15 Specman/AMS Integration

15.1 Real Type Reference

The ***e*** language supports the **real** data type. Objects of type **real** are double precision floating point numbers, similar to the precision of a C type **double**.

This section contains the following topics:

- "Real Data Type Usage" on page 15-1
- "Real literals" on page 15-2
- "Real Constants" on page 15-2
- "Conversion Between Real and Integer Data Types" on page 15-4
- "Conversions using the as_a() operator" on page 15-5
- "Real Data Type Precision, Data Conversion, and Sign Extension" on page 15-5
- "Packing Values for real Types" on page 15-6
- "Printing Real Values" on page 15-6
- "Arithmetic Routines Supporting Real Type" on page 15-6
- "Random Routines" on page 15-8
- "C Interface Macros that Support real Type Objects" on page 15-8
- "Real Type Limitations" on page 15-8

15.1.1 Real Data Type Usage

In any context expecting a numeric object, a **real** object is acceptable, except in the following cases:

- Both operands of the shift operators (<<, >>)
- Bitwise operators (|, &, ^)

- Bitwise routines
- Modulo (%)
- **odd()**
- **even()**

See Also

- "Real Type Reference" on page 15-1

15.1.2 Real literals

Real literals are numbers that have a decimal point or an exponential part or both. If a decimal point exists, there must be digits on both sides of the decimal point. Underscores can be added for readability and are ignored.

Examples of **real** literals are:

Real Constant	Value
5.3876e4	53,876
4e-11	0.00000000004
1e+5	100,000
7.321E-3	0.007321
3.2E+4	32,000
0.5e-6	0.0000005
0.45	0.45
6.e10	60,000,000,000

See Also

- "Real Type Reference" on page 15-1

15.1.3 Real Constants

The following **real** constants are defined in Specman both in ***e*** code and in C code that includes an h file created by Specman:

Table 15-1 Mathematical Constants

Constant	Value
SN_M_E	e
SN_M_LOG2E	Logarithm base 2 of e
SN_M_LOG10E	Logarithm base 10 of e
SN_M_LN2	Natural logarithm of 2
SN_M_LN10	Natural logarithm of 10
SN_M_PI	PI
SN_M_TWO_PI	2*PI
SN_M_PI_2	PI/2
SN_M_PI_4	PI/4
SN_M_1_PI	1/PI
SN_M_2_PI	2/PI
SN_M_2_SQRTPI	2/sqrt(PI)
SN_M_SQRT2	sqrt(2)
SN_M_SQRT1_2	sqrt(1/2)

Note All Specman mathematical constants are prefixed by SN_M_.

Table 15-2 Physical Constants

Constant	Value
SN_P_Q	Charge of electron in coulombs
SN_P_C	Speed of light in vacuum in meters/sec
SN_P_K	Boltzmann’s constant in joules/kelvin
SN_P_H	Planck’s constant in joules*sec

Table 15-2 Physical Constants

Constant	Value
SN_P_EPS0	Permittivity of vacuum in farads/meter
SN_P_U0	Permeability of vacuum in henrys/meter
SN_P_CELSIUS0	Zero Celsius in kelvin

Note All Specman physical constants are prefixed by SN_P_.

See Also

- "Real Type Reference" on page 15-1

15.1.4 Conversion Between Real and Integer Data Types

Automatic casting is performed between the **real** type and the other numeric types.

Converting a **real** type object to an integer type object uses the following process:

1. The object is first converted to type **int** (bits:*) with the value of the largest integer whose absolute value is less than or equal to the absolute value of the **real** object.
2. The object is then converted to the expected integer type.

Additional rules apply to converting **real** objects to integer objects:

- If the object's floating-point value is infinity (inf), negative infinity (-inf), or Not-a-Number (NaN), an error will be emitted when trying to convert to an integer value.
- When converting an integer object to the **real** type, the object is converted to the value closest to the integer value that can be represented in the double precision format.

See Also

- "Real Type Reference" on page 15-1

15.1.5 Conversions using the as_a() operator

Converting a non-numeric scalar type object to a **real** type object using the **as_a()** operator uses the following process:

1. The scalar type object is first converted to an integer value.

2. The object is then converted to a **real** value according to process and rules listed in Section 15.1.4, “Conversion Between Real and Integer Data Types,” on page 15-4.

Additional rules apply to converting non-numeric scalar objects to **real** objects using the **as_a()** operator:

- When converting a string value to **real** using the **as_a()** operator, the string is parsed as if it was a **real** literal, and the value of the **real** literal is returned.
- If the string does not conform to the definition of a **real** literal, an error is emitted.

See Also

- “Real Type Reference” on page 15-1

15.1.6 Real Data Type Precision, Data Conversion, and Sign Extension

The rules for deciding precision, performing data conversion, and sign extension are as follows:

- Determine the context of the expression. The context may be comprised of up to three types.
- If all types involved in an expression and its context are integer values of 32 bits in width or less:
    - The operation is performed in 32 bits.
    - If any of the types are unsigned, the operation is performed with unsigned integers.

