Prospect

Version 8.0.7

Expressions Technical Reference

Note

Before using this information and the product it supports, read the information in Notices on page 145.

This edition applies to version 8.0.7, release 6, modification 1 of IBM Prospect and to all subsequent releases and modifications until otherwise indicated in new editions.

Table of Contents

1 About This Documentation

The *Expressions Technical Reference* contains detailed information about regular expressions. The information included in this document includes the following:

- General expressions
- Advanced expressions

This guide was last updated July 2, 2010.

Please see the current release notes on this product for a list of revision dates for all IBM Prospect publications.

Audience

This guide is intended for technicians and engineers who use the IBM Prospect software to manage and analyze the performance of a telecommunication network. In general, the reader of this guide is referred to as "*you*." By contrast, "*we*" refers to the IBM Prospect development and technical staff who support this product.

Required Skills & Knowledge

This guide is intended for users who have knowledge and skills in the following:

- High school level mathematics
- Basic statistics
- The network that generates the data with which the IBM Prospect software works

Document Conventions

This document uses the typographical conventions shown in the following table:

Table 1: General Document Conventions

Format	*Examples*	*Description*
ALL UPPERCASE	• GPS • NULL • MYWEBSERVER	Acronyms, device names, logical operators, registry keys, and some data structures.
Underscore	See Document Conventions	For links within a document or to the Internet. Note that TOC and index links are not underscored. Color of text is determined by browser settings.
Bold	• **Note**: The busy hour determiner is...	Heading text for Notes, Tips, and Warnings.
SMALL CAPS	• The STORED SQL dialog box... • ...click VIEW... • In the main GUI window, select the FILE menu, point to NEW, and then select TRAFFIC TEMPLATE.	Any text that appears on the GUI.
Italic	• A *busy hour* is... • A web server *must* be installed... • See the *User Guide*	New terms, emphasis, and book titles.
`Monospace`	• `./wminstall` • `$ cd /cdrom/cdrom0` • `/xml/dict` • `http://java.sun.com/products/` • `addmsc.sh` • `core.spec` • Type `OK` to continue.	Code text, command line text, paths, scripts, and file names. Text written in the body of a paragraph that the user is expected to enter.

`Monospace Bold`	**`[root] # pkginfo | grep -i perl`**
	`system Perl5 On-Line Manual Pages`
	`system Perl 5.6.1 (POD Documentation)`
	`system Perl 5.6.1`
		For contrast in a code example to show lines the user is expected to enter.

`<Monospace italics>`	`# cd <oracle_setup>`	Used in code examples: command-line variables that you replace with a real name or value. These are always marked with arrow brackets.
`[square bracket]`	`log-archiver.sh  [-i][-w][-t]`	Used in code examples: indicates options.

User Publications

IBM Prospect software provides the following user publications in HTML or Adobe Portable Document Format (PDF) formats.

Table 2: IBM Prospect User Documentation

Document	Description
Administration Guide	Helps an administrator configure and support IBM Prospect core server software to analyze network performance and perform other network or database management tasks.
Administrator's Quick Reference Card	Presents the principal tasks of a IBM Prospect core server administrator in an easy-to-use format.
Expressions Technical Reference	Provides detailed information about expressions used in special calculations for reports.
Installation Guide	Instructions for installing and configuring the IBM Prospect software.
Open Interface API Guide	Describes how the Open Interface tool enhances your access to information about database peg counts and scenarios.
Performance Data Reference	Provides detailed information including entity hierarchies, peg counts, primitive calculations, and forecast expressions specific to your organization.
Release Notes	Provides technology-specific and late-breaking information about a given IBM Prospect release and important details about installation and operation.
Server Preparation Guide	Provides instructions for installing and setting up Solaris and Oracle software before you install IBM Prospect software.
Server Sizing Tool Guide	Helps an administrator use the sizing tool to calculate the system space needed for the IBM Prospect software and database.
User Guide	Provides conceptual information and procedures for using IBM Prospect software for performance and trending analysis.
Web Administration Guide	Helps administrators configure and maintain the IBM Prospect Web server using IBM Prospect administration software and command-line tools.
Web User Guide	Instructions for using the Web user interface for performance analysis.

Viewing the Desktop Client Help Publications

To view the desktop client Help publications, select a guide from the HELP menu of the IBM Prospect graphical user interface or press F1 for context-sensitive Help. To update the Help files, click the HELP menu on the IBM Prospect Explorer, and select UPDATE ALL HELP FILES.

When Help files are updated, they are downloaded automatically from the IBM Prospect server to the IBM Prospect client. A message box notifies you when this download occurs.

Viewing the Publications in PDF

All of the user publications are available in Adobe Portable Document Format (PDF). To open a PDF, you need the Adobe Acrobat Reader. You can download Adobe Acrobat Reader free of charge from the Adobe Web site. For more details about the Acrobat Reader, see the Adobe Web site http://www.adobe.com/ .

Viewing the Publications in IBM Information Center

All of the IBM Prospect publications, including Release Notes, are available online from the IBM Information Center website as follows:

http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp?topic=/com.ibm.netcool_pm.doc/IBM_Prospect_060308.htm

2 General Expressions

Expressions calculate and compare attribute values. Expressions provide support for attribute comparison, mathematical operators, type conversion, conditional operators, and functions. Attributes can either be constants (often referred to as *literals*), or Application Gateway Package Attributes.

An expression consists of a number of tokens, each of which has a type (such as Integer and String). There are four kinds of *tokens*: attributes, constants, keywords, and operators. Spaces, horizontal and vertical tabs, new lines, form feeds, and comments (collectively referred to as *white space*) are ignored except where they serve to separate tokens. Some white space is required to separate otherwise adjacent attributes, keywords, and constants.

Syntax

```
[[# comments]]

[[Unary]]
[[Binary]]
[[Conditional]]
[[Attribute]]
[[Constant]]
[[Function]]
[[( Expression )]]
```

Environment Variables

Variable	*Description*	*Comments*
WM_NEW_EXPRESSION_SYNTAX	Allows only new date and time constants (that is, dates and times enclosed in single quotes) when defined. In addition, the following characters are no longer supported in attribute names: - + * / % ^ : ?	This mechanism exists for backward compatibility. You should set WM_NEW_EXPRESSION_SYNTAX and migrate any existing attribute names and date or time constants to the new format.

Elements

Element	*R/O*	*Description*
comments	O	The character # starts a comment, which terminates with the next newline character.
Unary	O	Expression in the form: *operator expression*.
Binary	O	Expression in the form: *expression operator expression*
Conditional	O	Expression in the form: *expression ? expression : expression*
Attribute	O	Name of a gateway attribute with optional relation specifiers.
Constant	O	Literal Integer, Float, String, Date, and Time values.
Function	O	Expression in the form: functionName (*expr1, ..., exprN*)

Remarks

The following case-insensitive tokens are reserved for use as keywords in expressions:

```
AND OR (STRING) (INT) (FLOAT)
```

The following characters are used as operators for punctuation:

```
! - + * / % ^ : ? , () " ' > < =
```

The following character combinations are also used as operators:

```
== <> != <= >= && ||
```

Example

The following are expressions that range from simple logical and arithmetic expressions to more complex data extraction expressions.

Logical Expressions:

```
Id < 10
Id < 10 AND Id > 1000
Type == "CDMA"
ActivatedDate < sysdate ()
(STRING)Id = "*101*"
```

Arithmetic Expressions:

```
Id + " Activated on " + ActivatedDate
(DroppedCalls / TotalCalls) * 100
sysdate () - ActivatedDate > 0
AlarmTime < systime () - (60 * 60 * 6)
"Name: \"" + Name + "\""
```

Conditional Expressions:

```
Traffic == 0 ? "NO TRAFFIC" : (STRING)Traffic
Date - sysdate () < 0 ? "Active for " + (sysdate () - Date) + " days
: "Will go active in " + (Date - sysdate () ) + " days"
```

Function Expressions:

```
(Traffic / sum(package() .Traffic)) * 100
max(Bscs.Btss.Alarms.Priority)
sum(Bscs.Btss, (DroppedCalls / TotalCalls( * 100))
count ()
count(package (Switch)[count(package(this)[Vendor == "ERICSSON"])>0))
max(Bscs.Btss, mean(TrafficSlices[STime >= '17:00' AND ETime <
'19:00'].Traffic))
```

Related Topics

Binary Operators
Constants
Filters
Functions
Gateway Attribute
Operators
Unary Operators

- Operator (Unary)

Applies To

Unary Operators

Description

Multiplies operand by -1. Operand must be numeric.

Syntax

```
[[- operand]]
```

Examples

Expression	Result
-10	-10
--10	10
-(10 * 20)	-200

Related Topic

Operators

! Operator

Applies To

Unary Operators

Description

Returns `1` if the value of the operand is `0`. Otherwise, it returns `0`. Operand must be numeric.

Syntax

```
[[! operand]]
```

Examples

Expression	Result
!1	0

Expression	*Result*
`!0`	`1`
`!(10 < 20)`	`0`
`!(10 > 20)`	`1`

Related Topic

Operators

== *Operator (also =)*

Applies To

Logical Operators

Description

Returns `1` if *`operand1`* is equal to *`operand2`*, or `0` when the two operands are not equal.

When testing string attributes, *`operand2`* is treated as a regular expression (if it is a valid one) or a string otherwise. In order to support shell-like tests, the following substitutions are made:

The character `*` becomes `.*`

The character `?` becomes `.`

Syntax

```
[[operand1 == operand2]]
[[operand1 = operand2]]
```

Examples

Expression	*Result*
`10 == 10`	`1`
`"Hello" == "Goodbye"`	`0`
`"Expressions" == "*press*"`	`1`
`"Expressions" == "*press?"`	`0`

Related Topics

Binary Operators
Operators

!= Operator (also <>)

Applies To

Logical Operators

Description

Returns `1` if *operand1* is not equal to *operand2*, or `0` if the two operands are equal.

When testing string attributes, *operand2* is treated as a regular expression (if it is a valid one) or a string otherwise. In order to support shell-like tests, the following substitutions are made:

The character `*` becomes `.*`

The character `?` becomes `.`

Syntax

```
[[operand1 != operand2]]
```

or

```
[[operand1 <> operand2]]
```

Examples

Expression	*Result*
`10 != 10`	`0`
`"Hello" != "Goodbye"`	`1`
`"Expressions" != "*press*"`	`0`
`"Expressions" != "*press?"`	`1`

Related Topics

Binary Operators
Operators

< Operator

Applies To

Logical Operators

Description

Returns `1` if *operand1* is less than *operand2*, and `0` in all other cases.

Syntax

```
[[operand1 < operand2]]
```

Examples

Expression	*Result*
`10 < 10`	`0`
`10 < 20`	`1`
`"aardvark" < "zebra"`	`1`
`'Nov 22 1963' < 'Nov 22 1963'`	`0`

Related Topics

Binary Operators
Operators

<= *Operator*

Applies To

Logical Operators

Description

Returns `1` if *operand1* is less than or equal to *operand2*, and `0` in all other cases.

Syntax

```
[[operand1 <= operand2]]
```

Examples

Expression	*Result*
`10 <= 10`	`1`
`20 <= 10`	`0`
`'Nov 22 1963' <= 'Nov 22 1963'`	`1`

Related Topics

Binary Operators
Operators

> Operator

Applies To

Logical Operators

Description

Returns `1` if operand1 is greater than operand2, and `0` in all other cases. > is a binary operator that expresses a logical relationship.

Syntax

```
[[operand1 > operand2]]
```

Examples

Expression	*Result*
`10 > 5`	`1`
`10 > 10`	`0`
`'Nov 22 1963' > 'Nov 22 1963'`	`0`
`"aardvark" > "zebra"`	`0`

Related Topics

Binary Operators
Operators

>= Operator

Applies To

Logical Operators

Description

Returns `1` if *`operand1`* is greater than or equal to *`operand2`*, and `0` in all other cases.

Syntax

```
[[operand1 >= operand2]]
```

Examples

Expression	*Result*
`10 >= 5`	`1`
`10 >= 10`	`1`
`'Nov 22 1963' >= 'Nov 22 1963'`	`1`
`"aardvark" >= "zebra"`	`0`

Related Topics

Binary Operators
Operators

+ Operator

Applies To

Arithmetic Operators

Description

Adds the values of *operand1* and *operand2*. If either operand is a string, then the remaining operand is converted to a string.

If you add an integer *I* to a date *D*, then the result is a Date *I* days later than *D*.

If you add an integer *I* to a time *T*, then the result is a Time *I* seconds later than *T*.

Syntax

```
[[operand1 + operand2]]
```

Result Type

The following table specifies valid operand types and their result types. When commutative = 1, you can switch *operand1* type and *operand2* type, and they produce the same result type.

Type1	*Type2*	*Result Type*	*Commutative*
Int	Int	Int	1
Int	Float	Float	1

Type1	*Type2*	*Result Type*	*Commutative*
Int	String	String	1
Float	Float	Float	1
Float	String	String	1
Date	String	String	1
Date	Int	Date	1
Time	String	String	1
Time	Int	Time	1
String	String	Time	1

Examples

Expression	*Result*
`2 + 0`	`2`
`2 + 2`	`4`
`2 + -1`	`1`
`-2 + -2`	`-4`
`'Sep 10 1998' + 7`	`'Sep 17 1998'`
`"String" + " + " + "String"`	`String + String`
`"String + Int " + 10`	`String + Int 10`
`"String + Float " + 10.2`	`String + Float 10.2`
`"String + Date " + '10/10/98'`	`String + Date 10/10/98`
`"String + Time " + '12:00:00'`	`String + Time 12:00:00`

Related Topics

Binary Operators
Operators

- *Operator (Binary)*

Applies To

Arithmetic Operators

Description

Subtracts the values of *operand2* from *operand1*.

If you subtract an integer *I* from a date *D*, then the result is a Date *I* days earlier than *D*.

If you subtract an integer *I* from a time *T*, then the result is a Time *I* seconds earlier than *T*.

Subtracting a Date from a Date results in an integer that is the number of days between the two dates. If *operand1* < *operand2*, then the result is a negative value.

Syntax

```
[[operand1 - operand2]]
```

Result Type

The following table specifies valid operand types and their result types. When commutative = 1, you can switch *operand1* type and *operand2* type, and they produce the same result type.

Type1	*Type2*	*Result Type*	*Commutative*
Int	Int	Int	1
Int	Float	Float	1
Float	Float	Float	1
Date	Date	Int	1
Date	Int	Date	0
Time	Int	Time	0

Examples

Expression	*Result*
`2 - 0`	`2`
`2 - 2`	`0`
`2 - (-1)`	`3`
`-2 - 2`	`-4`
`'Sep 10 1998' - 'Sep 3 1998'`	`7`
`'Sep 3 1998' - 'Sep 10 1998'`	`-7`

Expression	Result
'Sep 10 1998' - 7	'Sep 3 1998'
'12:00:00' - 60	'11:59:00'

Related Topics

Binary Operators
Operators

* Operator

Applies To

Arithmetic Operators

Description

Multiplies the value of *operand1* by the value of *operand2*.

Syntax

```
[[operand1 * operand2]]
```

Result Type

The following table specifies valid operand types and their result types. When commutative = 1, you can switch *operand1* type and *operand2* type, and they produce the same result type.

Type1	Type2	Result Type	Commutative
Int	Int	Int	1
Int	Float	Float	1
Float	Float	Float	1

Examples

Expression	Result
2 * 0	0
2 * 2	4

Expression	*Result*
`2 * -1`	`-2`
`-2 * -2`	`4`

Related Topics

Binary Operators
Operators

/ Operator

Applies To

Arithmetic Operators

Description

Divides the value of *operand1* by the value of *operand2*. If the value of *operand2* is `0`, then a warning is produced and the result of the division is `NULL`.

