



DELIVERY MANAGER EXPRESSION LANGUAGE

Technical Manual



Syntax and Technical Reference

13 July 2009



Document Revision History

Date	Document Version	Major changes
13/7/09	0.1	Initial draft.
16/7/09	0.2	Update following new functions and introducing arrays.
24/7/09	0.3	Introduced compound expressions
28/7/09	0.4	Introduced more array functions
1/9/09	0.5	New functions: UpperBandIndex, LowerBandIndex and Percentage
19/5/10	0.6	Changes and additions for DM 2.8, including changes to date representations

Table of Contents

1	SYNTAX	5
1.1	Introduction	5
1.2	Keywords & Operators	5
1.3	Types	6
1.4	Constants	7
1.5	Variables	7
1.6	Functions	7
1.7	Expressions & Compound Expressions	7
1.8	Basic Examples	8
2	FUNCTION REFERENCE	9
2.1	Mathematical Functions	9
2.1.1	ABS(numberValue)	9
2.1.2	CEILING(numberValue)	9
2.1.3	FLOOR(numberValue)	9
2.1.4	FORMAT(numberValue,formatString)	9
2.1.5	INT(numberValue)	9
2.1.6	LN(value)	10
2.1.7	MAX(numberValue1[,...(numberValueN)])	10
2.1.8	MIN(numberValue1[,...(numberValueN)])	10
2.1.9	PERCENTAGE(value,percentage)	10
2.1.10	POWER(numberValue,powerValue)	10
2.1.11	RAND(lowerValue,upperValue)	10
2.1.12	ROUND(numberValue,numPlaces)	11
2.1.13	SIGN(numberValue)	11
2.1.14	SQRT(numberValue)	11
2.1.15	VAL(stringValue)	11
2.2	String Functions	12
2.2.1	CONCAT(value1[,...valueN])	12
2.2.2	INDEXOF(value,findValue)	12
2.2.3	LASTINDEXOF(value,findValue)	12
2.2.4	LEFT(value,length)	12
2.2.5	LEN(value)	12
2.2.6	LIKE(value,pattern)	13
2.2.7	LOWER(value)	13
2.2.8	MID(value,start[,length])	13
2.2.9	PADLEFT(value,paddingText,length)	13
2.2.10	PADRIGHT(value,paddingText,length)	13
2.2.11	REPEAT(value,count)	14
2.2.12	REPLACE(value,findValue,newValue)	14
2.2.13	REVERSE(value)	14
2.2.14	RIGHT(value,length)	14
2.2.15	SPLIT(value,delimiter,index)	15
2.2.16	TITLE(value)	15
2.2.17	TRIM(value)	15
2.2.18	UPPER(value)	15
2.3	Date & Time Functions	16
2.3.1	ADDDAYS(dateValue,count)	16
2.3.2	ADDHOURS(dateValue,count)	16
2.3.3	ADDMINUTES(dateValue,count)	16
2.3.4	ADDWEEKS(dateValue,count)	16
2.3.5	ADDYEARS(dateValue,count)	16
2.3.6	AFTER(dateValue1,dateValue2)	16
2.3.7	BEFORE(dateValue1,dateValue2)	16
2.3.8	CHANGETIMEZONE(dateValue,timeZone)	17
2.3.9	DATEDIFF(dateValue1,dateValue2,mode)	17
2.3.10	DAYOFMONTH(dateValue)	17



