Document Number: X3J16/92-0132, WG21/N0208
Date: 11/20/92
Project: Programming Language C++
Ref Doc: X3J16/92-0082, WG21/N0159
Reply To: Michael J. Vilot
ObjectCraft, Inc.
P.O. Box 6066
Nashua NH 03063 USA
mjv@objects.mv.com

Report from the Library working group
Work done July – November 1992

This report summarizes the progress made by the Library working group between the Toronto and Boston meetings, and the work done at the Boston meeting. Section 1 of the report provides details of the progress between the meetings. Section 2 provides details of the activities at the latter meeting. The final Section 3 concludes with a summary of work planned for the next meeting.

The main body of the report concentrates on the key discussions and consensus reached by the Library working group. Appendices contain the details. Appendix A lists the open issues the working group is actively pursuing. Appendix B lists the pending items that have been identified, but not yet pursued. Appendix C records the decisions on which the working group has reached consensus.

1. Progress Between the Meetings

Working group members volunteered to complete each of the work items identified at the last meeting. The results of most of these activities are documents, usually in the form of a proposal to the full X3J16 committee. Discussions on the `x3j16-lib` electronic mail reflector provide feedback and comment on the work items, as well as raise new issues.

Documents

The documents produced by working group members are intended to become proposals to X3J16 for inclusion into the Working Paper. Most have been in preparation for several meetings, undergoing revision after obtaining feedback from working group meetings and elsewhere.

17.1 Language Support

Mike Vilot provided a revised proposal, which clarifed the semantics of program-supplied `operator new()` and `operator delete()` functions, as well as the *new-handler*, *unexpected*, and *terminate* functions (92–0084/N0161).

17.2 String

Uwe Steinmüller provided a revised proposal that included the results of discussions at the London meeting (92–0090/N0167).

17.3 Input/Output

Jerry Schwarz provided version 5 of the proposal, including initial support for locales (92–0102/N0179).

17.5 Containers

Chuck Allison provided a revised bitset proposal (92–0051R1/N0128).

Electronic Mail

Comments directed at `x3j16-lib@redbone.att.com` and archived there represent the "conversation" among Library working group members between meetings. Comments this time focused on evolving proposals.

Library consistency

The group discussed class and member naming style as inconsistent across current proposals, and some suggestions for a common style (x3j16-lib-312, 316–318, 321, 329–330). The roup discussed the topic further at the Boston meeting, and agreed upon a standard "boilerplate" for class and function descriptions.

Type *boolean*

The topic of a standard boolean type generated more heat than light (x3j16-lib-313–315, 319, 325–328, 331). The group reached no consensus. The following message sums up the attitude of several participants in the discussion:

```
>
> Uwe Steinmueller is so right.
>
> It is a pity that so much time is wasted on discussing items which can be
> solved as enums or even #defines, where we have much more important things
> to discuss!
>
> For all those who like formal proposals:
>
> enum boolean { FALSE, TRUE };
>
> Or did I miss something very important?
>
> Michael Daeumling   73477.2366@compuserve.com
> Softeam GmbH
```

String class

Uwe Steinmüller posted a revised version of the String class and an implementation. There were several comments on the details of the proposal (x3j16-lib-299, 300, 302, 305, 335, 337–344).

Iostream comments

Jerry Schwarz answered some questions about the details of the Input/Output proposal (x3j16-lib-298, 332–336).

Bits classes

Chuck Allison posted a revised version of the proposal (x3j16-lib-277).

Dynarray class

Uwe Steinmüller posted a revised version of the DynArray class and an implementation (x3j16-lib-306, 307).

2. Activity at the Meeting

The discussion at each meeting generally follows the topics outlined for the Library portion of the Working Paper. Some subgroups have been formed to work in parallel with the rest of the working group (such as for strings and iostreams).

This section of the report summarizes the key discussions, issues raised, and decisions reached during the week.

General

The Library working group is making progress, refining proposals to the point where they can be submitted for a vote by X3J16 and W21. The discussion of the Language Support, String, Input/Output and Bits proposals at the Toronto meeting cleared up many details, and the discussion at the Boston meeting continued to refine many details.

