Document Number: X3J16/92-0082, WG21/N0159
Date: 9/20/92
Project: Programming Language C++
Ref Doc: X3J16/92-0042, WG21/N0119
Reply To: Michael J. Vilot
ObjectCraft, Inc.
Nashua NH 03063 USA
mjv@objects.mv.com

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

This report summarizes the progress made by the Library working group between the London and Toronto meetings, and the work done at the Toronto 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 propsal 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.

Due to the increasing size and activity of the C++ community, suggestions for the standard C++ library sometimes come from outside X3J16 circles. Some of these other activities are described here.

**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 (92–0043/N0120).

Jerry Schwarz provided an analysis of exceptions and operator new, including a summary of existing implementations (92–0058/N0135).

### 17.2 String

Uwe Steinmüller provided a revised proposal that included the results of discussions at the London meeting (92–0045/N0122).

### 17.3 Input/Output

Jerry Schwarz provided version 4 of the proposal, including initial support for `wchar_t` types (92–0059/N0136).

### 17.4 ISO C Library

Thomas Plum and P.J. Plauger proposed adopting a standard definition for `localedef` files, along the lines of the version described in Plauger's book *The Standard C Library* (92–0049/N0126).

Philippe Gautron submitted a proposal from AFNOR, requesting an official ISO group to synchronize the evolution of the C and C++ libraries (92–0069/N0146).

### 17.5 Containers

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

Use Steinmüller provided a revised dynarray (92–0046/N0123).

**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.

String class

Uwe Steinmüller posted his revised strings proposals, including code for an implementation, and received several comments. (`x3j16-lib-275, 276, 288, 291, 293, 294`)

Bits classes

Chuck Allison posted his revised `bits` and `bit_string` classes. (`x3j16-lib-277`)

operator new() behavior

Jerry Schwarz requested additional information to complete his survey of the behavior of existing implementations of `operator new()`, and received several replies. (`x3j16-lib-278, 279, 280, 281, 282, 283`)

thread safety

Aron Insinga raised the issue of whether static objects in the C++ library inhibited thread-safe implementations. (`x3j16-lib-284, 285, 286, 287`)

ISO C Library synchronization

Tom Plum posted Philippe Gautron's request to form an official ISO committee to synchronize the development of the C and C++ libraries. Jerry Schwarz replied. (`x3j16-lib-289, 290`)

Environment (argv/argc) access

Steven Parkes posted a request for a standard way to access the environment variables, including access from constructors of static objects. Steve Clamage replied with a good description of the issues that made it difficult or impossible to satisfy the request under the current language rules (due to indeterminate static initialization order). (`x3j16-lib-295, 296, 297`)

**Other Discussions**

P.J. Plauger presented a talk entitled "C and C++ Libraries" at the *C Plus C++ at Work* conference, Secaucus NJ, April 92. In it, he described his view of what the design criteria should be for the standard C++ library, as well as its contents.

The design criteria included:

- A library used by all C++ (and C) programmers should favor efficiency over doctrinal purity.
- It should encompass the functionality most likely to become common in C within the next few years.
- It should include a number of middle-level classes as a bridge between C and C++ programming styles.
- It should include just those high-level classes most likely to be widely used.
- There should be no surprising performance overheads.
- Most important, it should be tried out *before* it's been standardized.

The contents included:

- The Standard C library
- plus `float` and `long double` math functions and the WG14 additions for large character sets
- plus overloaded names for the math and (wide) character classification functions
- language-support functions
- stream classes, including multibyte support
- string, bitstring, and container classes
- a family of complex classes

An upcoming book from Prentice-Hall, due out in early 1993, will contain an implementation of this library.

## 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 adoption of the proposal to include the C library (92–0024R1/N0101) at the London meeting freed up some resources to devote to the other proposals.

Much of the group's discussion focused on the Language Support proposal. A separate sub-group discussed the strings proposal. The groups discussed National Language Set issues for both strings and iostreams, at both meetings.

