

15. COMDALE LIBRARY FUNCTIONS OVERVIEW

15.1 Introduction

Comdale Functions allow you to access within your knowledge base, many of the features and functionality of the expert system development tool that you need to develop useful applications. With these functions you may:

- Control the on-screen appearance of your application
- Control the inference process
- Manipulate data and represent knowledge in ways which are similar to the methods used by humans in their thought process.

Comdale Functions may be used in condition and conclusion statements of rules as well as in procedures and facets of keyword triplets.

Comdale Functions are categorized as:

1. Inference control functions
2. String functions
3. Presentation functions
4. Mathematical functions
5. Statistical functions
6. Data/time functions
7. Dynamic variable functions
8. Knowledge base management functions
9. Miscellaneous functions

15.2 Inference Control Functions

Inference Control Functions control the inference process These functions allow the developer to control the inference process. Activities such as dictating how rules should be fired, as well as when to evoke forward and backward chaining can be achieved through the use of the inference control functions.

Example:

```
IF tank.level.high
THEN APPLYRULE (tank.level.critical)
THEN FIND ("tank.temperature.*")
```

“If the tank level is high then run all rules which are dependent on tank.level.critical, then investigate all situations which may decide on different status/values for the tank temperature”.

15.3 String Functions

String functions are used to manipulate string constants and variables. These functions allow the developer to manipulate string constants and variables. With string functions you may return the length of a string, concatenate two strings, etc.

Example:

```
IF STRLEN (tank.name.@string) > 6
THEN tank.name.too_long is TRUE cf = 100
```

“if the length of the string held by `tank.name.@string` is greater than 6 characters then assign the logical triplet `tank.name.too_long 100` degree of certainty”

15.4 Presentation Functions

Presentation Functions manipulate the user interface. These functions allow the developer flexibility in manipulating the user interface. With presentation functions you may display hypertext documents, interactive graphics as well as generate reports, etc.

Example:

```
PAINT (“tankgra.bmp”)
TEXT (“tank level is now: !$tank.level.@f$!”)
```

Displays a bitmap (bmp) graphic image and notify the user of the current value of `tank.level.@float` .

15.5 Mathematical Functions

Mathematical functions are used in numerical expressions. These functions allow the developer to perform mathematical computations. With mathematical functions, you can perform algebraic, trigometric, transcendental and other calculations in mathematical expressions.

Example:

```
IF tank.level.@float >= SQRT (tank.height.@float * 20)
THEN tank.temp.@integer = value.temperature.@i ^ 6

IF SIN (tank.level.@float + ABS (2.5 – tank.height.@f)
<= COS (sensor.reading.@f) + .34
THEN .....
ELSE .....
```

15.6 Statistical Functions

Statistical functions are used to do statistical computations These functions allow the developer to perform statistical computations. With statistical functions you can calculate averages and standard deviations.

Example:

```
IF AVG (tank.level.@f tank2.level.@f) > 20
AND .....
THEN .....
```

15.7 Date/Time Functions

Date/Time functions are used to manipulate date and time variables With Date/Time functions the developer can manipulate date and time variables.

Example:

```
IF n.n.@i = YEARDAY (xmas.day.@date) - YEARDAY ($current_date)
THEN TEXT (“!$n.n.@i$! days till Christmas”)
```

15.8 Dynamic Variable Functions

Dynamic functions manipulate subsets of classes With dynamic variable functions, the developer can manipulate dynamic lists of objects. You can perform functions such as extracting a list of objects in a class which meet a specific condition, then perform various operations on this list.

Examples:

```
IF ?member = ANY ({tank}.level.@f = 24)
THEN ALL (?member.valve.shut is TRUE)
```

“if any member of the class tank has a level equal to 24 then shut all those tank valves”

15.9 Knowledge Base Management Functions

Knowledge Base management functions are used to manipulate active and inactive knowledge bases during inference With Knowledge Base Management functions the developer can control the use of various knowledge bases during the inference process. You can perform functions such as loading or unloading a new knowledge base, making a specific knowledge base that is already in memory current, etc.

Example:

```
IF tank.level.critically_high
THEN tank1.valve.shut is TRUE
THEN LOAD (“emergency.knw”)
```

“if tank1 level is critically high then shut tank1 valve and load a new knowledge base which has an emergency strategy for operation.”

15.10 Miscellaneous Functions

Miscellaneous Functions Miscellaneous functions are assorted functions which perform diverse tasks such as running an external executable file, saving a consultation session, etc.

Example:

```
IF tank.level.high  
THEN ACTIVATE (“model.exe”)
```

“if the tank level is high then run an external model”.

15.11 Comdale Functions

15.11.1 Overview

This section is a reference of all the functions available to the developer during the creation of a COMDALE/X application. In this manual the functions will appear in alphabetical order according to their function name. For convenience, the functions are listed below sorted according to function type.

The sections in this part of the chapter are organized as follows:

- Functions sorted by Type;
- Functions Layout Syntax;
- Nomenclature;
- Functions in Alphabetic Order.

15.11.2 Function page layout syntax

15.11.2.1 Function Name

Shows the name of the function at the top of the page.

Function Name

TAN

15.11.2.2 In-depth Information

Shows information about the function:

Indepth Information

Format
Description
Arguments
Return Value
Comments
Example

Format:

the layout of the function, and how it is to be used.

Description

a description of the functionality of the function

Arguments

in depth explanation of the functions arguments, if required.

Return Value

the value(s) the function returns, if any.

Comments

additional information that may be of used, if required.