2.3.11	DAYOFWEEK(dateValue)	17
2.3.12	DAYOFYEAR(dateValue)	17
2.3.13	ENDOFDAY(dateValue)	17
2.3.14	ENDOFMONTH(dateValue)	18
2.3.15	FORMATDATE(dateValue,formatPattern)	18
2.3.16	HOUR(dateValue)	19
2.3.17	ISSAMEDAY(dateValue1,dateValue2)	19
2.3.18	ISWEEKEND(dateValue)	19
2.3.19	LASTDAYOFWEEK(dateValue,dayNumber)	19
2.3.20	MINUTE(dateValue)	19
2.3.21	MONTH(dateValue)	19
2.3.22	NEXTDAYOFWEEK(dateValue,dayNumber)	19
2.3.23	NOW()	19
2.3.24	PARSEDATE(stringValue,formatPattern)	20
2.3.25	RANGECONTAINSDAY(dateFrom,dateTo,dayNum)	20
2.3.26	SECOND(dateValue)	20
2.3.27	STARTOFDAY(dateValue)	20
2.3.28	STARTOFMONTH(dateValue)	20
2.3.29	TODATE(stringValue)	20
2.3.30	TODAY()	20
2.3.31	TOMORROW()	20
2.3.32	YEAR(dateValue)	21
2.3.33	YESTERDAY()	21
2.4	Postcode Functions	22
2.4.1	CLEANPOSTCODE(countryCode,postCode)	22
2.4.2	COUNTRYCODE(value)	22
2.4.3	ISSAMETRADINGBLOCK(country1,country2)	22
2.4.4	ISVALIDPOSTCODE(countryCode,postCode)	22
2.4.5	POSTCODELEVEL(countryCode,postCode)	23
2.4.6	POSTCODEPART(countryCode,postCode,level)	23
2.5	General functions	24
2.5.1	ARRAYCONTAINS(variableName,value [,caseSensitiveFlag])	24
2.5.2	ARRAYFIND(variableName,value [,caseSensitiveFlag])	24
2.5.3	ARRAYSIZE(variableName)	24
2.5.4	DECODE(value,test1,answer1[,...testN,answerN] [,elseAnswer])	24
2.5.5	EVAL(expression)	24
2.5.6	IIF(booleanValue,trueResult,falseResult)	24
2.5.7	ISIN(matchValue,value[,...(valueN)])	25
2.5.8	LOOKUP(variableName)	25
2.5.9	LOWERBANDINDEX(matchValue,value[,...(valueN)])	25
2.5.10	SET(variableName,newValue)	25
2.5.11	UPPERBANDINDEX(matchValue,value[,...(valueN)])	26

1 SYNTAX

1.1 Introduction

The expression language is similar to some spreadsheet's function languages. Each expression will return a value, but that expression can be quite complex. This section will introduce the basic operator set and the requirements of the language itself.

Although an expression is expected to be on a single line, it need not be. White-space (tabs, spaces, new lines etc.) can be used to make them easier to read.

1.2 Keywords & Operators

The language supports the following operators:

+	Addition (or concatenation, in the case of a string operand)
-	Subtraction (or negation, in its unary use)
*	Multiplication
/	Division
%	Modulo remainder
^	Bit-wise XOR
&	Bit-wise AND
	Bit-wise OR
==	Equality
!=	Inequality
<	Less than
<=	Less than or equal to
>	More than
>=	More than or equal to
&&	Logical And
	Logical Or
!	Logical Not (unary)
()	Precedence alteration parenthesis
[]	Zero-based array index parenthesis

1.3 Types

The language supports strings and real numbers. Date-stamps are supported, and are implemented as strings. Similarly, boolean values are represented by numbers.

Numbers are stored as ANSI/IEEE Standard 754-1985.

The boolean value `true` is defined as "-1", while `false` is defined as "0".

Date-stamps are represented as strings, and are always presented in the following format:

```
yyyy-MM-dd'T'HH:mm:ssZ
```

The system is very tolerant when it comes to assigning values when calling functions that require date-stamps. You can use any of the following unambiguous date/time formats directly:

```
yyyy-MM-dd HH:mm:ss  
yyyy-MM-dd HH:mm  
HH:mm yyyy-MM-dd  
yyyy-MM-dd  
HH:mm dd MMM yyyy  
dd MMM yyyy  
HH:mm dd/MM/yyyy  
dd/MM/yyyy
```

Strings can be of any length, and are terminated with single or double quotes. The terminator character can also appear inside the string by making use of the back-slash ("\") escape-character.

The escape character can also be used to represent non-text characters, and ASCII symbols. The common escape-sequences are:

<code>\"</code>	Double-quote
<code>\'</code>	Single-quote
<code>\\</code>	Back-slash
<code>\n</code>	New-line (line-feed)
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\0nn</code>	Specific character (where "nn" is the octal of the ASCII value)

1.4 Constants

The language implements a number of commonly used constants. These are listed below:

True and False

Mon, Tue, Wed, Thu, Fri, Sat and Sun

Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov and Dec

1.5 Variables

A variable is a named entity that can be used as an operand, and can represent a string or a number. The name must start with a letter (or underscore). The rest of the name can be made up of letters, numbers and underscores.

If the name represents an array of items, the individual items can be accessed via a zero-based index, enclosed in square-brackets.

Names are not case-sensitive.

1.6 Functions

A function is distinguished from a variable by it having brackets containing arguments immediately following the name. If the function does not take any arguments, the brackets are empty.