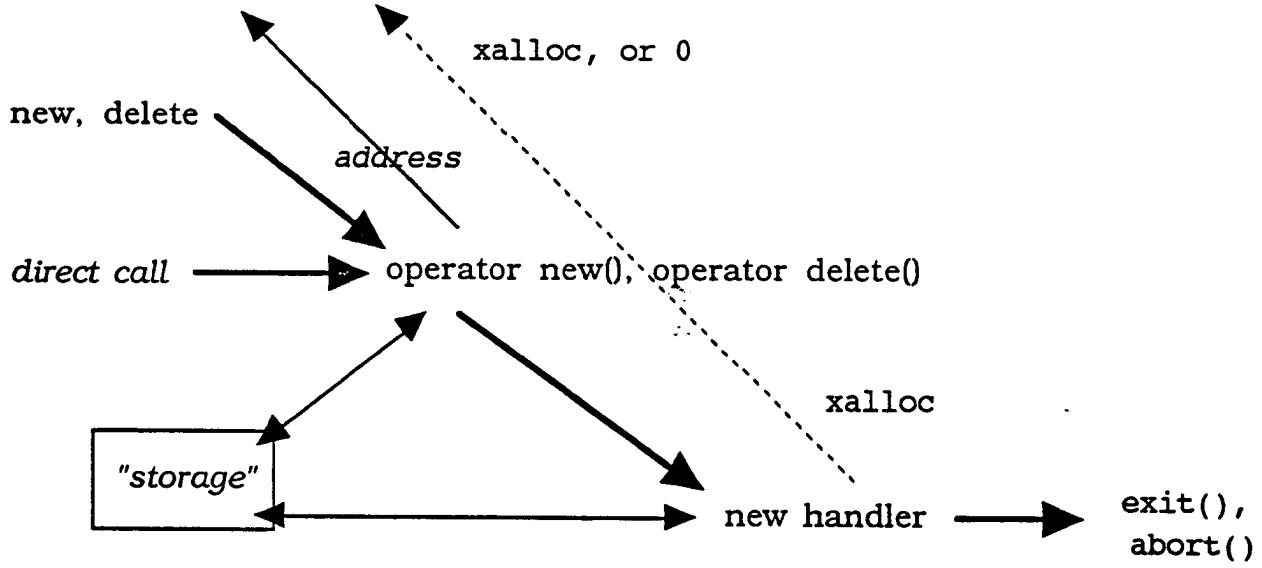
### 17.1 Language Support

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*.

P.J. Plauger and Dan Saks rewrote the proposal, and exposed a deeper conceptual issue. Their background with C encouraged them to use the word "object" for the result of `operator new()`, while C++ makes a sharp distinction between an "object" and its storage. This exposed the fact that the Working Paper has no vocabulary of concepts for describing storage and its management. This makes it difficult to express the intended operation and constraints on any program-supplied version of `operator new()` and `operator delete()`, and the intended side-effects of a *new-handler* function.

The following diagram illustrates this interaction, as well as the effect of having a *new-handler* throw an exception:




The figure helped clarify the discussion. The group decided not to pursue a rigorous definition of "storage," and turned to the implications of exceptions thrown by the default *new-handler*.

Jerry Schwarz discussed the issues in document 92–0058/N0135. The Library working group discussed his analysis, and decided to adopt his suggestion (2), leaving the possibility that a *new-expression* could return null as undefined behavior. Arkady Rabinov of Apple pointed out that this was crucial for supporting the transition from existing code to the proposed semantics.

In the discussion before the full X3J16 committee, Martin O'Riordan of Microsoft raised the most vocal criticism of the proposed change. The key point of his argument was an insistence upon standardizing existing practice.

Clearly, this change breaks all existing implementations. Since existing implementations do not support exceptions, they do not throw exceptions upon running out of storage. The decision involves a value judgement: is the existing practice of returning null to indicate failure worth replacing with exceptions, the language's own error reporting mechanism?

The following code examples helped clarify the discussion:

(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;
```

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

```
p = new T;
assert(p);
*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 question is whether there is more extant code of type (1) than of type (3). The former will be "fixed," in the sense that they will now terminate in a well-defined way rather than by their erroneous behavior. The latter will be "broken," in the sense that their checks will not catch the exceptions. The sense of the Library working group, voiced by Jerry Schwarz, is that there is far more code of type (1) than of any other type. This change would therefore be helping to expose latent defects, in much the same manner as type-safe linkage did.

Type (2) programs continue to work because their installed *new-handlers* do not throw exceptions, and type (4) programs are, we assume, not yet prevalent. But a secondary question is whether the style of type (4) programs is desirable or "in the spirit of C++." Jan Gray of Microsoft raised the point that defining the semantics to throw an exception means a C++ program must bear the costs of the exception mechanism even if it never uses exceptions itself. This is probably a specious argument, since all conforming implementations will have to provide the language feature anyway, but it does illustrate the concern.

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 undefined behavior for just such backward compatibility.

