Document:	WG14 N1180
Subject:	TR 18037 defect log
Source:	TR 18037 editor
Date:	March 2005

Defect 1:
Problem: Using the suffix 'k' to stand for 'accum' with the base 'strto' for numeric conversion functions yields a conflict with the existing C function 'strtok'.

Solution: Rename the fixed-point numeric conversion functions to have a base of 'strtofx' instead of 'strto'. The new names will be 'strtofxhr', 'strtofxr', 'strtofxlr', 'strtofxhk', 'strtofxk', 'strtofxlk', 'strtofxuhr', 'strtofxur', 'strtofxulr', 'strtofxuhk', 'strtofxuk', and 'strtofxulk'.

Affected sections in TR18037: 4.1.7 (2 times), 4.1.9, replacement text for 7.18a.6.8 (many times), replacement text for 7.19.6.2 para 12.

Defect 2:
Problem: TR 18037 requires that overflow handling be done before rounding, which causes problems for alternative overflow modes such as modular wraparound. (When the overflow mode is saturation, the order in which rounding and overflow handling are performed has no effect on the end-result.)

Solution: Specify that rounding should be done before overflow handling.

Changes:

- Clause 4.1.3, para 3: strike '(after any overflow handling)'.

Replacement text for 5.2.4.2.3:

- Replace in para 6 (staring with 'If the result type of an arithmetic operation') the text 'and then overflow handling and rounding' by 'and then rounding and overflow handling'
- Strike from para 9 (starting with 'If (after any overflow handling)') the text '(after any overflow handling)'

Defect 3:
Problem: 4.1.6.2.1 (Binary arithmetic operations), last para, the text on the divi functions has 'yielding a fixed-point type result' while the divi functions have an integer result.

Solution: Change 'yielding a fixed-point type result'; this must be 'yielding an integer type result'

Change: Modify 4.1.6.2.1 para 5 accordingly.

Defect 4:
Problem: The type-generic macro definition sections (4.1.7.6 and 7.18a.6.7) are incomplete and possibly wrong (see N1071 for more information)

Solution: The generic function names should be 'absfx', 'countlsfx' and 'roundfx'; 7.18a.6.7 should better explain which underlying functions are selected.

Changes: in the replacement text for 7.18a.6.7:

- Change the type font for 'fx' from bold italic to bold (3 times)
- Add to 7.18a.6.7:
  For each macro, use of the macro invokes the function whose corresponding real type and type domain is the real type of the first generic argument. If the real type of the first generic argument is not a fixed-point type, the behavior is undefined.

Defect 5:
Problem: The replacement text for 6.2.6.3 para 3, last sentence says 'integer types' while it should say 'fixed-point types' (twice).

Solution: Make the change

Change: Replace the last sentence of the replacement text for 6.2.6.3 para 3 with: The width of a fixed-point type is the same but including any sign bit; thus for unsigned fixed-point types the two values are the same, while for signed fixed-point types the width is one greater than the precision.

Defect 6:
Problem: The replacement text for 7.18a.6.1 (on fixed-point arithmetic support functions) does not specify what happens if an integer result overflows.

Solution: Undefined behavior is implied by default in the C standard. Mention in the descriptive text that this should result in undefined behavior.

Change: In 4.1.6.2.1 para 5, add the following sentence to the end of the paragraph: If an integer result of one of these functions overflows, the behavior is undefined.

Defect 7:
Problem: The description of the fixed-point rounding functions in the replacement text for 7.18a.6.3 require that fractional bits beyond the rounding point are set to zero in the result. This should not apply when saturation has occurred.

Solution: Replace the text as proposed.

Change: Modify the third sentence of the description of 7.18a.6.3 to read: 'When saturation has not occurred, fractional bits beyond the rounding point are set to zero in the result.'

Defect 8:
Problem: Consider

```
// file 1
register REG_A int reg_a;

// file 2
extern int reg_a;
int main() { return reg_a; }
```

According to the second new constraints for 6.7.1 this is not allowed:

> If an object is declared with a named-register storage-class specifier, every declaration of that object shall include the same named-register storage-class specifier.

The 'shall' implies that a diagnostic is required here. However, so far C compilers have not been required to diagnose such issues across translation units. Is this really the intention?

Solution: The intent is to require a diagnostic for different named-register storage-class specifier declarations within a single translation unit for the same object.

Changes:

- Change in the second of the new constraint paragraphs for 6.7.1 the words 'every declaration of that object' to 'every declaration of that object within the same translation unit'.
- In the new text for 6.7.1.1, add at the beginning of the last paragraph (paragraph 6) the sentence: 'If an object is declared with a named-register storage-class specifier, every declaration of that object shall include the same named-register storage-class specifier.'