Much of the group's discussion focused on the Language Support and Input/Output proposals, since they were ready for consideration by the full X3J16 committee. The group also discussed the strings and bits proposals, and these should be ready for a vote at the Portland meeting. The group also discussed a consistent naming and formatting style for the proposals, to simplify the Editor's job when incorporating elements of the standard library into the Working Paper.

The group also discussed consistency and formatting style for the classes and functions in the standard library. The naming convention for classes would be all lower-case, with no separators between distinct words (e.g. `fstreambuf`). The naming convention for members would be all lower-case, with underscores as separators between distinct words (e.g. `dynarray::get_at`).

The group decided to describe functions in the library using a format similar to that of theC standard:

Synopsis
Provide a function prototype, including *exception-specification*, as applicable.
Description
Define pre-conditions, actions (including changes to object state), and side effects. Define post-conditions for normal return.
Returns
Describe the range of result values upon normal return.
Exceptions
Describe the post-condition failure that led to the exception, and the value(s) of the exception object thrown.

17.1 Language Support

The two issues that received the most attention in this proposal were the wording and the decision to change the default behavior of the implementation-supplied *new-handler*.

The wording changes focused on providing clear and unambiguous specifications of each function. This turned out to be difficult, because the functions `operator new()` and `operator delete()` can be replaced by arbitrary functions in a C++ program. Similarly, arbitrary functions can be installed as a *new-handler*, *terminate-function*, or *unexpected-function*.

Jerry Schwarz discussed the issues in document 92-0058/N0135. At the Toronto meeting, the Library working group discussed his analysis, and decided (by a vote of 13 for throwing `xalloc`, 0 for returning null, and 3 for doing both) to adopt his suggestion (2), leaving the possibility that a *new-expression* could return null as undefined behavior. Arkady Rabinov of Apple pointed out at that time that this was crucial for supporting the transition from existing code to the proposed semantics.

In the discussion before the full X3J16 committee in Toronto, Martin O'Riordan of Microsoft raised the most vocal criticism of the proposed change. At the Boston meeting, Alan Sloane of Sun repeated the criticism. The key point of his argument was an insistence upon standardizing existing practice. The example was a memory allocation system that made allocation requests and depended upon a null return value, as in example (3), below.

(1) erroneous program — never checks

```
T* p = 0;
p = new T;  // might fail
*p;         // error
```

(2) "careful" program, v.1: — provide own checks and/or handling

```
extern void recover();
set_new_handler(recover);
p = new T;
*p;         // safe if we get here
```

(3) "careful" program, v. 2: — checks for null, but not exception

```
p = new T;
if (!p) do_something_clever();
else ... *p ...
```

(4) "careful" program, v. 3: — checks for exception, but not for null

```
try {
  p = new T;
} catch (xalloc& x) {
  // recover & goto retry, return, or call exit/abort
}
*p;
```

The main objection is the required change from a type (3) program to a type (4), with the concern that catching all the failed attempts would be too expensive. With the revised wording to allow a null return as implementation-defined behavior, the existing code can be retained by converting it to a type (2) program.

The final resolution was to endorse the design of having the the implementation provide a default *new-handler* that throws an `xalloc` exception if it cannot free storage. Existing behavior could be re-established trivially with a C++ program that calls `set_new_handler(0)` to unset the default *new-handler*, and returning null from a *new-expression* would be allowed as implementation-defined behavior for just such backward compatibility.

On Friday, the X3J16 and WG21 committees voted to accept the proposal with the editorial changes identified during the week's discussion.

17.2 String

At the Toronto meeting, the group had decided (by a vote of 8 in favor, 2 against) to revise the proposal, with several string classes (each focused on a specific aspect of representation) and well-defined conversions among them. The revised proposal concentrated instead on a single class, since the author did not agree with the consensus of the working group.

The discussion centered on the details of the proposal, and reaching agreement on the facilities provided by a `text` class.

