

# Expressions Technical Reference Prospect®



Copyright © 2007 Vallent Corporation and/or its subsidiaries All rights reserved.

Information in these materials is furnished for informational use only, is subject to change without notice and does not represent a commitment on the part of Vallent Corporation ("Vallent"). These materials, as well as the software described herein ("Software"), are furnished under license; there is no transfer of title. The Software is subject to the license agreement that accompanies or is included with the Software, which specifies the permitted and prohibited uses of the Software. Any unauthorized duplication or use of Vallent Software, in whole or in part, in print or in any other storage and retrieval system is prohibited.

Furthermore, the purchase of Vallent products shall not be deemed to grant either directly or by implication, estoppel or otherwise, any license under the copyrights, trademarks, patents or patent applications of Vallent as may arise by operation of law in the sale of a product.

It is possible that these materials may contain references to, or information about, Vallent products or services that are not announced or available in your country. Such references or information must not be construed to mean that Vallent intends to announce or make available such products or services in your country.

These materials are protected by copyright laws and international copyright treaties; no part of these materials may be copied, distributed, downloaded, modified, posted, published, reproduced, reused, re-posted, stored in a retrieval system, transcribed, translated into any language or computer language, transmitted, uploaded or otherwise used in any form or by any means (electronic, mechanical, recording or otherwise) without the prior written permission of Vallent.

While reasonable efforts have been made to ensure the accuracy of these materials, Vallent assumes no responsibility or liability for any errors, inaccuracies or omissions that may appear in these materials or from the use of these materials. Use these materials at your own risk.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, VALLENT AND ITS SUPPLIERS DISCLAIM ANY AND ALL WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT, AND THOSE ARISING OUT OF USAGE OF TRADE OR COURSE OF DEALING, CONCERNING THESE MATERIALS. THESE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL VALLENT OR ITS SUPPLIERS (OR THEIR RESPECTIVE AGENTS, DIRECTORS, EMPLOYEES OR REPRESENTATIVES) BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, CONSEQUENTIAL, INCIDENTAL, DIRECT, INDIRECT, SPECIAL, ECONOMIC, PUNITIVE OR SIMILAR DAMAGES, OR DAMAGES FOR LOSS OF BUSINESS PROFITS, LOSS OF GOODWILL, BUSINESS INTERRUPTION, COMPUTER FAILURE OR MALFUNCTION, LOSS OF BUSINESS INFORMATION OR ANY AND ALL OTHER COMMERCIAL OR PECUNIARY DAMAGES OR LOSSES) ARISING OUT OF THE PURCHASE OR USE OF THESE MATERIALS, HOWEVER CAUSED AND ON ANY LEGAL THEORY OF LIABILITY (WHETHER IN TORT, CONTRACT OR OTHERWISE), EVEN IF VALLENT

HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY. Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

THESE MATERIALS ARE INTENDED FOR STANDARD COMMERCIAL USES; THEY MUST NOT BE USED IN ANY HAZARDOUS ENVIRONMENTS REQUIRING FAIL SAFE PERFORMANCE, SUCH AS IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION OR COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL, DIRECT LIFE SUPPORT MACHINES, OR WEAPONS SYSTEMS, IN WHICH THE FAILURE OF THESE MATERIALS COULD RESULT IN DEATH, PERSONAL INJURY, OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE. YOU HEREBY AGREE THAT THE USE OF THESE MATERIALS FOR ANY SUCH APPLICATION IS AT YOUR SOLE RISK. YOU AGREE TO INDEMNIFY AND HOLD VALLENT HARMLESS FROM ANY CLAIMS FOR LOSS, COST, DAMAGE, EXPENSE OR LIABILITY THAT MAY ARISE OUT OF OR IN CONNECTION WITH THE USE OF THESE MATERIALS IN SUCH APPLICATIONS.

Unless otherwise noted, all names of companies, products, street addresses, data, characters and persons contained herein are part of a completely fictitious scenario or scenarios, are designed solely to document the use of a Vallent product, and are in no way intended to represent any real individual, company, product or event.

Vallent, the Vallent logo, Vallent Control, Vallent Design, Vallent Pilot, Virtuo, Prospect, Metrica Performance Manager, ServiceAssure, NetworkAssure and BusinessAssure are either registered trademarks or trademarks of Vallent and/or its subsidiaries in the United States and/or other countries.

All other trademarks, trade names or company names referenced herein are used for identification only and are the property of their respective owners.

US Government Restricted Rights: These materials are provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the Government is subject to restriction as set forth in subparagraph (c)(1)(ii) of the rights in Technical Data and Computer Software clause at DFAR 252.227-7013, or as set forth in the particular department or agency regulations or rules which provide Vallent protection equivalent to or greater than the above-cited clause. The contractor/manufacturer is Vallent Corporation, 13431 NE 20th Street, Bellevue, WA 98005, USA.



---

---

# Table of Contents

<b>1</b>	<b>About This Documentation</b>	<b>9</b>
	Audience	9
	Required Skills & Knowledge	9
	Document Conventions	10
	User Publications	11
	Training and Technical Support	12
<b>2</b>	<b>General Expressions</b>	<b>13</b>
	- Operator (Unary)	16
	! Operator	16
	== Operator (also =)	17
	!= Operator (also <>)	18
	< Operator	18
	<= Operator	19
	> Operator	20
	>= Operator	20
	+ Operator	21
	- Operator (Binary)	22
	* Operator	24
	/ Operator	25
	% Operator	26
	^ Operator	27
	() Operator (Cast)	28
	abs	29
	AND Operator (also &&)	29
	arcCos	30
	arcSin	31
	arcTan	31
	Arithmetic Operators	32
	average	33
	Binary Operators	33
	ceil	34
	concat	34
	Conditional Operator (? :)	35
	Constants	36
	cos	37
	count	38
	Date Functions	39
	dateToString	40
	dayOfMonth	40
	dayOfYear	41
	decode()	41