Defect 9:
Problem: If _X and _Y are address spaces with _Y enclosing _X, after the declarations

```
_X char a;
_Y char *p = &a;
```

the dereference `*p` is undefined because of the way TR 18037 applies the notion of effective type (C Standard 6.5, paragraphs 6 and 7). This makes overlapping named address spaces unusable by strictly conforming code in most circumstances.

Solution: In the detailed changes to the C Standard, modify the definition of *effective type* (clause 6.5, paragraph 6) to exclude address space qualifiers, and restore the rules in paragraph 7 to their original form. (Note that the whole concept of *effective type* is used only in 6.5 and in one footnote elsewhere in the C Standard.) This makes TR 18037's definition of *additionally access-qualified version* of a type (in the new text for 6.2.5) unnecessary.

Changes: Change the replacement text for clause 6.5 of the C Standard to the following:

> **Clause 6.5 - Expressions**, replace the first two sentences of paragraph 6 with:
>
> The *effective type* of an object for an access to its stored value is the declared type of the object, if any, without any address-space qualifier that the declared type may have.[72)] If a value is stored into an object having no declared type through an lvalue having a type

that is not a character type, then the type of the lvalue, without any address-space qualifier, becomes the effective type of the object for that access and for subsequent accesses that do not modify the stored value.

Remove the sentence defining *additionally access-qualified version* from TR 18037 (first paragraph replacing paragraph 26 of 6.2.5).

Defect 10:
After close reading during the meeting it appeared that Issue 10 of N1071 was not a defect. The number is maintained for consistency with N1071.

Defect 11:
Problem: The current specification allows global named-registers to be initialized. It is however unclear when, and by whom this initialization should be done (one could imagine that the register storage onto which the variable maps does not really exist until some device is initialized by some user code).

Solution: Disallow initializers on named-register variables.

Change: Add the following new constraint to section 6.7: 'A declaration containing a named-register storage-class specifier shall not contain an initializer.'

Defect 12:
Problem: The new text for 6.7.2.1, requires that a specifier-qualifier-list in the declaration of a member of a structure or union shall not include an address space qualifier. This disallows for instance the type of a member of structure to be a pointer into a named address space.

Solution: The intention was to disallow members of a single structure/union to have different address qualifiers.

Change: Modify the constraint for 6.7.2.1 to be: 'Within a structure or union specifier, the type of a member shall not be qualified by an address space qualifier.'

Defect 13:
Problem: TR 18037 does not alter the definition of integer constant expression in para 6 of 6.6.

Solution: This is an oversight which, for consistency reasons, should be corrected.

Change: Add new replacement text for 6.6 to change the first sentence of para 6 as follows: insert before 'and floating' the text 'fixed-point constants that are the immediate operand of casts'

Defect 14:
Problem: The new text for 6.5.8 (relational operators) and 6.5.9 (equality operators) add as a constraint: 'If the two operands are pointers into different address spaces, the address spaces must overlap.'. Such a constraint is missing for 6.5.6 (additive operators), where it is relevant for pointer subtraction.

Solution: Add the requested constraint.

Change: Add the following constraint to 6.5.6 : 'For subtraction, if the two operands are pointers into different address spaces, the address spaces must overlap.'

Defect 15:
Problem: The third sentence of para 5 of 4.1.6.2.1 start with 'The generic function names ...'; the word 'generic' might cause confusion with 'type-generic'.

Solution: Change 'generic function names' to just 'names'.

Change: Modify 4.1.6.2.1 para 5 third sentence accordingly.

Defect 16:
Problem: The first sentence of 4.1.6.2.1 para 5 (' According to the rules above, the result type of an arithmetic operation where (at least) one operand has a fixed-point type is always a fixed-point type.') is wrong as it does not take operands with floating-point type into account.

Solution: As it is the intention to only discuss integer types and fixed-point types in this paragraph, change the sentence accordingly.

Change: Modify the first sentence of 4.1.6.2.1 para 5 to read: 'According to the rules above, the result type of an arithmetic operation where one operand has a fixed-point type and the other operand has an integer or a fixed-point type is always a fixed-point type.'

Defect 17:
Problem: TR 18037 has in many places the text fragment 'overflow and rounding', but has also the text 'rounding and overflow'.

Solution: Defect 2 has established that the required order is first to do rounding and then to do overflow handling; make the text consistent by replacing the text fragment 'overflow and rounding' by 'rounding and overflow' throughout the document.

Change: Make the required change in many places (including the title of 4.1.3 and A.4).

Defect 18:
Problem: In 6.5, the first sentence of the replacement text for the description section of 7.8a.4.6 lists a number of functions. The second sentence of the same paragraph has a similar list of functions that should be in the same order as in the first sentence, but which is not.

Solution: Reorder the list in the second sentence.