The group felt that the string proposal would be ready for a vote at the Portland meeting, and a definite text class proposal would be ready for discussion at that meeting.

17.3 Input/Output

Revision 5 of the iostreams proposal included most of the results of discussions at the Toronto meeting. Most of the changes related to support for locales and wide characters.

At the Toronto meeting, the group agreed that locale processing belonged in the stream classes. The main question was how to design the "right" behavior: should all streams use a single, global locale setting (as in the C library), or should they each have their own? The conclusion was to do both: the default was for each stream to have no locale setting enabled. Any stream in such a state would look to the global locale setting. A program can explicitly set a stream to a specific locale setting.

In this way, C++ programs can reflect C program behavior (I/O with global locale state), but also support multiple locales in the same program (not possible in C).

Jerry also maintained that streambufs should retain their `char` buffer design. The difficult design issue was how to support seeking to arbitrary positions, if such a position lands in the middle of a multibyte sequence involving shift states. Accounting for shift states and file positions complicates the `streampos` and `streamoff` "types."

At the Boston meeting, P.J. Plauger raised concerns about the implementability of the proposed MSE addendum to ISO C. His concern for the iostreams proposal was that it entailed "getting ahead" of the C standard, and that the proposed iostreams interface might be the wrong one. After some discussion, the group decided to take a "wait and see" approach, and voted 7–1 to remove the wording regarding locales and shift states from the Input/Output proposal.

Jerry Schwarz dissented from this decision, but agreed to identify the wording changes necessary.

On Friday, the X3J16 and WG21 committees voted to accept the proposal with the editorial changes identified during the week's discussion.

17.4 ISO C Library

The group discussed a needed revision to disallow implementations to write `malloc()` and/or `calloc()` in terms of `operator new()`, and `free()` in terms of `operator delete()`. This is to avoid an infinite cycle if a C++ program implements `operator new()` in terms of `malloc()` and/or `calloc()`, and `operator delete()` in terms of `free()`.

The specific changes are:

§17.4.10.2: `malloc/calloc/relloc` shall not call `::operator new()`
change footnote 30
§7, page 17–6: The relationship ... is unspecified,* except that `calloc`, `malloc`, and `realloc` shall not call `::operator new()`, and `free` shall not call `::operator delete()`;
* For example, `::operator new()` might be written in terms of `malloc`. On the other hand, ...

The group did not discuss the proposed normative addendum to ISO C.

The group briefly discussed the `localedef` proposal, and decided to wait to see what WG14 votes on at their December meeting. It seems likely that a `localedef` class will become part of the standard C++ library.

17.5 Containers

The group discussed a number of details in the `bits` and `bit_string` classes. A key difference between the two classes is their bit-ordering conventions. The `bits` class uses right-to-left ordering, consistent with the integral types. The `bit_string` class uses left-to-right ordering, consistent with arrays and strings. The group decided against providing conversions between the two classes, since it is possible to iterate over them and set individual bits.

The group did not discuss the DynArray proposal.

3. Work Plan

The Library working group will continue to refine proposals for submission to the full X3J16/WG21 committee. By the next meeting, two more documents should present proposals in almost-final form.

17.2 String

Pete Becker agreed to revise the strings proposal, and to write up a proposal for class `text`.

The strings proposal should be ready for an X3J16/WG21 vote at the Portland meeting.

17.3 Input/Output

Jerry Schwarz agreed to revisit the proposal regarding locales and MSE shift states.

17.5 Containers

Chuck Allison agreed to revise the bitset proposal, and to prepare a brief description of the standard "boilerplate" for describing classes and functions in the standard library..

The bits proposal should be ready for an X3J16/WG21 vote at the Portland meeting.

Uwe Steinmüller agreed to revise the DynArray proposal.

A. Open Issues

This appendix lists the issues the Library working group is actively trying to resolve. It also lists some of the issues active in other working groups that are relevant to the Library portion of the Working Paper.

Other WG Issues

This section describes issues raised by the Library working group that should be addressed by other X3J16 working groups.