**EXPRESSIONS TECHNICAL REFERENCE**  
Prospect®

---

degrees	42
distance	42
exp	43
Filters	44
firstDayOfMonth	44
firstIndex	45
floor	45
Functions	46
Gateway Attribute	47
Geometry Functions	49
getDateFromTime	50
getenv	50
getTimeFromDate	51
hour	52
hourGMT	52
inPolygon	53
ipAddressClass	53
ipAddressToInt	54
ipApplyNetmask	55
ipApplyNetmaskMerge	56
ipByteOfAddress	57
ipBytesToAddress	57
ipCommonNetmask	58
ipCommonNetmaskBits	59
ipDefaultNetmask	59
ipDefaultNetmaskBits	60
Ip Functions	61
ipIntToAddress	62
ipIsBroadcast	62
ipIsLoopback	63
ipIsNormal	64
ipIsPrivate	64
ipIsUnknown	65
ipIsValid	66
isLeapYear	66
ipNetmaskMerge	67
isNotNull	68
isNull	68
lastIndex	69
length	69
ln	70
log	71
Logical Operators	71
matchesRE	72
Math Functions	73
max	73
mean	74

---

**EXPRESSIONS TECHNICAL REFERENCE**  
Prospect®

---

median	75
min	76
minute	77
minuteGMT	78
mode	79
month	80
monthName	80
nextDate	81
nullDate	81
nullFloat	82
nullInt	83
nullString	83
nullTime	84
nullValue	85
numToString	85
Operators	86
OR Operator (also   )	88
package	89
Package Functions	89
parameterAsString	91
pi	93
previousDate	93
protect	94
radians	94
replaceRE	95
replaceAllRE	96
reverse	96
round	97
second	98
seconds	98
sin	99
sqr	100
sqrt	100
stddev	101
String Functions	102
stringToDate	103
stringToFloat	103
stringToInt	104
stringToTime	105
subString	106
subStringRE	106
sum	107
sysdate	108
System Functions	109
systemtime	109
tan	110
Time Functions	111

---

Updated: 2007-07-02

**EXPRESSIONS TECHNICAL REFERENCE**  
Prospect®

---

timeToString .....	111
tokenize .....	112
toLower .....	113
toUpper .....	113
Trig Functions .....	114
trunc .....	114
Unary Operators .....	115
unescape .....	116
validDate .....	116
validTime .....	117
variance .....	117
vsum() .....	119
weekDay .....	119
weekDayName .....	120
year .....	120
<b>3 Advanced Expressions .....</b>	<b>123</b>
Erlang Functions .....	123
Kaufman-Roberts Functions .....	130
Forecast Reports .....	132
Building Expressions from Forecast Functions .....	138
Bridge Functions .....	139
Instance Name Mapping Functions .....	140
<b>Index .....</b>	<b>143</b>



# 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 02 July 2007.

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

## ***Audience***

This guide is intended for technicians and engineers who use the 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 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 Prospect software works

## Document Conventions

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

**Table 1:** General Document Conventions

<b>Format</b>	<b>Examples</b>	<b>Description</b>
ALL UPPERCASE	<ul style="list-style-type: none"> <li>• GPS</li> <li>• NULL</li> <li>• MYWEBSERVER</li> </ul>	Acronyms, device names, logical operators, registry keys, and some data structures.
<u>Underscore</u>	See <a href="#">Document Conventions</a>	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.
<b>Bold</b>	<ul style="list-style-type: none"> <li>• <b>Note:</b> The busy hour determiner is...</li> </ul>	Heading text for Notes, Tips, and Warnings.
SMALL CAPS	<ul style="list-style-type: none"> <li>• The STORED SQL dialog box...</li> <li>• ...click VIEW...</li> <li>• In the main GUI window, select the FILE menu, point to NEW, and then select TRAFFIC TEMPLATE.</li> </ul>	Any text that appears on the GUI.
<i>Italic</i>	<ul style="list-style-type: none"> <li>• A <i>busy hour</i> is...</li> <li>• A web server <i>must</i> be installed...</li> <li>• See the <i>User Guide</i></li> </ul>	New terms, emphasis, and book titles.
Monospace	<ul style="list-style-type: none"> <li>• <code>./wminstall</code></li> <li>• <code>\$ cd /cdrom/cdrom0</code></li> <li>• <code>/xml/dict</code></li> <li>• <code>http://java.sun.com/products/</code></li> <li>• <code>addmsc.sh</code></li> <li>• <code>core.spec</code></li> <li>• Type OK to continue.</li> </ul>	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.
<b>Monospace Bold</b>	<pre>[root] # pkginfo   grep -i perl system Perl5 On-Line Manual Pages system Perl 5.6.1 (POD Documenta- tion) system Perl 5.6.1</pre>	For contrast in a code example to show lines the user is expected to enter.
<Mono- space italics>	<pre># cd &lt;oracle_setup&gt;</pre>	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]	<pre>log-archiver.sh [-i] [-w] [-t]</pre>	Used in code examples: indicates options.

## User Publications

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

**Table 2:** Prospect User Documentation

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

## Viewing the Desktop Client Help Publications

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

When Help files are updated, they are downloaded automatically from the Prospect server to the 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/>.

## ***Training and Technical Support***

Both training and technical support are available for Prospect software. For technical support, contact us at [wmcustomersupport@vallent.com](mailto:wmcustomersupport@vallent.com). For training, contact us at [training@vallent.com](mailto:training@vallent.com).

## 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

<i>Variable</i>	<i>Description</i>	<i>Comments</i>
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

<i>Element</i>	<i>R/O</i>	<i>Description</i>
comments	O	The character # starts a comment, which terminates with the next newline character.
Unary	O	Expression in the form: <i>operator expression</i> .
Binary	O	Expression in the form: <i>expression operator expression</i>
Conditional	O	Expression in the form: <i>expression ? expression : expression</i>
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: <i>functionName (expr1, ..., exprN)</i>

## 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**

<i>Expression</i>	<i>Result</i>
-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**

<i>Expression</i>	<i>Result</i>
!1	0



<i>Expression</i>	<i>Result</i>
!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

<i>Expression</i>	<i>Result</i>
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**

<b><i>Expression</i></b>	<b><i>Result</i></b>
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
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.

