Doc No: SC22/WG21/N2851
J16/09-0041
Date: 2009-03-06
Reply to: Robert Klarer
IBM Canada, Ltd.
klarer@ca.ibm.com

Changes to the Decimal TR since the PDTR Ballot

Document N2849 supersedes N2732, which was the subject of the PDTR ballot. This document is a comprehensive list of the differences between the two documents, apart from minor changes to formatting and presentation. These changes are also identified using red text and ~~strikethrough text~~ in N2849 itself.

1. Change the second paragraph of "2 Conventions" as follows:

Although this report describes extensions to the *C++ standard library*, vendors may choose to implement these extensions in the C++ language translator itself. This practice is permitted so long as all well-formed programs are accepted by the implementation, and the semantics of those programs which do not have undefined or implementation-defined behavior are ~~the same~~ as ~~the same as they would be had~~ if the extensions had taken the form of a library. [Note: This allows, for instance, an implementation to produce a different result when the extension is implemented in the C++ language translator, for programs that are ill-formed when the extension is implemented as a library.]

The result of deriving a user-defined type from `std::decimal::decimal32`, `std::decimal::decimal64`, or `std::decimal::decimal128` is undefined.

2. In "3.1 Characteristics of decimal floating-point types," change Table 1 as follows:

Format	**decimal32**	**decimal64**	**decimal128**
Coefficient length in digits	7	16	34
Maximum Exponent (E_{max})	97	385	6145
Minimum Exponent (E_{min})	-94	-382	-6142

3. Change "3.2.2.4 Conversion to integral type" as follows:

```
operator long long() const;
```

~~**Returns**~~**Effects:** Returns the result of the conversion of `*this` to the type `long long`~~, as if performed by the expression `llroundd32(*this)` while the decimal rounding direction mode [3.5.2] `FE_DEC_TOWARD_ZERO` is in effect.~~ by discarding the fractional part (i.e., the value is truncated toward zero). If the value of the integral part cannot be represented by the integer type or *this has infinite value or is NAN, the "invalid" floating-point exception shall be raised and the result of the conversion is unspecified.

4. Change "3.2.3.4 Conversion to integral type" as follows:

```
operator long long() const;
```

~~**Returns**~~**Effects:** Returns the result of the conversion of `*this` to the type `long long`~~, as if performed by the expression `llroundd64(*this)` while the decimal rounding direction mode [3.5.2] `FE_DEC_TOWARD_ZERO` is in effect.~~ by discarding the fractional part (i.e., the value is truncated toward zero). If the value of the integral part cannot be represented by the integer type or *this has infinite value or is NAN, the "invalid" floating-point exception shall be raised and the result of the conversion is unspecified.

5. **Change "3.2.4.4 Conversion to integral type" as follows:**

```
operator long long() const;
```

~~**Returns**~~**Effects:** Returns the result of the conversion of `*this` to the type `long long`~~, as if performed by the expression `llroundd128(*this)` while the decimal rounding direction mode [3.5.2] `FE_DEC_TOWARD_ZERO` is in effect.~~ by discarding the fractional part (i.e., the value is truncated toward zero). If the value of the integral part cannot be represented by the integer type or *this has infinite value or is NAN, the "invalid" floating-point exception shall be raised and the result of the conversion is unspecified.

6. Change the Effects clause of "3.2.5 Initialization from coefficient and exponent" as follows:

Effects: ~~If the value *coeff* x 10^*exponent* is outside of the range of values that can be represented by the return type, plus or minus HUGE_VAL_D32, HUGE_VAL_D64, or HUGE_VAL_D128 is returned (according to the return type and the sign of *coeff*) and the value of the macro ERANGE is stored in errno. If the result underflows, plus or minus DEC32_MIN, DEC64_MIN, or DEC128_MIN is returned (according to the return type and the sign of *coeff*), and the value of ERANGE is stored in errno. Otherwise,~~ Returns an object of the appropriate decimal floating-point type with the value *coeff* x $10^{exponent}$, rounded as in IEEE-754, if necessary. If an overflow condition occurs, then the value of the macro `ERANGE` is stored in `errno`.