If both operands are integers, then an integer division is performed and the following equation is satisfied:

```
((a / b) * b) + (a % b) = a
```

A floating point division can be forced by explicitly casting one of the values to a float.

Syntax

```
[[operand1 / operand2]]
```

Result Type

The following table specifies valid operand types and their result types. When commutative = 1, you can switch operand1 type and operand2 type, and they produce the same result type.

Type1	*Type2*	*Result Type*	*Commutative*
Int	Int	Int	1
Int	Float	Float	1
Float	Float	Float	1

Examples

Expression	*Result*
`5 / 2`	`2`
`5.0 / 2`	`2.5`
`4 / 2`	`2`
`4 / 0`	`NULL`
`5 / (FLOAT)2`	`2.5`

Related Topics

Binary Operators
Operators

% Operator

Applies To

Arithmetic Operators

Description

Yields the remainder of dividing the value of *operand1* by the value of *operand2*. Both operands must be integers. If the value of *operand2* is `0`, then a warning is produced and the result of the division is `NULL`.

The modulus operator satisfies the following equation:

```
((a / b) * b) + (a % b) = a
```

Syntax

```
[[operand1 % operand2]]
```

Result Type

The following table specifies valid operand types and their result types. When commutative = 1, you can switch *operand1* type and *operand2* type, and they produce the same result type.

Type1	*Type2*	*Result Type*	*Commutative*
Int	Int	Int	1

Examples

Expression	*Result*
`5 % 2`	`1`
`4 % 2`	`0`
`4 % 0`	`NULL`

Related Topics

Binary Operators
Operators

^ *Operator*

Applies To

Arithmetic Operators

Description

Raises the value of *operand1* to the power of *operand2*. If the value of *operand1* is less than zero, then the value of *operand2* must be integral. If it is greater or equal to zero, then it is cast to an integer. If *operand1* is zero and *operand2* is negative, then a warning is produced and `NULL` is returned.

Syntax

```
[[operand1 ^ operand2]]
```

Result Type

The following table specifies valid operand types and their result types. When commutative = 1, you can switch *operand1* type and *operand2* type, and they produce the same result type.

Type1	*Type2*	*Result Type*	*Commutative*
`Int`	`Int`	`Float`	1
`Int`	`Float`	`Float`	1
`Float`	`Float`	`Float`	1

Examples

Expression	*Result*
`0 ^ 0`	`0`
`1 ^ 0`	`1`
`0 ^ -1`	`NULL`

Related Topics

Binary Operators
Operators

() Operator (Cast)

Applies To

Unary Operators

Description

Converts an operand from one type to another. Any attribute can be converted to a string, but only numeric attributes can be cast to integers and floats. Casting from a float to an integer truncates the fractional part.

Syntax

```
[[(STRING) | (INT) | (FLOAT) operand]]
```

Examples

Expression	*Result*
`(STRING) '10/10/10'`	`10/10/10 # locale specific`
`(FLOAT)10`	`10.0`
`(INT) 10.245`	`10`

Related Topics

Operators

abs

Applies To

Math Functions

Description

Returns the absolute value of numeric attribute *N*. The result type is the same as the type of the argument. (for instance, If *N* is an integer, then an integer is returned.)

Syntax

```
NUMBER abs(NUMBER N)
```

Examples

Expression	*Result*
abs(10)	10
abs(-10)	10
abs(1.0 * 10)	10.000000
abs(1.0 * -10)	10.000000

AND Operator (also &&)

Applies To

Logical Operators

Description

Returns 1 if both operands are nonzero, and 0 in all other cases. Both operands must be numeric.

Syntax

```
[[operand1 AND operand2]]
```

or

```
[[operand1 && operand2]]
```

Examples

Expression	*Result*
`1 And 1`	`1`
`0 And 1`	`0`
`0 && 0`	`0`

Related Topics

Binary Operators
Operators

arcCos

Applies To

Trig Functions

Description

Computes the principal value of the arc cosine of *N*. If *N* does not fall within the range `[-1, 1]`, then `NULL` is returned. The value returned is in the range `[0, pi ()]`.

Syntax

```
FLOAT arcCos(NUMBER N)
```

Examples

Expression	*Result*
`arcCos(1)`	`0.000000`
`arcCos(-1)`	`3.141593`
`arcCos(0.5)`	`1.047198`
`arcCos(0)`	`1.570796`
`arcCos(-0.5)`	`2.094395`

Related Topics

arcTan
arcSin
cos

parameterAsString
radians

arcSin

Applies To

Trig Functions

Description

Computes the principal value of the arc sine of *N*. If *N* does not fall within the range `[-1, 1]`, then `NULL` is returned. The value returned is in the range `[-pi()/2, pi()/2]`.

Syntax

```
FLOAT arcSin(NUMBER N)
```

Examples

Expression	*Result*
`arcSin(1)`	`1.570796`
`arcSin(-1)`	`-1.570796`
`arcSin(-0.5)`	`-0.523599`
`arcSin(0.5)`	`0.523599`
`arcSin(0)`	`0.000000`

Related Topics

arcCos
arcTan
parameterAsString
radians
sin

arcTan

Applies To

Trig Functions

Description

Computes the principal value of the arc tangent of *N*.

The value returned is in the range `[-pi () / 2, pi () / 2]`

Syntax

```
FLOAT arcTan(NUMBER N)
```

Examples

Expression	*Result*
`arcTan(1)`	`0.785398`
`arcTan(-1)`	`-0.785398`
`arcTan(-0.5)`	`-0.463648`
`arcTan(0.5)`	`0.463648`
`arcTan(0)`	`0.000000`

Related Topics

arcCos
arcSin
parameterAsString
tan
radians

Arithmetic Operators

Applies To

Binary Operators

Description

Returns the result of a mathematical expression. The type of the result is based on the type of the two operands. Arithmetic Operators perform basic implicit typecasting.

Syntax

```
[[operand1 operator operand2]]
```

Related Topics

+ Operator
- Operator (Binary)
* Operator
/ Operator

% Operator
^ Operator

average

Applies To

Package Functions

Description

Alias for mean. See mean for details.

Syntax

```
FLOAT average(RELATION_ATTR A)
FLOAT average(RELATION R, EXPRESSION E)
```

Related Topics

mean

Binary Operators

Applies To

Operators

Description

A logical or arithmetic expression that takes two operands. Binary operators perform basic implicit type casting, as described in the section Operators.

Logical binary operators return either `1` (TRUE) or `0` (FALSE). Arithmetic binary operators return a value based on the types of its two operands.

Syntax

```
[[operand1 operator operand2]]
```

Related Topics

Arithmetic Operators
Logical Operators

ceil

Applies To

Math Functions

Description

Returns the smallest integer value not less than *N*, expressed as a float.

Syntax

```
FLOAT ceil(NUMBER N)
```

Examples

Expression	*Result*
`ceil(1.3)`	`2.000000`
`ceil(1.5)`	`2.000000`
`ceil(1.7)`	`2.000000`
`ceil(-1.3)`	`-1.0000000`
`ceil(-1.5)`	`-1.0000000`
`ceil(-1.7)`	`-1.0000000`

Related Topics

floor
round
trunc

concat

Applies To

String Functions

Description

Concatenates the string representations of the specified attributes. This is the function equivalent of `a1 + a2 + .... + aN`.

Syntax

```
STRING concat(ATTR a1, ATTR a2, ..., aN)
```

Examples

Expression	*Result*
`concat("Current Date is ", sysdate ())`	`Current Date is 09/25/98`
`concat("Hello ",  2, " you")`	`Hello 2 you`

Related Topic

+ Operator

Conditional Operator (? :)

Applies To

General Expressions

Description

Allows a simple IF.THEN.ELSE statement. That is, if *`test_expr`* returns a nonzero result, then the value of *`expr1`* is used. Otherwise, the value of *`expr2`* is used. Both *`expr1`* and *`expr2`* must return the same type.

Conditionals can be nested to any depth.

Syntax

`[[`*`test_expr`* `?` *`expr1`* `:` *`expr2`*`]]`

Examples

Expression	*Result*
`isNULL(10 / 0) ? "NULL" : (STRING) (10 / 0)`	`NULL`
`10  <  0 ?  "true"`	`false`

The following expression is invalid because `NULL` and `10 / 2` are of different types. (`NULL` is a string and `10 / 2` is an integer.)

```
isNull (10 / 0) ? "NULL" : (10 / 2) = "NULL"
```

Constants

Applies To

General Expressions

Description

Constants (often referred to as literals) are values that do not change. There are five types of constants: integer, float, string, date, and time.

Syntax

```
[[ Integer | Float | String | Date | Time ]]
```

Elements

Element	*Description*
Integer Constant	Consists of a sequence of decimal digits.
Float Constant	Consists of a sequence of decimal digits and exactly one period '.'
String Constant	Consists of a sequence of characters surrounded by double quotes.
Date Constant	Consists of a sequence of characters surrounded by single quotes.
Time Constant	Consists of a sequence of characters surrounded by single quotes.

Remarks

A time constant must contain the characters colon (`:`) or period (`.`) or the strings `am` or `pm` (case insensitive). Any constant enclosed between single quotation marks that is not a time is assumed to be a date.

When the environment variable `WM_NEW_EXPRESSION_SYNTAX` is defined, date and time constants must be enclosed between single quotation marks.

When the environment variable `WM_NEW_EXPRESSION_SYNTAX` is not defined, then single quotes are not required for date or time formats.

Note: You should set `WM_NEW_EXPRESSION_SYNTAX` and modify any existing filters to use the new date and time formats.

Examples

Integer Constants:

```
10
```

```
9999
1024556
```

Float Constants:

```
10.2
.234
1.
1.045
0.45
```

String Constants:

```
"Hello World"
"Number1"
"12"
```

Date Constants that are valid in a US English speaking locale:

```
'January 9 1962'
'Jan 9, 62'
'1/9/62'
'1-9-62'
'09Jan62'
'010962'
```

Time Constants:

```
'12:30.45pm'
'1:10 AM'
'13:45:30'
'12.30.45pm'
'PM 3:15'
```

Old Date Constant Formats:

```
1/9/62
1-9-62
```

Old Time Constant Format

```
13:45:30
```

cos

Applies To

Trig Functions

Description

Computes the cosine of *N* (in radians), expressed as a float.

Syntax

```
FLOAT cos(NUMBER N)
```

Examples

Expression	*Result*
cos(pi ())	-1.000000
cos(0)	1.000000
cos(radians (90))	0.000000
cos(radians (270))	0.000000
cos(pi() /2)	0.000000

Related Topics

arcCos
degrees
radians
sin
tan

count

Applies To

Package Functions

Description

Returns the number of objects found in the package or relations specified by `R`. If `R` is not specified, count returns the number of objects in the current package.

Syntax

```
INT count([RELATION R])
```

Examples

The following expressions are all equivalent when evaluating an `Msc` Object.

Expression	*Result*
count()	3 # Current package

Expression	*Result*
`count(package())`	`3  # Current package`
`count(package(this))`	`3  # Current package`
`count(package(Msc))`	`3  # Named package`
`count(package()[Vendor = "LUCENT"].Bscs.Btss)`	`7`
`count(package()[Vendor = "ERICS-SON"].Bscs.Btss`	`0`
`count(package(Bsc)[Vendor != Msc.Vendor])`	`1`

`count(Bscs.Btss)`	`MSC1:  4`
	`MSC2:  3`
	`MSC3:  3`

Related Topics

Functions
sum

Date Functions

Applies To

Functions

Description

Functions that operate on date parameters. Time parameters are automatically converted to date types.

Related Topics

dateToString
dayOfYear
dayOfMonth
firstDayOfMonth
getTimeFromDate
isLeapYear
month
monthName
nextDate
previousDate
stringToDate
weekDayName
weekDay

year
validDate

dateToString

Applies to

Date Functions

Description

Returns a string created by applying format to D. If F is not given, default for locale is used. If D is NULL or invalid, or F is NULL or, invalid, NULL is returned. Format is same as `strftime(3C)`.

Syntax

```
STRING dateToString(DATE|TIME D [,STRING F])
```

Examples

Expression	*Result*
`dateToString ('1/9/99', "%B %b %d")`	`January 9, 1999'`

Related Topics

stringToDate
timeToString

dayOfMonth

Applies to

Date Functions

Description

Returns the day of the month (1-31) for D. NULL if D is NULL or invalid.

Syntax

```
INT dayOfMonth(DATE|TIME D)
```

Examples

Expression	*Result*
`dayOfMonth('1/9/99')`	`9`

Related Topics

dayOfYear
firstDayOfMonth

dayOfYear

Applies to

Date Functions

Description

Returns the day of the year (1-366) for D. NULL if D is NULL or invalid.

Syntax

```
INT dayOfYear(DATE|TIME D)
```

Examples

Expression	*Result*
`dateOfYear ('2/9/99')`	`40`

Related Topics

dayOfMonth

decode()

The `decode()` function is equivalent to the `IF:ELSE IF:ELSE` or `SWITCH` statements. It is interpreted as `IF:THEN:ELSEIF:THEN:ELSE`. You can have multiple `ELSEIF` statements to achieve the required evaluations.

Syntax

```
decode(test_expr, expr1, return1, expr2, ..., exprN, returnN,returnX)
```

Examples

Expression	*Result*
`decode(NBType, "I", HOVERCNT, null-Int())`	`If NBType is internal then the int value of HOVERCNT is returned otherwise a NULL value of type int is returned.`

degrees

Applies To

Trig Functions

Description

Converts *N* radians to degrees. Satisfies the equation:

`degrees(N) == N / (pi() / 180.0)`

Syntax

`FLOAT degrees(NUMBER N)`

Examples

Expression	*Result*
`degrees(pi())`	`180.000000`
`degrees(2 * pi())`	`360.000000`

Related Topics

parameterAsString
radians

distance

Applies To

Geometry Functions

Description

Calculates the distance between the two points (X1, Y1) and (X2, Y2). The attributes making up each point (for example, X1 and Y1) must be of the same type, either NUMBER or STRING. However, it is valid for the points to be of different types (for example, (X1, Y1) could be NUMBERs and (X2, Y2) could be STRINGs).

If the attribute type is STRING, the attributes are interpreted as Longitude/Latitude coordinates and are converted to a UTM grid coordinate using the current UTM zone. Please refer to the application-specific documentation for details on how to set the UTM zone.

Syntax

```
FLOAT distance(ATTR X1, ATTR Y1, ATTR X2, ATTR Y2)
```

Examples

```
distance(UTM_X1, UTM_Y1, UTM_X2, UTM_Y2)
distance(100, 100, 200, 200)
distance("23'57'23W", "10'15'17N", "22'57'23W", "11'15'17N")
distance("23'57'23W", "10'15'17N", 10.234, 5.987)
```

Related Topics

Geometry Functions
hour

elemprotect

Description

Computes the expression prior to aggregating over element. Behaves exactly as `protect` for Classic Client.

Syntax

```
elemprotect (expression)
```

Examples

In the following examples, `UDC X` aggregation type is:

- Element aggregation `(E) type = Avg`
- Time aggregation `(T) type = Avg`

`A`, `B`, and `C` are representative field names and have aggregation of `Sum` for *time* and *element*.

Therefore, Avg_B^E represents the element aggregation from `Peg B (Sum)`.

