3.3.4 precisionDecimal

[Definition:] The **precisionDecimal** datatype represents the numeric value and (arithmetic) precision of decimal numbers which retain precision; it also includes values for positive and negative infinity and for "not a number", and it differentiates between "positive zero" and "negative zero". This datatype is introduced to provide a variant of decimal thatfrom which may be derived datatypes closely correspondsing to the floating-point decimal datatypes described by [IEEE 754-2008]; it also permits derivation of a datatype closely corresponding to Java BigDecimal, although implementation is permitted to be confined to a much smaller value space.. Precision of values is retained and values are included for two zeroes, two infinities, and not-a-number.

Note: Users wishing to implement useful operations for this datatype (beyond the equality and order specified herein) are urged to consult [IEEE 754-2008].

Informally, the precision of a value is denoted by the number of decimal digits after the decimal point in its ·lexical representations·. The numbers 2 and 2.00, although numerically equal, have different precision (0 and 2 respectively). The precision of a value is derived from the ·lexical representation· using rules described in Lexical Mapping (§3.3.4.2). Values retain their precision, but that precision plays no part in any operations defined in this specification other than identity: specifically, it does not affect equality or ordering comparisons. Precision may play a role in arithmetic operations, but that is outside the scope of this specification.

The datatype precisionDecimal draws its name from a common usage of the term 'precision' to mean the degree of accuracy with which a quantity is recorded. In this usage, writing a number as "2" or as "2.00" is taken as recording the value with less or more precision.

Note: See the conformance note in Partial Implementation of Infinite Datatypes (§5.4), which applies to this datatype.

Editorial Note: Priority Feedback Request

The precisionDecimal datatype is a ‘feature at risk’. It may be or dropped at the end of the Candidate Recommendation period. The determination of whether to retain it or remove it will depend both on the existence of two independent implementations in the context of this specification and on the degree of uptake of [IEEE 754-2008] in the industry. Possible outcomes include retention of precisionDecimal, dropping precisionDecimal, rearrangement of the type hierarchy in this area, and merger with the existing decimal datatype.

3.3.4.1 Value Space

Properties of precisionDecimal Values

·***numericalValue***·
a decimal number, ***positiveInfinity***, ***negativeInfinity*** or ***notANumber***

·***arithmeticPrecisionscale***·
an integer or ***absent***; ***absent*** if and only if *·numericalValue·* is a ·special value·.

·***sign***·
positive, ***negative***, or ***absent***; must be ***positive*** if *·numericalValue·* is positive or ***positiveInfinity***,

must be ***negative*** if *·numericalValue·* is negative or ***negativeInfinity***, must be ***absent*** if and only if *·numericalValue·* is ***notANumber***.

Note: The *·sign·* property is redundant except when *·numericalValue·* is zero; in other cases, the *·sign·* value is fully determined by the *·numericalValue·* value.

Note: As explained below, '`NaN`' is the lexical representation of the precisionDecimal value whose *·numericalValue·* property has the ·special value· ***notANumber***. Accordingly, in English text we use 'NaN' to refer to that value. Similarly we use 'INF' and '–INF' to refer to the two values whose *·numericalValue·* properties have the ·special values· ***positiveInfinity*** and ***negativeInfinity***. These three precisionDecimal values are also informally called "not-a-number", "positive infinity", and "negative infinity". The latter two together are called "the infinities".

Note: The datatype defined here is intended to allow the derivation of less general datatypes corresponding to the decimal formats defined by [IEEE 754-2008]. Those formats can be viewed as representing values other than the infinities and NaN as triples (***s***, ***q***, ***m***), where ***s*** is the *sign* bit, ***q*** is the *exponent* (an integer), and ***m*** is the *significand* (also an integer). (The "*q*" form of IEEE's *exponent* is used when treating the *significand* as an integer.) The precisionDecimal *·numericalValue·* is (–1 ^ ***s***) × (10 ^ ***q***) × ***m*** and the *·scale·* is ***q***. Conversely, for nonzero finite values the *sign* ***s*** is the *·numericalValue·* divided by its absolute value, the integer *significand* ***m*** is | *·numericalValue·* | / (10 ^ *·scale·*) , and, of course, the *exponent* ***q*** is the *·scale·*.

The single NaN of precisionDecimal corresponds both to the signaling NaN and to the quiet NaN of [IEEE 754-2008], which permits the use of a single NaN when values are being transmitted from one system to another via ·lexical representations·.

The individual decimal formats defined by [IEEE 754-2008] are characterized by the range of values allowed for the *exponent* and by the number of decimal digits available for the *significand*, which IEEE terms the *precision* of the format. In any given IEEE 754 decimal format there may be multiple representations representing the same numerical value, one with its significand using the maximum number of significant digits available in the format, and the others with fewer significant digits (when that is possible). Note that in some cases these distinct representations will result in distinct results in operations defined by IEEE 754.

For datatypes derived from precisionDecimal, setting the totalDigits facet is equivalent to restricting the number of decimal digits available for the signifcand, and setting the minScale and maxScale facets amounts to controlling the possible values of the exponent q.

Note: precisionDecimal also allows the derivation of the Java BigDecimal class, which corresponds in essentials to a datatype derived from precisionDecimal by limiting the allowed scale, by eliminating the special values, and by ignoring the difference between +0 and –0.