Example

at least one example of the function used in a rule format

15.11.2.3 Compatibility Symbols

Displays the compatibility of the function with Comdale's software. The icons represent:

C/X function supported in COMDALE/X (QNX and MS-WIN versions)

C/C function supported by the COMDALE/C Expert System extension to Process Vision

PV function supported by the Process Vision cc_batch, and al_admin modules

15.11.3 Nomenclature

The following notation is used in this chapter to describe the argument type and the return type.

15.11.3.1.1 \$constant

\$constant is a pre-defined constant used within the system. Allowable constants are discussed with each function as required.

15.11.3.1.2 condition

A conditional expression is one which utilizes a combination of Boolean logic, mathematical or lexical comparisons. The condition consists of a SINGLE comparison which is reduced to TRUE or FALSE.

The condition MUST be placed within double quotes.

Allowable options are:

ACCEPTABLE CONDITION ENTRIES FOR TEMPORAL REASONING FUNCTIONS

Comparison	&	AND
		OR
	= =	EQUAL
	!=	NOT EQUAL
	>	GREATER THAN
	<	LESS THAN
	>=	GREATER THAN OR EQUAL
	<=	LESS THAN OR EQUAL
Variables	Any NUMERIC keyword triplet (@float or @integer type triplet)	

15.11.3.1.3 date

Date is used to represent a date keyword-triplet. e.g. year (date) means that the function 'year' accepts a date keyword-triplet in the format [object.attribute.@date](#) (or. @dt)

15.11.3.1.4 numeric

Numerical is used to represent a numerical constant, or a numerical keyword-triplet. e.g. cos (numeric) means that the function 'cos' can accept a numerical constant like 100 or a numerical keyword-triplet such as [object.attribute.@float](#) (or .@f) or [object.attribute.@integer](#) (or @i.). In addition a numeric may also be a numerical expression or a function that returns a numerical value. A numerical constant may be real or integer.

15.11.3.1.5 pattern

A pattern is a string containing the wild card character ‘*’.

For example: “tank.*.*” is a pattern that represents all keyword-triplets with the object name “tank”.

A pattern must be enclosed by double quotes.

15.11.3.1.6 string

A string is used to represent a string constant, or a string keyword-triplet. e.g. STRLEN (string) means that the function ‘STRLEN’ can accept a string constant like “string” or a string keyword-triplet such as [object.attribute.@string](#) (or [.@s](#)). In addition, a string can also be a string function that returns a string value. When a string constant is used it must be enclosed within double quotes.

15.11.3.1.7 time

Time is used to represent time keyword-triplets. e.g., HOUR (time) means that the function ‘HOUR’ accepts a time keyword-triplet such as [object.attribute.@time](#) (or [.@t](#)).

15.11.3.1.8 triplet

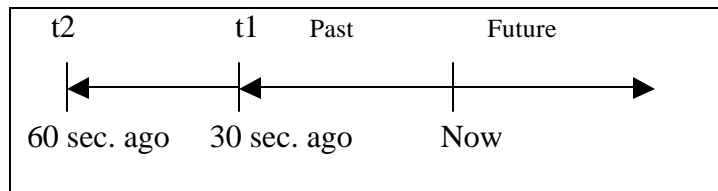
Triplet is used to represent a keyword-triplet of any type. e.g. [tank.level.@f](#), [tank.color.@s](#), [tank.delivery.@date](#) or [tank.level.high](#).

15.11.3.1.9 ...(ellipses)

ellipses (...) are used to indicate that a list of arguments may be included in the function. The list of arguments are of the same type as in the function argument list. If there is a limit to the number of arguments which a function can parse, it will be indicated in the description of the function.

15.11.3.1.10 t1, t2

t1, t2 specifies the time range over which temporal reasoning functions are evaluated. The following figure describes the convention used for all past history functions.



Example:

The following function script is used to describe the rate of change of a triplet variable over a period from 30 to 60 seconds. The figure below shows how the time period is specified for this function call.

```
RATEOFCHANGE (“a_line.temperature.@float”, 30, 60)
```

As a rule t1 must be less than t2 and both should be specified in seconds back from the present time.

15.11.4 Available functions – sorted by type

15.11.4.1 CLASS Mathematical Functions

CLASSMAX	returns the max. value in a given class
CLASSMIN	returns the min. value in a given class
CLASSMAXOBJECT	returns the object in the class with the maximum value
CLASSMINOBJECT	returns the object in the class with the minimum value
CLASSTOTAL	returns the sum of the class

15.11.4.2 CLASS Statistical functions

CLASSAVG	returns the average of the class
CLASSTDDEV	returns the standard deviation of the class

15.11.4.3 CLASS Special Functions

ANY	applies or checks a condition against a class
ALL	applies or checks a condition against a class

15.11.4.4 CLASS Triplet Information Functions

CLASSNUMOBJECT	returns the number of objects in a class
CLASSTOLIST	returns a list of objects in a class

15.11.4.5 Date/Time Functions

DAY	returns the current date of the month
ELAPSEDTIME	returns the time difference between two times
HOUR	returns the current hour
MINUTE	returns the current minute
MONTH	returns the current month
SECOND	returns the current second
WEEKDAY	returns the current day of the week
YEAR	returns the current year
YEARDAY	returns the total days elapsed this year

15.11.4.6 Dynamic Object/List Functions

LISTADDOBJECT	adds an object to a dynamic list
LISTALLOBJECT	returns all objects in dynamic list to a string
LISTCLEAR	empties a dynamic list
LISTCOMMONOBJECT	returns common objects in 2 lists
LISTFIRSTOBJECT	returns first object in dynamic list
LISTHASOBJECT	tests whether list has a specific object
LISTLASTOBJECT	returns last object in dynamic list
LISTMERGE	merges two lists together
LISTNTHOBJECT	lists the nth object in the list
LISTNUMOBJECT	returns the number of objects in a dynamic list
LISTREMOVEOBJECT	removes an object from a dynamic list
LISTREVERSE	reverses the objects in a dynamic list
LISTSORTNAME	sorts list into alphabetical order by object name
LISTUNCOMMONOBJECT	returns uncommon objects from 2 lists

