the \*C Standard Library
! .ix "\*C Standard Library"
  can report a failure by throwing an exception of the type(s) described in
their
  .B Throws:
  paragraph and/or their
***************
*** 1509,1516 ****
  (_lib.alg.c.library_) meet this condition.
  .Fe
  .P
! Any of the functions defined in the \*C Standard library
! .ix "\*C~Standard~library exception specifications"
  that do not have an
  .I exception-specification
  may throw implementation-defined exceptions.\*f
--- 1597,1607 ----
  (_lib.alg.c.library_) meet this condition.
  .Fe
  .P
! .\" 17.3.4.8 Restrictions on exception handling [lib.res.on.exceptions]
! .\" CD2 comment UK 682
! .\" Steve Rumsby
! Any of the functions defined in the \*C Standard Library
! .ix "\*C~Standard~Library exception specifications"
  that do not have an
  .I exception-specification
  may throw implementation-defined exceptions.\*f
diff -rc copyof~1/lib-iterators librar~1/lib-iterators
*** copyof~1/lib-iterators    Tue Jul 15 17:24:36 1997
--- librar~1/lib-iterators    Wed Jul 16 22:26:52 1997
***************
*** 168,174 ****
  .CW j
  is reachable from
  .CW i .
! The result of the application of the algorithms in the library to invalid
ranges is
  undefined.
  .P
  All the categories of iterators require only those functions that are
realizable for a given category in
--- 168,177 ----
  .CW j
  is reachable from
  .CW i .
! .\" [lib.iterator.requirements] 24.1 Iterator requirements
! .\" CD2 comment Germany 22
! .\" Steve Rumsby
! The result of the application of functions in the library to invalid
ranges is
  undefined.
  .P
  All the categories of iterators require only those functions that are
realizable for a given category in
***************
*** 314,320 ****
```

```
  .P
  A class or a built-in type
  .CW X
! satisfies the requirements of an output iterator if the following
expressions are
  valid, as shown in Table \n+(Tn:
  .Ts "Output iterator requirements"
  .na
--- 317,330 ----
  .P
  A class or a built-in type
  .CW X
! .\" 24.1.2  Output iterators [lib.output.iterators]
! .\" CD2 comment Germany _2412/
! .\" Steve Rumsby
! satisfies the requirements of an output iterator
! if
! .CW X
! is an Assignable type (_lib.container.requirements_) and also
! the following expressions are
  valid, as shown in Table \n+(Tn:
  .Ts "Output iterator requirements"
  .na
***************
*** 982,992 ****
      reverse_iterator  operator--(int);
  .Ce
  .Cb
!     reverse_iterator  operator+ (Distance n) const;
!     reverse_iterator& operator+=(Distance n);
!     reverse_iterator  operator- (Distance n) const;
!     reverse_iterator& operator-=(Distance n);
!     Reference operator[](Distance n) const;
    };
  }
  .Ce
--- 992,1005 ----
      reverse_iterator  operator--(int);
  .Ce
  .Cb
! .\" 24.4.1.1 [lib.reverse.iterator]
! .\" CD2 comment UK 642
! .\" Steve Rumsby
!     reverse_iterator  operator+ (difference_type n) const;
!     reverse_iterator& operator+=(difference_type n);
!     reverse_iterator  operator- (difference_type n) const;
!     reverse_iterator& operator-=(difference_type n);
!     reference operator[](difference_type n) const;
    };
  }
  .Ce
***************
*** 1347,1353 ****
      typedef Container container_type;
      explicit back_insert_iterator(Container& x);
      back_insert_iterator<Container>&
!       operator=(const typename Container::value_type& value);
  .Ce
  .Cb
      back_insert_iterator<Container>& operator*();
--- 1360,1368 ----
      typedef Container container_type;
      explicit back_insert_iterator(Container& x);
      back_insert_iterator<Container>&
```

```
! .\" CD2 comment Germany_244
! .\" Steve Rumsby
!       operator=(const typename Container::reference value);
  .Ce
  .Cb
      back_insert_iterator<Container>& operator*();
***************
*** 1377,1383 ****
  .ix "[back__insert__iterator] [operator=]"
  .Pb
  back_insert_iterator<Container>&
!   operator=(const typename Container::value_type& value);
  .Pe
  .La Effects:
  .CW container.push_back(value);
--- 1392,1400 ----
  .ix "[back__insert__iterator] [operator=]"
  .Pb
  back_insert_iterator<Container>&
! .\" CD2 comment Germany_244
! .\" Steve Rumsby
!   operator=(const typename Container::reference value);
  .Pe
  .La Effects:
  .CW container.push_back(value);
***************
*** 1424,1430 ****
      typedef Container container_type;
      explicit front_insert_iterator(Container& x);
      front_insert_iterator<Container>&
!       operator=(const typename Container::value_type& value);
  .Ce
  .Cb
      front_insert_iterator<Container>& operator*();
--- 1441,1449 ----
      typedef Container container_type;
      explicit front_insert_iterator(Container& x);
      front_insert_iterator<Container>&
! .\" CD2 comment Germany_244
! .\" Steve Rumsby
!       operator=(const typename Container::reference value);
  .Ce
  .Cb
      front_insert_iterator<Container>& operator*();
***************
*** 1452,1458 ****
  .ix "[front__insert__iterator] [operator=]"
  .Pb
  front_insert_iterator<Container>&
!   operator=(const typename Container::value_type& value);
  .Pe
  .La Effects:
  .CW container.push_front(value);
--- 1471,1479 ----
  .ix "[front__insert__iterator] [operator=]"
  .Pb
  front_insert_iterator<Container>&
! .\" CD2 comment Germany_244
! .\" Steve Rumsby
!   operator=(const typename Container::reference value);
  .Pe
  .La Effects:
  .CW container.push_front(value);
***************
```

```
*** 1500,1506 ****
      typedef Container container_type;
      insert_iterator(Container& x, typename Container::iterator i);
      insert_iterator<Container>&
!       operator=(const typename Container::value_type& value);
  .Ce
  .Cb
      insert_iterator<Container>& operator*();
--- 1521,1529 ----
      typedef Container container_type;
      insert_iterator(Container& x, typename Container::iterator i);
      insert_iterator<Container>&
! .\" CD2 comment Germany_244
! .\" Steve Rumsby
!       operator=(const typename Container::reference value);
  .Ce
  .Cb
      insert_iterator<Container>& operator*();
***************
*** 1532,1538 ****
  .ix "[insert__iterator] [operator=]"
  .Pb
  insert_iterator<Container>&
!   operator=(const typename Container::value_type& value);
  .Pe
  .La Effects:
  .Cb
--- 1555,1563 ----
  .ix "[insert__iterator] [operator=]"
  .Pb
  insert_iterator<Container>&
! .\" CD2 comment Germany_244
! .\" Steve Rumsby
!   operator=(const typename Container::reference value);
  .Pe
  .La Effects:
  .Cb
diff -rc copyof~1/lib-numerics librar~1/lib-numerics
*** copyof~1/lib-numerics     Tue Jul 15 17:24:44 1997
--- librar~1/lib-numerics     Wed Jul 16 20:27:18 1997
***************
*** 269,286 ****
      template<class X> complex<T>& operator/=(const complex<X>&);
    };