[*Note:* In cases where the desired coefficient is greater than `ULLONG_MAX` or less than `LLONG_MIN`, it will be preferable to initialize an object of decimal floating-point type by extracting its value from a string literal using one of the `strtod` functions or iostreams. Also, see 4.1 --*end note.*]

7. Add new functions to "3.2.6 Conversion to generic floating-point type"

```
float decimal32_to_float (decimal32 d);
float decimal64_to_float (decimal64 d);
float decimal128_to_float(decimal128 d);
float decimal_to_float(decimal32  d);
float decimal_to_float(decimal64  d);
float decimal_to_float(decimal128 d);
```

Returns: If `std::numeric_limits<float>::is_iec559 == true`, returns the result of the conversion of *d* to `float`, performed as in IEEE 754-2008. Otherwise, the returned value is implementation-defined. See 4.2.

```
double decimal32_to_double (decimal32 d);
double decimal64_to_double (decimal64 d);
double decimal128_to_double(decimal128 d);
double decimal_to_double(decimal32  d);
double decimal_to_double(decimal64  d);
double decimal_to_double(decimal128 d);
```

Returns: If `std::numeric_limits<double>::is_iec559 == true`, returns the result of the conversion of *d* `double`, performed as in IEEE 754-2008. Otherwise, the returned value is implementation-defined. See 4.2.

8. In "3.3 Additions to header `<limits>`", change the values of `numeric_limits<decimal::decimal32>:: min_exponent` and `numeric_limits<decimal::decimal32>:: max_exponent` in the example:

```
static const int min_exponent   = -94;
static const int min_exponent10 = min_exponent;
static const int max_exponent   = 97;
static const int max_exponent10 = max_exponent;
```

9. In "3.4 Headers <cfloat> and <float.h>", remove the outdated reference to `<cdecfloat>`:

The header <c~~dec~~`float`> is described in [tr.c99.cfloat].

10. Change the title of subclause "3.4.1 Additions to h~~H~~eader <cfloat> synopsis", and update some macro values as follows:

```
// minimum exponent:
#define DEC32_MIN_EXP    -94
#define DEC64_MIN_EXP   -382
#define DEC128_MIN_EXP -6142

// maximum exponent:
#define DEC32_MAX_EXP    97
#define DEC64_MAX_EXP   385
#define DEC128_MAX_EXP 6145
```

11. Change the title of subclause "3.4.2 Additions to h~~H~~eader <float.h> synopsis."

12. Correct the copy-paste bug in "3.4.6 Minimum positive subnormal value" as follows:

```
#define DEC32_SUBNORMAL  implementation-defined
```

Expansion: an rvalue of type `decimal32` equal to the minimum positive finite number that can be represented by an object of type `decimal32`; exactly equal to 0.000001×10^{-95}

```
#define DEC64_SUBNORMAL  implementation-defined
```

Expansion: an rvalue of type `decimal64` equal to the minimum positive finite number that can be represented by an object of type `decimal64`; exactly equal to $0.000000000000001 \times 10^{-383}$

```
#define DEC128_SUBNORMAL implementation-defined
```

Expansion: an rvalue of type `decimal128` equal to the minimum positive finite number that can be represented by an object of type `decimal128`; exactly equal to $0.000000000000000000000000000000001 \times 10^{-6143}$

13. Correct typo in "3.5.2 Rounding modes" as follows:

These macros are used by the `fe_dec_getround` and `fe_dec_setround` functions for getting and setting the rounding mode to be used in decimal floating-point operations.

14. Update "3.6 Additions to <cmath> and <math.h> " as follows:

The elementary mathematical functions declared in the standard C++ header `<cmath>` are overloaded by this Technical Report to support the decimal floating-point types. The macros `HUGE_VAL_D32`, `HUGE_VAL_D64`, `HUGE_VAL_D128`, `DEC_INFINITY`, and `DEC_NAN` are defined for use with these functions. ~~With the exception of `sqrt`, `fmax`, and `fmin`, the accuracy of the result of a call to one of these functions is implementation-defined:~~ The following math functions are completely specified by 754-2008 and are correctly rounded:

`sqrt`, `fma`, `fabs`, `fmax`, `fmin`, `ceil`, `floor`, `trunc`, `round`, `rint`, `lround`, `llround`, `ldexp`, `frexp`, `ilogb`, `logb`, `scalbn`, `scalbln`, `copysign`, `nextafter`, `remainder`, `isnan`, `isinf`, `isfinite`, `isnormal`, `signbit`, `fpclassify`, `isunordered`, `isgreater`, `isgreaterequal`, `isless`, `islessequal`, `quantize`, and `samequantum`.