Change: Change in 6.5 (detailed changes for the Basic I/O Hardware Addressing) the second sentence of the description section in 7.8a.4.6 to start with: ' The functions are equivalent to ioand, ioor, ioxor, ioandl, ioorl, and ioxorl, respectively, ...'.

Defect 19:
Problem: Sec 7.18a.2 introduces a set of typedefs, and describes a convention for the return type of bits'*fx*': either int_*fx*_t, or uint_*fx*_t.

Sec 7.18a.6.5 lists the 12 functions for bits'*fx*', all of which make use of the form 'int_*fx*_t'. According to 7.18a.2 the last six should be of the form 'uint_*fx*_t'.

Solution: change the last 6 function prototypes in the Synopsis of 7.18a.6.5 to:

```
uint_uhr_t bitsuhr(unsigned short fract f);
uint_ur_t bitsur(unsigned fract f);
uint_ulr_t bitsulr(unsigned long fract f);
uint_uhk_t bitsuhk(unsigned short accum f);
uint_uk_t bitsuk(unsigned accum f);
uint_ulk_t bitsulk(unsigned long accum f);
```

Defect 20:
Problem: the text on the countls function in 4.1.7.3 and 7.18a.6.4 reads:

> The integer return value of the above functions is defined as follows:
> -if the value of the fixed-point argument f is non-zero, the return value is the largest integer k for which the expression f<<k does not overflow;
> -if the value of the fixed-point argument is zero, an integer value is returned that is at least as large as N-1, where N is the total number of (nonpadding) bits of the fixed-point type of the argument.
>
> Note: if the value of the fixed-point argument is zero, the recommended return value is exactly N-1.

In the 'argument is zero' case, for a signed fixed-point type, the notion 'nonpadding bits' includes the sign bit (see 6.2.6.3); this implies that the N for signed types is one larger that the N for the corresponding unsigned types; this is wrong (it suggests that shifting into the sign bit does not generate an overflow). In stead of '(nonpadding) bits', the notion 'value bits' should be used.

Solution: in 4.1.7.3 and 7.18a.6.4, replace in the 2nd bullet '(nonpadding)' by 'value'.

Defect 21:
Problem: the text on the countls function in 4.1.7.3 and 7.18a.6.4 reads:

> The integer return value of the above functions is defined as follows:
> -if the value of the fixed-point argument f is non-zero, the return value is the largest integer k for which the expression f<<k does not overflow;
> -if the value of the fixed-point argument is zero, an integer value is returned that is at

least as large as N-1, where N is the total number of (nonpadding) bits of the fixed-point type of the argument.

Note: if the value of the fixed-point argument is zero, the recommended return value is exactly N-1.

From the definition it is clear that for instance
countlsur(UFRACT_EPSILON) == (UFRACT_FBIT - 1)
and
countlsk (ACCUM_EPSILON) == (ACCUM_IBIT + ACCUM_FBIT - 1)
If the text '(nonpadding) bits' is replaced by 'value bits' (see Defect 20), then the text requires that
countlsr(0.0r) >= (N - 1)
where the latter value equals countls(FRACT_EPSILON).
This seems counterintuitive; one would expect the value of countlsr(0.0r) to be one less than countls(FRACT_EPSILON).

Solution: change in 4.1.7.3 and 7.18a.6.4 the text of the 2nd bullet and the Note as follows:
-if the value of the fixed-point argument is zero, an integer value is returned that is at least as large as N, where N is the total number of value bits of the fixed-point type of the argument.

Note: if the value of the fixed-point argument is zero, the recommended return value is exactly N.

Defect 22:
Problem: the bitwise integer to fixed-point functions (in 7.18a.6.6) do not use the int_*fx*_t and uint_*fx*_t integer types; the text in 4.1.7.5 is already correct.

Solution:
-change the Synopsis section of 7.18a.6.6 to read:

```
#include <stdfix.h>
short fract hrbits(int_hr_t n);
fract rbits(int_r_t n);
long fract lrbits(int_lr_t n);
short accum hkbits(int_hk_t n);
accum kbits(int_k_t n);
long accum lkbits(int_lk_t n);
unsigned short fract uhrbits(uint_uhr_t n);
unsigned fract urbits(uint_ur_t n);
unsigned long fract ulrbits(uint_ulr_t n);
unsigned short accum uhkbits(uint_uhk_t n);
unsigned accum ukbits(uint_uk_t n);
unsigned long accum ulkbits(uint_ulk_t n);
```

-remove from the of 7.18a.2 the words 'as return types'
-change in 7.18a.2 the first sentence after the list to read:

The integer types **int_*fx*_t** and **uint_*fx*_t** are the return types of the corresponding **bits*fx*** functions and are chosen so that the return value can hold all the necessary bits; the ***fx*bits** functions use these integer types as types for their parameters.