! template<class T> complex<T> operator+(const complex<T>&, const T&);
! template<class T> complex<T> operator+(const T&, const complex<T>&);
! template<class T> complex<T> operator-(const complex<T>&, const T&);
! template<class T> complex<T> operator-(const T&, const complex<T>&);
! template<class T> complex<T> operator*(const complex<T>&, const T&);
! template<class T> complex<T> operator*(const T&, const complex<T>&);
! template<class T> complex<T> operator/(const complex<T>&, const T&);
! template<class T> complex<T> operator/(const T&, const complex<T>&);
! template<class T> complex<T> operator==(const complex<T>&, const T&);
! template<class T> complex<T> operator==(const T&, const complex<T>&);
! template<class T> complex<T> operator!=(const complex<T>&, const T&);
! template<class T> complex<T> operator!=(const T&, const complex<T>&);
  .Ce
  .P
  The class
--- 269,290 ----
      template<class X> complex<T>& operator/=(const complex<X>&);
    };
```

```
! .\" 26.2.2 [lib.complex]
! .\" CD2 comment UK 613
! .\" Steve Rumsby
!   template<class T> complex<T> operator+(const complex<T>&, const T&);
!   template<class T> complex<T> operator+(const T&, const complex<T>&);
!   template<class T> complex<T> operator-(const complex<T>&, const T&);
!   template<class T> complex<T> operator-(const T&, const complex<T>&);
!   template<class T> complex<T> operator*(const complex<T>&, const T&);
!   template<class T> complex<T> operator*(const T&, const complex<T>&);
!   template<class T> complex<T> operator/(const complex<T>&, const T&);
!   template<class T> complex<T> operator/(const T&, const complex<T>&);
!   template<class T> complex<T> operator==(const complex<T>&, const T&);
!   template<class T> complex<T> operator==(const T&, const complex<T>&);
!   template<class T> complex<T> operator!=(const complex<T>&, const T&);
!   template<class T> complex<T> operator!=(const T&, const complex<T>&);
! };
  .Ce
  .P
  The class
***************
*** 621,627 ****
      s.flags(o.flags());
      s.imbue(o.getloc());
      s.precision(o.precision());
!     s << '(' << x.real() << "," << x.imag() << ')' << ends;
      return o << s.str();
  }
  .Ce
--- 625,634 ----
      s.flags(o.flags());
      s.imbue(o.getloc());
      s.precision(o.precision());
! .\" 26.2.6 complex non-member operations [lib.complex.ops]
! .\" CD2 comment CD2-26-003
! .\" Steve Rumsby
!     s << '(' << x.real() << "," << x.imag() << ')';
      return o << s.str();
  }
  .Ce
***************
*** 899,909 ****
    template<class T> valarray<bool> operator>=(const valarray<T>&, const
T&);
    template<class T> valarray<bool> operator>=(const T&, const
valarray<T>&);
  .Ce
  .Cb
-   template<class T> T min(const valarray<T>&);
-   template<class T> T max(const valarray<T>&);
- .Ce
- .Cb
    template<class T> valarray<T> abs  (const valarray<T>&);
    template<class T> valarray<T> acos (const valarray<T>&);
    template<class T> valarray<T> asin (const valarray<T>&);
--- 906,919 ----
    template<class T> valarray<bool> operator>=(const valarray<T>&, const
T&);
    template<class T> valarray<bool> operator>=(const T&, const
valarray<T>&);
  .Ce
+ .\" 26.3.1  Header <valarray> synopsis [lib.valarray.synopsis]
+ .\" CD2 comment Germany _263/
+ .\" Steve Rumsby
+ .\" .Cb
+ .\"   template<class T> T min(const valarray<T>&);
+ .\"   template<class T> T max(const valarray<T>&);
+ .\" .Ce
  .Cb
    template<class T> valarray<T> abs  (const valarray<T>&);
    template<class T> valarray<T> acos (const valarray<T>&);
    template<class T> valarray<T> asin (const valarray<T>&);
***************
*** 975,981 ****
  Implementations introducing such replacement types shall provide
  additional functions and operators as follows:
  .LI
! for every functions taking a
  .CW "const valarray<T>&" ,
  identical functions taking the replacement types shall be added;
  .LI
--- 985,994 ----
  Implementations introducing such replacement types shall provide
  additional functions and operators as follows:
  .LI
! .\" 26.3.1 [lib.valarray.synopsis]
! .\" CD2 comment UK-Ed 706
! .\" Steve Rumsby
! for every function taking a
  .CW "const valarray<T>&" ,
  identical functions taking the replacement types shall be added;
  .LI
***************
*** 1878,1884 ****
      void operator>>=(const valarray<T>&) const;
  .Ce
  .Cb
!     void fill(const T&);
     ~slice_array();
    private:
      slice_array();
--- 1891,1899 ----
      void operator>>=(const valarray<T>&) const;
  .Ce
  .Cb
! .\" CD2 comment Germany _263/
! .\" Steve Rumsby
!     void operator= (constT&);
     ~slice_array();
    private:
      slice_array();
***************
*** 1987,1993 ****
  .H4 "\&\f7slice_array\fP\& fill function" lib.slice.arr.fill
  .ix "[slice__array] [fill]"
  .Pb
! void fill(const T&);
  .Pe
  .P
  This function has reference semantics, assigning the value of its
argument
--- 2002,2010 ----
  .H4 "\&\f7slice_array\fP\& fill function" lib.slice.arr.fill
  .ix "[slice__array] [fill]"
  .Pb
! .\" CD2 comment Germany _263/
! .\" Steve Rumsby
! void operator= (constT&);
  .Pe
  .P
  This function has reference semantics, assigning the value of its
argument
***************
*** 2154,2160 ****
      void operator>>=(const valarray<T>&) const;
  .Ce
  .Cb
!     void fill(const T&);
     ~gslice_array();
    private:
      gslice_array();
--- 2171,2179 ----
      void operator>>=(const valarray<T>&) const;
  .Ce
  .Cb
! .\" CD2 comment Germany _263/
! .\" Steve Rumsby
!     void operator= (constT&);
     ~gslice_array();
    private:
      gslice_array();
***************
*** 2256,2262 ****
  .H4 "\&\f7gslice_array\fP\& fill function" lib.gslice.array.fill
  .ix "[gslice__array] [fill]"
  .Pb
! void fill(const T&);
  .Pe
  .P
  This function has reference semantics, assigning the value of its
argument
--- 2275,2283 ----
  .H4 "\&\f7gslice_array\fP\& fill function" lib.gslice.array.fill
  .ix "[gslice__array] [fill]"
  .Pb
! .\" CD2 comment Germany _263/
! .\" Steve Rumsby
! void operator= (constT&);
  .Pe
  .P
  This function has reference semantics, assigning the value of its
argument
***************
*** 2287,2293 ****
      void operator>>=(const valarray<T>&) const;
  .Ce
  .Cb
!     void fill(const T&);
     ~mask_array();
    private:
      mask_array();
--- 2308,2316 ----
      void operator>>=(const valarray<T>&) const;
  .Ce
  .Cb
! .\" CD2 comment Germany _263/
! .\" Steve Rumsby
!     void operator=(const T&);
     ~mask_array();
    private:
      mask_array();
***************
*** 2383,2389 ****
```

