Doug Wood
Tivoli Systems Inc

Directory string representation for floating point values

Status of this Memo

Distribution of this memo is unlimited. It is filed as <draft-ietf-ldapext-float-00.txt >, and expires on June, 2000. Please send comments to the authors.

1. Abstract

This draft defines a way that floating point values may be represented as directory (ASCII) strings such that standard ordering rules can be used to sort the strings into the correct collating sequence for their numeric value. The representation is intended for use in X.500 like directories, and has been developed to support mapping of the DMTF Common Information Model.

2. Introduction

X.500 directories provide for use definable syntaxes, matching and ordering rules. This provides for the definition of schema supporting any type and structure of data. The definition of the LDAP protocol [RFC 2251] has encouraged the creation new generation of directories that support the X.500 structure, but donÆt support some of X.500Æs more heavy weight feature. Among the unsupported feature are user definable syntaxes. This restricts schema designers to syntaxes provided by the directory vendor. These typical do not include a syntax for floating point values.

This draft defines a an ASCII format for floating point values that allow them to be stored in attributes with Directory String syntax. And allows the standard case insensitive ordering rule to sort them in the correct collating sequence for their numeric value. In addition, attributes are defined which can be used for values stored in the format described below, or as superiors for user defined attributes. The attributes are provided both as a convenience, and as a method to document the storage format used.

The format is defined specifically to support mapping the DMTF Common Information Model to directory schema, but also has general applicability.

3. String format

Because string comparison is positional, it is necessary to define a fix format for representing the mantissa, and the exponent. Because the collating sequence for string comparison is left to right, the most significant portion of the representation must be on the left. There are four separate cases that must be handled.

- o Negative mantissa and positive exponent
- o Negative mantissa and negative exponent
- o Positive mantissa and negative exponent
- o Positive mantissa, and positive exponent

The above list is ordered by the desired collating sequence from smallest value to largest value. A single representation does not provide the correct collating sequence for all cases. Therefor it is necessary to sort by case, and then to sort within each case. To accomplish this, the cases are number from 1 to 5 as follows:

1. Negative mantissa and positive exponent
2. Negative mantissa and negative exponent
3. Zero
4. Positive mantissa and negative exponent
5. Positive mantissa, and positive exponent

For symmetry, zero is treated as its own case instead of a special sub-case of case 4.

A 64 bit float has a range of 1.7976931348623158e+308 to 2.2250738585072014e-308[1]. To represent this as a string, three digits are required for the exponent, and 17 for the mantissa not including the decimal point. The directory representation is fixed format, zero padded, blank separated, with the most significant fields on the left. The first character in the string is the case number. For readability, it is followed by a blank. Next is a 3 digit exponent, again followed by a blank. Next are a single digit, a decimal point, and 16 digits of decimal.

```
+-+-+---+-+-+-+----------------+
| | |Exp| | | |16 digits       |
+-+-+---+-+-+-+----------------+
|c| |nnn| |n|.|nnnnnnnnnnnnnnnn|
+-+-+---+-+-+-+----------------+
```

The way each of the fields is interpreted varies with the case. The cases are examined in reverse order so the simplest may be examined first.

3.1 Positive mantissa and positive exponent (case 5)

This is relatively straightforward. The exponent field has the exponent value expressed as a 3-digit integer string. It is zero padded to the left if necessary. The mantissa field as a seventeen-digit decimal string with exactly 1 digit to the left of the decimal point for a total of 18 characters. It is zero padded to the right if necessary.

Notes:

- The first digit is a 5 to indicate the case
- There is exactly one digit to the left of the decimal place. It is always non zero.
- Positions 2 through 4 have the exponent. It is right justified, and zero padded to the left if it is less than three digits
- Spaces are added to aid human readability
- No signs are required for the exponent or the mantissa because they are expressed in the case number

3.2 Positive mantissa and negative exponent (case 4)

When the exponent is negative, larger whole number values for the exponent produce smaller actual values. For this case, a string comparison of the numeric representation of the exponent yields the reverse of the desired collating sequence. To flip the collating sequence, the value of the exponent is added to 999, and the result stored as the exponent. No sign is stored. The sign of both the exponent and mantissa are indicated by the case character.

3.3 Zero (case 3)

The case number is sufficient to insure the correct collating sequence. To insure equality comparisons work correctly, all remaining digits are zero.

3.4 Negative mantissa and negative exponent (case 2)

When both the exponent and the mantissa are negative, the collating order for the exponent is correct. A larger exponent yields a number that is closer to zero and therefor larger. However, the collating sequence for the mantissa is reversed. To flip the collating sequence for the mantissa it is added to 10, and the result stored.

3.5 Negative mantissa and positive exponent (case 1)

When both the exponent and the mantissa are negative, the collating sequence is flipped for both of them. This is achieved by adding the exponent to 999, and the mantissa to 10.

4. Examples

Value	Representation
3.25e5	5 005 3.2500000000000000
8.4e-5	4 994 8.4000000000000000
8.4e-7	4 992 8.4000000000000000
7.23e-7	4 992 7.2300000000000000
0.0e0	3 000 0.0000000000000000
-4.25e-4	2 004 5.7500000000000000
-6.35e-4	2 004 3.6500000000000000
-6.35e-3	2 003 3.6500000000000000
-4.0e104	1 895 6.0000000000000000
-4.0e105	1 894 6.0000000000000000
-6.0e105	1 894 4.0000000000000000

5. 32 bit vs. 64 bit values

Both 32 and 64 bit floating point values are in common usage. To allow comparisons between the two, both are stored in the 64-bit format described above. This implies a greater degree of precision than is actually available for 32-bit values. The directory mapping described below provides implicit documentation of the actual precision of a value.

6. Directory mapping

The intent of the mapping is to simulate a new syntax. The advantage of this approach is it may be utilized without any changes to existing directory servers. To foster that illusion, and to aid in documentation, two new attributes are defined. cimFloat32, and cimFloat64. All floating-point attributes are derived from one of these. They are defined as:

```
cimFloat32
   {
                                      ; Need OID assigned
        NAME               æcimFloat32Æ
        DESC       '32 bit float encoded as sortable float format'
        EQUALITY   caseIgnoreeMatch
        ORDERING   caseIgnoreOrderingMatch
        SYNTAX     1.3.6.1.4.1.1466.115.121.1.15
    )

    cimFloat64
    (
                                      ; Need OID assigned
        NAME       æcimFloat64Æ
        DESC       'Æ64 bit float encoded as sortable float formatÆ
        EQUALITY   caseIgnoreeMatch
        ORDERING   caseIgnoreOrderingMatch
        SYNTAX     1.3.6.1.4.1.1466.115.121.1.15
    )
```

7. References

[1] From sys/limits.h on AIX 4.3

8. Acknowledgement

This work is a product of the DMTF LDAP Mapping Working Group and has benefited from many comments and discussions during this groups meetings.

9. Authors' Addresses

Doug Wood
Tivoli Systems
9025 North River Rd.
Indianapolis, IN 46240-7622
Dawood@Tivoli.com