The naming convention is the same as for variables, and names are also not case sensitive.

1.7 Expressions & Compound Expressions

An expression can return a string, a boolean, or a number.

There are occasions where you might want to pre-calculate something before using it within an expression. This is particularly useful when the calculation is used repeatedly.

You can separate expressions using semi-colons. This is known as a compound expression. The resultant value is the value of the last expression.

1.8 Basic Examples

```
3 * (1 + 2.23)
Len( "Hello world" )
"There are " + count + " parcels in the system."
iif( month(now()) == MAR , "This month is March" , "This month is not March" )
decode( data[fieldIndex] , 1 , "One" , 2 , "Two" , "Something else" )
set( "p" , 1 ) ; p + 1
```

2 FUNCTION REFERENCE

2.1 Mathematical Functions

2.1.1 ABS(numberValue)

This will return the absolute value of a real number. If a negative value is provided, the sign will be removed.

2.1.2 CEILING(numberValue)

This will return the nearest whole number above the specified value.

2.1.3 FLOOR(numberValue)

This will return the nearest whole number below the specified value.

2.1.4 FORMAT(numberValue,formatString)

This will format the provided number into a string, giving your own format.

e.g.

```
FORMAT( 1234.5 , "#,##0.00" )
```

Would result in a string containing:

```
1,234.50
```

2.1.5 INT(numberValue)

This will return the integer portion of the given number.

2.1.6 LN(value)

Returns the natural log of the specified value.

2.1.7 MAX(numberValue1[,...(numberValueN)])

This will return the maximum value from a set. The set can have one or more entries in.

e.g.

```
MAX(5,2,5,2,6,5,3,6,1,4,35,345)
```

Would result in:

345

2.1.8 MIN(numberValue1[,...(numberValueN)])

Similar to MAX, this will return the minimum value in a given set.

2.1.9 PERCENTAGE(value,percentage)

This will return the percentage of the specified value.

2.1.10 POWER(numberValue,powerValue)

This will return the specified number raised to a certain power.

i.e. n^p

e.g.

```
POWER( 3 , 2 )
```

Would result in:

9

2.1.11 RAND(lowerValue,upperValue)

This will return a random whole number between two values (inclusive).

2.1.12 ROUND(numberValue,numPlaces)

This will round a real number to a specific number of places.

2.1.13 SIGN(numberValue)

This will return the sign of the number. Positive numbers are returned as "1", while negative numbers are returned as "-1".

2.1.14 SQRT(numberValue)

This will return the square root of the specified number.

2.1.15 VAL(stringValue)

This will convert a string into a number.

2.2 String Functions

2.2.1 CONCAT(value1[,...valueN])

This will concatenate a series of values.

2.2.2 INDEXOF(value,findValue)

This will return the index of the first occurrence of the find value in the string. If it is not found, this function will return -1.

2.2.3 LASTINDEXOF(value,findValue)

This will return the index of the last occurrence of the find value in the string.

2.2.4 LEFT(value,length)

This will return the left-most characters of a string. If the number of characters requested is longer than the original string, the whole string will be returned.

2.2.5 LEN(value)

This will return the number of characters in the string.

2.2.6 LIKE(value,pattern)

This will return `true` if a value matches the pattern. The pattern can use wild-card operators such as "*" or "?".

e.g.

```
LIKE( "myfilename.txt" , "*.txt" )
```

Would return `true`.

2.2.7 LOWER(value)

This will return the specified value turned into lower-case.

2.2.8 MID(value,start[,length])

This will return a middle portion of the string, starting from a specific character position. If you omit the length, the rest of the string will be returned.

2.2.9 PADLEFT(value,paddingText,length)

This will pad a string out to the desired length using a specific piece of text. Typically, that piece of text comprises a single character.

e.g.

```
PADLEFT( "1234" , "0" , 6 )
```

Would return:

```
001234
```

2.2.10 PADRIGHT(value,paddingText,length)

This will pad a string out to the desired length using a specific piece of text. Typically, that piece of text comprises a single character.

2.2.11 REPEAT(value,count)

This will return a string with the given value repeated a number of times.

E.g.

```
REPEAT( "ABC" , 3 )
```

Would result in:

```
ABCABCABC
```

2.2.12 REPLACE(value,findValue,newValue)

This will replace all instances of a find-value with a new-value.

e.g.

```
REPLACE( "ABAAABAAABAA" , "B" , "CC" )
```

Would result in:

```
ACCAAACCAAACCAA
```

2.2.13 REVERSE(value)

This will reverse the characters in the given string.