Example 1:

Standard expression:

```
UDC X = (A+B) * C
```

This is evaluated as follows:

```
result = (Sum_A^E(Sum_A^T(A)) + Sum_B^E(Sum_B^T(B))) * Sum_C^E(Sum_C^T(C))
```

Example 2:

Same expression using `elemprotect`:

```
UDC X = elemprotect((A+B) * C)
```

This is evaluated as follows:

```
result = Avg_X^E (((Sum_A^T(A) + Sum_B^T(B)) * Sum_C^T(C))
```

Related Topics

protect

exp

Applies To

Math Functions

Description

Calculates the exponential of *N*, expressed as a float. Satisfies the equation:

```
exp(N) = exp(1) ^ N
```

Syntax

```
FLOAT exp(NUMBER N)
```

Examples

Expression	*Result*
`exp(1)`	`2.718282`
`exp(1) == exp(1) ^ 1`	`1`
`exp(2) == exp(1) ^ 2`	`1`

Related Topic

ln

Filters

Applies To

General Expressions

Description

A filter is an expression that returns a numeric value to indicate TRUE or FALSE. FALSE is represented by the value `0`; TRUE is represented by the value `1`.

Examples

Expression	*Result*
`10 > 20`	`0`
`10 < 20`	`1`
`length("hello") == 5`	`1`
`'10/10/98' < '10/10/97'`	`0`

firstDayOfMonth

Applies to

Date Functions

Description

Returns the day of the year (1-366) for the first day of month M (1-12) for the year of D. NULL if D is NULL or invalid, or if M is not in the range 1-12.

Syntax

```
INT firstDayOfMonth(DATE|TIME D, INT M)
```

Examples

Expression	*Result*
`firstDayOfMonth('1/9/99', 3)`	`60`

Related Topics

dayOfMonth

firstIndex

Applies To

String Functions

Description

Returns the start index of the first occurrence of *T* in test *S*. The first character in *S* has an index of 0. If *T* cannot be found in *S*, then -1 is returned.

Syntax

```
INT firstIndex(STRING S, STRING T)
```

Examples

Expression	*Result*
`firstIndex("Hello World Hello", "Hello")`	0
`firstIndex("Hello World Hello", "World")`	6
`firstIndex("Hello World Hello", "Today")`	-1

Related Topics

lastIndex
subString

floor

Applies To

Math Functions

Description

Largest integer value not greater than *N*, expressed as a float.

Syntax

```
FLOAT floor(NUMBER N)
```

Examples

Expression	*Result*
`floor(1.3)`	`1.000000`
`floor(1.5)`	`1.000000`
`floor(1.7)`	`1.000000`
`floor(-1.3)`	`-2.0000000`
`floor(-1.5)`	`-2.0000000`
`floor(-1.7)`	`-2.0000000`

Related Topics

ceil
round
trunc

Functions

Applies To

General Expressions

Description

A function is a built-in algorithm that takes zero or more comma-separated arguments and returns a (possibly `NULL`) result.

The number and type of the arguments are specific to each function and are described in the function reference pages. In general, the arguments to a function can be any valid expression, including another function call.

Syntax

```
[[function_name (arg1, arg2, ..., argN )]]
```

Elements

Elements	*Description*
function_name	Function names begin with a letter and continue with any alphanumeric character or underscore.

argX	Any valid expression. Must match the type required by the function.

Examples

In the following example, the function `max` is specified to discover the maximum number of Priority 1 alarms.

Expression	*Result*
`isNull (10 / (2 - 2))`	`1`
`sqrt(2 ^ 2)`	`2.0`
`subString("Hello World", 6, 5)`	`World`
`length(subString("Hello World", 6, 5))`	`5`

Related Topics

Date Functions
Gateway Attribute
Geometry Functions
Ip Functions
Math Functions
Package Functions
String Functions
System Functions
Time Functions
Trig Functions

Gateway Attribute

Applies To

General Expressions

Description

Allows users to access attributes from the current Application Gateway object being processed (for example, the object for the current form being displayed). The attributes can contain relation specifiers with optional filters as well as an attribute name, and can also access objects in other packages.

Attribute Tokenizing

Attribute tokens must begin with a letter or underscore character. They cannot be white space (except within parenthesis (`()`) and square brackets (`[]`)), or angle-brackets (<>) and equal sign (=) characters.

Any characters are supported within parenthesis (`()`) and square brackets (`[]`). When the first parenthesis (`(`) character is encountered, the parser checks for keywords and function names, so they are not permitted. For example, the following are invalid attribute names:

```
and(rev) .Id
sysdate(rev) .Id
```

If the environment variable `WM_NEW_EXPRESSION_SYNTAX` is defined, then the following characters are not supported within attribute names:

```
- + * / % ^ : ?
```

Note: This mechanism is only provided for compatibility with earlier versions of IBM Prospect. Users must set `WM_NEW_EXPRESSION_SYNTAX` and migrate any existing attribute names to use valid characters. Support for the old syntax will be removed in the next release.

Simple Attributes

Any attribute within the current package is a simple attribute. For example, assuming we are evaluating a Switch object and given the package definition:

```
PACKAGE Switch
{
     STRING Id
     STRING Vendor
     DATE Activated
}
```

the following are examples of simple attribute tokens:

```
Id
Vendor
Activated
```

Attributes Using One-to-One Relations

If an attribute name is prefixed with a relation attribute name from the current package, then the relation is traversed and the attribute is retrieved from the object at the other end of the relation.

For example, assuming you are evaluating a `Bsc` object using the previously defined package Switch and

```
PACKAGE Bsc
{
```

```
        STRING    Id
        STRING    Vendor
        DATE      Activated
        RELATION  Switch Switch REVERSE Bscs
}
```

then the following attribute tokens would access attributes of the Switch object to which the Bsc object is related:

```
Switch.Id
Switch.Vendor
Switch.Activated
```

Accessing Every Object in a Package

It can be useful to access every object in a package, rather than just the ones in a relation for the current object. This is supported by using the following syntax:

```
package()                 # Current Package
package(this)             # Current Package
package(package_name)      # Package package_name
```

It provides a pseudo-relation that contains every object in the current or specified package. Based on this syntax, the following are valid attribute tokens:

```
package() .Id
package(this) [Vendor  =  "vendor"] .Id
package(Bsc)   [Vendor   =  "vendor"] .Btss [Activated < '10/10/98'] .Id
```

Related Topic

Constants

Geometry Functions

Applies To

Functions

Description

Functions that perform geometric and spatial calculations.

Related Topics

distance
inPolygon

getDateFromTime

Applies to

Time Functions

Description

Returns a date value from T. Returns NULL if T is NULL or not valid.

Syntax

```
DATE getDateFromTime(TIME T)
```

Examples

Expression	*Result*
`getDateFromTime('09Jan1999 1:10AM') == '09Jan1999'`	1

Related Topics

getTimeFromDate

getenv

Applies To

System Functions

Description

Returns the value of the environment variable `S`. If `S` is not defined and optional argument `D` is specified, then `D` will be returned; otherwise `NULL` will be returned.

Note: that it is more efficient to use the optional argument `D` to supply a default value than it is to use the nullValue function to test the result of the getenv function.

Syntax

```
STRING getenv(STRING S [, STRING D])
```

Examples

Assume that the following environmental variable is set:

```
GREETING = "Hello"
```

Expression	*Result*
`getenv("GREETING")`	`"Hello"`
`getenv("GREETING", "Good Morning")`	`"Hello"`
`getenv("MY_GREETING")`	`NULL`
`getenv("MY_GREETING", "Good Morning")`	`"Good Morning"`

Related Topic

nullValue

getTimeFromDate

Applies to

Date Functions

Description

Returns the time value with the same date as D and the time of day 00:00:00. NULL if D is not valid or NULL.

Syntax

```
TIME getTimeFromDate(DATE D)
```

Examples

Expression	*Result*
`getTimeFromDate('1/9/99') == '1/9/99 00:00:00'`	`1`

Related Topics

getDateFromTime

hour

Applies to

Time Functions

Description

Returns the hour of T. Returns NULL if T is NULL or invalid.

Syntax

```
INT hour(TIME T)
```

Examples

Expression	*Result*
`hour('12.30.45pm')`	`12`

Related Topics

hourGMT
minute
second

hourGMT

Applies to

Time Functions

Description

Returns the hour of T in Coordinated Universal Time (UTC) (The internationally recognized name for Greenwich Mean Time (GMT)). Returns NULL if T is NULL or invalid.

Syntax

```
INT hourGMT(TIME T)
```

Examples

Assume time zone is Pacific Standard Time (PST)

Expression	*Result*
`hourGMT('13.30.45pm')`	`20`

Related Topics

hour
minuteGMT

inPolygon

Applies To

Geometry Functions

Description

Returns `1` if the point `(X, Y)` is inside the polygon `P`, and returns `0` in all other cases including polygon `P` not existing. Attributes `X` and `Y` must be of the same type (either a number or a string).

If the type is `STRING`, the `X`, `Y` attributes are interpreted as Longitude/Latitude coordinates and will be converted to a UTM grid coordinate using the current UTM zone. Please refer to application-specific documentation on how to set the UTM zone, and how to create named polygons.

Syntax

```
int inPolygon(ATTR X, ATTR Y, STRING P)
```

Examples

```
inPolygon(UTM_X, UTM_Y, "TestPolygon1")
inPolygon("10.234", "5.987", "TestPolygon1")
inPolygon("23'57'23W", "10'15'17N", "TestPolygon1")
```

Related Topics

distance
Geometry Functions

ipAddressClass

Applies to

Ip Functions

Description

Returns the name of the class that ADDR belongs to. This is determined by examining the high-order bits. STD 5 (RFC 791)

High order bits	*Class*	*Address range*
0	A	1.n.n.n-126.n.n.n
10	B	128.1.n.n-191.254.n.n
110	C	192.1.1.n-223.254.254.n

High order bits	*Class*	*Address range*
1110	D	224.n.n.n-239.n.n.n
111	E	240.n.n.n-254.n.n.n

Loopback addresses (127.n.n.n) are class A. Classes D and E are reserved extensions and are not normally used.

Returns NULL is ADDR is NULL or invalid.

Syntax

```
STRING   ipAddressClass(STRING ADDR)
```

Examples

Expression	*Result*
`ipAddressClass("204.57.184.198")`	`C`
`ipAddressClass("www.vallent.com")`	`NULL`
`ipAddressClass("172.24.129.9")`	`B`

Related Topics

ipIsValid

ipAddressToInt

Applies to

Ip Functions

Description

Converts the 32-bit address to a 32-bit integer corresponding to an unsigned big-endian integer with the same bytes as the address. If ADDR is not a valid address returns NULL.

Syntax

```
INT ipAddressToInt(STRING ADDR)
```

Examples

Expression	*Result*
`ipAddressToInt("204.57.184198")`	`3426334918`
`ipAddressToInt("www.vallent.com")`	`NULL`

Related Topics

ipIntToAddress

ipApplyNetmask

Applies to

Ip Functions

Description

Returns the net address of ADDR by applying MASK. If MASK is not given, use default for class of ADDR. MASK can be given as a STRING in form of nnn.nnn.nnn.nnn where each nnn is an integer from 0 to 255, or it can be given as the number of network bits, from 0 to 32. If either the ADDR or mask is invalid NULL is returned. If the class of ADDR is not A, B, or C, MASK must be supplied.

Syntax

```
STRING  ipApplyNetmask(STRING ADDR [, STRING | INT MASK])
```

Examples

Expression	*Result*
`ipApplyNetmask("10.57.180.169", "255.255.176.0")`	`10.57.176.0`
`ipApplyNetmask("10.57.180.169", 20)`	`10.57.176.0`
`ipApplyNetmask("10.57.180.169")`	`10.0.0.0`
`ipApplyNetmask("254.10.15.1", "255.255.0.0")`	`254.10.0.0`
`ipApplyNetmask("254.10.15.1")`	`NULL`
`ipApplyNetmask("10.57.180.169", "255.255.0.255")`	`NULL`
`ipApplyNetmask("10.57.180.169", 33)`	`NULL`

Related Topics

ipAddressClass
ipIsValid

ipApplyNetmaskMerge

Applies to

Ip Functions

Description

Returns the IP Address using NET for the high-order (network) bits and HOST for the low-order (host) bits using MASK to define which bits to use for the network. If host is not given 0.0.0.0 is used. MASK must be a STRING of four dotted bytes in the form nnn.nnn.nnn.nnn where nnn is an integer from 0 to 255 or an INT from 0 to 32. If MASK is not given, and NET is class A, B, or C, the default net mask for that class is used. If NET, HOST, or MASK is invalid, or if NET is not class A, B, or C, and MASK is not given, returns NULL.

Syntax

```
STRING   ipNetmaskMerge(STRING NET [,STRING HOST [,STRING | INT MASK]])
```

Examples

Expression	*Result*
`ipNetmaskMerge("10.57.180.169")`	`10.0.0.0`
`ipNetmaskMerge("172.16.1.1", "192.168.179.12")`	`172.16.176.0`
`ipNetmaskMerge("172.16.1.1","192.168.179.12","255.0.0.0")`	`172.168.176.0`
`ipNetmaskMerge("172.16.1.1", "192.168.179.12", 8)`	`172.168.176.0`
`ipNetmaskMerge("10.57.180.169", 20)`	`10.57.176.0`
`ipNetmaskMerge("254.10.15.1", "255.255.0.0")`	`254.10.0.0`
`ipNetmaskMerge("254.10.15.1")`	`NULL`
`ipNetmaskMerge ("10.57.180.169","192.168.179.12","255.255.0.255")`	`NULL`
`ipNetmaskMerge("10.57.180.169","192.168.179.12",33)`	`NULL`

Related Topics

ipAddressClass
ipIsValid

ipByteOfAddress

Applies to

Ip Functions

Description

Returns the value of the byte INDEX from ADDR. Default value of BYTE is 3. Returns NULL if ADDR is not valid.

Syntax

```
INT    ipByteOfAddress(STRING ADDR [, INT INDEX])
```

Examples

Expression	*Result*
`ipByteOfAddress("204.57.184.198")`	`198`
`ipByteOfAddress("204.57.184.198",`	`204`
`ipByteOfAddress("www.vallent.com",1)`	`NULL`

Related Topics

ipAddressToInt

ipBytesToAddress

Applies to

Ip Functions

Description

Creates an address using the bytes given. If a byte is not given it defaults to 0. Each byte must be an INT from 0 to 255 or a STRING from 0 (or 000) to 255. Any invalid bytes will result in NULL.

Syntax

```
STRING    ipBytesToAddress([ATTR B0 [, ATTR B1 [, ATTR B2 [, ATTR B3]]]])
```

Examples

Expression	*Result*
`ipBytesToAddress()`	`0.0.0.0`
`ipBytesToAddress("127", 0, "000", 1)`	`127.0.0.1`
`ipBytesToAddress("255", "255", 255)`	`255.255.255.0`
`ipBytesToAddress("127", 0, "0", 327)`	`NULL`

ipCommonNetmask

Applies to

Ip Functions

Description

Returns the common network bits for the two addresses in the form of a net mask. If either of the two addresses is invalid, returns NULL.

Syntax

```
STRING    ipCommonNetmask(STRING ADDR1, STRING ADDR2)
```

Examples

Expression	*Result*
`ipCommonNetmask("10.57.180.169", "10.57.184.198")`	`255.255.176.0`
`ipCommonNetmask("10.57.176.169", "267.0.0.1")`	`NULL`

Related Topics

ipCommonNetmaskBits
ipIsValid

ipCommonNetmaskBits

Applies to

Ip Functions

Description

Returns the common network bits for the two addresses as number of common high-order bits. If either of the two addresses is invalid, returns NULL.

Syntax

```
INT    ipCommonNetmaskBits(STRING ADDR1, STRING ADDR2)
```

Examples

Expression	*Result*
`ipCommonNetmaskBits("10.57.180.169", "10.57.184.198")`	`20`
`ipCommonNetmaskBits("10.57.176.169", "267.0.0.1")`	`NULL`

Related Topics

ipCommonNetmask
ipIsValid

ipDefaultNetmask

Applies to

Ip Functions

Description

Returns the default netmask for the class of ADDR. NULL if ADDR is not valid, or is class D or E. Internet Standards 5 (Request for Comments 791) — STD5 (RFC 791).