<i>Type1</i>	<i>Type2</i>	<i>Result Type</i>	<i>Commutative</i>
Int	Int	Int	1
Int	Float	Float	1

**EXPRESSIONS TECHNICAL REFERENCE**  
Prospect®

General Expressions

---

<i>Type1</i>	<i>Type2</i>	<i>Result Type</i>	<i>Commutative</i>
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

<i>Expression</i>	<i>Result</i>
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.

<i>Type1</i>	<i>Type2</i>	<i>Result Type</i>	<i>Commutative</i>
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
'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.

<i>Type1</i>	<i>Type2</i>	<i>Result Type</i>	<i>Commutative</i>
Int	Int	Int	1
Int	Float	Float	1
Float	Float	Float	1

## Examples

<i>Expression</i>	<i>Result</i>
2 * 0	0
2 * 2	4



<i>Expression</i>	<i>Result</i>
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.

<i>Type1</i>	<i>Type2</i>	<i>Result Type</i>	<i>Commutative</i>
Int	Int	Int	1
Int	Float	Float	1
Float	Float	Float	1

## Examples

<i>Expression</i>	<i>Result</i>
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.

<i>Type1</i>	<i>Type2</i>	<i>Result Type</i>	<i>Commutative</i>
Int	Int	Int	1

## Examples

<i>Expression</i>	<i>Result</i>
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.

<i>Type1</i>	<i>Type2</i>	<i>Result Type</i>	<i>Commutative</i>
Int	Int	Float	1
Int	Float	Float	1
Float	Float	Float	1

## Examples

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
(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**

<b><i>Expression</i></b>	<b><i>Result</i></b>
<i>abs</i> (10)	10
<i>abs</i> (-10)	10
<i>abs</i> (1.0 * 10)	10.000000
<i>abs</i> (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

<i>Expression</i>	<i>Result</i>
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  $[0, \text{pi} ()]$ .

## Syntax

FLOAT `arcCos`(NUMBER *N*)

## Examples

<i>Expression</i>	<i>Result</i>
<code>arcCos(1)</code>	0.000000
<code>arcCos(-1)</code>	3.141593
<code>arcCos(0.5)</code>	1.047198
<code>arcCos(0)</code>	1.570796
<code>arcCos(-0.5)</code>	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**

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
<code>arcTan(1)</code>	0.785398
<code>arcTan(-1)</code>	-0.785398
<code>arcTan(-0.5)</code>	-0.463648
<code>arcTan(0.5)</code>	0.463648
<code>arcTan(0)</code>	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**

<i>Expression</i>	<i>Result</i>
<code>ceil (1.3)</code>	2.000000
<code>ceil (1.5)</code>	2.000000
<code>ceil (1.7)</code>	2.000000
<code>ceil (-1.3)</code>	-1.0000000
<code>ceil (-1.5)</code>	-1.0000000
<code>ceil (-1.7)</code>	-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  $a_1 + a_2 + \dots + a_N$ .

### **Syntax**

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

## Examples

<i>Expression</i>	<i>Result</i>
<code>concat("Current Date is ", sysdate())</code>	Current Date is 09/25/98
<code>concat("Hello ", 2, " you")</code>	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

<i>Expression</i>	<i>Result</i>
<code>isNull(10 / 0) ? "NULL" : (STRING) (10 / 0)</code>	NULL
<code>10 &lt; 0 ? "true"</code>	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

<i>Element</i>	<i>Description</i>
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

<i>Expression</i>	<i>Result</i>
<code>cos(pi ())</code>	-1.000000
<code>cos(0)</code>	1.000000
<code>cos(radians (90))</code>	0.000000
<code>cos(radians (270))</code>	0.000000
<code>cos(pi () /2)</code>	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.

<i>Expression</i>	<i>Result</i>
<code>count ()</code>	3 # Current package

<i>Expression</i>	<i>Result</i>
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 = "ERICSON"] .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**

<i>Expression</i>	<i>Result</i>
<code>dateToString ('1/9/99', "%B %b %d")</code>	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

<i>Expression</i>	<i>Result</i>
<code>dayOfMonth('1/9/99')</code>	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

<i>Expression</i>	<i>Result</i>
<code>dateOfYear ('2/9/99')</code>	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

<i>Expression</i>	<i>Result</i>
<code>decode (NBType, "I", HOVERCNT, null-Int())</code>	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

<i>Expression</i>	<i>Result</i>
<code>degrees (pi ())</code>	180.000000
<code>degrees (2 * pi ())</code>	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](#)

## 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

<i>Expression</i>	<i>Result</i>
<code>exp(2) == exp(1) ^ 2</code>	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

<i>Expression</i>	<i>Result</i>
<code>10 &gt; 20</code>	0
<code>10 &lt; 20</code>	1
<code>length("hello") == 5</code>	1
<code>'10/10/98' &lt; '10/10/97'</code>	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

<i>Expression</i>	<i>Result</i>
<code>firstDayOfMonth('1/9/99', 3)</code>	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

<i>Expression</i>	<i>Result</i>
<code>firstIndex("Hello World Hello", "Hello")</code>	0
<code>firstIndex("Hello World Hello", "World")</code>	6
<code>firstIndex("Hello World Hello", "Today")</code>	-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

<i>Expression</i>	<i>Result</i>
<code>floor(1.3)</code>	1.000000
<code>floor(1.5)</code>	1.000000
<code>floor(1.7)</code>	1.000000
<code>floor(-1.3)</code>	-2.000000
<code>floor(-1.5)</code>	-2.000000
<code>floor(-1.7)</code>	-2.000000

## 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

<i>Elements</i>	<i>Description</i>
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.