Editorial WG Issues

New expr.	Jul 92	92–0082/N0159	§5.3 should be revised to guarantee that any exception thrown by `operator new()` is propagated through the new expression.

Core Language WG Issues

Varargs	Nov 90	90–0109	What is the effect of passing a C++ object with a constructor to/from a C/C++ function with an ellipsis specifier as an argument list?
Temps' Lifetime	Nov 91	??	Lifetime of temporaries needs to be specified, to allow implicit conversion of `strings` to `char*`. [Proposal: 92–0020/N0098]
New expr.	Mar 92	??	What happens if an exception occurs in a constructor during dynamic allocation? e.g. `T* p = new T(args);` Does the memory remain allocated? Is the implementation required to call `operator delete()` before propagating the exception?

Templates	Jul 92	92–0082/N0159	How does one specify the definition for friend functions of templates that use only expression arguments?
			e.g.
			`template <int n> class bits {`
			`friend int f(bits<n>&);`
			`};`
			
			Is it:
			`template <int n>`
			`f(bits<n>&) { ... }`
			
			or:
			`template <class T>`
			`f(T&) { ... }`
			
			[see: 92–0014/N0092, §4.1.2]

Environment WG Issues

Startup	Mar 92	92–0042/N0019	Need to define the point at which library functions become available for use, including how the *implementation* uses them. For example, `operator new()` and `operator delete()` can be provided by the implementation (the default versions), or replaced by a C++ program. If the implementation uses these functions for its own storage management, it should use the program-supplied versions.
Static Init.	Mar 90	90–0052,0062	Library classes must take a consistent approach to providing static initialization (e.g. `cin`, `cout`, `cerr`). Currently, the order of initialization between thranslation units is undefined. Providing means for explicit initialization

			introduces additional mechanism, complexity, and performance overhead. [Proposal: 91–0143/N0076]
Mixed C/C++	Mar 90	90–0052,0062	Need to define the interaction of C and C++ features, including: I/O (`stdio` & `iostreams`) signals & exceptions `longjmp` & exceptions memory (`malloc`/`new`, `free`/`delete`) [Analysis: 91–0011]

Extensions WG Issues

Name Space	Mar 90	90–0052,0062	Need to make library facilities available, if possible without relying on preprocessor `#include` directives. Names introduced should not conflict with names introduced from other libraries. Rejected approaches include "funny" names (e.g. special prefixes or naming conventions), conditional compilation directives (`#ifdef`). Current approach relies on nested classes — not viable for templates. [Proposal: 92–0008/N0086] [see also: 91–0041]

General Library WG Issues

Conformance	Mar 90	90–0052,0062	Need to describe ways in which an implementation of the library can extend the classes as specified. For example, it should be possible to add private members. Other reasonable additions: defaulted extra arguments, private base classes. Questionable: using virtual derivation, making specified functions virtual if they are not specified that way.
Typedefs	Mar 90	90–0052,0062	Use of typedefs improve readibility (e.g. for *new-handler* function type), but intrude on programs' namespace. [But see 91–0047, p. 2]
Reentrant	Mar 90	90–0052,0062	Non-reentrant code hinders the ability to prove the standard library in multi-threaded environments. Should the library be *required* to be reentrant? Apparently breaks the C library. [see also: 92–0082/N0159]
Design	Nov 90	90–0109	Need to document Rationale for why the library is not a Smalltalk-like hierarchy. [see also 91–0020]
Spec.	Nov 91	91–0134/N0067	All functions and member functions specified in the C++ library should have exception-specifications to document what exceptions they might throw. Issue: since a conforming implementation might use dynamic storage to implement these functions, any function may throw an out-of-storage exception [see 92–0042/N0019].
Spec.	Nov 91	91–0134/N0067	The standard should be precise, using exact type definitions. Leaving "types" unspecified is too vague. [but see iostreams for backward compatibility]
Terms	Jul 92	92–0082/N0159	Need to define terms used in the standard, such as "reserved" and "region of storage," and "C-style struct." (?!)