Class Netmask:

A255.0.0.0
B255.255.0.0
C255.255.255.0

Syntax

```
STRING    ipDefaultNetmask(STRING ADDR)
```

Examples

Expression	*Result*
`ipDefaultNetmask("204.57.184.198")`	`255.255.255.0`
`ipDefaultNetmask("127.0.0.1")`	`255.0.0.0`
`ipDefaultNetmask("255.0.0.0")`	`NULL`

Related Topics

ipAddressClass
ipDefaultNetmaskBits
ipIsValid

ipDefaultNetmaskBits

Applies to

Ip Functions

Description

Returns the default net mask as number of bits for the class of ADDR. NULL if ADDR is not valid, or is class D or E. This number includes the class bits plus network bits. Internet Standards 5 (Request for Comments 791) — STD5 (RFC 791).

Class Netmask

A8(1+7)
B16(2+14)
C24(3+21)

Syntax

```
INT    ipDefaultNetmaskBits(STRING ADDR)
```

Examples

Expression	*Result*
`ipDefaultNetmaskBits("127.0.0.1")`	`8`
`ipDefaultNetmaskBits("www.vallent.com")`	`NULL`
`ipDefaultNetmaskBits("204.57.184.198")`	`24`

Related Topics

ipAddressClass
ipDefaultNetmask
ipIsValid

Ip Functions

Applies to

Date Functions

Description

Functions that operate on IP addresses. IP addresses are strings in the format of a.b.c.d where a, b, c, and d are numbers 0-255. These expressions only work for IPv4 (32 bit) addresses and will not work with IPv6 (128 bit). Details of the IPv4 specification can be found in on the Internet:
STD 2 / RFC 1700 (http://www.faqs.org/rfcs/std/std2.html)
STD 5 / RFC 791 (http://www.faqs.org/rfcs/std/std5.html)
BCP 5 / RFC 1918 (http://www.faqs.org/rfcs/bcp/bcp5.html)

Related Topics

ipAddressClass
ipAddressToInt
ipByteOfAddress
ipBytesToAddress
ipCommonNetmask
ipCommonNetmaskBits
ipDefaultNetmask
ipDefaultNetmaskBits
ipIntToAddress
ipIsBroadcast
ipIsLoopback
ipIsNormal
ipIsPrivate
ipIsUnknown
ipIsValid
ipNetmaskMerge

ipIntToAddress

Applies to

Ip Functions

Description

Converts an integer to a 32 bit IP address. NULL if ADDR in not in the range of a unsigned 32 bit integer (0 - 4294967295).

Syntax

```
STRING  ipIntToAddress(INT ADDR)
```

Examples

Expression	*Result*
`ipIntToAddress(3426334918)`	`204.57.184.198`
`ipIntToAddress(-3)`	`NULL`

Related Topics

ipAddressToInt

ipIsBroadcast

Applies to

Ip Functions

Description

Returns 1 if ADDR is a valid address and contains a broadcast (255) in the lower order bytes. NULL if ADDR is not valid. Internet Standards 2 (Request for Comments 1700) — STD2 (RFC 1700).

Syntax

```
INT   ipIsBroadcast(STRING ADDR)
```

Examples

Expression	*Result*
`ipIsBroadcast("255.255.255.255")`	`1`
`ipIsBroadcast("10.255.255.255")`	`1`
`ipIsBroadcast("10.1.255.255")`	`1`
`ipIsBroadcast("204.57.184.255")`	`1`

Expression	*Result*
`ipIsBroadcast("255.255.255.1")`	0
`ipIsBroadcast("127.0.0.1")`	0

Related Topic

ipIsValid

ipIsLoopback

Applies to

Ip Functions

Description

Returns 1 if ADDR is a valid address and is a loopback address. A loopback address is an address with a high-order byte of 127. NULL if ADDR is not valid. Internet Standards 2 (Request for Comments 1700) — STD5 (RFC 1700).

Syntax

```
INT   ipIsLoopback(STRING ADDR)
```

Examples

Expression	*Result*
`ipIsLoopback("127.0.0.1")`	1
`ipIsLoopback("10.25.251.1")`	0

Related Topic

ipIsValid

ipIsNormal

Applies to

Ip Functions

Description

Returns 1 if ADDR is valid, is not loopback, is class A, B, or C, is not this or unknown net, is not broadcast, can be routed, and does not contain 255 for any of it's bytes. NULL if ADDR is not valid.

Syntax

```
INT    ipIsNormal(STRING ADDR)
```

Examples

Expression	*Result*
`ipIsNormal("204.57.184.198")`	1
`ipIsNormal("127.57.184.198")`	0
`ipIsNormal("204.57.184.0")`	0
`ipIsNormal("204.57.184.255")`	0
`ipIsNormal("10.23.1.126")`	0

Related Topic

ipIsBroadcast
ipIsLoopback
ipIsPrivate
ipIsUnknown
ipIsValid

ipIsPrivate

Applies to

Ip Functions

Description

Returns 1 if ADDR is valid and is in one of the ranges reserved for private internets. These ranges are as follows:

```
10.0.0.010.255.255.255
172.16.0.0172.31.255.255
192.168.0.0192.168.255.255
```

Syntax

```
INT    ipIsPrivate(STRING ADDR)
```

Examples

Expression	*Result*
`ipIsPrivate("10.23.45.1")`	1
`ipIsPrivate("192.168.23.1")`	1
`ipIsPrivate("127.0.0.1")`	0
`ipIsPrivate("172.35.12.1")`	0

Related Topic

ipIsValid

iplsUnknown

Applies to

Ip Functions

Description

Returns 1 is ADDR is a valid address and any of the bytes of the address are 0. NULL if ADDR is not valid. Internet Standards 2 (Request for Comments 1700) — STD2 (RFC 1700).

Syntax

```
INT    ipIsUnknown(STRING ADDR)
```

Examples

Expression	*Result*
`ipIsUnknown("127.0.0.1")`	1
`ipIsUnknown("204.57.184.0")`	1
`ipIsUnknown("0.0.0.198")`	1
`ipIsUnknown("10.25.251.1")`	0

Related Topic

ipIsValid

ipIsValid

Applies to

Ip Functions

Description

Returns 1 if ADDR is an internet address in the form qqq.rrr.sss.ttt where qqq, rrr, sss, ttt represent 1 to 3 digit decimal integers from 0 to 255 inclusive. Only tests the format of ADDR; does not perform any other verification on the address. Returns 0 otherwise. Internet Standards 5 (Request for Comments 791) — STD5 (RFC 791).

Syntax

```
INT    ipIsValid(STRING ADDR)
```

Examples

Expression	*Result*
`ipIsValid("0.0.0.0")`	`1`
`ipIsValid("254.255.0.255")`	`1`
`ipIsValid("www.vallent.com")`	`0`
`ipIsValid("237.327.124.6")`	`0`

isLeapYear

Applies to

Date Functions

Description

Returns `1` if D is in a leap year, `0` if not, and `NULL` if D is NULL or invalid.

Syntax

```
INT isLeapYear(DATE|TIME D)
```

Examples

Expression	*Result*
`isLeapYear('1/9/99')`	`0`

Expression	Result
isLeapYear('1/9/00')	1

ipNetmaskMerge

Applies to

Ip Functions

Description

Returns the IP Address using NET for the high-order (network) bits and HOST for the low-order (host) bits using MASK to define which bits to use for the network. If host is not given 0.0.0.0 is used. MASK must be a STRING of four dotted bytes in the form nnn.nnn.nnn.nnn where nnn is an integer from 0 to 255 or an INT from 0 to 32. If MASK is not given, and NET is class A, B, or C, the default net mask for that class is used. If NET, HOST, or MASK are invalid, or if NET is not class A, B, or C and MASK is not given, returns NULL.

Syntax

```
STRING   ipNetmaskMerge(STRING NET [,STRING HOST [,STRING | INT MASK]])
```

Examples

Expression	Result
ipNetmaskMerge("10.57.180.169")	10.0.0.0
ipNetmaskMerge("172.16.1.1", "192.168.179.12")	172.16.176.0
ipNetmaskMerge ("172.16.1.1","192.168.179.12","255.0.0.0")	172.168.176.0
ipNetmaskMerge("172.16.1.1", "192.168.179.12", 8)	172.168.176.0
ipNetmaskMerge("10.57.180.169", 20)	10.57.176.0
ipNetmaskMerge("254.10.15.1", "255.255.0.0")	254.10.0.0
ipNetmaskMerge("254.10.15.1")	NULL
ipNetmaskMerge ("10.57.180.169","192.168.179.12","255.255.0.255")	NULL
ipNetmaskMerge("10.57.180.169","192.168.179.12",33)	NULL

Related Topics

ipAddressClass
ipIsValid

isNotNull

Description

Returns `1` if attribute `A` is not `NULL`. That is, if a value has been read from the data repository, or the result of the expression has a value.

Syntax

```
INT isNotNull (AttrA)
```

Examples

Assuming the attribute `DateAttr` does not have a `NULL` value:

Expression	*Result*
`isNotNull(DateAttr)`	`1`
`isNotNull(10 / 0)`	`1`
`isNotNull(DateAttr) ? "NULL" : (STRING)DateAttr`	`NULL`

isNull

Applies To

System Functions

Description

Returns `1` if attribute `A` is `NULL`. That is, if no value has been read from the data repository, or the result of an expression has no value.

Syntax

```
INT isNull(ATTR A)
```

Examples

Assuming the attribute `DateAttr` has a `NULL` value:

Expression	*Result*
`isNull(DateAttr)`	`1`
`isNull(10 / 0)`	`1`

Expression	*Result*
`isNull(DateAttr) ? "NULL" : (STRING)DateAttr`	NULL

lastIndex

Applies To

String Functions

Description

Returns the start index of the last occurrence of *T* in test *S*. The first character in *S* has an index of `0`. If *T* cannot be found in *S*, then `-1` is returned.

Syntax

```
INT lastIndex(STRING S, STRING T)
```

Examples

Expression	*Result*
`lastIndex("Hello World Hello", "Hello")`	12
`lastIndex("Hello World Hello", "World")`	6
`lastIndex("Hello World Hello", "Today")`	-1

Related Topics

firstIndex
subString

length

Applies To

String Functions

Description

Returns the number of characters in the specified string.

Syntax

```
INT length(STRING S)
```

Examples

Expression	*Result*
`length("Hello")`	`5`
`length("Date is " + '09/25/98')`	`16`

Related Topic

subString

ln

Applies To

Math Functions

Description

Calculates the natural logarithm of `N` expressed as a float. If *`N`* `<= 0.0`, then a warning is output and `NULL` is returned. Satisfies the equation:

`ln(exp (`*`N`*`)) == `*`N`*

Syntax

```
FLOAT ln(NUMBER N)
```

Examples

Expression	*Result*
`ln(exp (1))`	`1.000000`
`ln(exp (2))`	`2.000000`
`ln(-1.0)`	`NULL`
`ln(exp (3))`	`3.000000`

Related Topics

exp
log

log

Applies To

Math Functions

Description

Calculates the base 10 logarithm of *N* expressed as a float. If *N* <= 0.0, then a warning is output and NULL is returned.

Syntax

```
FLOAT log(NUMBER N)
```

Examples

Expression	*Result*
log(1)	0.000000
log(10)	1.000000
log(100)	2.000000
log(1000)	2.000000
log(-1.0)	NULL

Related Topic

ln

Logical Operators

Applies To

Binary Operators

Description

Returns either 1 (TRUE) or 0 (FALSE). Logical Operators perform basic implicit type casting.

Syntax

```
[[ operand1 operator operand2]]
```

Related Topics

== Operator (also =)
!= Operator (also <>)

< Operator
<= Operator
> Operator
>= Operator
AND Operator (also &&)
OR Operator (also ||)

matchesRE

Applies To

String Functions

Description

Returns `1` if a substring of the first parameter *`S`* matches the conditions in the regular expression of the second parameter *`R`*. If an offset index *`I`* is given in a third parameter, the search of the string will begin at the character with that index. If *`I`* is less than 0, then 0 is used.

The first character in *`S`* has an index of 0.

Use `\` to escape characters `\ + * ? . [ ] ^ $`.

Syntax

```
INT matchesRE(STRING S, STRING R [,INT I])
```

Examples

Expression	*Result*
`matchesRE("TFL010043", "TFL")`	`1`
`matchesRE("TFL010043", "[0-9]+")`	`1`
`matchesRE("012-01098", "[A-Z]+")`	`0`

Related Topics

replaceRE
String Functions
subString
subStringRE

Math Functions

Applies To

Functions

Description

General math functions that operate on numeric arguments.

Related Topic

abs
ceil
exp
floor
ln
log
numToString
parameterAsString
round
sqr
sqrt
trunc

max

Applies To

Package Functions

Description

Returns the maximum value from a number of values found by following relations. If no objects can be found, then `NULL` is returned.

In the first syntax entry, *A* is in the format *rel1.rel2. ... .relN.Attr* and the result is the maximum of all the attributes found. The return type is the same as the type of the `Attr`.

In the second syntax entry, *R* is in the format *rel1.rel2. ... .relN* and expression *E* is any expression that is valid for objects of the type found in *relN*. The result is the maximum value of expression *E* when *E* is applied to every object found by following the relation specifier *R*. The result type is the same as the result type of *E*.

Syntaxes

```
ATTR max(RELATION_ATTR A)

ATTR max(RELATION R, EXPRESSION E)
```

Examples

Expression	*Result*
`max(package(Bts).Activated)`	`'10/10/98'`
`max(package(Bts), TotalCalls - DroppedCalls)`	`8000`
`max(package(Bts),` `((TotalCalls - DroppedCalls) / (FLOAT)TotalCalls) * 100)`	`99.333333`

Evaluating the expression:

```
max(Bscs.Btss), ((TotalCalls - DroppedCalls) / (FLOAT) TotalCalls) * 100)
```

gives these results for each `Msc`:

```
MSC1:  99.333333
MSC2:  98.666667
MSC3:  90.000000
```

Related Topics

mean
median
min

mean

Applies To

Package Functions

Description

Returns the mean of a number of values found by following relations.

If no objects can be found, then `NULL` is returned.

In the first syntax example, *`A`* is in the format *`rel1.rel2. ... .relN.Attr`*, and the result is the mean of all the attributes found. `Attr` must be a numeric attribute.

In the second syntax example, *`R`* is in the format *`rel1.rel2. ... .relN`*, and expression *`E`* is any expression that is valid for objects of the type found in *`relN`*. The result is the mean of expression *`E`* values, when *`E`* is applied to every object found by following the relation specifier *`R`*. The return type of *`E`* must be numeric.

Syntax

```
FLOAT mean(RELATION_ATTR A)

FLOAT mean(RELATION R, EXPRESSION E)
```

Examples

Expression	*Result*
`mean(package(Bts), DroppedCalls / (FLOAT)TotalCalls * 100)`	`25.409957`

Evaluating the expression:

```
mean(Bscs.Btss.DroppedCalls)
```

gives these results for each `Msc`:

```
MSC1:  40.393939
MSC2:  15.841270
MSC3:  15.000000
```

Related Topics

average
max
median
min
minute

median

Applies To

Package Functions

Description

Returns the central of a number of values found by following relations, when that list is in ascending order.

If the number of values found is odd, then the middle item is returned. If the number of values found is even and the attribute or expression type is numeric, then the mean of the two central values is returned. Otherwise, the smaller of the two central values is returned.

If no objects are found, then median returns `NULL`.