### 17.2 String

Much of the critique of this proposal centered on internationalization concerns. Greg Colvin wondered if the generality of allowing a NUL character (0x00) at arbitrary string locations was worth the added complexity and performance degradation. Jerry Schwarz wanted to remove the dependency on a single, global locale setting, in favor of per-string settings. These discussions revealed a number of design decisions that had to be made for string class design.

P.J. Plauger's suggestion for "a modest set of string classes" in his *C Plus C++ at Work* presentation provided a key insight for the Library working group. By organizing the analysis of string operations around the following table (originally sketched out at the March meeting in London), the group was able to see that the existing string class proposal was focusing on too many aspects of the problem simultaneously:

| representation | existing | class |
|---|---|---|
| individual bit | unsigned,<br>operators | bits,<br>bit_string |
| bytes<br>chars<br>(counted)<br>w/locales | mem*()<br>str*(),<br>strn*()<br>collate, xfrm | *string*<br>*string*<br>*string*<br>*string* |
| wchar_t | wstr*(),<br>wc*() | wstring |
| multi-byte | mb*() | (encode/decode) |

The group decided to revise the proposal, with several string classes (each focused on a specific aspect of representation) and well-defined conversions among them. Jerry Schwarz expressed his opinion that the key concern for each class was encapsulating the details of storage management.

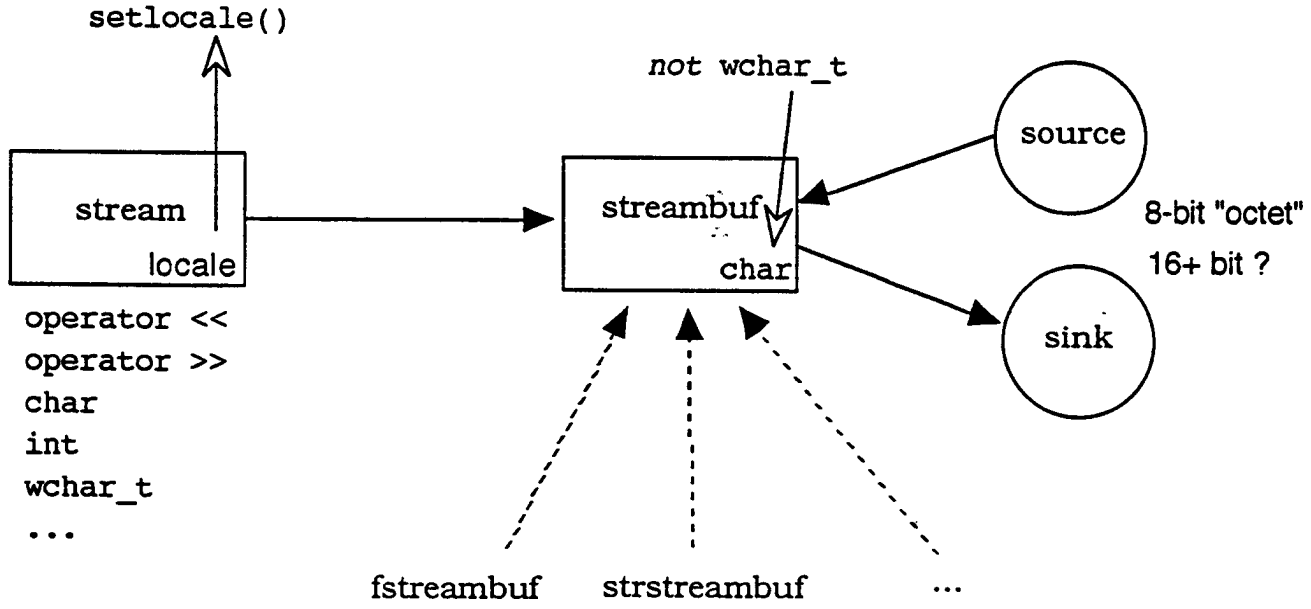
Tom Plum proposed having the string classes use the "most appropriate" representation for the locale. After discussing the idea, the group proposed a class `text` as a "higher level" string. The basic string classes would each concentrate on the most effective implementation of one kind of character sequence. The `text` objects could choose one of the other kinds of strings for their representation, based on the characteristics Tom had described.

### 17.3 Input/Output

Revision 4 of the iostreams proposal included most of the results of discussions at the London meeting. Most of the changes related to support for wide characters.