<i>Expression</i>	<i>Result</i>
<code>isNull (10 / (2 - 2))</code>	1
<code>sqrt(2 ^ 2)</code>	2.0
<code>substring("Hello World", 6, 5)</code>	World
<code>length(subString("Hello World", 6, 5))</code>	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 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"
```

<i>Expression</i>	<i>Result</i>
<code>getenv("GREETING")</code>	"Hello"
<code>getenv("GREETING", "Good Morning")</code>	"Hello"
<code>getenv("MY_GREETING")</code>	NULL
<code>getenv("MY_GREETING", "Good Morning")</code>	"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

<i>Expression</i>	<i>Result</i>
<code>getTimeFromDate('1/9/99') == '1/9/99 00:00:00'</code>	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**

<i>Expression</i>	<i>Result</i>
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)

<i>High order bits</i>	<i>Class</i>	<i>Address range</i>
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
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
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

<b>Expression</b>	<b>Result</b>
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

<b>Expression</b>	<b>Result</b>
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



<i>Expression</i>	<i>Result</i>
<code>ipNetmaskMerge ("10.57.180.169", "192.168.179.12", "255.255.0.255")</code>	NULL
<code>ipNetmaskMerge ("10.57.180.169", "192.168.179.12", 33)</code>	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

<i>Expression</i>	<i>Result</i>
<code>ipByteOfAddress ("204.57.184.198")</code>	198
<code>ipByteOfAddress ("204.57.184.198",</code>	204
<code>ipByteOfAddress ("www.vallent.com", 1)</code>	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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
<code>ipCommonNetmaskBits ("10.57.180.169", "10.57.184.198")</code>	20
<code>ipCommonNetmaskBits ("10.57.176.169", "267.0.0.1")</code>	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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
<code>ipDefaultNetmaskBits("127.0.0.1")</code>	8
<code>ipDefaultNetmaskBits("www.vallent.com")</code>	NULL
<code>ipDefaultNetmaskBits("204.57.184.198")</code>	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 is not in the range of a unsigned 32 bit integer (0 - 4294967295).

### **Syntax**

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

### **Examples**

<b><i>Expression</i></b>	<b><i>Result</i></b>
<code>ipIntToAddress(3426334918)</code>	204.57.184.198
<code>ipIntToAddress(-3)</code>	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

<i>Expression</i>	<i>Result</i>
<code>ipIsBroadcast("255.255.255.255")</code>	1
<code>ipIsBroadcast("10.255.255.255")</code>	1
<code>ipIsBroadcast("10.1.255.255")</code>	1
<code>ipIsBroadcast("204.57.184.255")</code>	1
<code>ipIsBroadcast("255.255.255.1")</code>	0
<code>ipIsBroadcast("127.0.0.1")</code>	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

<i>Expression</i>	<i>Result</i>
<code>ipIsLoopback("127.0.0.1")</code>	1
<code>ipIsLoopback("10.25.251.1")</code>	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**

<b><i>Expression</i></b>	<b><i>Result</i></b>
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

<i>Expression</i>	<i>Result</i>
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](#)

## *ipIsUnknown*

## Applies to

[Ip Functions](#)

## Description

Returns 1 if 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

<i>Expression</i>	<i>Result</i>
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 qqg.rrr.sss.ttt where qqg, 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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
<code>isLeapYear('1/9/99')</code>	0
<code>isLeapYear('1/9/00')</code>	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

<i>Expression</i>	<i>Result</i>
<code>ipNetmaskMerge("10.57.180.169")</code>	10.0.0.0
<code>ipNetmaskMerge("172.16.1.1", "192.168.179.12")</code>	172.16.176.0
<code>ipNetmaskMerge ("172.16.1.1","192.168.179.12","255.0.0.0")</code>	172.168.176.0
<code>ipNetmaskMerge("172.16.1.1", "192.168.179.12", 8)</code>	172.168.176.0
<code>ipNetmaskMerge("10.57.180.169", 20)</code>	10.57.176.0
<code>ipNetmaskMerge("254.10.15.1", "255.255.0.0")</code>	254.10.0.0
<code>ipNetmaskMerge("254.10.15.1")</code>	NULL
<code>ipNetmaskMerge ("10.57.180.169","192.168.179.12","255.255.0.255")</code>	NULL
<code>ipNetmaskMerge("10.57.180.169","192.168.179.12",33)</code>	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:

<i>Expression</i>	<i>Result</i>
<code>isNotNull (DateAttr)</code>	1
<code>isNotNull (10 / 0)</code>	1
<code>isNotNull (DateAttr) ? "NULL" : (STRING) DateAttr</code>	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:

<i>Expression</i>	<i>Result</i>
<code>isNull(DateAttr)</code>	1
<code>isNull(10 / 0)</code>	1
<code>isNull(DateAttr) ? "NULL" : (STRING)DateAttr</code>	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**

<i>Expression</i>	<i>Result</i>
<code>lastIndex("Hello World Hello", "Hello")</code>	12
<code>lastIndex("Hello World Hello", "World")</code>	6
<code>lastIndex("Hello World Hello", "Today")</code>	-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

<i>Expression</i>	<i>Result</i>
length ("Hello")	5
length ("Date is " + '09/25/98')	16

## Related Topic

[subString](#)

## In

## Applies To

[Math Functions](#)

## Description

Calculates the natural logarithm of  $N$  expressed as a float. If  $N \leq 0.0$ , then a warning is output and NULL is returned. Satisfies the equation:

$\ln(\exp(N)) == N$

## Syntax

FLOAT ln (NUMBER N)

## Examples

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
<code>matchesRE ("TFL010043", "TFL")</code>	1
<code>matchesRE ("TFL010043", "[0-9]+")</code>	1
<code>matchesRE ("012-01098", "[A-Z]+")</code>	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
<code>max(package(Bts).Activated)</code>	<code>'10/10/98'</code>
<code>max(package(Bts), TotalCalls - DroppedCalls)</code>	8000
<code>max(package(Bts), ((TotalCalls - DroppedCalls) / (FLOAT)TotalCalls) * 100)</code>	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
<code>min(package(Bts).Activated)</code>	<code>'08/01/98'</code>
<code>min(package(Bts), TotalCalls - DroppedCalls)</code>	10
<code>min(package(Bts), ((TotalCalls - DroppedCalls) / (FLOAT)TotalCalls) * 100)</code>	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 the 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  $m_1$  and  $m_2$  are the modes, then  $m_1$  is returned when  $m_1 \leq m_2$ . Otherwise,  $m_2$  is returned.

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

In the first syntax example,  $A$  is in the format  $rel1.rel2. \dots .relN.Attr$  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  $rel1.rel2. \dots .relN$  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
<code>mode(package (Bts) .Activated)</code>	'08/01/98'
<code>mode(package (Bts) .Vendor)</code>	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**

<i>Expression</i>	<i>Result</i>
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.

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
<code>nextDate('1/9/99', 3)</code>	<code>'1/12/99'</code>

## Related Topics

[previousDate](#)

## *nullDate*

### Applies To

[System Functions](#)

### Description

Returns a null value where type DATE is required.

### Syntax

