Doc Number:	X3J16/94-0110 WG21/N0497
Date:	March 30, 1994
Project:	Programming Language C++

Summary of Revised Clause 4 [conv]: Standard Conversions

Samuel C. Kendall
Sun Microsystems Laboratories, Inc.
sam.kendall@east.sun.com

This document is part of the work of Tom Plum's subgroup of the Core WG. Mike Cote and I had hoped to have a revised clause [conv] ready for the mailing. Unfortunately that document is not complete. Instead, this document summarizes the conversions without the semantics or the standardese.

Table 1 lists the conversions in their revised categories. Note that the conversions are not just from type to type, but

Table 1: Conversions

From	To	Notes
Lvalue Conversions		
lvalue "T"	rvalue "T"	
lvalue "array of T"	rvalue "pointer to T"	
rvalue "array of T"	rvalue "pointer to T"	[a]
lvalue "F"	rvalue "pointer to F"	F is a function type.[b]
Rvalue Conversions		
rvalue "T"	lvalue "T"	Except in the context of intializing an implied object parameter, T must be const.
Qualification Conversions		
lvalue "T1"	lvalue "T2"	T2 is T1 with cv-qualifiers added according to those complicated rules (not written down here).
rvalue "T1"	rvalue "T2"	
Promotions		
cv char[c] *cv* signed char *cv* unsigned char *cv* short *cv* unsigned short	*Implementation-defined, one of:* *cv* int *cv* unsigned int	
cv wchar_t *cv* enum E *(any enumeration type)*	*Implementation-defined, one of:* *cv* int *cv* unsigned int *cv* long *cv* unsigned long	

Table 1: Conversions (Continued)

From	To	Notes
cv bool	*cv* int	
cv float	*cv* double	
Standard[d] Conversions		
cv int *cv* unsigned int *cv* long *cv* unsigned long *cv* double *cv* long double	*cv* A	A is any arithmetic type (including bool). Exclude conversions from a type to itself.
"*cv* pointer to T" "*cv* pointer to member of class C of type T"	*cv* bool	
Constant 0 of type: *cv* int *cv* unsigned int *cv* long *cv* unsigned long	*cv* pointer to T "*cv* pointer to member of class C of type T"	
cv pointer to F	*Implementation-defined, one of:* "*cv* pointer to void" *no conversion*	F is a function type.
lvalue "*cv* D"	lvalue "*cv* B"	B is an unambiguous, accessible base class of D
rvalue "*cv* D"	rvalue "*cv* B"	B is an unambiguous, accessible base class of D
cv1 pointer to *cv2* D	*cv1* pointer to *cv2* B	B is an unambiguous, accessible base class of D.
cv1 pointer to *cv2* T	*cv1* pointer to *cv2* void	
cv pointer to member of class B of type T	*cv* pointer to member of class D of type T	B is an unambiguous, accessible base class of D.
Ellipsis Conversions		
rvalue "T"	. . .	T cannot be a function or array type.

a. It is odd and perhaps unfortunate that an rvalue array implicitly converts to a pointer, but that's the way the language is now. For example:

```
// ary-rv1.C, fnc-rv1.C
struct A { int a[10]; };
A f();
int* ip = f().a;                    // ok, rvalue int [10] --> rvalue int*
int (*ap)[10] = &f().a;             // ill-formed, explicit address of rvalue
```

The ARM, cfront 3.0, and Microsoft C++ agree.

b. Only lvalue functions (that is, only non-member functions) can convert to pointer type.

c. Many rules have seemingly gratuitous “*cv*”s tacked on to the From item. It is there so that cv-qualified rvalues behave as close to their nonqualified fellows as possible. This is vital. For example, if a “const char” could not be promoted, then the second initialization below would be ill-formed:

```
const char SPACE = ' ';
int i = SPACE;
```

For most of these conversions, I have put a matching “*cv*” onto the To item; I don’t know whether it should be there. Does a “const char” promote to a “const int” or an “int”? It doesn’t matter much, but it seemed simpler to put it there, for two reasons: first, to prevent weird loopholes from appearing that allow cv-qualifiers to be dropped; and second, because Tom WIlcox’s revised [over.ics.scs] says “standard conversions do not change the ... cv-qualification of the type.”

d. In the absence of a better name I have called this category “standard conversions”. However, the whole clause is titled “Standard Conversions”, so either this subset of standard conversions, or the clause itself, should get a different name. I recommend the clause be renamed “Implicit Conversions”.

also involve lvalue-ness and constant-ness. This has always been the case; I am just making it more explicit.

Each table entry is an rvalue unless otherwise noted (sometimes, for clarity, we redundantly note that an entry is an rvalue).

Unless noted, each From item can be a constant or a nonconstant (it doesn’t matter). All conversions preserve constant-ness; that is, if the From item is a constant, so is the To item.

We need a better word than “item” to refer to the endpoints of a conversion.

This revised clause omits two items of information in the present version of [conv]. First, [conv.arith], Arithmetic conversions. Second, under [conv.ptr] paragraph 1, the conversion of a constant 0 to a pointer when used with various operators. Both of these items belong in clause 5 [expr]. The new [conv] is about implicit conversions in the context of initialization only