17.1 Language Support Issues

Vote	Nov 92	92–0084/N0161	APPROVED

Design	Jul 92	92–0082/N0159	An allocation request of 0 is always supposed to return a "unique address." Should it simply be defined that `operator new(0) == operator new(1)` ?
Design	Jul 92	92–0082/N0159	Should the signature of the *new-handler* function be changed from `void (*)()` to `int (*)()`? This would break all existing implementations.

17.2 Strings Issues

Design	Nov 90	90–0109	Appropriate use of inheritance and/or templates in string class design. Simpler is better.
NLS	Nov 90	90–0109	Need to provide strings of National Language Set characters (i.e. `wchar_t`). [see also: 91–0027]
Locales	Mar 92	92–0042/N0119	Different kinds of comparisons for different kinds of strings: 1) fast, "raw" byte comparisons 2) execution character set collating order 3) fully locale-sensitive Need to consider locale-neutral operations and locale-sensitive operations separately (might be able to add one to the other through an appropriate use of inheritance).
NLS	Jul 92	92–0082/N0159	Should NULs (ASCII 0x00) be allowed anywhere in strings? Is the additional generality worth the complexity and performance degradation?
Locales	Jul 92	92–0082/N0159	Should strings depend on the global locale setting, or should they each have their own locale state?
Spec.	Jul 92	92–0082/N0159	Exceptions should be listed in *exception-specifications*, not comments. The "Exceptions" paragraph of each function should document the conditions under which the exception may be thrown, not simply list the exception name.

17.3 Input/Output Issues

Vote	Nov 92	92–0102/N0179	APPROVED
Design	Mar 92	92–0039/N0116	[see also: 92–0080/N0157] The treatment of `wchar_t` and multi-byte encodings are separate, but related, issues. The basic question is where to provide support. `wchar_t` at streams interface, multi-byte in external sources/sinks. Where to put encode/decode logic: streams or streambufs? Has implications for seeking, `streampos` type descriptions. Might just throw an exception on an attempt to seek to an illegal position.
Locale	Jul 92	92–0082/N0159	Should streams depend on the global locale setting, or should they each have their own locale state. Suggested resolution: have their own state default to "unset." When "unset," refer to global state, when "set," use own state.
Content	Nov 91	??	String stream

17.4 C Library Issues

Vote	Mar 92	92–0024R1/N0101	APPROVED
Content	Sep 91	91–0129/N0062	Request to review MSE addendum to ISO C. The addendum is available as 92–0087/N0087. [see also: 92–0035/N0112, 92–0036/N0113]

17.5 Containers Issues

Design	Jul 92	92–0082/N0159	Provide `operator[]` for dynarrays

B. Pending Items

This appendix lists issues which have been raised as possible work items for the Library working group. However, they are not being pursued at this time. The purpose of this appendix is to retain these items until they can be explicitly considered.

Content	Mar 90	90–0052	The standard C++ library should contain facilities for: inter-process communication network communication parallel tasks process table relational database access
Content	Mar 90	90–0062	The standard C++ library should contain facilities for: containers (lists, bags, sets, ...) memory managers (zones) garbage collection locale (time zone, currency, language, ...) date time currency task bit stream/bit map math (complex, vectors, matrices, infinite precision numbers, ...)
Content	Nov 91	91–0124/N0057	§17.1, Language Support, should provide default versions of array new and array delete functions. [Proposals: 92–0055/N0132]
Content	Nov 91	91–0134/N0067	The standard C++ library should contain facilities to replace the C `assert()` macro-based facility. It should be based on templates and exceptions. [Proposal: 92–0030/N0107]
Content	Nov 91	91–0134/N0067	The standard C++ library should contain facilities for: Common Language Independent Data Types NCEG numeric types
Content	Nov 91	91–0133/N0066	The standard C++ library should include facilities for concurrency, in the form of the extended language μC++. [Analysis: 91–0130/N0063]
Content	Mar 92	92–0042/N0019	The C library facilities (specifically, math, string, and MSE functions) should be made "more convenient" by using function overloadding. The standard C++ library should contain a `complex` type. The standard C++ library should contain template classes for sorting and searching.
Content	Mar 92	92–0042/N0019	The standard C++ library should contain a template function `renew`, to simulate the C library function `realloc` (plus invoke the appropriate constructors).
Content	Apr 92	92–0049/N0126	The standard C++ library should define a standard `localedef` format, based on the solution in P.J. Plauger's book, *The Standard C Library*.
Content	Jul 92	92–0076/N0153	The standard C++ library should contain mathematic array abstractions, to allow implementations to optimize for numeric-intensive environments.
Content	Aug 92	92–0074/N0151	§17.1, Language Support, should contain classes to support extended run-time type information classes: `Type_info, ExtTypeInfo, MemberInfo, DeclInfo, MemberIter, DeclIter`