```
  .H4 "\&\f7mask_array\fP\& fill function" lib.mask.array.fill
  .ix "[mask__array] [fill]"
  .Pb
! void fill(const T&);
  .Pe
  This function has reference semantics, assigning the value of its
  argument to the elements of the
--- 2406,2414 ----
  .H4 "\&\f7mask_array\fP\& fill function" lib.mask.array.fill
  .ix "[mask__array] [fill]"
  .Pb
! .\" CD2 comment Germany _263/
! .\" Steve Rumsby
! void operator=(const T&);
  .Pe
  This function has reference semantics, assigning the value of its
  argument to the elements of the
***************
*** 2413,2419 ****
      void operator>>=(const valarray<T>&) const;
  .Ce
  .Cb
!     void fill(const T&);
     ~indirect_array();
    private:
      indirect_array();
--- 2438,2446 ----
      void operator>>=(const valarray<T>&) const;
  .Ce
  .Cb
! .\" CD2 comment Germany _263/
! .\" Steve Rumsby
!     void operator=(const T&);
     ~indirect_array();
    private:
      indirect_array();
***************
*** 2484,2490 ****
  int addr[] = {2, 3, 1, 4, 4};
  valarray<size_t> indirect(addr, 5);
  valarray<double> a(0., 10), b(1., 5);
! array[indirect] = b;
  .Ce
  results in undefined behavior since element 4 is specified twice in the
  indirection.
--- 2511,2520 ----
  int addr[] = {2, 3, 1, 4, 4};
  valarray<size_t> indirect(addr, 5);
  valarray<double> a(0., 10), b(1., 5);
! .\" 26.3.9.2 [lib.indirect.array.assign]
! .\" CD2 comment UK 705
! .\" Steve Rumsby
! a[indirect] = b;
  .Ce
  results in undefined behavior since element 4 is specified twice in the
  indirection.
***************
*** 2534,2540 ****
  .H4 "\&\f7indirect_array\fP\& fill function" lib.indirect.array.fill
  .ix "[indirect__array] [fill]"
  .Pb
! void fill(const T&);
  .Pe
  .P
  This function has reference semantics, assigning the value of its
argument
--- 2564,2572 ----
  .H4 "\&\f7indirect_array\fP\& fill function" lib.indirect.array.fill
  .ix "[indirect__array] [fill]"
  .Pb
! .\" CD2 comment Germany _263/
! .\" Steve Rumsby
! void operator=(const T&);
  .Pe
  .P
  This function has reference semantics, assigning the value of its
argument
***************
*** 2600,2606 ****
             BinaryOperation \f6binary_op\fP);
  .Pe
  .La Effects:
! Initializes the accumulator
  .CW acc
  with the initial value
  .CW init
--- 2632,2641 ----
             BinaryOperation \f6binary_op\fP);
  .Pe
  .La Effects:
! .\" 26.4.1 Accumulate [lib.accumulate]
! .\" Library issue 26-002
! .\" Steve Rumsby
! Computes its result by initializing the accumulator
  .CW acc
  with the initial value
  .CW init
***************
*** 2619,2624 ****
--- 2654,2665 ----
  difficulty of defining the result of reduction on an empty sequence by
always requiring an initial value.
  .Fe
  .La Requires:
+ .\" 26.4.1 Accumulate [lib.accumulate]
+ .\" Library issue 26-002
+ .\" Steve Rumsby
+ T must meet the requirements of CopyConstructible
(_lib.copyconstructible_)
+ and Assignable (_lib.container.requirements_) types.
+ .P
  .CW binary_op
  shall not cause side effects.
  .\"===
***************
*** 2654,2659 ****
--- 2695,2703 ----
  .CW "[first2, first2 + (last - first)) "
  in order.
  .La Requires:
+ .\" 26.4.2 Inner product [lib.inner.product]
+ .\" Library issue 26-002
+ .\" Steve Rumsby
  .CW binary_op1
  and
  .CW binary_op2
diff -rc copyof~1/lib-strings librar~1/lib-strings
*** copyof~1/lib-strings      Tue Jul 15 17:24:48 1997
```

```
--- librar~1/lib-strings      Wed Jul 16 23:47:08 1997
***************
*** 109,115 ****
  .I "character type" ,
  that precede the termination null character type
  value
! .CW charT(0) .
  .H3 "Character traits requirements" lib.char.traits.require
  .P
  In Table \n+(Tn,
--- 109,117 ----
  .I "character type" ,
  that precede the termination null character type
  value
! .\" CD2 comment France 16
! .\" Steve Rumsby
! .CW charT() .
  .H3 "Character traits requirements" lib.char.traits.require
  .P
  In Table \n+(Tn,
***************
*** 193,206 ****
  _
  X::length(p)    \f5size_t\fP      T{
  yields: the smallest \&\f5i\fP\& such
! that \&\f5X::eq(p[i],charT(0))\fP\& is true.
  T}  linear
  _
  X::find(p,n,c)  T{
  \f5const\ X:: char_type*\fP
  T}  T{
  yields: the smallest \&\f5q\fP\& in [\f5p,p+n\fP)
! such that \&\f5X::eq(q,c)\fP\& is true, zero otherwise.
  T}  linear
  _
  X::move(s,p,n)  T{
--- 195,213 ----
  _
  X::length(p)    \f5size_t\fP      T{
  yields: the smallest \&\f5i\fP\& such
! .\" CD2 comment France 16
! .\" Steve Rumsby
! that \&\f5X::eq(p[i],charT())\fP\& is true.
  T}  linear
  _
  X::find(p,n,c)  T{
  \f5const\ X:: char_type*\fP
  T}  T{
  yields: the smallest \&\f5q\fP\& in [\f5p,p+n\fP)
! .\" 21.1.2  Character traits requirements
[lib.char.traits.require]
! .\" CD2 comment France 17
! .\" Steve Rumsby
! such that \&\f5X::eq(*q,c)\fP\& is true, zero otherwise.
  T}  linear
  _
  X::move(s,p,n)  T{
***************
*** 232,238 ****
  T}  linear
  _
  X::not_eof(e)   \f5int_type\fP    T{
! yields: \&\f5e\fP\& if \&\f5X::eq(e,X::eof())\fP\&
  is false, otherwise a value \&\f5f\fP\& such that
  \&\f5X::eq(f,X::eof())\fP\& is false.
  T}  constant
--- 239,248 ----
  T}  linear
  _
  X::not_eof(e)   \f5int_type\fP    T{
! .\" 21.1.2  Character traits requirements [lib.char.traits.require]
! .\" CD2 comment 21-007
! .\" Steve Rumsby
! yields: \&\f5e\fP\& if \&\f5X::eq_int_type(e,X::eof())\fP\&
  is false, otherwise a value \&\f5f\fP\& such that
  \&\f5X::eq(f,X::eof())\fP\& is false.
  T}  constant
***************
*** 265,271 ****
  \&\f5X::eq_int_type(X::to_int_type(c),
    X::to_int_type(d))\fP; otherwise, yields
  true if \&\f5e\fP\& and \&\f5f\fP\& are both
! copies of \&\f5X::eof()\fP.
  T}  constant
  _
  T{
--- 275,284 ----
  \&\f5X::eq_int_type(X::to_int_type(c),
    X::to_int_type(d))\fP; otherwise, yields
  true if \&\f5e\fP\& and \&\f5f\fP\& are both
! copies of \&\f5X::eof()\fP;
! otherwise, yields false if one of \&\f5e\fP\& and \&\f5f\fP\&
! are copies of \&\f5X::eof()\fP\& and the other is not;
! otherwise the value is unspecified.
  T}  constant
  _
  T{
***************
*** 290,296 ****
  .P
  The struct template
  .Cb
! template<class charT> struct char_traits { };
  .Ce
  shall be provided in the header
  .CW <string>
--- 303,312 ----
  .P
  The struct template
  .Cb
! .\" 21.1.2  Character traits requirements
[lib.char.traits.require]
! .\" CD2 comment Japan 3
! .\" Steve Rumsby
! template<class charT> struct char_traits;
  .Ce
  shall be provided in the header
  .CW <string>
***************
*** 324,332 ****
  .CW eof() .
  The type
  .CW int_type
! represents another character container type
  which can hold end-of-file to be used as a return type
! of the iostream class member functions.
  .Pb
  typedef OFF_T off_type;
  .Pe
--- 340,358 ----
  .CW eof() .
  The type
  .CW int_type
! .\" 21.1.3  traits typedefs [lib.char.traits.typedefs]
! .\" CD2 comment France 18
! .\" Steve Rumsby
! represents a character container type
  which can hold end-of-file to be used as a return type
! of the iostream class member functions.\*f
! .Fs
! If
! .CW "eof()"
! can be held in
! .CW char_type
! then some iostreams operations may give surprising results.
! .Fe
  .Pb
  typedef OFF_T off_type;
  .Pe
***************
*** 460,466 ****