15.11.4.7 Import/Export functions

ASK	query user for information
ASKCC	query the database directly for a keyword triplet
ASKUSER	query user for the value of a keyword triplet
ASNDEFAULT	assigns the default value to a pattern of triplets
ASNUNKNOWN	assigns the value of unknown to a pattern of triplets
ASNCERTAINTY	assigns a certainty to a pattern of triplets
CONNECT	connects to an external data source via DDE
DISCONNECT	disconnects a DDE link made with CONNECT
EXPORT	stores data to a file (user specified format)
GETPOINTDATA	get a triplet from the point database
IMPORT	retrieves data from a file (specific format)
PUTPOINTDATA	put a triplet into the point database
READ	reads data from an external source via DDE link
WRITE	writes data to an external source via DDE link

15.11.4.8 Inference Control Functions

APPLYRULE	fire a pattern of rules
ASNTHRESHOLD	assign the threshold value for the system
ASNUSERLEVEL	assign the user level for the current user
ACTIVATE	run an executable file
BACKWARDCHAIN_ENABLED	test whether backward chaining is enabled
CYCLE	runs the system in a cyclic manner
DATASOURCE_ENABLED	test whether DataSource is enabled
DISABLEBACKWARDCHAIN	enables or disables backward chaining
DISABLEDATASOURCE	enables or disables DataSource
DISABLEFORWARDCHAIN	enables or disables forward chaining
DISABLEIFCHANGE	enables or disables IfChange
FIND	attempts to instantiate given keyword-triplets
FORGET	un-instantiates keyword-triplets of a given pattern
FREERULE	frees previously executes rules
FORWARDCHAIN_ENABLED	enables or disables forward chaining
IFCHANGE_ENABLED	enables or disables IfChange
GOTO	fires the rule(s) of the given name pattern
HALT	terminates the session and shows results
IGNORE	disallows firing of rules of given pattern
MACRO	fires the rule(s) of the given name pattern
DONOTHING	this function takes no action
PAUSE	pauses system until user acknowledges a dialog box
REDO	restarts the system
RUN_PROCEDURE	runs the procedure(s) specified by the pattern
SLEEP	suspends the inference engine for a period of time
WAIT	disallows a pattern of rules from being fired for a give time

15.11.4.9 Knowledge Base Management functions

LOAD	loads a knowledge base file
RUN	starts a knowledge base file
RUN_CONTINUE	runs another knowledge base then returns
UNLOAD	unloads a knowledge base file

15.11.4.10 Inheritance functions

INHERITPARENTVALUE allows a triplet to inherit a value from its parent class
INHERITSIBLINGVALUE allows a triplet to inherit a value from a sibling in the same class

15.11.4.11 Mathematical Functions

ABS returns the absolute value of its argument
ACOS returns the arc cosine of its argument
ASIN returns the arc sine of its argument
ATAN returns the arc tangent of its argument
CEIL returns the argument rounded up to the next whole number
COS returns the cosine of its argument
COSH returns the hyperbolic cosine of its argument
EXP returns the value of e to the power of the argument
FLOOR returns the argument rounded down to the next whole number
LOG returns the natural logarithm of its argument
LOG10 returns the logarithm (base 10) of its argument
MAX returns the maximum value of a list of triplets
MIN returns the minimum value of a list of triplets
MODULUS returns the remainder of a division statement
NOT logical not
POW raises one number to the power of another
RAND returns a random number
SIN returns the sine of its argument
SINH returns the hyperbolic arc sine of its argument
SQRT returns the square root of its argument
TAN returns the tangent of its argument
TANH returns the hyperbolic tangent of its argument

15.11.4.12 Presentation functions

ANIMATE displays an animation sequence
DISPLAY displays a hypertext document
FORM displays a user created form
FORMAT returns a formatted string
PAINT displays a picture
TEXT displays a text message

15.11.4.13 System Information Functions

GETPASSWORD returns the password of the knowledge base
GETREPORTPROCEDURE returns the name of the report procedure
GETSTARTPROCEDURE returns the name of the start procedure
GETSTARTRULE returns the name of the start rule
GETSTARTRULESET returns the name of the start rule set
SETPASSWORD sets the password for the knowledge base
SETREPORTPROCEDURE sets the name of the report procedure
SETSTARTPROCEDURE sets the name of the start procedure
SETSTARTRULESET sets the name of the start rule set
SAVESESSION saves the current knowledge base
THRESHOLD sets the threshold for the system
USERLEVEL sets the user level for the current session

15.11.4.14 String Functions

STRCOMPARE	compares 2 strings (case sensitive)
STRING_TO_NUMBER	converts a string to a number
STRCONCAT	concatenates 2 strings together
STRCOPY	copies a string into a keyword-triplet
STRICOMPARE	compares 2 strings (case-insensitive)
STRLOWERCASE	converts a string to lower case
STRNOCOMPARE	compares the first n characters of 2 strings
STRNCOPY	copies the first n characters of a string to a keyword triplet
STRNICOMPARE	compares the first n characters of 2 strings (case insensitive)
STRLEN	returns the length of a string
STRUPPERCASE	converts a string to upper case

15.11.4.15 Statistical functions

AVG	returns the average of a list of triplets
STDDEV	returns the standard deviation of a list of triplets
VARIANCE	returns the variance of a list of triplets