C. Decisions

This appendix records the decisions and rationale for the decisions on which the Library working group has reached consensus. The decisions are listed by topic, indicating the meeting at which they were made and the document recording the decision.

General Library Decisions

Goals	Mar 90	90–0052,0062	The library portion of the standard will describe interfaces, not implementations. For example, the iostreams classes will document protected members, since they represent an interface to derived classes.
Criteria	Mar 90	90–0052,0062	Contents of the standard library should be based on existing practice, to the greatest extent feasible. There is a dilemma: addition of templates and exceptions significantly influences library design, and present C++ library practice does not use them.
Criteria	Nov 90	90–0109	Classes proposed for the standard library should have been implemented and used before accepted. How much use constitutes an acceptable level of "prior art" has not been defined.
Criteria	Mar 90	90–0052,0062	Contents of the standard C++ library will **not** provide "C++ bindings" to other standards. That is the responsibility of the respective standards involved.
Design	Mar 91	91–0047	The classes in the standard library should not be part of a singly-rooted class hierarchy in the Smalltalk tradition. Their design should emphasize static type checking and mechanisms other than inheritance (such as templates) as appropriate.
Design	Mar 92	92–0042/N0019	The container classes will emphasize CDT (concrete data type) designs, using templates as appropriate.
Design	Nov 91	91–0134/N0067	The standard library will include predefined exceptions. [see: 91–0116/N0049]
Formal Spec.	Mar 90	90–0052,0062	Elements of the standard library should use formal specification techniques where applicable. Decision (evolved over several proposals): use precise English to document function actions and post conditions. Use C++ (including *exception-specifications*) to specify the details of the interface. [see: 91–0036, 0038, 0046]
Names	Mar 91	91–00030,0047	Header "file" names will have **no** trailing suffix. The mapping of these names onto file names is, as ever, implementation-dependent.
Design	Mar 92	92–0042/N0019	C++ headers can `#include` others (necessary for iostreams). C headers will retain C library rules of not allowing such inclusion.
Names	Nov 92	92-0132/N0208	The naming convention for classes would be all lower-case, with no separators between distinct words (e.g. `fstreambuf`). The naming convention for members would be all lower-case, with underscores as separators between distinct words (e.g. `dynarray::get_at`).
Format	Nov 92	92-0132/N0208	Use a format similar to that of theC standard: **Synopsis** Provide a function prototype, including *exception-specification*, as applicable. **Description** Define pre-conditions, actions (including changes to object state), and side effects. Define post-conditions for normal return. **Returns** Describe the range of result values upon

normal return.
Exceptions Describe the post-condition failure that led to the exception, and the value(s) of the exception object thrown.

17.1 Language Support Decisions

Content	Nov 91	91–0134/N0067	The standard library will provide predefined exceptions, including a base class xmsg [see 91–0116/N0049].
Error Design	Nov 91	91–0134/N0067	The default *new-handler* should throw an out-of-memory exception.
Design	Jul 92	92–0082/N0159	[Issue Mar 92 92–0042/N0019]: Should the default behavior of `operator new()` be changed to always throw an exception if it cannot satisfy the memory request? Decision: allow returning null as an unspecified or undefined behavior, for backward compatibility.
Content	Jul 92	92–0082/N0159	The placement version of `operator new()` will be specified and reserved (i.e. not replaceable, but overloadable).