2.2.14 RIGHT(value,length)

This will return the right-most characters of a string. If the number of characters requested is longer than the original string, the whole string will be returned.

2.2.15 SPLIT(value,delimiter,index)

This will split a value according to a regular expression pattern, and will return a specific part of the matched value.

e.g.

```
SPLIT( "A,BC,D,EFG" , "," , 2 )
```

Would return:

D

2.2.16 TITLE(value)

This will return the value in title case.

e.g.

```
TITLE( "fred bLoGgs" )
```

Would result in:

Fred Bloggs

2.2.17 TRIM(value)

This will remove leading and trailing white space from the value.

2.2.18 UPPER(value)

This will return the string in uppercase letters.

2.3 Date & Time Functions

2.3.1 ADDDAYS(dateValue,count)

This will return the date with a certain number of days added (or subtracted to it).

2.3.2 ADDHOURS(dateValue,count)

This will return the date with a certain number of hours added (or subtracted to it).

2.3.3 ADMINUTES(dateValue,count)

This will return the date with a certain number of minutes added (or subtracted to it).

2.3.4 ADDWEEKS(dateValue,count)

This will return the date with a certain number of weeks added (or subtracted to it).

2.3.5 ADDYEARS(dateValue,count)

This will return the date with a certain number of years added (or subtracted to it).

2.3.6 AFTER(dateValue1,dateValue2)

This will return true if the first value is after the second value.

2.3.7 BEFORE(dateValue1,dateValue2)

This will return true if the first value is before the second value.

2.3.8 CHANGETIMEZONE(dateValue,timeZone)

This will return the date & time in the specified time-zone.

2.3.9 DATEDIFF(dateValue1,dateValue2,mode)

This will return the difference between two dates, in the units dictated by the mode.

Mode	Meaning
1	Seconds
2	Minutes
3	Hours
4	Days
5	Weeks
6	Months

2.3.10 DAYOFMONTH(dateValue)

This will return the number of the day within the month.

2.3.11 DAYOFWEEK(dateValue)

This will return the day of the week as a number (Sunday is represented as 1). You are advised to make use of the constants.

2.3.12 DAYOFYEAR(dateValue)

This will return the number of days that have elapsed since the start of the year.

2.3.13 ENDOFDAY(dateValue)

This will set the time portion of the date to be 23:59:59.

2.3.14 ENDOFMONTH(dateValue)

This will set the date to be the end of the month, and will set the time portion to be the end of the day.

2.3.15 FORMATDATE(dateValue,formatPattern)

This will format the date using the pattern specified.

Letter	Date or Time Component	Examples
G	Era designator	AD
y	Year	1996; 96
M	Month in year	July; Jul; 07
w	Week in year	27
W	Week in month	2
D	Day in year	189
d	Day in month	10
F	Day of week in month	2
E	Day in week	Tuesday; Tue
a	Am/pm marker	PM
H	Hour in day (0-23)	0
k	Hour in day (1-24)	24
K	Hour in am/pm (0-11)	0
h	Hour in am/pm (1-12)	12
m	Minute in hour	30
s	Second in minute	55
S	Millisecond	978
z	Time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	-800

e.g.

```
FORMATDATE( "16/05/2009" , "dd MMM yyyy" )
```

Would return:

```
16 May 2009
```

2.3.16 HOUR(dateValue)

This will return the hour of the day (using the 24-hour clock).

2.3.17 ISSAMEDAY(dateValue1,dateValue2)

This will return true if the dates are on the same day.

2.3.18 ISWEEKEND(dateValue)

This will return true if the date is a Saturday or a Sunday.

2.3.19 LASTDAYOFWEEK(dateValue,dayNumber)

This will return a date representing the last particular day. For example, you could use this to determine the date of last Monday.

2.3.20 MINUTE(dateValue)

This will return the minute portion of the date.

2.3.21 MONTH(dateValue)

This will return the month portion of the date.

2.3.22 NEXTDAYOFWEEK(dateValue,dayNumber)

This will return a date representing the next particular day. For example, you could use this to determine the date of next Monday.

2.3.23 NOW()

This will return the current date & time.

2.3.24 PARSEDATE(stringValue,formatPattern)

This will return a date after being parsed from a string. It uses the same pattern as the `FORMATDATE()` function.

2.3.25 RANGECONTAINSDAY(dateFrom,dateTo,dayNum)

This will return true if the date range species contains the day indicated. The `dayNum` is the same values as used by `DAYOFWEEK()`.