15.11.4.16 Temporal Reasoning Functions

ALLTIMES	returns the number of times a condition has occurred over the last t seconds
ALLTIMESAFTER	tests if a condition occurred at all times between t to present
ALLTIMESBEFORE	tests if a condition occurred at all times before time t2
ALLTIMESDURING	tests if a condition occurred at all times during t1 to t2
ANYTIMEAFTER	tests if a condition occurred between time t and the present
ANYTIMEBEFORE	tests if a condition occurred at any time before t1
ANYTIMEDURING	tests if a condition occurred at anytime during t1 tot2
ATTIME	tests if a condition was met at time t or t seconds age
CERTAINTYAT	returns the certainty of a triplet t1 seconds ago
FIRSTTIME	returns the time that the condition first occurred
JUSTAFTER	tests if a condition occurred only at times between t to present
JUSTBEFORE	tests if a condition occurred only at times before time t2
JUSTDURING	tests if a condition occurred only at times during the period t1 to t2
LASTTIME	returns the time in seconds that the condition last occurred
RATEOFCHANGE	calculates the rate of change of a variable between t1 and t2 seconds ago
RELATIVECHANGE	calculates the relative change of a variable between t1 and t2 seconds ago
TIMEAVERAGE	calculates the average value of a variable between t1 and t2 seconds ago
TIMEHIGHEST	calculates the highest value of a variable between t1 and t2 seconds ago
TIMELOWEST	calculates the lowest value of a variable between t1 and t2 seconds ago
VALUEAT	returns the value of the triplet t1 seconds ago
WHEN	returns the absolute time when this triplet last changed its value

15.11.4.17 Triplet Information Functions

AGE	returns the age of a triplet in seconds
ATTR	returns the string of the attribute part of the keyword triplet name
KNOWN	returns whether the triplet is known
NOTKNOWN	returns whether the triplet is unknown
OBJ	returns the string of the object part of the keyword triplet name
VALUE	returns the string of the value part of the keyword triplet name
UNKNOWN	returns whether the triplet is unknown

15.11.5 Function compatibility chart

For completeness, COMDALE/X contains all functions that can be used with any COMDALE product such as:

- COMDALEX** - offline expert system and expert system development tool
- COMDALE/C** - online expert system add-on to Process Vision
- Process Vision** - complete SCADA system

Below is a table listing all functions and their applicability for each product.

Function	Compatability	Function	Compatability
ABS	C/X C/C PV	COS	C/X C/C PV
ACOS	C/X C/C PV	COSH	C/X C/C PV
ACTIVATE	C/X C/C PV	CYCLE	C/X C/C
AGE	C/C PV	DATASOURCE_ENABLED	C/X C/C
ALL	C/X C/C	DAY	C/X C/C
ALLTIMES	C/C PV	DISABLEBACKWARDCHAIN	C/X C/C
ALLTIMESAFTER	C/C PV	DISABLEDATASOURCE	C/X C/C
ALLTIMESBEFORE	C/C PV	DISABLEFORWARDCHAIN	C/X C/C
ALLTIMESDURING	C/C PV	DISABLEIFCHANGE	C/X C/C
ANIMATE	C/X	DISCONNCT	C/X
ANY	C/X C/C	DISPLAY	C/X
ANYTIMEAFTER	C/C PV	DONOTHING	C/X C/C
ANYTIMEBEFORE	C/C PV	ELAPSEDTIME	C/X C/C
ANYTIMEDURING	C/C PV	EXP	C/X C/C PV
APPLYRULE	C/X C/C	EXPORT	C/X C/C
ASIN	C/X C/C PV	FIND	C/X C/C
ASK	C/X	FIRSTTIME	C/C PV
ASKCC	C/C	FLOOR	C/X C/C PV
ASKUSER	C/X	FORGET	C/X C/C
ASNCERTAINTY	C/X C/C	FORM	C/X
ASNDEFAULT	C/X C/C	FORMAT	C/X C/C PV
ASNTHRESHOLD	C/X C/C	FORWARDCHAIN_ENABLED	C/X C/C
ASNUNKNOWN	C/X C/C	FREERULE	C/X C/C PV
ASNUSERLEVEL	C/X C/C	GETPASSWORD	C/X C/C
ATAN	C/X C/C PV	GETPOINTDATA	C/C
ATTIME	C/C PV	GETREPORTPROCEDURE	C/X C/C
ATTR	C/X C/C PV	GETSTARTPROCEDURE	C/X C/C
AVG	C/X C/C PV	GETSTARTRULE	C/X C/C
BACKWARDCHAIN_ENABLED	C/X C/C	GETSTARTRULESET	C/X C/C
CEIL	C/X C/C PV	GETSTARTRULE	C/X C/C
CERTAINTYAT	C/CPV	GETSTARTRULESET	C/X C/C
CLASSAVG	C/X C/C	GOTO	C/X C/C PV
CLASSMAX	C/X C/C	HALT	C/X C/C
CLASSMAXOBJECT	C/X C/C	HOURL	C/X C/C
CLASSMIN	C/X C/C	IFCHANGE_ENABLED	C/X C/C
CLASSMINOBJECT	C/X C/C	IGNORE	C/X C/C
CLASSNUMOBJECT	C/X C/C	IMPORT	C/X C/C
CLASSSTDDEV	C/X C/C	INHERITPARENTVALUE	C/X C/C
CLASSTOLIST	C/X C/C	INHERITSIBLINGVALUE	C/X C/C
CLASSTOTAL	C/X C/C	JUSTAFTER	C/C PV
CONNECT	C/X	JUSTBEFORE	C/C PV