```
DATE NullDate()
```

## Examples

<i>Expression</i>	<i>Result</i>
<code>isNull (NullDate ())</code>	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

<i>Expression</i>	<i>Result</i>
<code>isNull (NullFloat ())</code>	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**

<i>Expression</i>	<i>Result</i>
isNull (NullInt ())	1

### **Related Topic**

[isNull](#)  
[nullDate](#)  
[nullFloat](#)  
[nullValue](#)  
[nullString](#)  
[nullTime](#)  
[nullValue](#)

## ***nullString***

### **Applies To**

[System Functions](#)

### **Description**

Returns a null value where type STRING is required.

### **Syntax**

STRING NullString()

## Examples

<i>Expression</i>	<i>Result</i>
<code>isNull (NullString())</code>	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

<i>Expression</i>	<i>Result</i>
<code>isNull (NullTime())</code>	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.

<b><i>Expression</i></b>	<b><i>Result</i></b>
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

<i>Expression</i>	<i>Result</i>
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.

<b>Opera- tor</b>	<b>Description</b>	<b>Syntax</b>
!	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

<i>Opera- tor</i>	<i>Description</i>	<i>Syntax</i>
&&	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

<i>Expression</i>	<i>Result</i>
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

<i>Element</i>	<i>Description</i>
ATTR	Function specific return type.
<i>package_function</i>	Name of the function.
ATTR_SPECIFIER	In the format <i>rel1.rel2. ... .relN.Attr</i> . See <a href="#">Gateway Attribute</a> , sections “Attributes Through One-to-Many Relations” and “Accessing Every Object in a Package.”
RELATION_SPECIFIER	In the format <i>rel1.rel2. ... .relN</i> . See <a href="#">Gateway Attribute</a> , sections “Attributes Through One-to-Many 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.

<i><b>Expression</b></i>	<i><b>Result</b></i>
<code>parameterAsString("perf.start")</code>	24 May 1999
<code>parameterAsString("perf.duration")</code>	NULL
<code>parameterAsString("perf.duration", "NO VALUE")</code>	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.

<i><b>Expression</b></i>	<i><b>Result</b></i>
<code>parameterAsString("DateRange")</code>	24 May 1999 .. 25 May 1999
<code>parameterAsString("DateRange:start")</code>	24 May 1999
<code>parameterAsString("DateRange:end")</code>	25 May 1999
<code>parameterAsString("TimeRange")</code>	11:53:36 .. 11:55:36
<code>parameterAsString("TimeRange:date")</code>	1999/05/24 11:53:36 .. 1999/05/25 11:55:36
<code>parameterAsString("Time-Range:start_date")</code>	1999/05/24 11:53:36
<code>parameterAsString("TimeRange:end_date")</code>	1999/05/25 11:55:36
<code>parameterAsString("TimeRange:start")</code>	11:53:36
<code>parameterAsString("TimeRange:end")</code>	11:55:36
<code>parameterAsString("TimeSlice")</code>	00:00:00 (1/24) .. 23:59:59 (24/24)
<code>parameterAsString("TimeSlice:start")</code>	1
<code>parameterAsString("TimeSlice:end")</code>	24
<code>parameterAsString("TimeSlice:total")</code>	24

## Related Topics

[System Functions](#)

## *pi*

## Applies To

[Math Functions](#)

## Description

Returns the value of pi (3.14159265358979323846).

## Syntax

FLOAT `pi()`

## Examples

Expression	Result
<code>360 * (pi() / 180)</code>	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
<code>previousDate('1/9/99', 3)</code>	'1/5/99'

## Related Topics

[nextDate](#)

## *protect*

### Description

Prevents any expression from being warped, which reverses the order of expression processing. Because expressions are frequently warped, yielding undesirable results, they are often protected. See the *User Guide* for more information.

### Syntax

`protect (expression)`

### Examples

Expression	Result
<code>Cell.Peg3 = sum(Cell, protect(Cell.Peg2/ Cell.Peg1))</code>	Each value of the ratio Peg2/ Peg1 is evaluated indepen- dently and then summed
<code>protect sum(NBCell, HOATTHS) * 100.0 / HO_CAUSEALL</code>	<code>mean(ElementAnd- Time.Cell, ((sum(NBCell,HOATTHS ) *100)/HO_CAUSEALL))</code>
<code>protect ((a+b)*c)</code>	$SUM_Y^E ( Sum_Y^T ((a+b) * c ))$

## *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

<i>Expression</i>	<i>Result</i>
<code>radians (180)</code>	3.141593
<code>radians (360)</code>	6.283185
<code>radians (180) == pi ()</code>	1
<code>radians (360) == 2 * pi ()</code>	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

<i>Expression</i>	<i>Result</i>
<code>replaceRE ("TFL010043", "[A-Z]+")</code>	010043
<code>replaceRE ("TFL010043", "[A-Z]+", "CELL ")</code>	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

<b><i>Expression</i></b>	<b><i>Result</i></b>
<code>replaceRE("TFL010043", "[A-Z]+[0]")</code>	143
<code>replaceRE("TFL010043", "[A-Z]+", "CELL ")</code>	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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
<code>second ( '12.30.45pm' )</code>	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
<code>seconds(stringToTime("9Jan1999 1:10:15 AM", "%d%b%Y %I:%M:%S %p"))</code>	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
<code>sin(pi())</code>	0.000000
<code>sin(0)</code>	0.000000
<code>sin(pi()/2)</code>	1.000000
<code>sin(radians(90))</code>	1.000000
<code>sin(radians(270))</code>	-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:

$$\text{sqr}(N) == N ^ 2$$

### **Syntax**

FLOAT `sqr` (NUMBER *N*)

### **Examples**

<i>Expression</i>	<i>Result</i>
<code>sqr(0)</code>	0.000000
<code>sqr(1)</code>	1.000000
<code>sqr(2)</code>	4.000000
<code>sqr(-2)</code>	4.000000
<code>sqr(10)</code>	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:

$$\text{sqr}(\text{sqrt} (N)) == N$$

### **Syntax**

FLOAT `sqrt` (NUMBER *N*)

## Examples

<i>Expression</i>	<i>Result</i>
<code>sqrt(0)</code>	0.000000
<code>sqrt(1)</code>	1.000000
<code>sqrt(9)</code>	3.000000
<code>sqrt(-1)</code>	NULL
<code>sqr(sqrt(100))</code>	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:

$$\text{stddev} = \sqrt{\text{sum}(x^2)/n - \text{mean}(x)^2}$$

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
<code>stddev(package(Bts), DroppedCalls / (FLOAT) TotalCalls * 100)</code>	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

<i>Expression</i>	<i>Result</i>
<code>stringToDate("1/9/99") == '9Jan1999'</code>	1
<code>stringToDate("1999 Jan 9", "%Y %b %d")== '1/9/1999'</code>	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

<i>Expression</i>	<i>Result</i>
<code>stringToInt ("1234")</code>	1234
<code>stringToInt ("98659")</code>	98659



<i>Expression</i>	<i>Result</i>
<code>stringToInt("0XABCD")</code>	43981
<code>stringToInt("0x769A")</code>	30362
<code>stringToInt("0654")</code>	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

<i>Expression</i>	<i>Result</i>
<code>stringToTime("13:35:27")=='1:35:27pm'</code>	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**

<b><i>Expression</i></b>	<b><i>Result</i></b>
<code>subString("Hello World", 5)</code>	World
<code>subString("Hello World", 6, 5)</code>	World
<code>subString("Hello World", -1, 100)</code>	Hello World
<code>subString("Hello World", firstIndex("Hello World", "World"), length("World"))</code>	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
<code>subStringRE("TFL010043", "[A-Z]+")</code>	TFL
<code>subStringRE("012-01098", "[A-Z]+")</code>	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

<i>Expression</i>	<i>Result</i>
<code>sum(package(Bts), TotalCalls - DroppedCalls)</code>	23940.000000
<code>sum(package(Bts).DroppedCalls) / sum(package(Bts).TotalCalls) * 100</code>	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':

<i>Expression</i>	<i>Result</i>
'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
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

<i>Expression</i>	<i>Result</i>
<code>timeToString('12.30.45pm')</code>	12:30:45
<code>timeToString('12.30.45pm', "%l:%M %p")</code>	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

<i>Expression</i>	<i>Result</i>
<code>tokenize("10-20.50-30-40-50", "-", 2)</code>	30
<code>tokenize("10-20.50-30-40-50", "-.", 2)</code>	50
<code>tokenize("10-20.50-30-40-50", "-.", 10)</code>	
<code>tokenize("10 20 30 40 50-60 70.80", "-.", 7)</code>	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**

<b><i>Expression</i></b>	<b><i>Result</i></b>
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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
<code>trunc(1.3)</code>	1.000000
<code>trunc(1.5)</code>	1.000000
<code>trunc(1.7)</code>	1.000000
<code>trunc(-1.3)</code>	-1.000000
<code>trunc(-1.5)</code>	-1.000000
<code>trunc(-1.7)</code>	-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"
```