      static int compare(const char_type* s1, const char_type* s2, size_t
n);
      static size_t length(const char_type* s);
!     static const char_type* find(const char_type* s, int n,
                                   const char_type& a);
      static char_type* move(char_type* s1, const char_type* s2, size_t n);
      static char_type* copy(char_type* s1, const char_type* s2, size_t n);
--- 486,495 ----

      static int compare(const char_type* s1, const char_type* s2, size_t
n);
      static size_t length(const char_type* s);
! .\" [lib.char.traits.specializations.char] 21.1.4.1 struct
char_traits<char>
! .\" CD2 comment Germany 21-015
! .\" Steve Rumsby
!     static const char_type* find(const char_type* s, size_t n,
                                   const char_type& a);
      static char_type* move(char_type* s1, const char_type* s2, size_t n);
      static char_type* copy(char_type* s1, const char_type* s2, size_t n);
***************
*** 529,534 ****
--- 558,571 ----
  and
  .CW <
  respectively.
+ .\" [lib.char.traits.specializations.char] 21.1.4.1 struct
char_traits<char>
+ .\" CD2 comment Germany 21-015
+ .\" Steve Rumsby
+ .P
+ The member
+ .CW "eof()"
+ returns
+ .CW EOF .
  .H4 "\&\f7struct char_traits<wchar_t>\fP\&"
lib.char.traits.specializations.wchar.t
  .Cb
  namespace std {
***************
```

```
*** 546,552 ****

      static int compare(const char_type* s1, const char_type* s2, size_t
n);
      static size_t length(const char_type* s);
!     static const char_type* find(const char_type* s, int n,
                                   const char_type& a);
      static char_type* move(char_type* s1, const char_type* s2, size_t n);
      static char_type* copy(char_type* s1, const char_type* s2, size_t n);
--- 583,592 ----

      static int compare(const char_type* s1, const char_type* s2, size_t
n);
      static size_t length(const char_type* s);
! .\" [lib.char.traits.specializations.wchar.t] 21.1.4.2 struct
char_traits<wchar_t>
! .\" CD2 comment Germany 21-015
! .\" Steve Rumsby
!     static const char_type* find(const char_type* s, size_t n,
                                   const char_type& a);
      static char_type* move(char_type* s1, const char_type* s2, size_t n);
      static char_type* copy(char_type* s1, const char_type* s2, size_t n);
***************
*** 626,631 ****
--- 666,679 ----
  and
  .CW <
  respectively.
+ .\" [lib.char.traits.specializations.wchar.t] 21.1.4.2 struct
char_traits<wchar_t>
+ .\" CD2 comment Germany 21-015
+ .\" Steve Rumsby
+ .P
+ The member
+ .CW "eof()"
+ returns
+ .CW WEOF .
  .\"-----------------------------------------------------------------------
----------
  .H2 "String classes" lib.string.classes
  .P
***************
*** 821,826 ****
--- 869,930 ----
  .CW out_of_range
  (_lib.out.of.range_).
  .ix "[out__of__range]"
+ .\" 21.3 Template class basic_string [lib.basic.string]
+ .\" CD2 comment 21-001
+ .\" Steve Rumsby
+ .P
+ References, pointers, and iterators referring to the elements of a
+ basic_string sequence may be
+ invalidated by the following uses of that basic_string object:
+ .LI
+ As an argument to non-member functions
+ .CW "swap()"
+ (_lib.string.special_),
+ .CW "operator>>()"
+ (_lib.string.io_), and
+ .CW "getline()"
+ (_lib.string.io_).
+ .LI
+ As an argument to
+ .CW "basic_string::swap()" .
+ .LI
+ Calling
+ .CW "data()"
+ and
+ .CW "c_str()"
+ member functions.
+ .LI
+ Calling non-const member functions, except
+ .CW "operator[]()" , "at()" , "begin()" ,
+ .CW "rbegin()" , "end()" ,
+ and
+ .CW "rend()" .
+ .LI
+ Subsequent to any of the above uses except the forms of
+ .CW "insert()"
+ and
+ .CW "erase()"
+ which return iterators, the first call to non-const member functions
+ .CW "operator[]()" , "at()" , "begin()" , "rbegin()" , "end()" ,
+ or
+ .CW "rend()" .
+ .P
+ .N[
+ These rules are formulated to allow, but not require,
+ a reference counted implemenation.
+ A reference counted implementation must have the same semantics
+ as a non-reference counted implementation.
+ .E[
+ .Cb
+ string s1("abc");
+
+ string::iterator i = s1.begin();
+ string s2 = s1;
+
+ *i = 'a'; // Must modify only s1
+ .Ce
+ .E]
+ .N]
  .Cb
  namespace std {
    template<class charT, class traits = char_traits<charT>,
***************
*** 910,915 ****
--- 1014,1023 ----
      basic_string& append(size_type \f6n\fP, charT \f6c\fP);
      template<class InputIterator>
        basic_string& append(InputIterator \f6first\fP, InputIterator
\f6last\fP);
+ .\" 21.3  Template class basic_string
[lib.basic.string]
+ .\" CD2 comment France 22
+ .\" Steve Rumsby
+     void push_back(const charT);
  .Ce
  .Cb
      basic_string& assign(const basic_string&);
***************
*** 1078,1084 ****
  T}
  size()    \&\f6rlen\fP\&
  capacity()      a value at least as large as \f5size()\fP
! get_allocator() \&\f6str\fP\&.get_allocator()
  .TE
  .Te
  .Pb
--- 1186,1195 ----
  T}
  size()    \&\f6rlen\fP\&
  capacity()      a value at least as large as \f5size()\fP
! .\" 21.3.1  basic_string constructors
[lib.string.cons]
! .\" CD2 comment France 23
! .\" Steve Rumsby
! .\" get_allocator()   \&\f6str\fP\&.get_allocator()
  .TE
  .Te
  .Pb
***************
*** 1310,1315 ****
--- 1421,1428 ----
  .Pe
  .La Returns:
  The maximum size of the string.
+ .La Note:
+ See Container requirements table (_lib.container.requirements_).
  .\"
  .ix "[basic__string] [resize]"
  .Pb
***************
*** 1372,1379 ****
  .CW capacity()
  is greater or equal to the argument of
  .CW reserve   .
! Reallocation invalidates all the references, pointers, and iterators
referring
! to the elements in the sequence.
  .\"
  .ix "[basic__string] [clear]"
  .Pb
--- 1485,1502 ----
  .CW capacity()
  is greater or equal to the argument of
  .CW reserve   .
! .\" 21.3.3 basic_string capacity [lib.string.capacity]
! .\" CD2 comment 21-001
! .\" Steve Rumsby
! .\" Reallocation invalidates all the references, pointers, and iterators
referring
! .\" to the elements in the sequence.
! .\" 21.3.3 [lib.string.capacity]
! .\" CD2 comment 21-004
! .\" Steve Rumsby
! .La Throws:
! .CW length_error
! if
! .CW "\f6res_arg\fP > max_size()" .
  .\"
  .ix "[basic__string] [clear]"
  .Pb
***************
*** 1399,1411 ****
  const_reference operator[](size_type \f6pos\fP) const;
  reference       operator[](size_type \f6pos\fP);
  .Pe
! .La Effects:
! The reference returned is invalid after any subsequent call to
! .CW c_str() ,
! .CW data() ,
! or any
! .CW \f1non-\fPconst
! member function for the object.
  .La Returns:
  If
  .CW "\&\f6pos\fP\& < size()" ,
--- 1522,1537 ----
  const_reference operator[](size_type \f6pos\fP) const;
  reference       operator[](size_type \f6pos\fP);
  .Pe
! .\" 21.3.4 basic_string elements access [lib.string.access]
! .\" CD2 comment 21-001
! .\" Steve Rumsby
! .\" .La Effects:
! .\" The reference returned is invalid after any subsequent call to
! .\" .CW c_str() ,
! .\" .CW data() ,
! .\" or any
! .\" .CW \f1non-\fPconst
! .\" member function for the object.
  .La Returns:
  If
  .CW "\&\f6pos\fP\& < size()" ,
***************
*** 1416,1422 ****
  the
  .CW const
  version returns
! .CW traits::eos() .
  Otherwise, the behavior is undefined.
  .ix "undefined"
  .\"
--- 1542,1550 ----
  the
  .CW const
  version returns
! .\" CD2 comment 21-002
! .\" Steve Rumsby
! .CW charT() .
  Otherwise, the behavior is undefined.
  .ix "undefined"
  .\"
***************
*** 1655,1661 ****
  .La Returns:
  .CW
insert(\f6pos\fP,basic_string<charT,traits,Allocator>(\f6n\fP,\f6c\fP)) .
  .Pb
! iterator insert(iterator \f6p\fP, charT \f6c\fP);
  .Pe
  .La Requires:
  \f6p\fP is a valid iterator on
--- 1783,1792 ----
  .La Returns:
  .CW
insert(\f6pos\fP,basic_string<charT,traits,Allocator>(\f6n\fP,\f6c\fP)) .
  .Pb
! .\" 21.3.5.4 basic_string::insert [lib.string::insert]
! .\" CD2 comment 21-005
! .\" Steve Rumsby
! iterator insert(iterator \f6p\fP, charT \f6c\fP = charT());
  .Pe
  .La Requires:
```