Function	Compatability	Function	Compatability
JUSTDURING	C/C PV	SAVESESSION	C/X
KNOWN	C/X C/C	SECOND	C/X C/C
LASTTIME	C/C PV	SETPASSWORD	C/X C/C
LISTADDOBJECT	C/X C/C	SETREPORTPROCEDURE	C/X C/C
LISTALLOBJECT	C/X C/C	SETSTARTPROCEDURE	C/X C/C
LISTCLEAR	C/X C/C	SETSTARTRULESET	C/X C/C
LISTCOMMONOBJECT	C/X C/C	SIN	C/X C/C PV
LISTFIRSTOBJECT	C/X C/C	SINH	C /X C/C PV
LISTHASOBJECT	C/X C/C	SLEEP	C/X C/C
LISTLASTOBJECT	C/X C/C	SQRT	C/X C/C PV
LISTMERGE	C/X C/C	STDDEV	C/X C/C PV
LISTNTHOBJECT	C/X C/C	STRCOMPARE	C/X C/C
LISTNUMOBJECT	C/X C/C	STRCONCAT	C/X C/C
LISTREMOVEOBJECT	C/X C/C	STRCOPY	C/X C/C
LISTREVERSE	C/X C/C	STRICOMPARE	C/X C/C
LISTSORTNAME	C/X C/C	STRING_TO_NUMBER	C/X C/C
LISTUNCOMMONOBJECT	C/X C/C	STRLEN	C/X C/C
LOAD	C/X C/C	STRLOWERCASE	C/X C/C
LOG	C/X C/C PV	STRNCOMPARE	C/X C/C
LOG10	C/X C/C PV	STRNCOPY	C/X C/C
MACRO	C/X C/C PV	STRNICOMPARE	C/X C/C
MAX	C/X C/C PV	STRUPPERCASE	C/X C/C
MIN	C/X C/C PV	TAN	C/X C/C PV
MINUTE	C/X C/C	TANH	C/X C/C PV
MODULUS	C/X C/C PV	TEXT	C/X C/C PV
MONTH	C/X C/C	THRESHOLD	C/X C/C
NOT	C/X C/C PV	TIMEAVERAGE	C/C PV
NOTKNOWN	C/X C/C	TIMEHIGHEST	C/C PV
OBJ	C/X C/C PV	UNKNOWN	C/X C/C
PAINT	C/X	UNLOAD	C/X
PAUSE	C/X	USERLEVEL	C/X C/C
POW	C/X C/C PV	VALUE	C/X C/C PV
PUTPOINTDATA	C/C	VALUEAT	C/C PV
RAND	C/X C/C PV	VARIANCE	C/X C/C PV
RATEOFCHANGE	C/C PV	WAIT	C/X C/C
READ	C/X	WEEKDAY	C/X C/C
REDO	C/X C/C	WHEN	C/C PV
RELATIVECHANGE	C/C PV	WRITE	C/X
RUN	C/X	YEAR	C/X C/C
RUN_CONTINUE	C/X	YEARDAY	C/X C/C
RUN_PROCEDURE	C/X C/C PV		

15.12 Process Vision Functions

All Process Vision, COMDALE/X and COMDALE/C functions are listed on the following pages.

Format

ABS (numeric)

Description

This function returns the absolute value of the numerical argument.

Return Value

A numerical constant.

Example**Rule in C/X, C/C, or cc_batch**

If absolute.value.needed is TRUE
THEN `return.value.@f` = ABS (`variable.of_interest.@f`)

If ABS (`tank30.level.@f`) > 50
THEN TEXT (“Tank 30 level above HI mark”, “warning1”)

al_admin condition procedure

ABS (`tank.level.@f`) > 23

Format

ACOS (numeric)

Description

This function returns the arc-cosine of the numerical argument in the range 0 to pi radians.

Return Value

This return value is a real numerical constant in radians.

Comments

The argument must have units of radians.

If the argument is less than -1 or greater than 1 the value 0 will be returned.

Example

IF new.calculation.needed is TRUE

THEN `trig.argument.@f = ACOS (acos.of_trig_argument.@f)`

Format

ACTIVATE (string)

Description

This function executes any valid application.

Arguments

String is an expression containing the name of an executable and/or its path

Return Value

TRUE upon successful completion of the ACTIVATE string, otherwise FALSE.

Comments

This function performs successfully, if the application can run under the current environment

QNX tasks

If the executable is a QNX Windows native task, the windows associated with this executable will open. If the started executable is a QNX task then it will run concurrently with other tasks but will not be displayed in a QNX SHELL window. To run a custom script file in a shell, ACTIVATE the /windows/bin/wterm program with the appropriate options.

Remember to chmod a+x the script file to give each an executable bit for use with ACTIVATE

IMPORTANT: QNX environments should always use the syntax:

ACTIVATE ("command &")

to ensure the task starts up in the background

DOS tasks

If the executable is a DOS task and you use the activate command from Comdale/X for MS-Windows, then a DOS shell will automatically open to run the program. To run a DOS Application in a window, use Microsoft's PIF Editor to allow a DOS executable in Windows. If the executable is a Windows application, starting it will open its associated window.

Example**QNX Windows:**

```
If windows_clock.visible.needed is TRUE  
THEN ACTIVATE ("/windows/apps/wclock &")
```

QNX Shell:

```
If pulse.action.required is TRUE  
THEN ACTIVATE ("coil_setting.fifteen.@f15 &")
```

MS DOS:

```
If archiving.of_directory.needed is TRUE  
THEN ACTIVATE ("pkzip -a -eX archive.zip*.knw")
```

MS-Windows:

```
If start.excel.now is TRUE  
THEN ACTIVATE ("c:\excel\excel")
```


Format

ALL ({class}.attribute.value<legal-operand><constant | triplet> ...)