<b><i>Expression</i></b>	<b><i>Result</i></b>
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

<i>Expression</i>	<i>Result</i>
<code>validDate('2/29/99')</code>	0
<code>validDate('2/29/00')</code>	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

<i>Expression</i>	<i>Result</i>
<code>validTime('09Jan1999 1:10AM')</code>	1
<code>validTime('29Feb1999 25:77AM')</code>	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:

$$\text{variance} = \text{sum}(x^2)/n - \text{mean}(x)^2$$

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
<code>variance(package(Bts), DroppedCalls / (Float)TotalCalls * 100)</code>	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 collection 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**

<b><i>Expression</i></b>	<b><i>Result</i></b>
<code>vsum(1, 2, 3, 0)</code>	Not NULL – The zero forces the result to never be NULL and does not affect the calculated summation
<code>vsum(1, 2, 3)</code>	Not NULL
<code>vsum(0,0,0)</code>	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**

<b><i>Expression</i></b>	<b><i>Result</i></b>
<code>weekDay ('1/9/99')</code>	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

<i>Expression</i>	<i>Result</i>
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

<i>Expression</i>	<i>Result</i>
<code>year('1Jan1999')</code>	1999

## Related Topics

[month](#)

**EXPRESSIONS TECHNICAL REFERENCE**  
Prospect®

General Expressions

---

## 3 Advanced Expressions

This section covers the additional advanced expressions available within the 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(timestamp, 'YYYY-MM-DD HH24:MI:SS')`

### Related Topics

[Erlang Functions](#)  
[Kaufman-Roberts Functions](#)  
[Forecast Reports](#)  
[Bridge Functions](#)  
[Instance Name Mapping Functions](#)

### 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 use the field either directly in their report templates, or in UDCs.

### avgDelayAllCallsC

#### Signature

`FLOAT avgDelayAllCallsC(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 avrgDelayDelayedCallsC(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.

<i><b>Attribute</b></i>	<i><b>Description</b></i>
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.

## **criticalTrafficOfferedC**

### **Signature**

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

<i><b>Attribute</b></i>	<i><b>Description</b></i>
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)`

<i><b>Attribute</b></i>	<i><b>Description</b></i>
A	Offered traffic in Erlangs.