Documents 92–0039/N0116, from Tatsunori Hashimoto, and 92–0080/N0157, from Norihiro Kumagai, helped clearly illustrate the design impact of wide characters and their multibyte encodings. The Library working group was able to discuss the various proposed designs.

The following diagram helped illustrate the discussion:



There are really three separate issues:

1. How to support locales when formatting streams
2. How and where to encode/decode wide characters to/from multibyte representation
3. Whether/how to support "direct mode" transfers between internal wide characters and external wide (i.e. non-multibyte) representations.

The goal is to provide facilities to make NLS streams as convenient as ASCII streams:

```
istream in(" ... ");
string s;
in >> s;     // current ASCII behavior (will be locale-sensitive)

wstring w;
in >> w;          // do the right thing (decode)

istream wide(" ... ");  // might have 64-bit characters
wide >> w;              // "direct mode"
```

Jerry Schwarz reiterated the design concepts behind iostreams. The stream classes are responsible for formatting, while streambuf classes manage consumption, buffering, and production of (char) character buffers.

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."

Simply throwing an exception for an invalid positioning request might be the simplest alternative.

Supporting "direct mode" transfers (wchar_t <—> non-MSE external) would not be supported by char-based streambufs. It may be possible to define a wchar_t-based streambuf to support such a mode.

### 17.4 ISO C Library

The AFNOR request to create a group to synchronize the evolution of the C and C++ libraries was discussed. The group's consensus was that the C++ library will track the C library anyway, and the group did not have the resources to spare on yet another committee.

P.J. Plauger requested the Library working group review the proposed normative addendum to ISO C. The "penultimate" proposal would be available from the Japanese delegation to WG14 soon after WG21's Toronto meeting. Comments at the WG21 Boston meeting could be relayed to WG14's December meeting. Library working group members agreed to review the proposal for impact on the C++ library.

The `localedef` proposal was not discussed.

### 17.5 Containers

Reviewing the bitset proposal encountered a Core Language issue regarding friends of template classes.

```
template<unsigned n>
class bits {
  ...
  friend int operator==(const bits<n>&, const bits<n>&);
};

// can't define a template function using expression n
template <unsigned n>
int operator==(const bits<n>&, const bits<n>&);
```

```
// the legal alternative intrudes on the global name space
template<class T>
int operator==(const T&, const T&);
// and still requires a definition for each instantiation of bits
```

Basically, this is the same issue Philippe Gautron discussed in London (see: 92–0014/N0092, §4.1.2).

The dynarray proposal received some negative comments. The most serious was that the functionality made little sense without an `operator[]` defined. Simply returning a reference to an element does not distinguish between lvalue and rvalue uses, while providing an intermediate "helper" class seems inefficient. Jerry Schwarz pointed out that standard classes might be optimized as a special case by an implementation.

The group did not reach a clear consensus on whether the proposal should be tabled or continued, but it was clear that it needed more work.

## 3. Work Plan

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

### 17 Library Introduction

Mike Vilot agreed to develop the wording for the introduction to the Library chapter, and to develop a Rationale statement for the library.

### 17.1 Language Support

Mike Vilot agreed to revise the proposal.

### 17.2 String

Uwe Steinmüller agreed to revise the strings proposal.

Pete Becker agreed to write up a proposal for class `text`.

### 17.3 Input/Output

Jerry Schwarz agreed to revise the proposal.

### 17.4 ISO C Library

All working group members agreed to review the MSE proposal, per P.J. Plauger's request.

### 17.5 Containers

Chuck Allison agreed to revise the bitset 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*`.<br>[Proposal: 92–0020/N0098] |
| New expr. | Mar 92 | ?? | What happens if an exception occurs in a constructor during dynamic allocation?<br>e.g. `T* p = new T(args);`<br>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?<br>e.g.<br>`template <int n> class bits {`<br>`friend int f(bits<n>&);`<br>`};`<br><br>Is it:<br>`template <int n>`<br>`f(bits<n>&) { ... }`<br><br>or:<br>`template <class T>`<br>`f(T&) { ... }`<br><br>[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.<br>[Proposal: 91–0143/N0076] |
| Mixed C/C++ | Mar 90 | 90–0052,0062 | Need to define the interaction of C and C++ features, including:<br>I/O (`stdio` & `iostreams`)<br>signals & exceptions<br>`longjmp` & exceptions<br>memory (`malloc`/`new`, `free`/`delete`)<br>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.<br>[Proposal: 92–0008/N0086]<br>[see also: 91–0041] |
| wchar_t | Mar 92 | 92–0047/N0124 | The type `wchar_t` should be made a distinct type for the pruposes of overloading. C only requires that it be one of the existing integral types, which is not sufficiently portable. for C++<br>Proposal: 92–0047/N0124 |
| enums | Nov 91 | 91–0134/N0067 | Allw overloading/operations on enumerations as distinct types. Currently, and operation on enums reverts to `int`. Checking has to be done dynamically. Full classes are seen as too inefficient and/or layout incompatible with `int`s.<br>Proposal: 91–0139/N0072<br>see also: 92–0070/N0070 |