The accuracy of other math functions is implementation defined and the implementation may state that the accuracy is unknown.

15. Change "3.6.2 <cmath> macros" as follows:

```
#define HUGE_VAL_D32        implementation-defined
```

Expansion: a positive rvalue of type `decimal32` `representing infinity`.

```
#define HUGE_VAL_D64        implementation-defined
```

Expansion: a positive rvalue of type `decimal64`~~`, not necessarily representable as a decimal32`~~ `representing infinity`.

```
#define HUGE_VAL_128        implementation-defined
```

Expansion: a positive rvalue of type `decimal128`~~`, not necessarily representable as a decimal64`~~ `representing infinity`.

```
#define DEC_INFINITY        implementation-defined
```

Expansion: an rvalue of type `decimal32` representing infinity.

```
#define DEC_NAN             implementation-defined
```

Expansion: an rvalue of type `decimal32` representing quiet NaN.

```
decimal32  abs(decimal32  d);
```

16. Change "3.7 Additions to <cstdio> and <stdio.h>" as follows:

```
      H  Specifies that any following a, A, e, E, f, F, g, or G
conversions specifier applies to a decimal32 argument.
      D  Specifies that any following a, A, e, E, f, F, g, or G
conversions specifier applies to a decimal64 argument.
      DD Specifies that any following a, A, e, E, f, F, g, or G
conversions specifier applies to a decimal128 argument.
```

17. Change "3.10.2.2 extended_num_get virtual functions" as follows:

Effects: The input characters will be interpreted as described in [lib.facet.num.get.virtuals], and the resulting value will be stored in *val*. For conversions to type decimal32, decimal64, and decimal128, the conversion specifiers are `%Hg`, `%Dg`, and `%DDg`, respectively.

18. Change "Table 4 -- Length modifier" as follows:

type	length modifier
decimal32	H~~D~~
decimal64	D
decimal128	~~L~~DD

19. Change "3.11 Type traits" as follows:

However, the following expressions shall all yield ~~`true`~~ the same Boolean value, where *dec* is one of `decimal32`, `decimal64`, or `decimal128`:

~~`tr1::is_arithmetic<`*dec*`>::value == tr1::is_fundamental<`*dec*`>::value == tr1::is_scalar<`*dec*`>::value == !tr1::is_class<`*dec*`>::value == tr1::is_pod<`*dec*`>::value`~~

```
tr1::is_arithmetic<dec>::value
tr1::is_fundamental<dec>::value
tr1::is_scalar<dec>::value
!tr1::is_class<dec>::value
tr1::is_pod<dec>::value
```

[*Note:* The behavior of the type trait `std::tr1::is_floating_point` is not altered by this Technical Report. --*end note*]

~~The following expression shall yield `true` where *dec* is one of `decimal32, decimal64, or decimal128:`~~

~~`is_pod<`*dec*`>::value`~~

20. Remove subclause “4.1 Use of <decfloat.h>”:

~~To aid portability to C++, it is recommended that C programmers `#include` the header file `<decfloat.h>` in those translation units that make use of the decimal floating types. This ensures that the equivalent C++ floating-point types will be available, should the program source be ported to C++.~~

21. Change “4.3 [now 4.2, see above] Conversions” as follows:

In C, objects of decimal floating-point type can be converted to generic floating-point type by means of an explicit cast. In C++ this is not possible. Instead, the following functions ~~`decimal_to_long_double`, `decimal32_to_long_double`, `decimal64_to_long_double`, and `decimal128_to_long_double`~~ should be used for this purpose:

```
decimal_to_float    decimal_to_double    decimal_to_long_double
decimal32_to_float  decimal32_to_double  decimal32_to_long_double
decimal64_to_float  decimal64_to_double  decimal64_to_long_double
decimal128_to_float decimal128_to_double decimal128_to_long_double
```

C programmers who wish to maintain portability to C++ should use these ~~`decimal32_to_long_double`, `decimal64_to_long_double`, and `decimal128_to_long_double`~~ forms instead of the cast notation.