```
  \f6p\fP is a valid iterator on
***************
*** 1681,1690 ****
  .CW *this .
  .CW [\f6first\fP,\f6last\fP)
  is a valid range.
! .La Effects:
! inserts copies of the characters in the range
! .CW [\f6first\fP,\f6last\fP)
! before the character referred to by \f6p\fP.
  .\"
  .H4 "\&\f7basic_string::erase\fP\&" lib.string::erase
  .ix "[basic__string] [erase]"
--- 1812,1826 ----
  .CW *this .
  .CW [\f6first\fP,\f6last\fP)
  is a valid range.
! .\" 21.3.5.4  basic_string::insert [lib.string::insert]
! .\" Library issue 21-018
! .\" Steve Rumsby
! .\" .La Effects:
! .\" inserts copies of the characters in the range
! .\" .CW [\f6first\fP,\f6last\fP)
! .\" before the character referred to by \f6p\fP.
! .La Returns:
! .CW "insert(\f6p\fP,basic_string<
charT,traits,Allocator>(\f6first\fP,\f6last\fP))" .
  .\"
  .H4 "\&\f7basic_string::erase\fP\&" lib.string::erase
  .ix "[basic__string] [erase]"
***************
*** 1945,1951 ****
  elements equal the corresponding elements of the string controlled by
  .CW *this
  and whose last element is a null character specified by
! .CW traits::eos() .
  .La Requires:
  The program shall not alter any of the values stored in the array.
  Nor shall the program treat the returned value as a valid pointer value
--- 2081,2089 ----
  elements equal the corresponding elements of the string controlled by
  .CW *this
  and whose last element is a null character specified by
! .\" CD2 comment 21-002
! .\" Steve Rumsby
! .CW charT() .
  .La Requires:
  The program shall not alter any of the values stored in the array.
  Nor shall the program treat the returned value as a valid pointer value
***************
*** 1953,1962 ****
  .CW basic_string
  that designates the same object as
  .CW this .
! .La Notes:
! Uses
! .CW traits::eos() .
! .ix "[char__traits] [eos]"
  .\"
  .ix "[basic__string] [data]"
  .Pb
--- 2091,2102 ----
  .CW basic_string
  that designates the same object as
  .CW this .
! .\" CD2 comment 21-002
! .\" Steve Rumsby
! .\" .La Notes:
! .\" Uses
! .\" .CW traits::eos() .
! .\" .ix "[char__traits] [eos]"
  .\"
  .ix "[basic__string] [data]"
  .Pb
***************
*** 2659,2665 ****
  Begins by constructing a
  .CW sentry
  object \&\f6k\fP\& as if by
! .CW "basic_istream<charT,traits>::sentry \&\f6k\fP(\f6is\fP)" .
  If
  .CW bool(\f6k\fP)
  is true, it calls
--- 2799,2807 ----
  Begins by constructing a
  .CW sentry
  object \&\f6k\fP\& as if by
! .\" 21.3.7.9 [lib.string.io]
! .\" Library issue 21-011
! .CW "basic_istream<charT,traits>::sentry \&\f6k\fP(\f6is\fP, true)" .
  If
  .CW bool(\f6k\fP)
  is true, it calls
diff -rc copyof~1/lib-support librar~1/lib-support
*** copyof~1/lib-support      Tue Jul 15 17:24:52 1997
--- librar~1/lib-support      Tue Jul 15 21:39:38 1997
***************
*** 92,97 ****
--- 92,102 ----
  accepts a restricted set of \&\f6type\fP\& arguments in this
International Standard.
  .ix "[offsetof]"
  \&\f6type\fP\& shall be a POD structure or a POD union (_class_).
+ .\" 18.1 Types [lib.support.types]
+ .\" CD2 comment CD2-18-001
+ .\" Steve Rumsby
+ The result of applying the offsetof macro to a field which is
+ a static data member, a function member is undefined.
  .Xr
  subclause _expr.sizeof_, Sizeof,
  subclause _expr.add_, Additive operators,
***************
*** 132,138 ****
  that they are usable as
  integral constant expressions.
  .P
! Non-fundamental types, such as
  .CW complex<T>
  (_lib.complex_), shall not have specializations.
  .DS L
--- 137,146 ----
  that they are usable as
  integral constant expressions.
  .P
! .\" 18.2.1 [lib.limits]
! .\" Library issue 18-008
! .\" Steve Rumsby
! Non-fundamental standard types, such as
```