---

<i><b>Attribute</b></i>	<i><b>Description</b></i>
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**

Erlang-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)

<b>Attribute</b>	<b>Description</b>
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)

<b>Attribute</b>	<b>Description</b>
Acar	$(1 - PT) * A$
PT	Probability of delay greater than T1.



---

<i><b>Attribute</b></i>	<i><b>Description</b></i>
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

<b>Term</b>	<b>Description</b>
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)**

<i><b>Term</b></i>	<i><b>Description</b></i>
Pearson Correlation Coefficient	The Pearson algorithm computes a correlation coefficient returning a value in the range $-1 \leq c \leq 1$ that measures how closely the data-points in an XY distribution fit to a straight line. An $\text{abs}(\text{coefficient}) \geq 0.8$ 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)`

<i><b>Parameter</b></i>	<i><b>Description</b></i>
<code>instance_id</code>	Identifier (ID) of the element instance being evaluated.
<code>tstamp</code>	The interval start of a weekly report.
<code>n</code>	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)`

<i><b>Parameter</b></i>	<i><b>Description</b></i>
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)
```

<i><b>Parameter</b></i>	<i><b>Description</b></i>
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 := growthPerSecond * 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 forth).

### ***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.

**EXPRESSIONS TECHNICAL REFERENCE**  
Prospect®

Advanced Expressions

---

# Index

– Operator	
binary	22
unary	16

## Symbols

!= Operator	18
! Operator	16
% Operator	26
&& Operator	29
( ) Operator (Cast)	28
* Operator	24
+ Operator	21
/ Operator	25
< > Operator	18
< Operator	18
<= Operator	19
= = Operator	17
= Operator	17
> Operator	20
>= Operator	20
? : Operator	35
^ Operator	27
Operator	88

## A

abs	
absolute value	29
add	107
+ operator	21
AND Operator	29
Application Gateway	
Gateway Attribute	47
package functions	89
arcCos	30
arcSin	31
arcTan	31
Arithmetic Operators	32
arithmetic operators	
binary	33
attributes	
Gateway Attribute	47

audience	9
average	74
avrgDelayAllCallsC	123
avrgDelayDelayedCallsC	124

## B

base 10 log	71
Binary Operators	33
boolean	
Filters	44
bridgeList	139

## C

capacityB	124
capacityP	125
case	
toLower	113
toUpper	113
ceil	34
circuits	125
circuitsP	125
concat	
string functions	34
Conditional Operator ( ? : )	35
Constants	36
converting	
() operator (cast)	28
stringToFloat	103
stringToInt	104
cos	37
count	38
criticalTrafficCarriedC	125
criticalTrafficOfferedC	126

## D

date	
sysdate	108
Date Functions	39
dateToString	40
dayOfMonth	40
dayOfYear	41
decode()	41
degrees	
convert to radians	94

**EXPRESSIONS TECHNICAL REFERENCE**  
Prospect®

---

converting from radians . . . . .	42	WM_FCAST_DIMENSION . . . . .	134
delayProbabilityC . . . . .	126	WM_FCAST_FINALTHRESH . . .	136
distance . . . . .	42	WM_FCAST_GROWTH . . . . .	135
divide		WM_FCAST_PABH . . . . .	133
% operator . . . . .	26	WM_FCAST_SAMPLES . . . . .	137
/ operator . . . . .	25	WM_FCAST_THRESH . . . . .	135
documentation			
assumptions about prior knowledge . . .	9	<b>F</b>	
font usage . . . . .	10	Filters . . . . .	44
typographical conventions . . . . .	10	finding	
user . . . . .	11	max . . . . .	73
viewing HTML Help . . . . .	11	min . . . . .	76
viewing PDF . . . . .	12	firstDayOfMonth . . . . .	44
<b>E</b>		firstIndex . . . . .	45
environment variable		floating point	
getenv . . . . .	50	stringToFloat . . . . .	103
equality		floor . . . . .	45
!= operator . . . . .	18	font usage	
< operator . . . . .	18	documentation . . . . .	10
<= operator . . . . .	19	forecast functions . . . . .	132
== operator . . . . .	17	forecast reports . . . . .	132
> operator . . . . .	20	Function	
>= operator . . . . .	20	bridgeList . . . . .	139
Erlang B		instanceFDN . . . . .	140
capacityB . . . . .	124	instanceName . . . . .	140
circuits . . . . .	125	scenarioInstance . . . . .	141
gos . . . . .	127	Functions . . . . .	46
Erlang C		functions	
avgDelayAllCallsC . . . . .	123	Kaufman-Roberts . . . . .	130
avgDelayDelayedCallsC . . . . .	124	mathematical . . . . .	73
criticalTrafficCarriedC . . . . .	125		
criticalTrafficOfferedC . . . . .	126		
delayProbabilityC . . . . .	126		
notAcceptDelayProbabilityC . . . . .	128		
numCircuitsC . . . . .	128		
Erlang Functions . . . . .	123		
exp . . . . .	43		
explicit casting . . . . .	86		
exponent			
^ operator . . . . .	27		
Expression			
WM_FCAST_BDBH . . . . .	133		
WM_FCAST_CCOTHRESH . . . . .	135		
WM_FCAST_CORRELATION . . .	137		
WM_FCAST_CURRTHRESH . . .	136		
WM_FCAST_DAYS . . . . .	136		

---



<b>G</b>		<b>ip Functions</b> . . . . .	61
Gateway Attribute . . . . .	47	ipAddressClass . . . . .	53
Geometry Functions . . . . .	49	ipAddressToInt . . . . .	54
getDateFromTime . . . . .	50	ipApplyNetmask . . . . .	55
