Doc. No.: X3J16/96-0202R1
WG21/N1020R1
Date: November 12, 1996
Project: Programming Language C++
Reply to: Beman Dawes
beman@dawes.win.net

Kona Motions for Clauses 17 through 21

Revision 1 reflects motions as actually passed in Kona.

1) Motion (to resolve several issues from the Clause 21 (Strings) issues list - version 20):

Move we:

-- close issues 21-090, 095, and 111 from N1006 = 96-0188 without taking any action.

-- amend the WP as described in N1006 = 96-0188 by adopting the proposed resolution for issues 21-113, 115, and 116.

-- amend the WP and close issue 21-114 in N1006 = 96-0188 by replacing in 21.3.7.9 [lib.string.io] operator<< effects:

Behaves as if the function calls:

os.write(str.data(), str.size())

by:

Behaves as if the following is executed:

```
for(str::iterator i = str.begin(); i != str.end(); i++) {
      os.putc(*i);
}
```

2) Motion (to resolve an issue from the Clause 19 (Diagnostics) issues list - version 3):

Move we close issue 19-002 as described in N1018 = 96-0200 by amending the WP as follows:

Change Postcondition sections in 19.1.1 [lib.logic.error], 19.1.2 [lib.domain.error], 19.1.3 [lib.invalid.argument], 19.1.4 [lib.length.error], 19.1.5 [lib.out.of.range], 19.1.6 [lib.runtime.error], 19.1.7 [lib.range.error], 19.1.8 [lib.overflow.error], 19.1.9 [lib.underflow.error]

from: Postcondition: what() == what_arg.data()
to: Postcondition: strcmp(what(), what_arg.c_str()) == 0

3) Motion (to resolve several issues from the Clause 20 (Utilities) issues list - version 6):

Move we:

-- close issue 20-039 from N1000 = 96-0182 by amending the WP EqualityComparable requirements table in 20.1.1 [lib.equalitycomparable] from:

== is an equivalence relationship.

To:

== is an equivalence relationship, that is, it satisfies the following properties:

- For all a, a == a.
- If a == b, then b == a.
- If a == b and b == c, then a == c.

And changing "a and b" to "a, b, and c" in paragraph 1 of the same section.

-- close issues 20-042 and 043 from N1000 = 96-0182 by amending the WP with the following changes to 20.4.5.1 [lib.auto.ptr.cons]:

- Delete paragraph 1.
- Change paragraph 3 to:
  Requires: Y* can be implicitly converted to X*.
- Change paragraphs 6 and 7 to:
  Requires: Y* can be implicitly converted to X*. The expression delete get() is well formed.

  Effects: If *this is the same object as a there are no effects. Otherwise, call a.release(), and if *this owns *get()then delete get().
- Change paragraph 9 to:
  Postconditions: If *this is not the same object as a then *this holds the pointer returned from a.release(). *this owns *get() if and only if, as a precondition, a owns *a.
- Add a requires clause to the destructor:
  Requires: The expression delete get() is well formed.

-- close issue 20-044 from N1000 = 96-0182 by amending the WP as follows:

Section 20.4.1
Add:

```
allocator(const allocator<T>&) throw();
allocator& operator=(const allocator<T>&) throw();
```

Section 20.4.1.3
Add:

```
shared_allocator(const shared_allocator<T>&) throw();
shared_allocator<T>& operator=(const shared_allocator<T>&)
throw();
```

Section 20.4.5
Add:

```
auto_ptr(const auto_ptr<X>&) throw();
auto_ptr<X>& operator=(const auto_ptr<X>&) throw();
```

Section 20.4.5.1
Add the same two prototype as above.

Section 26.2.2
Add:

```
      complex(const complex<T>&)
      complex<T>& operator=(const complex<T>&);

Section 26.2.3
Add:
      In complex<float> declaration, add:
      complex<float>& operator=(const complex<float>&);

      In complex<double> declaration, add:
      complex<double>& operator=(const complex<double>&);

      In complex<long double> declaration, add:
      complex<long double>& operator=(const complex<long
double>&);
```

4) Motion (to resolve several issues from the Clause 18 (Language Support) issues list - version 6):

Move we:

-- close issues 18-031, 18-032 from N1017 = 96-0199 without taking any action.

-- close issue 18-030 from N1017 = 96-0199 by amending the WP:

- 17.3.1.1 paragraph 2 replace:

  All library entities shall be defined within the namespace std.

  with:

  All library entities except macros, operator new, and operator delete are defined within the namespace std or namespaces nested within namespace std.

- 18.4, 18.4.1.1 through 18.4.1.3 change "size_t" to "std::size_t".

-- close issue 18-033 from N1017 = 96-0199 by amending the WP as follows:

18.6.2.2 Type unexpected_handler [lib.unexpected.handler] change first bullet in `Required behavior' to:

--throw an exception that satisfies the exception specification (however, if the call to unexpected() is from the program rather than:
from the implementation, any exception may be thrown);

18.6.2.4 unexpected [lib.unexpected] replace existing section with:

```
void unexpected();
```

Called by the implementation when a function exits via an exception not allowed by its exception-specification (_except.unexpected_). May also be called directly by the program.

Effects: Calls the unexpected_handler function in effect immediately after evaluating the throw-expression (_lib.unexpected.handler_), if called by the implementation, or calls the current unexpected_handler function, if called by the program.

18.6.3.3 terminate [lib.terminate] replace existing section with:

void terminate();

Called by the implementation when exception handling must be abandoned for any of several reasons (_except.terminate_). May also be called directly by the program.

Effects: Calls the terminate_handler function in effect immediately after evaluating the throw-expression (_lib.terminate.handler_), if called by the implementation, or calls the current terminate_handler function, if called by the program.

5) Motion 5 was withdrawn.

6) Motion (to clarify namespace std usage):

Move we amend the WP by changing the first sentence of section 17.3.3.1 [lib.reserved.names] from:

It is undefined for a C++ program to add declarations or definitions to namespace std unless otherwise specified.

To:

It is undefined for a C++ program to add declarations or definitions to namespace std or namespaces within namespace std unless otherwise specified.

7) Motion (Compromise on Library template default arguments [T Plum]):

Move we amend the WP by adding the following wording to clause 17:

Throughout the C++ Library clauses (17 through 27), whenever a template member function is declared with one or more default arguments, this is to be understood as specifying a set of two or more overloaded template member functions. The version with the most parameters defines the interface; the versions with fewer parameters are to be understood as functions with fewer parameters, in which the corresponding default argument is substituted in-place.

[Example from _lib.set.cons_ 23.3.3.1

```
explicit set(const Compare& comp = Compare(),
                 const Allocator& = Allocator());
```

This declaration is to be understood as a shorthand for the following three declarations:

```
explicit set(const Compare& comp, const Allocator& );
explicit set(const Compare& comp);
explicit set();
```

In the second and third declarations, the default values Allocator() and Compare() are used in place of the missing explicit function parameters.]