```
  .CW complex<T>
  (_lib.complex_), shall not have specializations.
  .DS L
***************
*** 763,769 ****
--- 771,793 ----
  The contents are the same as the Standard C library header
  .CW <stdlib.h> ,
  with the following changes:
+ .\" 18.3 Start and termination [lib.support.start.term
+ .\" CD2 comment CD2-18-002
+ .\" Steve Rumsby
  .\"
+ .ix "[abort]"
+ .Pb
+ abort(void)
+ .Pe
+ .P
+ The function
+ .CW abort() has additional behavior in this International Standard:
+ .LI
+ The program is terminated without executing destructors for objects of
+ automatic or static storage
+ duration and without calling the functions passed to
+ .CW atexit()
+ (_basic.start.term_).
  .ix "[atexit]"
  .Pb
  atexit(void (*\&\f6f\fP\&)(void))
***************
*** 947,957 ****
  that displaces the default version defined by the
  \*C Standard library.
  .La "Required behavior:"
! Return a pointer to dynamically allocated storage (_basic.stc.dynamic_),
  or else throw a
  .CW bad_alloc
  .ix "[bad__alloc]"
  exception.
  .La "Default behavior:"
  .LI
  Executes a loop:
--- 971,985 ----
  that displaces the default version defined by the
  \*C Standard library.
  .La "Required behavior:"
! .\" 18.4.1.1[lib.new.delete.single]
! .\" CD2 comment UK 696
! .\" Steve Rumsby
! Return a non-null pointer to suitably aligned storage
(_basic.stc.dynamic_),
  or else throw a
  .CW bad_alloc
  .ix "[bad__alloc]"
  exception.
+ This requirement is binding on a replacement version of this function.
  .La "Default behavior:"
  .LI
  Executes a loop:
***************
*** 1284,1290 ****
--- 1312,1326 ----
  .La Effects:
  Constructs an object of class
  .CW bad_alloc .
+ .\" 18.4.2.1  Class bad_alloc [lib.bad.alloc]
+ .\" CD2 comment UK 697
+ .\" Steve Rumsby
  .\"
+ .La Notes:
+ The result of calling
+ .CW what()
+ on the newly constructed object is implementation-defined.
+ .ix "implementation-defined"
  .ix "[bad__alloc] [bad__alloc]"
  .ix "[bad__alloc] [operator=]"
  .Pb
***************
*** 1294,1311 ****
  .La Effects:
  Copies an object of class
  .CW bad_alloc .
- .La Notes:
- The result of calling
- .CW what()
- on the newly constructed object is implementation-defined.
- .ix "implementation-defined"
  .\"
  .ix "[bad__alloc] [what]"
  .Pb
  virtual const char* what() const throw();
  .Pe
  .La Returns:
! An implementation-defined value.
  .ix "implementation-defined [bad__alloc::what]"
  .\"----
  .H4 "Type \&\f7new_handler\fP\&" lib.new.handler
--- 1330,1346 ----
  .La Effects:
  Copies an object of class
  .CW bad_alloc .
  .\"
  .ix "[bad__alloc] [what]"
  .Pb
  virtual const char* what() const throw();
  .Pe
  .La Returns:
! .\" 18.4.2.1 /5 [lib.bad.alloc]
! .\" CD2 comment UK 664
! .\" Library issue 18-007
! .\" Steve Rumsby
! An implementation-defined NTBS.
  .ix "implementation-defined [bad__alloc::what]"
  .\"----
  .H4 "Type \&\f7new_handler\fP\&" lib.new.handler
***************
*** 1434,1439 ****
--- 1469,1478 ----
  const char* name() const;
  .Pe
  .La Returns:
+ .\" 18.5.1 /7 [lib.type.info]
+ .\" CD2 comment UK 664
+ .\" Library issue 18-007
+ .\" Steve Rumsby
  an implementation-defined value.
  .ix "implementation-defined [type__info::name]"
  .La Notes:
```