If the attribute or expression type is numeric, then the result type is a Float; in all other cases, it is the same type as the attribute or expression.

In the first syntax example, *A* is in the format *rel1.rel2. ... .relN.Attr*, and the result is the median of all the attributes found.

In the second syntax example, *R* is in the format *rel1.rel2. ... .relN* and expression *E* is any expression that is valid for objects of the type found in *relN*. The result is the median value of expression *E*, when *E* is applied to every object found by following the relation specifier *R*.

Syntax

```
ATTR median(RELATION_ATTR A)

ATTR median(RELATION R, EXPRESSION E)
```

Examples

Expression	*Result*
`median(package (Bts) .Activated)`	`'09/01/98'`
`median(package (Bsc) , DroppedCalls / (FLOAT) TotalCalls * 100)`	`16.250000`

Evaluating the expression:

```
median(Bscs.Btss.Activated)
```

gives these results for each `Msc`:

```
MSC1: 08/01/98
MSC2: 09/10/98
MSC3: 10/01/98
```

Related Topics

mean
minute

min

Applies To

Package Functions

Description

Returns the minimum value from a number of values found by following relations.

If no objects can be found, then `NULL` is returned.

In the first syntax example, A is in the format *rel1.rel2. ... .relN.Attr*, and the result is the minimum of all the attributes found. The result type is the same as the type of Attr.

In the second syntax example, *R* is in the format *rel1.rel2. ... .relN*, and expression *E* is any expression which is valid for objects of the type found in *relN*. The result is the minimum value of expression *E* when *E* is applied to every object found by following the relation specifier *R*. The result type is the same as the result type of *E*.

Syntax

```
ATTR min(RELATION_ATTR A)

ATTR min(RELATION R, EXPRESSION E)
```

Examples

Expression	*Result*
min(package(Bts).Activated)	'08/01/98'
min(package (Bts), TotalCalls - DroppedCalls)	10
min(package (Bts), ((TotalCalls - DroppedCalls) / (FLOAT)TotalCalls) * 100)	9.090909

Evaluating the expression:

```
min(Bscs.Btss), ((TotalCalls - DroppedCalls) / (FLOAT)TotalCalls) * 100)
```

gives these results for each Msc:

```
MSC1: 9.090909
MSC2: 57.142857
MSC3: 77.50000
```

Related Topics

max
mean
median
minute

minute

Applies to

Time Functions

Description

Returns tge hour of T. Returns NULL if T is NULL or invalid.

Syntax

```
INT minute(TIME T)
```

Examples

Expression	*Result*
`minute('12.30.45pm')`	30

Related Topics

hour
minuteGMT
second

minuteGMT

Applies to

Time Functions

Description

Returns the minute of T in Coordinated Universal Time (UTC) (The internationally recognized name for Greenwich Mean Time (GMT)). Returns NULL if T is NULL or invalid.

Syntax

```
INT minuteGMT(TIME T)
```

Examples

Assume time zone is Pacific Standard Time (PST)

Expression	*Result*
`minuteGMT('12.30.45pm')`	30

Related Topics

hourGMT
minute

mode

Applies To

Package Functions

Description

Returns the most common value from a number of values found by following relations.

If more than one mode is found, then the mode with the smallest value is returned. For example, if `m1` and `m2` are the modes, then `m1` is returned when `m1` <= `m2`. Otherwise, `m2` is returned.

If no objects can be found, then `NULL` is returned.

In the first syntax example, *`A`* is in the format *`rel1.rel2. ... .relN.Attr`* and the result is the most common of all the attributes found. The result type is the same as the type of `Attr`.

In the second syntax example, *`R`* is in the format *`rel1.rel2. ... .relN`* and expression *`E`* is any expression that is valid for objects of the type found in *`relN`*. The result is the most common value of expression *`E`*, when *`E`* is applied to every object found by following the relation specifier *`R`*. The result type is the same as the result type of *`E`*.

Syntax

```
ATTR mode(RELATION_ATTR A)

ATTR mode(RELATION R, EXPRESSION E)
```

Examples

Expression	*Result*
`mode(package (Bts) .Activated)`	`'08/01/98'`
`mode(package (Bts) .Vendor)`	LUCENT

Evaluating the expression:

```
mode(Bscs.Btss.Activated)
```

gives these results for each `Msc`:

```
MSC1: 08/01/98
MSC2: 09/10/98
MSC3: 10/01/98
```

Related Topics

mean
median

month

Applies to

Date Functions

Description

Returns the number of month (1-12) for D. NULL if D is NULL or invalid.

Syntax

```
INT month(DATE|TIME D)
```

Examples

Expression	*Result*
`month('1/9/99')`	`1`

Related Topics

dayOfMonth
monthName
year

monthName

Applies to

Date Functions

Description

Returns the name of month for D, based on locale. NULL if D is NULL or invalid.

Syntax

```
STRING monthName(DATE|TIME D)
```

Examples

The following example is for an english-speaking locale.

Expression	*Result*
`monthName('1/9/99')`	`January`

Related Topics

month
weekDayName

nextDate

Applies to

Date Functions

Description

Returns the date of the next occurrence of weekday W (1-7). NULL if D is NULL or invalid, or if W is not in range 1-7.

Syntax

```
DATE|TIME nextDate(DATE|TIME D, INT W)
```

Examples

Expression	*Result*
`nextDate('1/9/99', 3)`	`'1/12/99'`

Related Topics

previousDate

nullDate

Applies To

System Functions

Description

Returns a null value where type DATE is required.

Syntax

```
DATE NullDate()
```

Examples

Expression	*Result*
`isNull(NullDate())`	`1`

Related Topic

isNull
nullFloat
nullInt
nullString
nullTime
nullValue

nullFloat

Applies To

System Functions

Description

Returns a null value where type FLOAT is required.

Syntax

```
FLOAT NullFloat()
```

Examples

Expression	*Result*
`isNull(NullFloat())`	`1`

Related Topic

isNull
nullDate
nullInt
nullString
nullTime
nullValue

nullInt

Applies To

System Functions

Description

Returns a null value where type INT is required.

Syntax

```
INT NullInt()
```

Examples

Expression	*Result*
`isNull(NullInt())`	`1`

Related Topic

isNull
nullDate
nullFloat
nullString
nullTime
nullValue

nullString

Applies To

System Functions

Description

Returns a null value where type STRING is required.

Syntax

```
STRING NullString()
```

Examples

Expression	*Result*
`isNull(NullString())`	`1`

Related Topic

isNull
nullDate
nullFloat
nullInt
nullTime
nullValue

nullTime

Applies To

System Functions

Description

Returns a null value where type TIME is required.

Syntax

```
TIME NullTime()
```

Examples

Expression	*Result*
`isNull(NullTime())`	`1`

Related Topic

isNull
nullDate
nullFloat
nullInt
nullString
nullValue

nullValue

Applies To

System Functions

Description

If attribute `A` is NULL (for example, has no value, or traverses a null relationship) then `V` will be returned, otherwise `A` will be returned.

The return type of this function is the same as attribute A's type. Attribute `V` must be of the same type as Attribute `A`.

Syntax

```
ATTR nullValue(ATTR A, ATTR V)
```

Examples

Assume that the relation "Switch" is null for the following examples.

Expression	*Result*
`nullValue(10/0, 0)`	`0`
`nullValue(Switch.Id, "NO SWITCH")`	`"NO SWITCH"`
`nullValue("Not Null", "Null")`	`"Not Null"`

Related Topic

isNull

numToString

Applies To

Math Functions

Description

Formats the number *N* with *P* decimal places and a minimum field width of *W* character. *P* must take a value in the range `[0, 10]` and, if specified, *W* must take a value in the range `[1, 100]`. If the width of the String is *LW* characters, the String is left-padded with the character `0` until the String length is *W*.

Syntax

```
STRING numToString(NUMBER N, INT P [,INT W])
```

Examples

Expression	*Result*
`numToString(100, 0, 6)`	`000100`
`numToString(100.456789, 0)`	`100`
`numToString (100.456789, 3)`	`100.457`
`numToString(100.456789, 3, 10)`	`000100.457`

Operators

Applies To

General Expressions

Description

Operators are used in expressions to perform arithmetic or logical operations. There are two types of operators: unary and binary. Unary operators use one operand in the format:

```
operand operator
```

Binary operators use two operands in the format:

```
operand1 operator operand2
```

Unary operators are right associative; binary operators are left associative. This means that the following expression

```
a - (INT)b - c < -d
```

should be read as

```
((a - (INT)b) - c) < (-d)
```

Type Casting

Some operators can, depending on their operands, cause conversion of the operand value from one type to another. This process is known as *implicit casting*. Most binary operators use a set of standard type conversions whereby if either operand is of type `FLOAT`, then the other operand is converted to a `FLOAT`. Any variation from this rule is described in the relevant operator section.

You can also force conversion of an operand value from one type to another. This process is known as *explicit casting*. The different types of explicit casts are described in Unary Operators. You can use both implicit and explicit casting in expressions.

Syntax

The following table summarizes the precedence among operators. Each operator in a block shares the same precedence and is evaluated from left to right. An operator in a higher block has a higher precedence than operators in a lower block.

For example

```
a + b * c  ==>  a + (b * c)
```

because * has a higher precedence than +. Also

```
a + b - c  ==> (a + b) - c
```

because + and – have the same precedence and because + is left associative.

Operator	*Description*	*Syntax*
!	Not	! expr
-	Unary minus (negation)	- expr
()	Explicit Cast	(type) expr
^	Power	expr ^ expr
*	Multiply	expr * expr
/	Divide	expr / expr
%	Modulo (remainder)	expr % expr
+	Addition	expr + expr
-	Subtraction	expr - expr
==	Equality	expr = = expr
=	Equality	expr = expr
!=	Inequality	expr != expr
<>	Inequality	expr != expr
<	Less than	expr < expr
<=	Less than or equal to	expr <= expr
>	Greater than	expr > expr
>=	Greater than or equal to	expr >= expr
?:	Conditional Expression	expr ? expr : expr
AND	Logical AND	expr AND expr

Operator	*Description*	*Syntax*
`&&`	Logical AND	`expr && expr`
`OR`	Logical OR	`expr OR expr`
`||`	Logical OR	`expr || expr`

Remarks

Values of attributes in an Application Gateway object are sometimes `NULL`. That means no data was read from the database, ASCII file or other repository source. If you apply an operator to a `NULL` attribute, the `NULL` is returned. You can use the function `isNull()` to determine if an attribute or expression has a value.

Related Topics

() Operator (Cast)
Binary Operators
Gateway Attribute
Unary Operators

OR Operator (also ||)

Applies To

Logical Operators

Description

Returns `0` if both operands are zero, and `1` in all other cases. Both operands must be numeric. OR and || are binary operators that express logical relationships.

Syntax

```
[[operand1 OR operand2]]

[[operand1 || operand2]]
```

Examples

Expression	*Result*
`1 OR 1`	`1`
`0 OR 1`	`1`
`0 OR 0`	`0`

Related Topics

Binary Operators
Operators

package

Applies To

Package Functions

Description

Returns all objects in the current or specified package. It can prefix other attribute specifiers.

The first and second syntax example returns all the objects in the current package.

The third syntax example returns all the objects in the specified package.

Syntax

```
RELATION package()

RELATION package(this)

RELATION package(package_name)
```

Related Topic

Gateway Attribute

Package Functions

Applies To

Functions

Description

Functions that operate on a number of objects in an Application Gateway package or relation. With the exception of count and package, all the package methods share the same general syntax.

Syntax

```
ATTR package_function(ATTR_SPECIFIER A)
ATTR package_function(RELATION_SPECIFIER R, EXPRESSION E)
```

Elements

Element	*Description*
ATTR	Function specific return type.
package_function	Name of the function.
ATTR_SPECIFIER	In the format *rel1.rel2. ... .relN.Attr*. See Gateway Attribute, sections Attributes Using One-to-One Relations and Accessing Every Object in a Package.
RELATION_SPECIFIER	In the format *rel1.rel2. ... .relN*. See Gateway Attribute, sections Attributes Using One-to-One Relations and Accessing Every Object in a Package.
EXPRESSION	Any expression. This must be valid for the type (package) of objects returned by the RELATION_SPECIFIER.

Examples

The examples used by the functions within this group rely on Application Gateway packages. All the examples use the following package definitions:

```
PACKAGE Msc
{
     STRING   Id
     STRING   Vendor
     DATE     Activated
}

PACKAGE Bsc
{
     STRING   Id
     STRING   Vendor
     Date     Activated
     RELATION  Msc  Msc  REVERSE  Bscs
}

PACKAGE Bts
{
     STRING    Id
     STRING    Vendor
     DATE      Activated
     INT       DroppedCalls
     INT       TotalCalls
     RELATION  Vsc  Vsc  REVERSE  Btss
}
```

And the following package data:

```
Msc:
     MSC1     LUCENT   08/01/98
```

```
      MSC2      NORTEL    09/01/98
      MSC3      LUCENT    10/01/98

Bsc
    BSC1A      LUCENT    08/01/98        MSC1
    BSC2A      NORTEL    09/01/98        MSC2
    BSC2B      LUCENT    09/10/98        MSC2
    BSC3A      LUCENT    10/01/98        MSC3

Bts:
    BTS1A-1    LUCENT   08/01/98         1000      2000     BSC1A
    BTS1A-2    LUCENT   08/01/98         100       110      BSC1A
    BTS1A-3    LUCENT   08/10/98         2000      10000    BSC1A
    BTS1A-4    LUCENT   08/10/98         20        3000     BSC1A
    BTS2A-1    NORTEL   09/01/98         40        3000     BSC2A
    BTS2B-1    LUCENT   09/10/98         300       700      BSC2B
    BTS2B-2    LUCENT   09/10/98         200       6000     BSC2B
    BTS3A-1    LUCENT   10/01/98         100       800      BSC3A
    BTS3A-2    LUCENT   10/09/98         60        600      BSC3A
    BTS3A-3    LUCENT   10/10/98         450       2000     BSC3A
```

Related Topics

average
count
Gateway Attribute
max
mean
median
min
minute
package
stddev
sum
validDate

parameterAsString

Applies To

System Functions

Description

Returns the named parameter *P* from the current system parameter set as a STRING. If *P* cannot be found, then the value of *D* is returned (if *D* is specified), or NULL.

Refer to the application-specific documentation for details on how to create and set parameters.

Syntax

```
STRING parameterAsString(STRING P [, STRING D])
```

Examples

For this example, assume that the parameter `perf.duration` does not exist.

Expression	*Result*
`parameterAsString("perf.start")`	`24 May 1999`
`parameterAsString("perf.duration")`	`NULL`
`parameterAsString("perf.duration", "NO VALUE")`	`NO VALUE`

There are several parameter types that represent ranges of values: Date Range, Time Range, and Time Slice. To access these values, the parameter name can contain additional access strings. The following example illustrates how to use these parameter types and access strings.

Expression	*Result*
`parameterAsString("DateRange")`	`24 May 1999 .. 25 May 1999`
`parameterAsString("DateRange:start")`	`24 May 1999`
`parameterAsString("DateRange:end")`	`25 May 1999`
`parameterAsString("TimeRange")`	`11:53:36 .. 11:55:36`
`parameterAsString("TimeRange:date")`	`1999/05/24 11:53:36 .. 1999/05/25 11:55:36`
`parameterAsString("TimeRange:start_date")`	`1999/05/24 11:53:36`
`parameterAsString("TimeRange:end_date")`	`1999/05/25 11:55:36`
`parameterAsString("TimeRange:start")`	`11:53:36`
`parameterAsString("TimeRange:end")`	`11:55:36`
`parameterAsString("TimeSlice")`	`00:00:00 (1/24) .. 23:59:59 (24/24)`
`parameterAsString("TimeSlice:start")`	`1`
`parameterAsString("TimeSlice:end")`	`24`
`parameterAsString("TimeSlice:total")`	`24`

Related Topics

System Functions

pi

Applies To

Math Functions

Description

Returns the value of pi (3.14159265358979323846).