2.3.26 SECOND(dateValue)

This will return the seconds portion of the date.

2.3.27 STARTOFDAY(dateValue)

This will set the time in the specified date to be 00:00:00.

2.3.28 STARTOFMONTH(dateValue)

This will return the first day of the same month for the date specified.

2.3.29 TODATE(stringValue)

This will return a date from the string value specified. This is rather tolerant, and will accept a string using any of the formats listed in section 1.3.

2.3.30 TODAY()

This will return today's date.

2.3.31 TOMORROW()

This will return tomorrow's date.

2.3.32 YEAR(dateValue)

This will return the year portion of the date.

2.3.33 YESTERDAY()

This will return yesterday's date.

2.4 Postcode Functions

2.4.1 CLEANPOSTCODE(countryCode,postCode)

This will parse the input postcode, validate it and clean it using the country's standards.

e.g.

```
CLEANPOSTCODE("GBR","EC1r 4 P F")
```

Would result in:

```
EC1R 4PF
```

2.4.2 COUNTRYCODE(value)

This will attempt to identify the country from the name. You can use any of the ISO country codes as the value, or the name of the country in the user's language. It will return the ISO 3 letter code.

2.4.3 ISSAMETRADINGBLOCK(country1,country2)

This will return true if the two countries are in the same trading block.

e.g.

```
ISSAMETRADINGBLOCK("GBR","France")
```

Would result in:

```
True
```

2.4.4 ISVALIDPOSTCODE(countryCode,postCode)

This will return true if the postcode is valid.

2.4.5 **POSTCODELEVEL(countryCode,postCode)**

This will determine the postcode level. It is particularly useful when the country supports multiple levels.

e.g.

```
POSTCODELEVEL("GBR","EC1r 4 P F")
```

Would result in:

4

2.4.6 **POSTCODEPART(countryCode,postCode,level)**

This will return the portion of the postcode for the desired level. If the level exceeds that which the format supports, the maximum level will be returned. E.g. If you ask for a level 100 UK postcode, you will receive the standard level 4.

e.g.

```
POSTCODEPART("GBR","EC1r 4 P F",3)
```

Would result in:

EC1R 4

2.5 General functions

2.5.1 **ARRAYCONTAINS(variableName,value [,caseSensitiveFlag])**

This will return true if the named array contains the specified value. You can use the flag to specify that string searching should be case-sensitive.

2.5.2 **ARRAYFIND(variableName,value [,caseSensitiveFlag])**

This will return the zero-based index of the item in the array. If it is not found, this will return -1.

2.5.3 **ARRAYSIZE(variableName)**

This will return the number of elements in the named array.

2.5.4 **DECODE(value,test1,answer1[,...testN,answerN] [,elseAnswer])**

This will try to match a value against a given set of criteria. If it finds a match, the appropriate answer will be provided. If no match is possible, the else clause is returned (if present).

2.5.5 **EVAL(expression)**

This will evaluate (at run-time) the expression provided. The result is a string. If you want to turn it into a number, use the `VAL()` function to convert the result.

2.5.6 **IIF(booleanValue,trueResult,falseResult)**

The "in-line if" function will return one of two clauses, based on the input of the first parameter.

2.5.7 ISIN(matchValue,value[,...(valueN)])

This will return true if the match value is in the list of values.

2.5.8 LOOKUP(variableName)

This will look up the contents of the named variable. This allows you to programmatically compute the name of the variable at run-time. If the variable does not exist, this will return the empty string.

It also allows you to make use of variable names that would otherwise be illegal.

2.5.9 LOWERBANDINDEX(matchValue,value[,...(valueN)])

This will return the index of the item in the list with the value nearest to, but lower than the match value specified. This can be used with either numbers or strings.

e.g.

```
LOWERBANDINDEX (55, 10, 20, 30, 40, 50, 60, 70)
```

Would result in:

4

As the nearest value is 40, and that is at position 4 (10 is at position zero).

2.5.10 SET(variableName,newValue)

This will change the contents of the named variable (in the context) to the new value. The effect is immediate, and the return value is the new value.

e.g. If 15 was the value of the "counter" parameter

```
SET("counter", counter+2)
```

Would result in:

17



2.5.11 UPPERBANDINDEX(matchValue,value[,...(valueN)])

This will return the index of the item in the list with the value nearest to, but greater than the match value specified. This can be used with either numbers or strings.