      Note Decimal constants are treated as signed integers, whether they are negative or not. All other constants are treated as unsigned integers unless preceded by a hyphen.

    - Each operand is automatically cast, if necessary, to the required type.

      Note Casting of small negative numbers (signed integers) to unsigned integers produces large positive numbers.

- If all types are integer types, and any of the types is greater than 32 bits:
    - The operation is performed in infinite precision (int(bits:*)).
    - Each operand is zero-extended if it is unsigned, or sign-extended if it is signed, to infinite precision.

- If any of the types is a **real** type, then the operation is done in real precision, and all objects should first be converted according to the rules described above.

Pubs: this is a repeat of the info in 3.5.2 in the eref.

15.1.7 Packing Values for real Types

Real values take up 64 bits when packed. These bits are actual bit representation of the double value. The effect of the various packing options on real type objects is similar to their effect on an integer (bits:64) value.

See Also

- "Real Type Reference" on page 15-1

15.1.8 Printing Real Values

By default, **real** values are printed similar to the %g format of printf in C. When using formatted printing, you can use the %e, %f, and %g formats similar to C. Printing an integer value with these formats will cause an automatic conversion to the **real** type. Printing **real** values with integer formatting will cause an automatic conversion to int (bits:*).

See Also

- "Real Type Reference" on page 15-1

15.1.9 Arithmetic Routines Supporting Real Type

The following arithmetic routines support of **real** type objects:

Table 15-3 Arithmetic Routines Supporting real Types

Routine	Description
floor(real): real	Returns the largest integer that is less than or equal to the parameter.
ceil(real): real	Returns the smallest integer that is greater than or equal to the parameter.
round(real): real	Returns the closest integer to the parameter. In the case of a tie then it returns the integer with the higher absolute value.
log(real): real	Returns the natural logarithm of the parameter.
log10(real): real	Returns the base-10 logarithm of parameter.

Table 15-3 Arithmetic Routines Supporting real Types

Routine	Description
pow(real, real): real	Returns the value of the first parameter raised to the power of second one.
sqrt(real): real	Returns the square root of the parameter.
exp(real): real	Returns the value of e raised to the power of the parameter.
sin(real): real	Returns the sine of the parameter given in radians.
cos(real): real	Returns the cosine of the parameter given in radians.
tan(real): real	Returns the tangent of the parameter given in radians.
asin(real): real	Returns the arc sine of the parameter.
acos(real): real	Returns the arc cosine of the parameter.
atan(real): real	Returns the arc tangent of the parameter.
sinh(real): real	Returns the hyperbolic sine of the parameter.
cosh(real): real	Returns the hyperbolic cosine of the parameter.
tanh(real): real	Returns the hyperbolic tangent of the parameter.
asinh(real): real	Returns the inverse hyperbolic sine of the parameter.
acosh(real): real	Returns the inverse hyperbolic cosine of the parameter.
atanh(real): real	Returns the inverse hyperbolic tangent of the parameter.
atan2(real, real): real	Returns the arc tangent of the two parameters.
hypot(real, real): real	Returns the distance of the point defined by the two parameters from the origin.
is_nan(real): bool	Returns TRUE if the parameter’s value is Not-a-Number (NaN).
is_finite(real): bool	Returns TRUE if the parameter’s value is a finite real value that is, it is not infinity, negative infinity, or NaN).

Note For integer routines like **ilog()**, **ilog10()**, **ilog2()**, **ipow()**, and **isqrt()**, whose return type is based on the expected type, if the expected type is **real**, then the return type is **int** (bits:*).

See Also

- “Real Type Reference” on page 15-1

15.1.10 Random Routines

Specman supports the following routines to generate random **real** numbers:

Table 15-4

Routine	Description
rdist_uniform(from: real, to:real): real	Returns a random **real** number using uniform distribution in the range **from** to **to**.

See Also

- “Real Type Reference” on page 15-1

15.1.11 C Interface Macros that Support real Type Objects

The following C interface macros support **real** type objects:

Table 15-5 C Interface Macros Supporting real Type Objects

Macro	Description
`SN_REAL_NEW(c_double_value)`	Returns a Specman allocated **real** object with its value specified by the parameter c_double_value of type C double.
`SN_REAL_GET ()`	Receives an object of type SN_TYPE(real) and returns a C double value.
`SN_TYPE(real)`	Declares an ***e*** **real** type in C.

See Also

- “Real Type Reference” on page 15-1

15.1.12 Real Type Limitations

- The key of a keyed list cannot be of type **real**.
- The waveform related command **wave exp** cannot be used with **real** type objects.

See Also

- "Real Type Reference" on page 15-1