Syntax

```
FLOAT pi()
```

Examples

Expression	*Result*
`360 * (pi() / 180)`	`6.283185`

previousDate

Applies to

Date Functions

Description

Returns the date of the previous occurrence of weekday W (1-7). NULL if D is NULL or invalid, or if W is not in range 1-7.

Syntax

```
DATE|TIME previousDate(DATE|TIME D, INT W)
```

Examples

Expression	*Result*
`previousDate('1/9/99', 3)`	`'1/5/99'`

Related Topics

nextDate

protect

Description

Computes the expression prior to aggregating over element.

Syntax

```
protect (expression)
```

Examples

In the following examples, `UDC X` aggregation type is:

- Element aggregation `(E) type = Avg`
- Time aggregation `(T) type = Avg`

`A`, `B`, and `C` are representative field names and have aggregation of `Sum` for *time* and *element*.

Therefore, Avg_B^E represents the element aggregation from `Peg B (Sum)`.

Example 1:

Standard expression:

```
UDC X = (A+B) * C
```

This is evaluated as follows:

$$result = (Sum_A^E(Sum_A^T(A)) + Sum_B^E(Sum_B^T(B))) * Sum_C^E(Sum_C^T(C))$$

Example 2:

Same expression using `protect`

```
UDC X = protect((A+B) * C)
```

This is evaluated as follows:

$$result = Avg_X^E\ (((Sum_A^T(A) + Sum_B^T(B)) * Sum_C^T(C))$$

Related Topics

elemprotect

Note: If you are using IBM Prospect Web, then be aware `protect` behaves differently in that environment. See the IBM Prospect Web User's Guide for more details.

---

radians

Applies To

Trig Functions

Description

Converts *N* degrees to radians. Satisfies the equation:

```
radians(N) == N * (pi()  /  180.0)
```

Syntax

```
FLOAT radians(NUMBER N)
```

Examples

Expression	*Result*
`radians(180)`	`3.141593`
`radians(360)`	`6.283185`
`radians(180)  == pi()`	`1`
`radians(360)  == 2 * pi()`	`1`

Related Topics

degrees
parameterAsString

replaceRE

Applies To

String Functions

Description

Replaces substring of *S* that matches the regular expression condition given in a second parameter *R*, with the contents of a third parameter *U*. If no third parameter is provided, the matching substring is simply removed. If an offset index *I* is given in a fourth parameter, the search of the string will begin at the character with that index. If *I* is less than 0, then 0 is used.

The first character in *S* has an index of 0.

Use \ to escape characters \ + * ? . [ ] ^ $.

Syntax

```
STRING replaceRE(STRING S, STRING R [,STRING U [,INT I]])
```

Examples

Expression	*Result*
`replaceRE("TFL010043", "[A-Z]+")`	`010043`
`replaceRE("TFL010043", "[A-Z]+", "CELL ")`	`CELL 010043`

Related Topics

matchesRE
subString
subStringRE

replaceAllRE

Applies To

String Functions

Description

Replaces all substrings of *S* that match the regular expression condition given in a second parameter *R*, with the contents of a third parameter *U*. If no third parameter is provided, the matching substring is simply removed. If an offset index *I* is given in a fourth parameter, the search of the string will begin at the character with that index. If *I* is less than 0, then 0 is used.

The first character in *S* has an index of 0.

Use \ to escape characters \ + * ? . [ ] ^ $.

Syntax

```
STRING replaceAllRE(STRING S, STRING R [,STRING U [,INT I]])
```

Examples

Expression	*Result*
`replaceRE("TFL010043", "[A-Z]+[0]")`	`143`
`replaceRE("TFL010043", "[A-Z]+", "CELL ")`	`CELL 010043`

Related Topics

matchesRE
replaceRE
subString
subStringRE

reverse

Applies To

String Functions

Description

Reverses the order of characters in *S*.

Syntax

```
STRING reverse(STRING S)
```

Examples

Expression	*Result*
`reverse("Hello")`	`olleH`

round

Applies To

Math Functions

Description

Returns the integer value nearest *N* in the direction of the current `IEEE754` rounding mode, expressed as a float.

Syntax

```
FLOAT round(NUMBER N)
```

Examples

Expression	*Result*
round(1.3)	1.000000
round(1.5)	2.000000
round(1.7)	2.000000
round(-1.3)	-1.000000
round(-1.5)	-2.000000
round(-1.7)	-2.000000

Related Topics

ceil
floor
trunc

second

Applies to

Time Functions

Description

Returns the second (0-59) of minute of T. Returns NULL if T is NULL or invalid.

Syntax

```
INT second(TIME T)
```

Examples

Expression	*Result*
second('12.30.45pm')	45

Related Topics

hour
minute

seconds

Applies to

Time Functions

Description

Returns the number of seconds since 00:00:00 January 1, 1901 UTC for T. Returns NULL if T is NULL or invalid.

Syntax

```
FLOAT seconds(TIME T)
```

Examples

Expression	*Result*
`seconds(stringToTime("9Jan1999 1:10:15 AM", "%d%b%Y %I:%M:%S %p"))`	`3093325815.000000`

Note to Developer:

Return type is FLOAT because INT is too small and LONG is not yet supported.

sin

Applies To

Trig Functions

Description

Computes the sine of *N* (in radians), expressed as a float.

Syntax

```
FLOAT sin(NUMBER N)
```

Examples

Expression	*Result*
`sin(pi())`	`0.000000`
`sin(0)`	`0.000000`

Expression	*Result*
`sin(pi()/2))`	`1.000000`
`sin(radians(90))`	`1.000000`
`sin(radians(270))`	`-1.000000`

Related Topics

arcSin
cos
radians
tan

sqr

Applies To

Math Functions

Description

Calculates the square of *N*, expressed as a float. Satisfies the equation:

```
sqr(N)  == N ^ 2
```

Syntax

```
FLOAT sqr(NUMBER N)
```

Examples

Expression	*Result*
`sqr(0)`	`0.000000`
`sqr(1)`	`1.000000`
`sqr(2)`	`4.000000`
`sqr(-2)`	`4.000000`
`sqr(10)`	`100.000000`

Related Topics

^ Operator
sqr

sqrt

Applies To

Math Functions

Description

Calculates the square root of *N*, expressed as a float. If *N* is less than 0, then NULL is returned. Satisfies the equation:

```
sqr(sqrt (N))     == N
```

Syntax

```
FLOAT sqrt(NUMBER N)
```

Examples

Expression	*Result*
sqrt(0)	0.000000
sqrt(1)	1.000000
sqrt(9)	3.000000
sqrt(-1)	NULL
sqr(sqrt(100))	100.000000

Related Topics

^ Operator
sqrt

stddev

Applies To

Package Functions

Description

Returns the standard deviation of a number of values found by following relations. Standard deviation (the square root of variance) is calculated using the following equation:

```
stddev = sqrt(sum(x^2)/n - mean(x)^2)
```

where *x* is each attribute value or expression found.

If no objects can be found, then NULL is returned.

In the first syntax example, *A* is in the format *rel1.rel2. ... .relN.Attr* and the result is the standard deviation of all the attributes found. Attribute Attr must be numeric.

In the second syntax example, *R* is in the format *rel1.rel2. ... .relN* and expression *E* is any expression that is valid for objects of the type found in *relN*. The result is the standard deviation of expression *E* values, when *E* is applied to every object found by following the relation specifier *R*. The result type of *E* must be numeric.

Syntax

```
FLOAT stddev(RELATION_ATTR A)

FLOAT stddev(RELATION R, EXPRESSION E)
```

Examples

Expression	*Result*
stddev(package(Bts), DroppedCalls / (FLOAT) TotalCalls * 100)	27.056186

Evaluating the expression:

```
stddev(Bscs.Btss, DroppedCalls / (FLOAT)TotalCalls * 100)
```

gives these results for each Msc::

```
MSC1: 34.052261
MSC2: 19.120548
MSC3:  5.400617
```

Related Topics

sqr
sqrt
validDate

String Functions

Applies To

Functions

Description

A function that operates on string arguments.

Related Topics

concat
firstIndex
isLeapYear
lastIndex
length
matchesRE
replaceAllRE
replaceRE
reverse
stringToInt
subString
subStringRE
Time Functions
toLower
toUpper
unescape

stringToDate

Applies to

Date Functions

Description

Returns a date by parsing S.
If F is provided, it is used as the format.
If F is not given, default for locale is used.
Format is same as `strptime`. If S is NULL, F is NULL or invalid, or S and F do not match to produce a valid date, NULL is returned. Format definition is same as `strptime(3C)`.

Syntax

```
DATE stringToDate(STRING S [, STRING F])
```

Examples

Assume locale for time LC_TIME=en_US, d_fmt=%m/%d/%y

Expression	*Result*
`stringToDate("1/9/99") == '9Jan1999'`	1
`stringToDate("1999 Jan 9", "%Y %b %d")=='1/9/1999'`	1

Related Topics

dateToString
stringToTime

stringToFloat

Applies To

String Functions

Description

Converts a string into a floating point number. The string to convert must be in a valid format, otherwise NULL is returned.

Syntax

```
FLOAT stringToFloat(STRING S)
```

Examples

```
stringToFloat("34.05")
stringToFloat("0.0564")
stringToFloat("4.34+e13")
stringToFloat("1.234-E6")
```

An example of an invalid format is:

```
stringToFloat("abcd")
```

Related Topics

numToString
String Functions
stringToInt

stringToInt

Applies To

String Functions

Description

Converts a string into an integer number (of base 10). The string to convert must be in a valid format, otherwise NULL is returned.

If the string has the leading characters "0x" or "0X" it is assumed to be a hexadecimal number.

If the string has the leading character "0" it is assumed to be an octal number.

If the string does not have the leading characters mentioned above, it is assumed to be a decimal number.

Syntax

```
INT stringToInt(STRING S)
```

Examples

Expression	*Result*
`stringToInt("1234")`	`1234`
`stringToInt("98659")`	`98659`
`stringToInt("0XABCD")`	`43981`
`stringToInt("0x769A")`	`30362`
`stringToInt("0654")`	`428`

Example of an invalid format:

```
stringToInt("abcd")
```

Related Topics

isLeapYear
numToString
String Functions

stringToTime

Applies to

Time Functions

Description

Returns the time value created by parsing S. If F is provided it is used as the format. If F is not given default for locale is used. Format is same as `strptime`. If S is NULL, F is NULL or invalid, or S and F do not match to produce a valid date, NULL is returned. Format definition is same as `strptime(3C)`.

Syntax

```
TIME stringToTime(STRING S [, STRING F])
```

Examples

Assume locale for time LC_TIME=en_US, t_fmt=%H:%M:%S

Expression	*Result*
`stringToTime("13:35:27")=='1:35:27pm'`	`1`

Related Topics

stringToDate
timeToString

subString

Applies To

String Functions

Description

Extracts a substring of *S*, starting at index *I* with length *L*. If index *I* is less than 0, then 0 is used. If length *L* is less than or equal to 0, then `NULL` is returned. If `I + L` is greater than the length of *S*, then only the characters from *I* to the end of *S* are returned. If L is not given, characters from I to the end of S will be returned.

The first character in *S* has an index of 0.

Syntax

```
STRING subString(STRING S, INT I[, INT L])
```

Examples

Expression	*Result*
`subString("Hello World", 5)`	`World`
`subString("Hello World", 6, 5)`	`World`
`subString("Hello World", -1, 100)`	`Hello World`
`subString("Hello World", firstIndex("Hello World", "World"),length("World"))`	`World`

Related Topics

firstIndex
lastIndex
subStringRE
Time Functions

subStringRE

Applies To

String Functions

Description

Extracts a substring of *S* that matches the regular expression condition given in a second parameter *R*. If an offset index *I* is presented in a third parameter, the search of the string will begin at the character with that index. If *I* is less than 0, then 0 is used. A fourth parameter *L* can be used to restrict the length of the result. If length *L* is less than or equal to 0, then NULL is returned. If *I* + *L* is greater than the remaining length of *S*, then only the characters from *I* to the end of *S* are returned.

The first character in *S* has an index of 0.

Use \ to escape characters \ + * ? . [ ] ^ $.

Syntax

```
STRING subStringRE(STRING S, STRING R, INT I, INT L)
```

Examples

Expression	*Result*
`subStringRE("TFL010043", "[A-Z]+")`	TFL
`subStringRE("012-01098", "[A-Z]+")`	NULL

Related Topics

matchesRE
replaceAllRE
replaceRE
String Functions
subString

sum

Applies To

Package Functions

Description

Returns the sum of a number of values found by following relations. If no objects are found, then NULL is returned.

In the first syntax example, *A* is in the format *rel1.rel2. ... .relN.Attr* and the result is the sum of all the attributes found. Attribute Attr must be numeric.

In the second syntax example, *R* is in the format *rel1.rel2. ... .relN* and the expression *E* is any expression that is valid for objects of the type found in *relN*. The result is the sum of expression *E* values, when *E* is applied to every object found by following the relation specifier *R*. The result type of *E* must be numeric.

Syntax

```
FLOAT sum(RELATION_ATTR A)

FLOAT sum(RELATION R, EXPRESSION E)
```

Examples

Expression	*Result*
sum(package(Bts), TotalCalls - DroppedCalls)	23940.000000
sum(package(Bts).DroppedCalls) / sum(package(Bts).TotalCalls) * 100	15.136476

Evaluating the expression:

```
sum(Bscs.Btss.DroppedCalls) / sum(Bscs.Btss.TotalCalls) * 100
```

gives these results for each Msc:

```
MSC1:  20.648577
MSC2:  5.567010
MSC3:  17.941176
```

Related Topics

count
Functions
mean

sysdate

Applies To

System Functions

Description

Returns the current date.

Syntax

```
DATE sysdate()
```

Examples

The following example assumes that the current date is 'Sep 10 1998':

Expression	*Result*
'Sep 2 1998' < sysdate()	1
'Sep 2 1998' > sysdate() - 7	0

Related Topic

systime

System Functions

Applies To

Functions

Description

Functions commonly used that do not fit into other function groups (for example, String, Math, Trig, and Package functions).

Related Topics

getenv
nullDate
nullFloat
nullInt
nullString
nullTime
nullValue
parameterAsString
sysdate
systime

systime

Applies To

System Functions

Description

Returns the current time.

Syntax

```
TIME systime()
```

Examples

The following example assumes the current time is '4:00:00pm':

Expression	*Result*
`'10:00:00am' < systime()`	`1`
`'10:00:00am' > systime() - (60 * 60)`	`0`

Related Topic

sysdate

tan

Applies To

Trig Functions

Description

Computes the tangent of *N* (in radians), expressed as a float.

Syntax

```
FLOAT tan(NUMBER N)
```

Examples

Expression	*Result*
`tan(pi())`	`0.000000`

Expression	*Result*
`tan(0)`	`0.000000`
`tan(pi ()/2)`	`0.000000`

The following are approximations of positive and negative infinity:

```
tan(radians (90))      = 16331239353195370.000000
tan(radians (270))     = 5443746452065234.000000
```

Related Topics

arcTan
cos
radians
sin

Time Functions

Applies To

Functions

Description

Functions that operate on time parameters

Related Topics

getDateFromTime
hour
hourGMT
minute
minuteGMT
second
seconds
stringToTime
timeToString
validTime

timeToString

Applies to

Time Functions

Description

Returns a formatted string from T. If F is not given, default for locale is used. If T is NULL or invalid, or F is NULL or invalid, NULL is returned. Format is same as `strftime(3C)`.