Arguments

{class}.attribute.value - any legal class with attribute and value
legal operand - any of the following:
==, =, is, is_not

Description

This function is best used to check for a like condition across a class or a few classes. To achieve its goal the ALL function will walk through the named {class} applying the <legal-operand>.

Return Value

TRUE if ALL of the conditions are met or if ALL of the assignments are made. The return values of the ALL function are also used to build dynamic lists.

Example

```
IF ALL ({furnace}.status.on is TRUE and {furnace}.temperature > 850)
THEN TEXT ("All furnaces are on and above minimum temperatures" , "info")
```

In the above example the ALL function will check each object in the furnace class to make sure it is on and also to see if it's temperature is above 850.

```
IF all.names.needed is TRUE
THEN ?all_name = ALL ({employees}.name.@string)
```

This example builds a list which will contain all the objects in the class employees.

Format

ALLTIMES (t, condition)

only for use in Comdale/C and/or ProcessVision

Arguments

t is in seconds

Description

Returns the number of times a condition was met over the last t seconds.

Return Value

The number of times this condition was satisfied.

Example

```
IF ALLTIMES (500, "tank.level@float >= 26") >100
```

```
THEN TEXT ("The process has not been stable over the last 500 seconds", "Process Status")
```

In this example the condition `tank.level.@float >= 26` is checked.

Format

ALLTIMEAFTER (t, condition) only for use in Comdale/C and/or ProcessVision

Arguments

t is in seconds

condition is a conditional expression – see 1.3.3 Nomenclature

Description

Tests if a condition was met at all times between t seconds ago and the present.

Return Value

TRUE (100), FALSE (0)

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have All the points in the conditional configured in that database.

Example

```
IF ALLTIMESAFTER (100, “tank.level@float < 87”)
```

```
THEN TEXT (“The tank level has been below 87 for the last 100 seconds”, “Info”)
```

Format

ALLTIMESBEFORE (t1, t2, condition) only for use in Comdale/C and/or ProcessVision

Description

Tests if a condition occurred at all times before time t1.

The check is done as far in the past as time t2.

If the value of t2 is set as zero then the check is done as far back as history is kept for triplet.

Arguments

t1 is in seconds*

t2 is in seconds*

condition is a conditional expression – see Nomenclature

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Return Value

TRUE (100), FALSE (0)

Example

```
IF ALLTIMESBEFORE (10, 20, "pump_5.amperage_draw.@f >
```

```
Plant_average.amperage.@f")
```

```
THEN TEXT ("Pump 5'S amperage was aboce average between 10 and 20
```

```
Seconds ago", "Warning")
```

Format

ALLTIMESDURING (t1, t2, condition) only for use in Comdale/C and/or ProcessVision

Description

Tests if a condition occurred at all times during the period time t1 to t2.

Arguments

t1 is in seconds*

t2 is in seconds*

condition is a conditional expression – see Nomenclature

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Return Value

TRUE (100), FALSE (0)

Example

```
IF ALLTIMESDURING (20, 50, "furnace.temperature.@f > furnace.temp_setpoint.@f")  
THEN TEXT ("The furnace temperature was above its setpoint temperature for all time from  
20 seconds ago to 50 seconds ago.", "Furnace Status")
```

Format

ANIMATE (string_1, string_2, numeric_1, numeric_2, T | F, T | F)

Description

This function animates a sequence of bitmap files.

Under MS-DOS and Windows, these must be .bmp

Under QNX Windows they must be .scan RLE (Run Length Encoded) files.

Return Value

TRUE if the ANIMATE command completed successfully, otherwise FALSE.

Arguments

string_1 The first argument is a string containing the bitmap files' name without any extension.*

string_2 The second argument is a string containing the picture's title.

numeric_1 The third argument is the number of bitmap files(frames) to animate.

numeric_2 The fourth argument is the delay time (in seconds) between frames.

T | F The fifth argument is the cycle flag. Set this flag to TRUE if the animation is to repeat over and over. Set this flag to FALSE to animate only once.

T | F The sixth argument is the user-acknowledge flag. Set this flag to TRUE if you want a user to acknowledge (by clicking on "REMOVE") before leave this function. Set this flag to FALSE if a user acknowledgement is not required.

*Note: Bitmap files must have the same filenames, with extension ".001", ".002", ..., ".999" a maximum of 999 bitmaps is allowed.

Example

```
IF TRUE  
THEN ANIMATE ("ufo_bmp", "Title", 13, 0.5, T, F)
```

This example displays the bitmaps "ufo_bmp.001", "ufo_bmp.002", ..., "ufo_bmp.013". The animation will pause for 0.5 second between each bitmap and since the cycle flag is set to TRUE, the animation will repeat when "ufo_bmp.013" is reached. No user acknowledgement is required to continue on past this function and the animation will continue to run in the background as other rules of the knowledge base are executed.

Format

ANY ({class}.attribute.value <legal-operand> <constant | triplet> ...)

Arguments

{class}.attribute.value - any legal class with attribute and value

legal operand - any of the following;
==, =, is, is_not

Description

This function is best used to check for a like condition across a class or a few classes. To achieve its goal the ANY function will walk through the named {class} applying the <legal-operand>.

Return Value

TRUE if ANY of the conditions are met or if ANY of the assignments are made. The return values of the ANY function are also used to build dynamic lists.