```
***************
*** 1488,1493 ****
--- 1527,1540 ----
  .La Effects:
  Constructs an object of class
  .CW bad_cast .
+ .\" 18.5.2 [lib.bad.cast]
+ .\" CD2 comment UK 699
+ .\" Steve Rumsby
+ .La Notes:
+ The result of calling
+ .CW what()
+ on the newly constructed object is implementation-defined.
+ .ix "implementation-defined"
  .\"
  .ix "[bad__cast] [bad__cast]"
  .ix "[bad__cast] [operator=]"
***************
*** 1498,1515 ****
  .La Effects:
  Copies an object of class
  .CW bad_cast .
- .La Notes:
- The result of calling
- .CW what()
- on the newly constructed object is implementation-defined.
- .ix "implementation-defined"
  .\"
  .ix "[bad__cast] [what]"
  .Pb
  virtual const char* what() const throw();
  .Pe
  .La Returns:
! An implementation-defined value.
  .ix "implementation-defined [bad__cast::what]"
  .La Notes:
  The message may be a null-terminated multibyte string
(_lib.multibyte.strings_),
--- 1545,1561 ----
  .La Effects:
  Copies an object of class
  .CW bad_cast .
  .\"
  .ix "[bad__cast] [what]"
  .Pb
  virtual const char* what() const throw();
  .Pe
  .La Returns:
! .\" 18.5.2 /5 [lib.bad.cast]
! .\" CD2 comment UK 664
! .\" Library issue 18-007
! .\" Steve Rumsby
! An implementation-defined NTBS.
  .ix "implementation-defined [bad__cast::what]"
  .La Notes:
  The message may be a null-terminated multibyte string
(_lib.multibyte.strings_),
***************
*** 1547,1552 ****
--- 1593,1607 ----
  .La Effects:
  Constructs an object of class
  .CW bad_typeid .
+ .\" 18.5.3 [lib.bad.typeid]
+ .\" Note: comments referenced 18.5.2 [lib.bad.typeid]
+ .\" CD2 comment UK 700
+ .\" Steve Rumsby
+ .La Notes:
+ The result of calling
+ .CW what()
+ on the newly constructed object is implementation-defined.
+ .ix "implementation-defined"
  .\"
  .ix "[bad__typeid] [bad__typeid]"
  .ix "[bad__typeid] [operator=]"
***************
*** 1557,1573 ****
  .La Effects:
  Copies an object of class
  .CW bad_typeid .
- .La Notes:
- The result of calling
- .CW what()
- on the newly constructed object is implementation-defined.
- .ix "implementation-defined"
  .\"
  .ix "[bad__typeid] [what]"
  .Pb
  virtual const char* what() const throw();
  .Pe
  .La Returns:
  An implementation-defined value.
  .ix "implementation-defined [bad__typeid::what]"
  .La Notes:
--- 1612,1627 ----
  .La Effects:
  Copies an object of class
  .CW bad_typeid .
  .\"
  .ix "[bad__typeid] [what]"
  .Pb
  virtual const char* what() const throw();
  .Pe
  .La Returns:
+ .\" 18.5.3 /5 [lib.bad.typeid]
+ .\" CD2 comment UK 664
+ .\" Library issue 18-007
+ .\" Steve Rumsby
  An implementation-defined value.
  .ix "implementation-defined [bad__typeid::what]"
  .La Notes:
***************
*** 1704,1709 ****
--- 1758,1772 ----
  .La Effects:
  Constructs an object of class
  .CW bad_exception .
+ .\" 18.6.2.1  Class bad_exception [lib.bad.exception]
+ .\" Note, the original comment referenced 18.5.2 [lib.bad.exception]
+ .\" CD2 comment 701
+ .\" Steve Rumsby
+ .La Notes:
+ The result of calling
+ .CW what()
+ on the newly constructed object is implementation-defined.
+ .ix "implementation-defined"
  .\"
  .ix "[bad__exception] [bad__exception]"
  .ix "[bad__exception] [operator=]"
***************
*** 1714,1731 ****
  .La Effects:
  Copies an object of class
  .CW bad_exception .
- .La Notes:
- The result of calling
- .CW what()
- on the newly constructed object is implementation-defined.
- .ix "implementation-defined"
  .\"
  .ix "[bad__exception] [what]"
  .Pb
  virtual const char* what() const throw();
  .Pe
  .La Returns:
! An implementation-defined value.
  .ix "implementation-defined [bad__exception::what]"
  .La Notes:
  The message may be a null-terminated multibyte string
(_lib.multibyte.strings_),
--- 1777,1793 ----
  .La Effects:
  Copies an object of class
  .CW bad_exception .
  .\"
  .ix "[bad__exception] [what]"
  .Pb
  virtual const char* what() const throw();
  .Pe
  .La Returns:
! .\" 18.6.2.1 /5 [lib.bad.exception]
! .\" CD2 comment UK 664
! .\" Library issue 18-007
! .\" Steve Rumsby
! An implementation-defined NTBS.
  .ix "implementation-defined [bad__exception::what]"
  .La Notes:
  The message may be a null-terminated multibyte string
(_lib.multibyte.strings_),
diff -rc copyof~1/lib-utilities librar~1/lib-utilities
*** copyof~1/lib-utilities    Tue Jul 15 17:24:54 1997
--- librar~1/lib-utilities    Tue Jul 15 23:00:18 1997
***************
*** 299,312 ****
  _
  X a(b);         post: \f5Y(a) == b\fP
  _
! x.construct(p,t)      (not used)  Effect: \f5new((void*)p) T(t)\f5
  _
! x.destroy(p)    (not used)  Effect: \f5((T*)p)->~T()
  .TE
  .ad
  .Te
  .Fs
! It is intended that \&\f5a::allocate\fP\& be an efficient means
  of allocating a single object of type \&\f5T\fP\&, even when
  \&\f5sizeof(T)\fP\& is small.  That is, there is no need for a
  container to maintain its own ``free list''.
--- 299,318 ----
  _
  X a(b);         post: \f5Y(a) == b\fP
  _
```

```
! .\" 20.1.5 [lib.allocator.requirements]
! .\" CD2 comment UK 675
! .\" Steve Rumsby
! a.construct(p,t)      (not used)  Effect: \f5new((void*)p) T(t)\f5
  _
! a.destroy(p)    (not used)  Effect: \f5((T*)p)->~T()
  .TE
  .ad
  .Te
  .Fs
! .\" 20.1.5 [lib.allocator.requirements]
! .\" CD2 comment UK 692
! .\" Steve Rumsby
! It is intended that \&\f5a.allocate\fP\& be an efficient means
  of allocating a single object of type \&\f5T\fP\&, even when
  \&\f5sizeof(T)\fP\& is small.  That is, there is no need for a
  container to maintain its own ``free list''.
***************
*** 758,766 ****
  .P
  The library provides basic function object classes for all of the
comparison
  operators in the language (_expr.rel_, _expr.eq_).
! In all cases, type
! .CW T
! is convertible to type
  .CW bool .
  .ix "[equal__to]"
  .Pb
--- 764,774 ----
  .P
  The library provides basic function object classes for all of the
comparison
  operators in the language (_expr.rel_, _expr.eq_).
! .\" 20.3.3  Comparisons [lib.comparisons]
! .\" CD2 comment UK 676
! .\" In all cases, type
! .\" .CW T
! .\" is convertible to type
  .CW bool .
  .ix "[equal__to]"
  .Pb
***************
*** 884,890 ****
  .Pb
  template <class Predicate>
    class unary_negate
!     : public unary_function<Predicate::argument_type,bool> {
  public:
    explicit unary_negate(const Predicate& \f6pred\fP);
    bool operator()(const argument_type& \f6x\fP) const;
--- 892,901 ----
  .Pb
  template <class Predicate>
    class unary_negate
! .\" 20.3.5 [lib.negators]
! .\" CD2 comment UK 693
! .\" Steve Rumsby
!     : public unary_function<typename Predicate::argument_type,bool> {
  public:
    explicit unary_negate(const Predicate& \f6pred\fP);
    bool operator()(const argument_type& \f6x\fP) const;
***************
*** 905,912 ****
```