Syntax

```
STRING timeToString(TIME T [,STRING F])
```

Examples

Assume locale for time LC_TIME=en_US, t_fmt=%H:%M:%S

Expression	*Result*
`timeToString('12.30.45pm')`	`12:30:45`
`timeToString('12.30.45pm', "%l:%M %p")`	`12:30 pm`

Related Topics

stringToTime

tokenize

Applies To

String Functions

Description

Returns the field number *F*, when string *S* is tokenized using any of the characters in string *D* as token delimiters. Each character in *D* is treated as a separate delimiter. The first field has a field number of 0.

If the string does not contain *F* fields, then the empty string "" is returned.

Syntax

```
STRING tokenize(STRING S, STRING D, INT F)
```

Examples

Expression	*Result*
`tokenize("10-20.50-30-40-50", "-", 2)`	`30`
`tokenize("10-20.50-30-40-50", "-.", 2)`	`50`

Expression	*Result*
`tokenize("10-20.50-30-40-50", "-.", 10)`	
`tokenize("10 20 30 40 50-60 70.80"," -.", 7)`	80

Related Topic

subString

toLower

Applies To

String Functions

Description

Changes all upper-case letters in the specified string to lower-case. Nonalphabetic characters remain unchanged.

Syntax

```
STRING toLower(STRING S)
```

Examples

Expression	*Result*
`toLower("UPPER CASE")`	`upper case`
`toLower("Mixed Case")`	`mixed case`
`toLower("Characters #6734&^")`	`characters #6734&^`
`toLower("Date is " + sysdate())`	`date is 09/25/98`

Related Topic

toUpper

toUpper

Applies To

String Functions

Description

Changes all lower-case letters in the specified string to upper-case. Nonalphabetic characters remain unchanged.

Syntax

```
STRING toUpper(STRING S)
```

Examples

Expression	*Result*
`toUpper("lower case")`	`LOWER CASE`
`toUpper("Mixed Case")`	`MIXED CASE`
`toUpper("Characters #6734&^")`	`CHARACTERS #6734&^`
`toUpper("Date is " + sysdate())`	`DATE IS 09/25/98`

Related Topic

toLower

Trig Functions

Applies To

Functions

Description

Trigonometric functions that operate on numeric arguments.

Related Topics

arcCos
arcSin
arcTan
cos
degrees
radians
sin
tan

trunc

Applies To

Math Functions

Description

Integer value of N, expressed as a float. Returns the same value as the expression:

```
(FLOAT)(INT)N
```

Syntax

```
FLOAT trunc(NUMBER N)
```

Examples

Expression	Result
trunc(1.3)	1.000000
trunc(1.5)	1.000000
trunc(1.7)	1.000000
trunc(-1.3)	-1.000000
trunc(-1.5)	-1.000000
trunc(-1.7)	-1.000000

Related Topics

() Operator (Cast)
ceil
floor
round

Unary Operators

Applies To

Operators

Description

Right associative operators that take one operand.

Syntax

```
[[ operator operand]]
```

Related Topics

() Operator (Cast)
- Operator (Unary)
! Operator

unescape

Applies to

String Functions

Description

Removes any embedded characters from string, replacing them with the ASCII character that the escape sequence represents. '\n' is replaced with new line, '\\' with '\', '\t' with tab, '\"' with ", and '\r' with return. If the pair does not define a special character, the sequence is replaced with the second character so that the only change is the removal of the ASCII escape character.

Syntax

```
STRING unescape(STRING S)
```

Examples

```
Assume
Street = "123 Oak Street\nSuite 210"
eMail = "\"John Q. User\" <jqp@juno.com>"
path = "C:\\Documents\\file.txt"
```

Expression	*Result*
`unescape(Street)`	`123 Oak Street` `Suite 210`
`unescape(eMail)`	`John Q. User <jqp@juno.com>`
`unescape(path)`	`C:\Documents\file.txt`

Related Topics

replaceRE
String Functions

validDate

Applies to

Date Functions

Description

Returns 1 if D is a valid date, 0 if D is not valid or NULL.

Syntax

```
INT validDate(DATE D)
```

Examples

Expression	*Result*
`validDate('2/29/99')`	0
`validDate('2/29/00')`	1

Related Topics

validTime

validTime

Applies to

Time Functions

Description

Returns 1 if T is a valid time. 0 if it is not valid or NULL.

Syntax

```
INT validTime(TIME T)
```

Examples

Expression	*Result*
`validTime('09Jan1999 1:10AM')`	1
`validTime('29Feb1999 25:77AM')`	0

Related Topics

validDate

variance

Applies To

Package Functions

Description

Returns the variance of a number of values found by following relations. Variance is calculated using the following equation:

```
variance = sum(x^2)/n - mean(x)^2
```

where *x* is each attribute value or expression result found.

If no objects can be found, then NULL is returned.

In the first syntax example, A is in the format *rel1.rel2. ... .relN.Attr* and the result is the variance of all the attributes found. Attribute Attr must be numeric.

In the second syntax example, *R* is in the format *rel1.rel2. ... .relN* and expression *E* is any expression that is valid for objects of the type found in *relN*. The result is the variance of expression *E* values, when *E* is applied to every object found by following the relation specifier *R*. The result type of *E* must be numeric.

Syntax

```
FLOAT variance(RELATION_ATTR A)

FLOAT variance(RELATION R, EXPRESSION E)
```

Examples

Expression	*Result*
variance(package(Bts), DroppedCalls / (FLOAT)TotalCalls * 100)	732.037184

Evaluating the expression:

```
variance(Bscs.Btss, DroppedCalls / (FLOAT)TotalCalls * 100)
```

gives these results for each Msc:

```
MSC1:  1159.556474
MSC2:  365.595364
MSC3:  29.166667
```

Related Topics

mean
stddev
sum

vsum()

Description

Applies vector summation semantics to a colleciton of scalar values. If a particular result is NULL, it is ignored. If at least one result is not-NULL, then the result is not NULL. If all values are NULL, then the result is NULL.

Syntax

```
vsum(field, ...)
```

field—One or more fields that are added.

Examples

Expression	*Result*
`vsum(1, 2, 3, 0)`	`Not NULL – The zero forces the result to never be NULL and does not affect the calculated summation`
`vsum(1, 2, 3)`	`Not NULL`
`vsume(0,0,0)`	`NULL – All values are NULL therefore the result is NULL.`

weekDay

Applies to

Date Functions

Description

Returns the day (1-7) of week for D. NULL if D is NULL or invalid.

Syntax

```
INT weekDay(DATE|TIME D)
```

Examples

Expression	*Result*
`weekDay('1/9/99')`	`6`

Related Topics

weekDayName

weekDayName

Applies to

Date Functions

Description

Returns the name of the day of the week for D. NULL if D is NULL or invalid.

Syntax

```
STRING weekDayName (DATE|TIME D)
```

Examples

Expression	*Result*
`weekDayName('1/9/99')`	`Saturday`

Related Topics

monthName
weekDay

year

Applies to

Date Functions

Description

Returns the year of D. NULL if D is NULL or invalid.

Syntax

```
INT year(DATE|TIME D)
```

Examples

Expression	*Result*
`year('1Jan1999')`	`1999`

Related Topics

month

3 Advanced Expressions

This section covers the additional advanced expressions available within the IBM Prospect reporting generation process. Some of these expressions are context sensitive; that is, they are only valid when running a particular kind of report (such as a Forecast Report).

Note: where TSTAMP is passed as a STRING type, it is assumed it is formatted as: to_char(tstamp, 'YYYY-MM-DD HH24:MI:SS')

Related Topics

Erlang Functions
Kaufman-Roberts Functions
Forecast Reports
Bridge Functions
Instance Name Mapping Functions
Trending Reports

Erlang Functions

The Erlang family of functions is used for traffic analysis and capacity planning. Most users do not write expressions containing these functions directly. Rather, the function would be implemented natively in the data dictionary and users simply uses the field either directly in their report templates, or in UDCs.

avrgDelayAllCallsC

Signature

```
FLOAT avrgDelayAllCallsC(INT N, FLOAT A, FLOAT T1, FLOAT T2)
```

Attribute	*Description*
A	Offered traffic in Erlangs.
N	Number of servers.
T1	Acceptable delay.

Attribute	Description
T2	Mean server holding time (Average call time).

Description

Using Erlang C, calculates the average delay on all calls.

Restrictions

None.

avrgDelayDelayedCallsC

Signature

```
FLOAT avrgDelayDalayedCallsC(INT N, FLOAT A, FLOAT T1, FLOAT T2)
```

Attribute	Description
A	Offered traffic in Erlangs.
N	Number of servers.
T1	Acceptable delay.
T2	Mean server holding time (Average call time).

Description

Using Erlang C, calculates the average delay on delayed calls.

Restrictions

None.

capacityB

Signature

```
FLOAT capacityB(INT NUM_CIRCUITS, FLOAT GOS)
```

Description

Erlang B capacity for a given configuration of circuits and grade of service (GOS).

Restrictions

None.

capacityP

Signature

```
FLOAT capacityP(INT NUM_CIRCUITS, FLOAT GOS)
```

Description

Poisson capacity for a given configuration of circuits and grade of service (GOS).

Restrictions

None.

circuits

Signature

```
INT circuits(FLOAT GOS, FLOAT OFFERED_TRAFFIC)
```

Description

Number of circuits required to meet or exceed the given grade of service (GOS) for offered load using Erlang B.

Restrictions

None.

circuitsP

Signature

```
INT circuitsP(FLOAT GOS, FLOAT OFFERED_TRAFFIC)
```

Description

Number of circuits required to meet or exceed the given grade of service (GOS) for offered load using Poisson.

Restrictions

None.

criticalTrafficCarriedC

Signature

```
FLOAT criticalTrafficCarriedC(INT N, FLOAT PT, FLOAT T1, FLOAT T2)
```

Attribute	*Description*
N	Number of servers.

Attribute	Description
PT	Probability of delay greater than T1.
T1	Acceptable delay.
T2	Mean server holding time (Average call time).

Description

Using Erlang C, calculates the critical carried traffic.

Returns (1 - PT) * A — where PT is the probability that the delay is greater than T1 and A is offered traffic in Erlangs.

Restrictions

None.

criticalTrafficOfferredC

Signature

```
FLOAT criticalTrafficOfferedC(INT N, FLOAT PT, FLOAT T1, FLOAT T2)
```

Attribute	Description
N	Number of servers.
PT	Probability of delay greater than T1.
T1	Acceptable delay.
T2	Mean server holding time (Average call time).

Description

Using Erlang C, calculates the critical offered traffic.

Restrictions

None.

delayProbabilityC

Signature

```
FLOAT delayProbabilityC(INT N, FLOAT A, FLOAT T1, FLOAT T2)
```

Attribute	Description
A	Offered traffic in Erlangs.

Attribute	*Description*
N	Number of servers.
T1	Acceptable delay.
T2	Mean server holding time (Average call time).

Description

Using Erlang C, calculates the probability of delay greater than zero.

Restrictions

None.

gos

Signature

```
FLOAT gos(INT NUM_CIRCUITS, FLOAT OFFERED_TRAFFIC)
```

Description

Erlang B derived grade of service (GOS) for given circuit/load configuration.

Restrictions

None.

gosCarried

Signature

```
FLOAT gosCarried (INT NUM_CIRCUITS, FLOAT CARRIED_TRAFFIC)
```

Description

Earlang-B derived GOS for given circuit and carried load configuration.

Restrictions

None.

gosCarriedP

Signature

```
FLOAT gosCarriedP (INT NUM_CIRCUITS, FLOAT CARRIED_TRAFFIC)
```

Description

Poisson derived GOS for given circuit and carried load configuration.

Restrictions

None.

gosP

Signature

```
FLOAT gosP(INT NUM_CIRCUITS, FLOAT OFFERED_TRAFFIC)
```

Description

Poisson-derived grade of service (GOS) for given circuit/load configuration.

Restrictions

None.

notAcceptDelayProbabilityC

Signature

```
FLOAT notAcceptDelayProbabilityC(INT N, FLOAT A, FLOAT T1,FLOAT T2)
```

Attribute	*Description*
`A`	Offered traffic in Erlangs.
`N`	Number of servers.
`T1`	Acceptable delay.
`T2`	Mean server holding time (Average call time).

Description

Using Erlang C, calculates the probability of delay greater than `T1`.

Restrictions

None.

numCircuitsC

Signature

```
FLOAT numCircuitsC(Float Acar, FLOAT PT, FLOAT T1, FLOAT T2)
```

Attribute	*Description*
`Acar`	`(1 - PT) * A`
`PT`	Probability of delay greater than `T1`.

Attribute	Description
T1	Acceptable delay.
T2	Mean server holding time (Average call time).

Description

Using Erlang C, calculates the number of servers.

Restrictions

None.

offTraffic

Signature

```
FLOAT offTraffic (INT NUMB_CIRCUITS, FLOAT CARRIED_TRAFFIC)
```

Description

Erlang-B offered load for given circuit and carried load configuration.

Restrictions

None.

offTrafficP

Signature

```
FLOAT offTrafficP (INT NUMB_CIRCUITS, FLOAT CARRIED_TRAFFIC)
```

Description

Poisson offered load for given circuits and carried load configuration.

Restrictions

None.

requiredCircuits

Signature

```
INT requiredCircuits (FLOAT NUM_CIRCUITS, FLOAT CARRIED_TRAFFIC, FLOAT
DESIGN_GOS)
```

Description

Number of circuits required to meet or exceed GOS for offered load using Erlang-B.

Restrictions

None.

requiredCircuitsP

Signature

```
INT requiredCircuitsP (FLOAT NUM_CIRCUITS, FLOAT CARRIED_TRAFFIC, FLOAT
DESIGN_GOS)
```

Description

Number of circuits required to meet or exceed GOS for the carried load using Poison.

Restrictions

None.

Kaufman-Roberts Functions

Kaufman-Roberts is a multi-dimensional Erlang method that you use when multiple services share a common resource pool. The Kaufman-Roberts functions compute the blocking probability when the total capacity of a link is composed of a different number of traffic flows or channels, and each flow or channel is smaller than the maximum capacity of the link.

Each `(A,B)` pair referred in the function expression describes a single traffic class, type, flow, or channel of traffic. All `A` values represent the arrival rate of traffic and are of type `DOUBLE`. The value for `A` must be between zero and one. All `B` values represent the traffic class. The traffic class is a numeric integer that reflects the number of basic units of traffic used by the channel and corresponds to the same basic unit represented by the capacity (C). The sequence `( a1, b1, ..., ak, bk )` describes the traffic characteristics in the system.

The following list details the parameters used in the Kaufman-Roberts functions:

- `DOUBLE P` — Represents a probability
- `INT C` — Represents a link capacity
- `INT IDX` — An index that specifies the traffic class to operate from the `( a1, b1, ... , ak, bk )` set. This value must be less than the number of `(A,B)` pairs in the function expression minus one (that is `k-1`). For example, if you have three `(A,B)` pairs, the `INT IDX` value must be less than two.

Example:

If the link capacity is 128 Kbps with channels of 32 Kbps, 48 Kbps, and 80 Kbps and assuming an arrival rate of 0.5 for each channel, the link would be described as:

C = 128

a1,b2 = 0.5, 32

a2,b2 = 0.5, 48

a3,b3 = 0.5, 80

Using the Kaufman-Roberts functions, you could calculate the blocking probability of the first channel as:

```
DOUBLE KaufmanPB( 0, 128, 0.5, 32, 0.5, 48, 0.5, 80)
```

INT KaufmanAllC

Signature

```
INT KaufmanAllC (DOUBLE P, a1, b1, ..., ak, bk)
```

Description

Calculates the minimum capacity C for the link where all services have a probability of blocking less than or equal to P.