Example

```
IF ANY ( {tank}.level.@float > 15)
THEN TEXT ("one of the tanks is above range", "warning1")
```

In the above example a class named {tank} may have many configured objects (like tank1, tank2, tank3, etc.). The premise of this example will look for `tank1.level.@float > 15`, then `tank2.level.@float > 15`, etc. If ANY of the triplets meet this condition then the ANY function will return TRUE and the TEXT in the conclusion will be executed.

```
IF ?above_level_tanks = ANY ( {tank}.level.@float > level.to_check_for.@float)
THEN problem.list.@string is LISTALLOBJECT (?above_level_tanks," ", "none")
THEN TEXT ("Problem tanks : !$ problem.list.@string $!", "warning1")
```

In the above example the ANY function is again used to look at the class {tank} but this time each object is checked against a variable (`level.to_check_for.@float`) instead of a constant. The return value of ANY in this example is used to build a dynamic list of object names which meet the condition in the ANY function.

If our initial conditions are:

Variable	Value
<code>Level.to_check_for.@float</code>	90
<code>Tank1.level.@float</code>	12
<code>Tank2.level.@float</code>	99
<code>Tank3.level.@float</code>	93
<code>Tank4.level.@float</code>	45
<code>Tank5.level.@float</code>	97

Then the above example would output a message to the operator: "tank2,tank3,tank5"

Format

ANYTIMEAFTER (t, condition) only for use in Comdale/C and/or ProcessVision

Description

Tests if a specified condition occurred between t seconds ago and the present time.

Return Value

TRUE (100), or FALSE (0)

Arguments

t is in seconds

condition is a conditional expression – see Nomenclature

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Return Value

TRUE (100), or FALSE (0)

Example

```
IF ANYTIMEAFTER (400, "controller.computer.on == 0")  
THEN TEXT ("The controller computer was off during the last 400 seconds", "Warning")
```

Format

ANYTIMEBEFORE (t1, t2, condition) only for use in Comdale/C and/or ProcessVision

Description

Tests if a condition occurred at anytime before t1. The check is done as far in the past as t2. If t2 is 0 then the check will be done as far back as history is kept for the triplets.

Arguments

t1 is in seconds

t2 is in seconds

condition is a conditional expression – see Nomenclature

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Return Value

TRUE (100), or FALSE (0)

Example

IF ANYTIMEBEFORE (20, 200,

“[dilithium_containment.field_in_testlas.@f < 300](#)”)

THEN TEXT (“Warp engine core meltdown imminent !”, “Beam me up Scotty”)

Format

ANYTIMEDURING (t1, t2, condition) only for use in Comdale/C and/or ProcessVision

Description

Tests if a condition occurred at anytime during t1 and t2.

Arguments

t1 is in seconds

t2 is in seconds

condition is a conditional expression – see Nomenclature

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Return Value

TRUE (100), or FALSE (0)

Example

```
IF ANYTIMEDURING (2, 400, "radiation.level_in_rads.@f > 20")  
THEN RUN_PROCEDURE ("Lower_Active_Output")
```

Format

APPLYRULE (pattern)

Description

This function uses a keyword-triplet pattern to indicate the next set of rules to be fired. The pattern must matching the pattern of the triplets in the rule(s) premise to be fired.

Return Value

None.

Example

```
IF tank.information.needed is TRUE
AND rules_to_be.fired.@s IS "tank.*.@f"
THEN APPLYRULE (rules_to_be.fired.@s)
```

```
IF tank.information.needed is TRUE
AND rules_to_be.fired.@s IS _NOT "tank.*.@f"
THEN APPLYRULE ("tank.*.@f")
THEN rules_to_be.fired.@s IS "tank.*.@f"
```

In the first rule the APPLYRULE function is passed a string keyword-triplet that contains the pattern "tank.*@f". All rules the containing keyword keyword-triplets with the object name "tank" and the value name "@f" in their premise will be fired.

Format

ASIN (numeric)

Description

This function calculates the arc-sine of the numerical argument in the range $-\pi/2$ to $\pi/2$ radians.

Return Value

This return value is a real numerical constant in radians.

Comments

This function will return 0 if the argument is less than -1 or greater than 1 .

Example

```
IF ASIN (sin_of.trig_argument.@f) > (3.141/4)
THEN RUN_PROCEDURE ("Lower_attack_angle")
```

Format

ASK (string, triplet)

Description

This function will display the question window with the string as the question for the triplet.

Return Value

None.

Comments

When using the ASK function within the Question Editor of Comdale/X \$OAV can be used in place of the keyword triplet as the second argument in the ASK function.

Example

ASK (“What is the value for the constant variable in the sine integral?”, [constant.argument.@f](#))
When COMDALE/X needs to obtain the value for the keyword-triplet [constant.argument.@f](#) from the user it will check to see if the keyword-triplet has a question facet defined. If the facet is defined with the ask function, COMDALE/X will ask the question provided in the ASK function.

Format

ASKUSER (triplet)

only for use in Comdale/C and/or ProcessVision

Description

This function asks the user to input a value for a keyword-triplet.

Return Value

None.

Comments

This function is not limited to @float values but can be used with all data types in the system.

Example

```
IF TRUE
THEN ASKUSER (tank.level.@f)
THEN tank.level.@f= tank.level.@f+ 10.0
```

This example forces the system to ask the user for a [tank.level.@f](#) and then adds 10 to the users answer.

Format

ASNCERTAINTY (triplet, numeric)

Description

This function assigns the numeric argument as the degree of certainty of the triplet argument.

Return Value

A real numerical constant representing the previous degree of certainty of the triplet argument.

Comments

The triplet argument must be a normal keyword-triplet.

Example

```
IF certainty.assignment.needed is TRUE  
THEN old.certainty.@f= ASNCERTAINTY (manipulated.variable.@f, new.certainty.@f)
```

The ASNCERTAINTY function assigns the value of the keyword-triplet `new.certainty.@f` as the certainty of the keyword-triplet “`manipulated.variable.@f`”. It returns the former certainty of the keyword-triplet “`manipulated.variable.@f`” which is then assigned as the value of the keyword-triplet “`old.certainty.@f`”.