17.2 Strings Decisions

Design	Nov 91	91–0134/N0067	The design of string class(e) will emphasize: hiding details of storage management providing existing operations conveniently provide NLS support (`wchar_t`, locales).
Design	Mar 92	92–0042/N0019	List of operations supported: construction assignment concatenation insert search, replace select compare convert memory management (e.g. pre-reserve)
Design	Mar 92	92–0042/N0019	Until there is a resolution on lifetime of temporaries from the Core WG, the string class(es) will provide no implicit conversion to `char*`.
Design	Mar 92	92–0042/N0019	String operations need to be overloaded on `char*` (and `signed/unsigned char*`), to prevent ambiguities, avoid spurious temporaries, and prevent porting problems among incompatible implementations.
Design	Jul 92	92–0082/N0159	Split the string class into several, and define the conversions between them. Each string class will concentrate on just one "kind of" string: raw memory, `char` strings, locale-sensitive strings, and `wchar_t` strings.
Content	Jul 92	92–0082/N0159	A class `text` will provide a higher level of abstraction. It will use one of the more basic strings, selecting among them "as appropriate" to the locale (probably under explicit program control).

17.3 Input/Output Decisions

Design	Mar 90	90–0052,0062	The design for iostreams will use AT&T Release 2.0 as a base line, minus the use of multiple inheritance, plus the use of templates and exceptions (as appropriate). Classes included:

			ios, istream, ostream, strstream, fstream manipulators streambuf, strstreambuf, fstreambuf
Static Init	Nov 90	90–0109	Provide nested class `ios::init` for explicit initialization. Programs requiring iostream support in static consstructors can creat an instance of this class, and subsequently use the facilities of iostreams.
Error Design	Nov 90	90–0109	Provide both alternatives: stream state and exceptions. Default is to use stream state (for backward compatibility), explicit option to throw exceptions instead.
NLS	Nov 91	91–0134/N0067	Provide stream inserters and extractors for `wchar_t` types.
Error Design	Nov 91	91–0134/N0067	Streambufs will have their own exceptions, distinct from `ios::failure`. Streambufs will unconditionally throw exceptions to report errors, streams will have the (program-selectable) option of using error state or throwing/propagating exceptions.
Spec.	Mar 92	92–0042/N0019	Iostreams will specify "types" vaguely, to allow conforming implementations to use `int`s (existing practice) or classes.
Others	Jun 92	92–0059/N0136	pp. 7 & 8 list changes in Revision 3 and Revision 4.

17.4 C Library Decisions

Mixed C/C++	Mar 92	92–0042/N0019	Made the interaction of C and C++ features "undefined", including: signals & exceptions longjmp & exceptions memory (`malloc`/new, `free`/delete) Will address I/O (`stdio` & `iostreams`) in iostreams, by describing the interaction of `cin`/`stdin`, `cout`/`stdout`, `cerr`/`stderr`).
Inclusion	Nov 91	91–0134/N0067	Alternatives (90–0109): by copy or by reference Decision: include the relevant portions of the ISO C standard by reference. Document with the appropriate document number and revision identifier/date. Text in the C++ standard will define how the C++ version differs from the C version of the same facility.
Design	Nov 91	91–0134/N0067	Options identified since Mar 90: 1. include "as is" (current practice) 2. minimal repairs to ensure type safety 3. revise to use available C++ features (e.g. overloading) 4. complete rewrite/leave out for "more appropriate" C++ solutions Decision: (2) [See also 91–0047]
Names	Nov 90	90–0109	C library reserves names, C++ library reserves signatures.

17.5 Containers Decisions

Content	Jul 90	90–0062	The Library WG is not to consider additional classes until proposals for 17.1–17.4 have been completed.
Content	Nov 91	91–0134/N0067	The standard C++ library will contain some container classes. Initial volunteers identified for: bitset and array/vector. [see: 91–0111/N0044]
Design	Mar 92	92–0042/N0019	The container class `bits` will focus on the concrete type, not its use in an abstract set ADT. Work on a `set` class is deferred.