Restrictions

None.

INT KaufmanMinC

Signature

```
INT KaufmanMinC (INT KaufmanMinC (INT IDX, DOUBLE P, a1, b1, ..., ak, bk)
```

Description

Calculates the minimum capacity C for the link where the service identified by IDX has a probability of blocking less than or equal to P.

Restrictions

None.

DOUBLE KaufmanMaxPB

Signature

```
DOUBLE KaufmanPB (INT C, a1, b1, ..., ak, bk)
```

Description

Calculates the maximum probability of blocking for all services on a link of capacity C.

Restrictions

None.

Forecast Reports

Forecast reports use a special class of functions, known as forecast functions, that are integrated with the report generation process to access regression analysis results. Regression analysis results are built internally by the application when the user requests a forecast type report. Table 3 details common terms used with forecast reports.

Table 3: Forecast Reports: Common Terms

Term	*Description*
BDAV5 —Average of top 5 busy day busy hours	Average of the top five values out of the seven that come from determining the busiest hour of each day, then ranking those daily values over the week.
Concepts Forecast Template	A traffic report template that contains fields that are specific to Forecast reports.
DABH — Daily Busy Hour	The busiest hour of the day, as defined by the attribute selected in the busy hour determiner. When used in a weekly context, this is further refined to represent the single-busiest hour of the week.
Date-time Scope	As Forecast Reports are all weekly based, the date-time scope should always cover complete weeks, with the first date falling on the start of a week as defined for your particular system. The behavior of the report generation process is undefined for date-time scopes that do not meet these criteria.
Forecast Attribute	Determines the type of busy-hour value that is used in the linear regression analysis, and also the value returned by the WM_FCAST_DIMENSION(...) function. Selected on the Report Editor, and can be one of DABH, PABH3, PABH5, BDAV5.
Forecast Field	A field to be used only in the context of a Forecast Report. These types of fields use special extensions to the Expression library that access forecast-related data (linear regression, pabh3, pabh5 values etc).
Linear Regression	Algorithm used to fit a straight line to a distribution of data points laid out on an XY Cartesian plane. The result is two attributes a, and b, which represent the intercept/slope pair for the equation y = a + bx, that describe the best fit of the line with respect to the data points.
PABH — Profile Average Busy Hour	A weekly busy hour determiner that is computed by averaging the values from the same hour each day to determine the busiest hour number over the entire week.
PABH3 —Top 3 average PABH	Average of the top three values out of the seven that come from the busiest average hour of the week.
PABH5 —Top 5 average PABH	Average of the top five values out of the seven that come from the busiest average hour of the week. This value is naturally always lower than PABH3.

Table 3: Forecast Reports: Common Terms (Continued)

Term	*Description*
Pearson Correlation Coefficient	The Pearson algorithm computes a correlation coefficient returning a value in the range -1 <= c <= 1 that measures how closely the data-points in an XY distribution fit to a straight line. An abs(coefficient) >= 0.8 is generally accepted as representing a good fit.
Start of Week	The first day of the week can fall on any weekday. This option is configured once for the entire system when it is installed and must never be modified after that. Weeks in the US are often considered to start on Sunday, whereas the ISO standard identifies Monday as the first day of a week.

The following functions are available for use in forecast reports. Function results are undefined if they are used in any other context.

WM_FCAST_PABH

Signature

```
FLOAT WM_FCAST_PABH(INT instance_id, STRING tstamp, INT n)
```

Parameter	*Description*
`instance_id`	Identifier (ID) of the element instance being evaluated.
`tstamp`	The interval start of a weekly report.
`n`	The number of values to return.

Description

Returns the N-day profile average busy hour for the given instance and time stamp.

Restrictions

This expression can be used only in the context of the focal entity of a forecast report.

Example

```
pabh3FieldName := WM_FCAST_PABH(instance_id, TimeAndElement.tstamp, 3)
pabh5FieldName := WM_FCAST_PABH(instance_id, TimeAndElement.tstamp, 5)
```

WM_FCAST_BDBH

Signature

```
FLOAT WM_FCAST_BDBH(INT instance_id, STRING tstamp)
```

Parameter	Description
instance_id	Identifier (ID) of the element instance being evaluated.
tstamp	The interval start of a weekly report.

Description

Returns the busy day busy hour for the given instance and time stamp, where time stamp is the interval start of (typically) a week.

Restrictions

This expression can only be used in the context of the focal entity of a forecast report.

Example

```
bdbhFieldName := WM_FCAST_BDBH(instance_id, TimeAndElement.tstamp)
```

WM_FCAST_DIMENSION

Signature

```
FLOAT WM_FCAST_DIMENSION(INT instance_id, STRING tstamp)
```

Parameter	Description
instance_id	Identifier (ID) of the element instance being evaluated.
tstamp	The interval start of a weekly report.

Description

Returns the selected dimension parameter for the given instance and time stamp. The dimension parameter can be one of PABH3, PABH5 or BDBH, depending on the selection made by the user in the forecast report definition. The value returned by this function is also used to compute the linear trend.

Restrictions

This expression can only be used in the context of the focal entity of a forecast report.

Example

```
dimensionFieldName := WM_FCAST_DIMENSION(instance_id, TimeAndElement.tstamp)
```

WM_FCAST_GROWTH

Signature

```
FLOAT WM_FCAST_GROWTH(INT instance_id)
```

Parameter	*Description*
instance_id	Identifier (ID) of the element instance being evaluated.

Description

Returns the growth factor, or slope computed as part of the linear regression analysis of the dimension parameter.

Restrictions

This expression can only be used in the context of the focal entity of a forecast report.

Example

```
growthFieldName := WM_FCAST_GROWTH(instance_id)
```

WM_FCAST_THRESH

Signature

```
FLOAT WM_FCAST_THRESH()
```

Description

Returns the forecast threshold value as entered on the report definition form.

Restrictions

This expression can only be used in the context of a forecast report.

WM_FCAST_CCOTHRESH

Signature

```
FLOAT WM_FCAST_CCOTHRESH()
```

Description

Returns the forecast correlation coefficient threshold value as entered on the report definition form.

Restrictions

This expression can only be used in the context of a forecast report.

WM_FCAST_CURRTHRESH

Signature

```
FLOAT WM_FCAST_CURRTHRESH()
```

Description

Returns the forecast current threshold value as entered on the report definition form.

Restrictions

This expression can only be used in the context of a forecast report.

WM_FCAST_FINALTHRESH

Signature

```
FLOAT WM_FCAST_FINALTHRESH()
```

Description

Returns the forecast final threshold value as entered on the report definition form.

Restrictions

This expression can only be used in the context of a forecast report.

WM_FCAST_DAYS

Signature

```
INT WM_FCAST_DAYS(INT n)
```

Parameter	*Description*
n	Determines which text box on the report definition form is used. Values can be 1, 2, or 3 for the N1, N2, and N3 text boxes, respectively.

Description

Returns the number of days to look ahead as configured on the report definition form. Return values are typically 60 for n=1, 90 for n=2, or 120 for n=3.

Restrictions

This expression can only be used in the context of a forecast report.

WM_FCAST_CORRELATION

Signature

```
FLOAT WM_FCAST_CORRELATION(INT instance_id)
```

Parameter	*Description*
instance_id	Identifier (ID) of the element instance being evaluated.

Description

Returns the Pearson correlation coefficient for the given instance. This can be used to make an expression that compares the correlation value against a threshold entered on the form. The resulting boolean, which is actually the integer 1 or 0, can be used in a template criteria expression to filter out the rows (Cells) that do not pass the test.

Restrictions

This expression can only be used in the context of the focal entity of a forecast report.

Example

```
CORRELATION := WM_FCAST_CORRELATION(instance_id)
CRTNTEST := WM_FCAST_CORRELATION(instance_id) > WM_FCAST_CCOTHRESH()
```

WM_FCAST_SAMPLES

Signature

```
FLOAT WM_FCAST_SAMPLES(INT instance_id)
```

Parameter	*Description*
instance_id	Identifier (ID) of the element instance being evaluated.

Description

Returns the number of data points used in the regression analysis for the given instance.

Restrictions

This expression can only be used in the context of the focal entity of a forecast report.

Building Expressions from Forecast Functions

Forecast functions are extremely useful when used together to create complex expressions.

Weekly growth rate

The forecast growth field is based on growth-rate per second. To turn this value into a weekly growth rate, the following formula can be used:

```
growthPerSecond := WM_FCAST_GROWTH(instance_id) growthPerWeek := growthPer-
Second * 3600 * 24 * 7
```

Most recent dimension parameter value

The most recent value of the dimension parameter (from the last week in the date-time scope) is available using this expression:

```
dimension := WM_FCAST_DIMENSION(instance_id, TimeAndElement.tstamp)
```

It is important to remember that the actual dimension value depends on the type of busy hour result selected for the report (PABH3, PABH5, BDBH, and so farth).

Comparing the correlation value against a threshold

The Pearson correlation coefficient reveals how closely the straight line approximates the actual data points. It is accessible by using the following expression:

```
correlation := WM_FCAST_CORRELATION(instance_id)
```

Filtering rows that don't exceed the final threshold

To filter out rows that don't exceed the final threshold, you can use the following expression combined with a criteria test in the template:

```
filter := dimension > WM_FCAST_FINALTHRESH()
```

Exhaustion date

An exhaustion date is some date (usually in the future) when an extrapolation of the current value and estimated growth rate would exceed a fixed threshold. This can be computed using the date-math capabilities:

```
exhaustionDate := dateToString((stringToDate(TimeAndElement.tstamp, "%Y-%m-
%d") + (INT)((WM_FCAST_FINALTHRESH() - dimension) / growthPerSecond), "%Y-
%m-%d")
```

Note: The growth rate is based on seconds, the same unit used in the date math operations.

Projection offset values

The report definition editor has three fields for the user to enter projection offsets in terms of days. These values can be accessed by using the WM_FCAST_DAYS(n) function to generate projected values:

```
projection_1 := dimension + (growthPerWeek / 7 * WM_FCAST_DAYS(1))
projection_2 := dimension + (growthPerWeek / 7 * WM_FCAST_DAYS(2))
projection_3 := dimension + (growthPerWeek / 7 * WM_FCAST_DAYS(3))
```

Bridge Functions

The following is a list of available bridge functions.

bridge

Signature

```
ATTR bridge(ATTR local, RELN r, ATTR remote, ATTR val)
```

Description

The bridge function is used to traverse a one-many relationship and extract a single value at the far end of the relationship for the first object found where an expression result at the local object equals a result at the remote object.

Restrictions

The bridge function can cause problems when used in conjunction with stored busy hour reports if the relation argument traverses entities above the cluster entity.

Caution: If you use this function on an entity that has a great number of instances, it could impair system performance and lead to extended report generation times.

Note: Prospect Web 1.3 does not support this function.

---

bridgeList

Signature

```
AttrList bridgeList(ATTR local, RELN r, ATTR remote)
```

Description

This function is related to the bridge function, but instead of returning an attribute value, it returns a list of objects that can then be used in an aggregation function such as sum as in:

```
sum(bridgeList(LocalKey, Cell.Antenna.Ant_Carrier, LocalKey), foo)
```

Restrictions

The bridgeList function can cause problems when used in conjunction with stored busy hour reports if the relation argument traverses entities above the cluster entity.

Caution: If you use this function on an entity that has a great number of instances, it could impair system performance and lead to extended report generation times.

Note: Prospect Web 1.3 does not support this function.

Instance Name Mapping Functions

The following is a list of instance name mapping functions.

reportName

Signature

```
STRING reportName(INT InstanceId)
```

Description

Returns the report name defined by the reportnameexpr field in wmn_entity for this entity type. If the entity is not a supported scenario entity, then the name of the first parent entity that is supported in the scenario is used, and LocalKey values are appended to complete the name.

instanceName

Signature

```
STRING instanceName(INT InstanceId)
```

Description

At the time of writing, this function is identical to reportName(INT InstanceId).

instanceFDN

Signature

```
STRING instanceFDN(INT InstanceId)
```

Description

Returns a fully distinguished name for the given object by concatenating the LocalKey values (from wmn_instance) for all the objects ancestors starting from System.

scenarioInstance

Signature

```
INT scenarioInstance(INT InstanceId)
```

Description

Returns the instance_id of the first ancestor entity that is supported by the scenario. If the entity of the instance is itself supported by the scenario, then that instance is returned.

Trending Reports

Trending reports, like forecasting reports, use a special class of functions known as trending functions that are integrated with the report generation process to access regression analysis results.

Regression analysis results are built internally by the application when the user includes one or more forecasting UDCs (a UDC whose expression consists only of a trend() or projectTime() function call) in a report.

Trending reports are always grouped by day, automatically if necessary.

trend

Signature

```
FLOAT trend(RELATION kci_to_trend, STRING bh_cluster_entity, STRING
stored_busy_hour_name, STRING regression_type, INT number_of_days_regression,
INT min_datapoints [OPTIONAL])
```

Description

Returns the best-fit straight line value of the `kci_to_trend` calculated for the particular timestamp. Instances must have at least `min_datapoints` days worth of data to be included in the result set.

Restrictions

1. This function must be the only expression component when used in a UDC.
2. The busy hour used (in arguments 2 and 3) must be at the report focal entity or below.
3. Argument `number_of_days_regression` must be greater or equal to argument `min_datapoints`.
4. The only valid regression type at this time is "LINEAR" (in argument 4).
5. If the optional argument `min_datapoints` is omitted, the system default "10" is used.

Note: The system default is configurable by using the `trending_admin.sh` script.

---

6. Any report containing a trending UDC will be automatically grouped by DAY.

Example

```
trend([BTS_Cell]![someField], "BTS_Cell", "trendingBH", "LINEAR", 30, 5)
```

The expression returns the predicted value of [BTS_Cell]![someField], based upon the last 30 days worth of data for that field at the time of the buy hour BTS_Cell.forecastingBH. Each instance in the final result has at least 5 days worth of busy hour data.

projectTime

Signature

```
FLOAT projectTime(RELATION kci_to_project, STRING bh_cluster_entity, STRING
stored_busy_hour_name, STRING regression_type, INT number_of_days_regression,
FLOAT limit, INT min_datapoints [OPTIONAL])
```

Description

Returns the number of days until the specified limit is breached. Instances must have at least `min_datapoints` days worth of data to be included in the result set.

Restrictions

1. This function must be the only expression component when used in a UDC.
2. The busy hour used (in arguments 2 and 3) must be at the report focal entity or below.
3. Argument `number_of_days_regression` must be greater or equal to argument `min_datapoints`.
4. The only valid regression type at this time is "LINEAR" (in argument 4).
5. If the optional argument `min_datapoints` is omitted, the system default "10" is used.

Note: The system default is configurable by using the `trending_admin.sh` script.

---

6. Any report containing a trending UDC will be automatically grouped by DAY.

Example

```
projectTime([BTS_Cell]![someField], "BTS_Cell", "trendingBH", "LINEAR", 30,
10, 5)
```

This expression returns the number of days until [BTS_Cell]![someField] reaches a value of 10, based upon the last 30 days worth of data for that field at the time of the buy hour BTS_Cell.forecastingBH. Each instance in the final result has at least 5 days worth of busy hour data.

getKCILimit

Signature

```
FLOAT getKCILimit(STRING entity, STRING field, STRING limit_name)
```

Description

Returns the value of the specified limit, as per defined in the specified field.

Restrictions

The `limit_name` must be one of: "OperationalLimit", "PlanningLimit", "BaseUtilization", "LeadTime"

Example

```
getKCILimit("BTS_Cell", "someField", "PlanningLimit")
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785,
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

Printed in the Republic of Ireland.