**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.<br>[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.<br>[see also: 92–0082/N0159] |
| Design | Nov 90 | 90–0109 | Need to document Rationale for why the library is not a Smalltalk-like hierarchy.<br>[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<br>[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

| | | | |
|---|---|---|---|
| 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 `new(0) == 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`).<br>[see also: 91–0027] |
| Locales | Mar 92 | 92–0042/N0119 | Different kinds of comparisons for different kinds of strings:<br>1) fast, "raw" byte comparisons<br>2) execution character set collating order<br>3) fully locale-sensitive<br>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

| | | | |
|---|---|---|---|
| Issues List | Mar 91 | 91–0028 | UNIX-specific items |
| Issues List | Nov 91 | 91–0134/N0067 | Issues identified:<br>1) national language set character data streams<br>2) file open modes, newline translation, etc.<br>3) wide character support<br>4) interaction with other standards (e.g. ASN.1)<br>5) names of headers (existing C++)<br>6) streampos, streamoff "types"<br>7) exceptions thrown by streambuf<br>8) mode flag "types" (enums and or-ing)<br>9) name space<br>10) I/O support for strings, wstrings<br>11) stdio/streams interaction |
| Content | Nov 91 | ?? | String stream |

| | | | |
|---|---|---|---|
| Design | Mar 92 | 92–0039/N0116 | [see also: 92–0080/N0157]<br>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?<br>Has implications for seeking, `streampos` type descriptions. Might just throw an exception on an attempt to seek to an illegal position. |
| Design | Jul 92 | ?? | Copy constructors |
| Spec. | Jul 92 | ?? | Use of *exception-specifications*. |
| Locale | Jul 92 | 92–0082/N0159 | Should streams depend on the global locale setting, or should they each have their own locale state.<br>Suggested resolution: have their own state default to "unset." When "unset," refer to global state, when "set," use own state. |

### 17.4 C Library Issues

| | | | |
|---|---|---|---|
| Final | Mar 92 | 92–0024R1/N0101 | Vote: APPROVED |
| Content | Sep 91 | 91–0129/N0062 | Request to review MSE addendum to ISO C.<br>The addendum is available as 92–0087/N0087.<br>[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:<br>inter-process communication<br>network communication<br>parallel tasks<br>process table<br>relational database access |
| Content | Mar 90 | 90–0062 | The standard C++ library should contain facilities for:<br>containers (lists, bags, sets, ...)<br>memory managers (zones)<br>garbage collection<br>locale (time zone, currency, language, ...)<br>date<br>time<br>currency<br>task<br>bit stream/bit map<br>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.<br>[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.<br>[Proposal: 92–0030/N0107] |
| Content | Nov 91 | 91–0134/N0067 | The standard C++ library should contain facilities for:<br>Common Language Independent Data Types<br>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++.<br>[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.<br>The standard C++ library should contain a `complex` type.<br>The standard C++ library should containe 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:<br>`class Type_info;`<br>`class ExtTypeInfo;`<br>`class MemberInfo;`<br>`class DeclInfo;` |

```
class MemberIter;
class 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 reatest 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. |

### 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:<br>hiding details of storage management<br>providing existing operations conveniently<br>provide NLS support (`wchar_t`) |
| Design | Mar 92 | 92–0042/N0019 | List of operations supported:<br>construction<br>assignment<br>concatenation<br>insert<br>search, replace<br>select<br>compare<br>convert<br>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).<br><br>Classes included:<br>ios, istream, ostream, strstream, fstream<br>manipulators<br>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 | Provde 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:<br>signals & exceptions<br>longjmp & exceptions<br>memory (`malloc`/`new`, `free`/`delete`)<br>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<br>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:<br>1. include "as is" (current practice)<br>2. minimal repairs to ensure type safety<br>3. revise to use available C++ features (e.g. overloading)<br>4. complete rewrite/leave out for "more appropriate" C++ solutions<br>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. |