```
  .Pb
  template <class Predicate>
    class binary_negate
!     : public binary_function<Predicate::first_argument_type,
!                              Predicate::second_argument_type, bool> {
    public:
      explicit binary_negate(const Predicate& \f6pred\fP);
      bool operator()(const first_argument_type&  \f6x\fP,
--- 916,926 ----
  .Pb
  template <class Predicate>
    class binary_negate
! .\" 20.3.5 [lib.negators]
! .\" CD2 comment UK 693
! .\" Steve Rumsby
!     : public binary_function<typename Predicate::first_argument_type,
!                              typename Predicate::second_argument_type,
bool> {
    public:
      explicit binary_negate(const Predicate& \f6pred\fP);
      bool operator()(const first_argument_type&  \f6x\fP,
***************
*** 945,959 ****
  .Cb
    template <class Operation>
    class binder1st
!     : public unary_function<Operation::second_argument_type,
!                             Operation::result_type> {
    protected:
      Operation                     op;
!     Operation::first_argument_type value;
  .Ce
  .Cb
    public:
!     binder1st(const Operation& \f6x\fP, const
Operation::first_argument_type& \f6y\fP);
      result_type operator()(const argument_type& \f6x\fP) const;
    };
  .Ce
--- 959,976 ----
  .Cb
    template <class Operation>
    class binder1st
! .\" (20.3.6.2) [lib.bind.1st]
! .\" CD2 comment UK 694
! .\" Steve Rumsby
!     : public unary_function<typename Operation::second_argument_type,
!                             typename Operation::result_type> {
    protected:
      Operation                     op;
!     typename Operation::first_argument_type value;
  .Ce
  .Cb
    public:
!     binder1st(const Operation& \f6x\fP, const typename
Operation::first_argument_type& \f6y\fP);
      result_type operator()(const argument_type& \f6x\fP) const;
    };
  .Ce
***************
*** 975,996 ****
    binder1st<Operation> bind1st(const Operation& \f6op\fP, const T&
\f6x\fP);
  .Pe
  .La Returns:
! .CW "binder1st<Operation>(\f6op\fP,
Operation::first_argument_type(\f6x\fP))" .
  .\"----
  .H4 "Template class \&\f7binder2nd\fP\&" lib.binder.2nd
  .ix "[binder2nd]"
  .Cb
    template <class Operation>
    class binder2nd
!     : public unary_function<Operation::first_argument_type,
!                             Operation::result_type> {
    protected:
      Operation                       op;
!     Operation::second_argument_type value;
  .Ce
  .Cb
    public:
!     binder2nd(const Operation& \f6x\fP, const
Operation::second_argument_type& \f6y\fP);
      result_type operator()(const argument_type& \f6x\fP) const;
    };
  .Ce
--- 992,1019 ----
    binder1st<Operation> bind1st(const Operation& \f6op\fP, const T&
\f6x\fP);
  .Pe
  .La Returns:
! .\" (20.3.6.2) [lib.bind.1st]
! .\" Library issue 20-005
! .\" Steve Rumsby
! .CW "binder1st<Operation>(\f6op\fP, typename
Operation::first_argument_type(\f6x\fP))" .
  .\"----
  .H4 "Template class \&\f7binder2nd\fP\&" lib.binder.2nd
  .ix "[binder2nd]"
  .Cb
    template <class Operation>
    class binder2nd
! .\" (20.3.6.4) [lib.bind.2nd]
! .\" CD2 issue UK 695
! .\" Steve Rumsby
!     : public unary_function<typename Operation::first_argument_type,
!                             typename Operation::result_type> {
    protected:
      Operation                       op;
!     typename Operation::second_argument_type value;
  .Ce
  .Cb
    public:
!     binder2nd(const Operation& \f6x\fP, const typename
Operation::second_argument_type& \f6y\fP);
      result_type operator()(const argument_type& \f6x\fP) const;
    };
  .Ce
***************
*** 1012,1018 ****
    binder2nd<Operation> bind2nd(const Operation& \f6op\fP, const T&
\f6x\fP);
  .Pe
  .La Returns:
! .CW "binder2nd<Operation>(\f6op\fP,
Operation::second_argument_type(\f6x\fP))" .
  .P
  .E[
  .Cb
--- 1035,1044 ----
    binder2nd<Operation> bind2nd(const Operation& \f6op\fP, const T&
\f6x\fP);
  .Pe
  .La Returns:
! .\" (20.3.6.4) [lib.bind.2nd]
! .\" Library issue 20-005
! .\" Steve Rumsby
! .CW "binder2nd<Operation>(\f6op\fP, typename
Operation::second_argument_type(\f6x\fP))" .
  .P
  .E[
  .Cb
***************
*** 1103,1109 ****
          : public unary_function<T*, S> {
    public:
      explicit mem_fun_t(S (T::*p)());
!     S operator()(T* p);
  };
  .Ce
  .P
--- 1129,1138 ----
          : public unary_function<T*, S> {
    public:
      explicit mem_fun_t(S (T::*p)());
! .\" 20.3.8 [lib.member.pointer.adaptors]
! .\" Library issue 20-006
! .\" Steve Rumsby
!     S operator()(T* p) const;
  };
  .Ce
  .P
***************
*** 1116,1122 ****
          : public binary_function<T*, A, S> {
    public:
      explicit mem_fun1_t(S (T::*p)(A));
!     S operator()(T* p, A x);
  };
  .Ce
  .P
--- 1145,1154 ----
          : public binary_function<T*, A, S> {
    public:
      explicit mem_fun1_t(S (T::*p)(A));
! .\" 20.3.8 [lib.member.pointer.adaptors]
! .\" Library issue 20-006
! .\" Steve Rumsby
!     S operator()(T* p, A x) const;
  };
  .Ce
  .P
***************
*** 1148,1154 ****
          : public unary_function<T, S> {
    public:
      explicit mem_fun_ref_t(S (T::*p)());
!     S operator()(T& p);
  };
  .Ce
  .P
--- 1180,1189 ----
```

```
          : public unary_function<T, S> {
    public:
      explicit mem_fun_ref_t(S (T::*p)());
! .\" 20.3.8 [lib.member.pointer.adaptors]
! .\" Library issue 20-006
! .\" Steve Rumsby
!     S operator()(T& p) const;
  };
  .Ce
  .P
***************
*** 1161,1167 ****
          : public binary_function<T, A, S> {
    public:
      explicit mem_fun1_ref_t(S (T::*p)(A));
!     S operator()(T& p, A x);
  };
  .Ce
  .P
--- 1196,1205 ----
          : public binary_function<T, A, S> {
    public:
      explicit mem_fun1_ref_t(S (T::*p)(A));
! .\" 20.3.8 [lib.member.pointer.adaptors]
! .\" Library issue 20-006
! .\" Steve Rumsby
!     S operator()(T& p, A x) const;
  };
  .Ce
  .P
***************
*** 1392,1398 ****
  namespace std {
    template <class OutputIterator, class T>
    class raw_storage_iterator
!     : public iterator<output_iterator_tag,void,void,void> {
    public:
      explicit raw_storage_iterator(OutputIterator \f6x\fP);
  .Ce
--- 1430,1439 ----
  namespace std {
    template <class OutputIterator, class T>
    class raw_storage_iterator
! .\" 20.4.2p1 [lib.storage.iterator]
! .\" CD2 comment 20-001
! .\" Steve Rumsby
!     : public iterator<output_iterator_tag,void,void,void,void> {
    public:
      explicit raw_storage_iterator(OutputIterator \f6x\fP);
  .Ce
***************
*** 1601,1608 ****
  .CW *this
  owns
  .CW *get()
! if and only if \f6a\fP owns
! .CW *\f6a\fP .
  .\"
  .Pb
  template<class Y> auto_ptr(const auto_ptr<Y>& \f6a\fP) throw();
--- 1642,1653 ----
  .CW *this
  owns
  .CW *get()
```

```
! .\" 20.4.5.1  auto_ptr constructors [lib.auto.ptr.cons]
! .\" CD2 comment Germany _20451
! .\" Steve Rumsby
! if and only if \f6a\fP owned
! .CW *\f6a\fP
! on entry.
  .\"
  .Pb
  template<class Y> auto_ptr(const auto_ptr<Y>& \f6a\fP) throw();
***************
*** 1621,1628 ****
  .CW *this
  owns
  .CW *get()
! if and only if \f6a\fP owns
! .CW *\f6a\fP .
  .\"
  .ix "[auto__ptr] [operator=]"
  .Pb
--- 1666,1677 ----
  .CW *this
  owns
  .CW *get()
! .\" 20.4.5.1  auto_ptr constructors [lib.auto.ptr.cons]
! .\" CD2 comment Germany _20451
! .\" Steve Rumsby
! if and only if \f6a\fP owned
! .CW *\f6a\fP
! on entry.
  .\"
  .ix "[auto__ptr] [operator=]"
  .Pb
```