getenv . . . . .	50	ipApplyNetmaskMerge . . . . .	56
getTimeFromDate . . . . .	51	ipByteOfAddress . . . . .	57
gos . . . . .	127	ipBytesToAddress . . . . .	57
gosP . . . . .	128	ipCommonNetmask . . . . .	58
<b>H</b>		ipDefaultNetmaskBits . . . . .	60
hour . . . . .	52	ipIntToAddress . . . . .	62
hourGMT . . . . .	52	ipIsBroadcast . . . . .	62
HTML Help format . . . . .	11	ipIsLoopback . . . . .	63
<b>I</b>		ipIsNormal . . . . .	64
if . . . then operator . . . . .	35	ipIsPrivate . . . . .	64
implicit casting . . . . .	86	ipIsUnknown . . . . .	65
In . . . . .	70	ipIsValid . . . . .	66
index		ipNetmaskMerge . . . . .	67
lastIndex . . . . .	69	isLeapYear . . . . .	66
inequality		isNotNull . . . . .	68
!= operator . . . . .	18	isNull . . . . .	68
<= operator . . . . .	19	<b>K</b>	
<operator . . . . .	18	Kaufman-Roberts functions . . . . .	130
> operator . . . . .	20	<b>L</b>	
>= operator . . . . .	20	lastIndex . . . . .	69
inPolygon . . . . .	53	length . . . . .	42, 69
instance		literals . . . . .	36
firstIndex . . . . .	45	locality	
instanceFDN . . . . .	140	inPolygon . . . . .	53
instanceName . . . . .	140	log	
integer		base 10 . . . . .	71
stringToInt . . . . .	104	natural . . . . .	70
		Logical Operators . . . . .	71
		logical operators	
		binary . . . . .	33
		OR . . . . .	88

**EXPRESSIONS TECHNICAL REFERENCE**  
Prospect®

---

<b>M</b>	
Math Functions	73
max	73
mean	74
median	75
min	76
minute	77
minuteGMT	78
mode	79
month	80
monthName	80
multiply	
* operator	24
<b>N</b>	
natural log	70
negative	
- operator	16
nextDate	81
notAcceptDelayProbabilityC	128
null	
isNotNull	68
isNull	68
NullDate	81
NullFloat	82
NullInt	83
NullString	83
NullTime	84
nullValue	85
numCircuitsC	128
numToString	85
<b>O</b>	
Operators	32, 33
logical	71
unary	86, 115
OR Operator	88
<b>P</b>	
package	89
Package Functions	89
packages	
object count	38
parameterAsString	91
PDF format	12
pi	93
Poisson	
capacityP	125
circuitsP	125
gosP	128
power	
^ operator	27
prerequisites	
assumptions in documentation	9
previousDate	93
product	
* operator	24
product support	12
product training	12
protect	94
publications	
user	11
<b>R</b>	
radians	
convert from degrees	94
converting to degrees	42
remainder	
% operator	26
replaceAllRE	96
reverse	96
round	97
rounding	
ceil	34
floor	45
round	97
trunc	114
<b>S</b>	
scenarioInstance	141
second	98
seconds	98
sin	99
skills required documentation	
assumptions about prior knowledge	9
software	9
sorting	
max	73
minimum	76

---

**EXPRESSIONS TECHNICAL REFERENCE**  
Prospect®

sqr	100	arcTan	31
sqrt	100	cos	37
square	100	degrees	42
square root	100	radians	94
standard deviation	101	sin	99
statistical operators		tan	110
mean	74	trunc	114
median	75	type	
mode	79	toLower	113
stddev	101	toUpper	113
variance	117	typographical conventions	10
stddev	101	<b>U</b>	
string		UDC Editor	
length	69	Kaufman-Roberts functions	130
numToString	85	Unary Operators	86, 115
reverse	96	unescape	116
subString	106	user publications	11
String Functions	102	<b>V</b>	
subString	106	validDate	116
stringToDate	103	validTime	117
stringToFloat	103	variance	117
stringToInt	104	vsum()	119
stringToTime	105	<b>W</b>	
subString	106	weekDay	119
subtract		weekDayName	120
– operator (binary)	22	WM_FCAST_BDBH	133
sum	107	WM_FCAST_CCOTHRESH	135
+ operator	21	WM_FCAST_CORRELATION	137
support	12	WM_FCAST_CURRTHRESH	136
sysdate	108	WM_FCAST_DAYS	136
System Functions	109	WM_FCAST_DIMENSION	134
sysTime	109	WM_FCAST_FINALTHRESH	136
<b>T</b>		WM_FCAST_GROWTH	135
tan	110	WM_FCAST_PABH	133
time		WM_FCAST_SAMPLES	137
sysTime	109	WM_FCAST_THRESH	135
Time Functions	111	<b>Y</b>	
timeToString	111	year	120
tokenize	112	<b>Z</b>	
toLower	113	zero	
toUpper	113	! operator	16
training	12	AND operator	29
Trig Functions	114		
trigonometric operators			
arcCos	30		
arcSin	31		

Updated: 2007-07-02

zone	
inPolygon .....	53



#### Corporate Headquarters

13431 NE 20th Street  
Bellevue, WA 98005 USA  
Phone: +1 425 564 8000  
Fax: +1 425 564 8001

#### EMEA

5300 Cork Airport  
Business Park  
Kinsale Road  
Cork, Ireland  
Phone: + 353 21 730 6000  
Fax: + 353 21 730 6024

Spencer House  
23 Sheen Road  
Richmond  
Surrey, UK, TW9 1BN  
Phone: +44 (0)20 8332 7400  
Fax: +44 (0)20 8332 7403

#### Asia Pacific

901B, Tower B, Uptown 5  
5 Jalan SS21/39,  
Damansara Uptown  
47400 Petaling Jaya  
Selangor, Malaysia  
Phone: +60 3 7712 7000  
Fax: +60 3 7726 7207

[www.vallent.com](http://www.vallent.com)  
Email: [info@vallent.com](mailto:info@vallent.com)

Vallent, Metrica, Prospect and ServiceAssure are registered trademarks or trademarks of Vallent Corporation and/or Vallent Software Systems UK in the United States and/or other countries. All other trademarks, trade names, company names, or products mentioned herein are the property of their respective owners. Copyright © 2000-2006 Vallent Corporation. All rights reserved.



Service Management

Performance Management

Professional Services