Equality and order for precisionDecimal are defined as follows:

Two numerical precisionDecimal values are ordered (or equal) as their *·numericalValue·* values are ordered (or equal). (This means that two zeroes with different *·sign·*s are *equal*; negative zeroes are *not* ordered less than positive zeroes.)

- INF is equal only to itself, and is greater than –INF and all numerical precisionDecimal values.

- –INF is equal only to itself, and is less than INF and all numerical precisionDecimal values.
- NaN is incomparable with all values, *including itself*.

  Note: Any value ·incomparable· with the value used for the four bounding facets (·minInclusive·, ·maxInclusive·, ·minExclusive·, and ·maxExclusive·) will be excluded from the resulting restricted ·value space·. In particular, when NaN is used as a facet value for a bounding facet, since no precisionDecimal values are ·comparable· with it, the result is a ·value space· that is empty. If any other value is used for a bounding facet, NaN will be excluded from the resulting restricted ·value space·; to add NaN back in requires union with the NaN-only space (which may be derived using the pattern '`NaN`').

  Note: As specified elsewhere, enumerations test values for equality with one of the enumerated values. Because NaN ≠ NaN, including NaN in an enumeration does not have the effect of accepting NaNs as instances of the enumerated type; a union with a NaN-only datatype (which may be derived using the pattern '`NaN`') can be used instead.

3.3.4.2 Lexical Mapping

precisionDecimal's lexical space is the set of all decimal numerals with or without a decimal point, numerals in scientific (exponential) notation, and the character strings '`INF`', '`+INF`', '`-INF`', and '`NaN`'.
Lexical Space

[4] *pDecimalRep* ::= *noDecimalPtNumeral* | *decimalPtNumeral* | *scientificNotationNumeral* | *numericalSpecialRep*

The *pDecimalRep* production is equivalent (after whitespace is removed) to the following regular expression:
```
(\+|-)?([0-9]+(\.[0-9]*)?|\.[0-9]+)([Ee](\+|-)?[0-9]+)?|(\+|-)?INF|NaN
```

The ·lexical mapping· for precisionDecimal is ·*precisionDecimalLexicalMap*·. The ·canonical mapping· is ·*precisionDecimalCanonicalMap*·.

For example, each of the ·lexical representations· shown below is followed by its corresponding value triple (·*numericalValue*·, ·*arithmeticPrecisionscale*·, and ·*sign*·) and ·canonical representation·:

- '`3`' (3 , 0 , ***positive***) '`3`'
- '`3.00`' (3 , 2 , ***positive***) '`3.00`'
- '`03.00`' (3 , 2 , ***positive***) '`3.00`'
- '`300`' (300 , 0 , ***positive***) '`300`'
- '`3.00e2`' (300 , 0 , ***positive***) '`300`'
- '`3.0e2`' (300 , –1 , ***positive***) '`3.0E2`'
- '`30e1`' (300 , –1 , ***positive***) '`3.0E2`'
- '`.30e3`' (300 , –1 , ***positive***) '`3.0E2`'

Note that the last three examples not only show different ·lexical representations· for the same value, but are of particular interest because values with negative precision·*scale*· *only* have ·lexical representations· in scientific notation.

> **Note:** [IEEE 754-2008] expects ·lexical representations· whose exact value is not in the ·value space· to be mapped to the nearest value that is in the ·value space·. When precisionDecimal is restricted, all ·lexical representations· of values dropped from the value space are dropped

from the lexical space. One result is that when precisionDecimal is restricted using the totalDigits or maxScale facets, non-zero digits beyond those required to exactly represent the intended value are not permitted by this specification.

[IEEE 754-2008] permits all case variants of '`INF`' and '`NaN`', as well as those of '`INFINITY`'; in many cases it permits language definitions to prescribe which variants are used. This specification explicitly chooses only '`INF`' and '`NaN`'. 754 also permits language definitions to prescribe whether '+' shall be used with positive values; this specification makes the '+' optional.

Note: The ·lexical representations· with "unnecessary least significant digits" representations are the only ones lost when precisionDecimal is restricted; shorter and simpler ·lexical representations· will not be eliminated by use of totalDigits or maxScale. (In contrast, if facets for controlling total digits and scale were available for the floating-point binary types, the effect of restriction would often be inconvenient. In the floating-point binary types, a simple decimal numeral will sometimes have no exact value and so the number will be rounded to a binary approximation. Any restriction of the value space which dropped that approximate value would automatically also drop the simple decimal numeral from the lexical space. For example, the number one-tenth is in the value space neither of float nor of double; the string '`0.1`' maps, in double, to 0.1000000000000000055511151231257827021181583404541015625. If the totalDigits, minScale, and maxScale facets were available for double (they are not) and were used to define the float type (again, they are not), the value just mentioned would be dropped, and the ·literal· '`0.1`' would be dropped along with it, instead of mapping (as in fact it does) to the value 0.100000001490116119384765625.

This interaction between datatype restriction and rounding has as a consequence that it will typically be more convenient for users if restricted-precision numeric types are derived from precisionDecimal than it would be if they were derived from float or double.

3.3.4.3 Facets

*   *   *   *   *