Format

ASNDEFAULT (pattern, ...)

Description

This function assigns the default to the keyword-triplets specified in the pattern.

Return Value

None.

Comments

The pattern argument must be a keyword-triplet pattern.

The maximum number of pattern arguments is 20.

Example

```
IF initialize.all.triplets is TRUE  
THEN ASNDEFAULT (“*.*.*”)
```

Format

ASNTHRESHOLD (numeric)

Description

This function assigns the numeric argument as the new threshold for the system.

Return Value

A real numerical constant representing the new threshold.

Comments

A threshold is defined as the acceptable level of certainty that a keyword-triplet has to have before it is used for inference (confidence level)

Example

```
IF threshold.assignment.needed is TRUE  
THEN threshold.level.@f= ASNTHRESHOLD (new.threshold.@f)
```

The ASNTHRESHOLD function assigns the value of the keyword-triplet [new.threshold.@f](#) as the confidence level of the system. It returns this confidence level which is then assigned as the value of the keyword-triplet [threshold.level.@f](#)

Format

ASNUNKNOWN (pattern, ...)

Description

This function sets keyword-triplets to unknown.

Return Value

None.

Comments

The pattern argument must be a keyword-triplet pattern.

The maximum number of pattern arguments is 20.

Example

```
IF TRUE  
THEN ASNUNKNOWN (tank.*.@f, "tank.*.@time")
```

Format

ASNUSERLEVEL (numeric)

Description

This function assigns the numeric argument as the new system userlevel.

Return Value

An integer numerical constant representing the new system userlevel.

Comments

The numeric argument must be in the range from 1 to 5.

If the argument is out of range, the system user level will be set to be the nearest valid userlevel value.

Example

IF userlevel.assignment.needed is TRUE

THEN `user.level.@i` = ASNUSERLEVEL (new.userlevel.@i)

The ASNUSERLEVEL function assigns the value of the keyword-triplet `new.userlevel.@i` as the userlevel of the system. It returns this userlevel which is then assigned as the value of the keyword-triplet `user.level.@i`.

Format

ATAN (numeric)

Description

This function calculates the arc-tangent of the numerical argument.

Return Value

A real numerical constant in radians.

Example

IF TRUE

THEN `polymer.modulus.@f = ATAN (polymer.creep.@f/polymer.stress.@f)`

Format

ATTIME (t, condition)

Description

Tests if a condition was met at time t seconds ago.

Arguments

t is in seconds

condition is a conditional expression-see Nomenclature

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Return Value

TRUE (100) if the value was available from a historical database, otherwise FALSE (0)

Example

IF ATTIME (300, [specific.constraint.@float > 15.35](#))

THEN TEXT (“The specified constraint was met 300 seconds ago.”, “Status”)

Format

ATTR (triplet)

Description

This function extracts the name of the attribute token from the keyword-triplet.

Return Value

A string constant with the attribute name of the triplet.

Comments

ATTR () is most useful when used in the embedded text of the ‘ask’ facet. If used in a facet, \$OAV can be used in place of the keyword triplet.

Example

```
IF attribute.manipulation.needed is TRUE  
THEN attribute.name.@s = ATTR (tank.width.@f)
```

Used in facet Question Editor:

```
ASK (“What is the value of the !$ATTR ($OAV) $! ?”, $OAV)
```

In the first example the ATTR function is passed a keyword-triplet `tank.width.@f`. It returns the attribute name “width” which is stored in the keyword-triplet “`attribute.@s`”

In the second example, the ATTR function is used embedded within !\$ and \$! delimiters in an ASK function which is entered in COMDALE/X’s Question Editor as part of the objects facets.

Format

AVG (numeric, ...)

Description

This function computes the average value of the numerical arguments.

Return Value

A real numerical constant.

Comments

This maximum number of numerical arguments for the function is 20.

Example

IF average.value.needed is TRUE

THEN `average.value.@f` = AVG (`tank.depth.@f`, `tank.height.@f`, `tank.width.@i`)

Format

BACKWARDCHAIN_ENABLED ()

Description

This function tests to see if backward chaining is enabled or not.

Return Value

TRUE if backward chaining is enabled, FALSE otherwise.

Comments

This function returns a Boolean measure of the status of the state of the backward chaining option. Use the function to test if backward chaining is enabled or not, and use DISABLEBACKWARDCHAIN () function to enable or disable this option.

Example

```
IF BACKWARDCHAIN_ENABLED ( )  
THEN DISABLEBACKWARDCHAIN (T)
```

Format

CEIL (numeric)

Description

This function calculates the ceiling of the numerical argument.

Return Value

An integer representing the numerical argument rounded up to the next whole number.

Example

```
IF rounded_up.value.needed is TRUE  
THEN rounded_up.value.@i= CEIL (variable.of_interest.@f)
```

Format

CERTAINTY (triplet)

Description

This function returns the certainty of the keyword triplet.

Return Value

A real numerical constant representing the certainty of the triplet. If the keyword-triplet is not instantiated, -9999.0 will be returned.

Comments

The argument must be a normal keyword-triplet.

Example

```
IF certainty.value.needed is TRUE  
THEN certainty.value.@f= CERTAINTY (variable\_of\_interest.@f)
```

The CERTAINTY function returns the certainty of the keyword-triplet “tank.depth.@f “ which is then assigned as the value of the keyword-triplet “certainty.value.@f “.

Format

CERTAINTYAT (triplet, t)

only for use in Comdale/C and/or ProcessVision

Description

Queries the certainty of the triplet at time t seconds ago

Arguments

t is in seconds

triplet can be a logical or numerical triplet

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database

Return Value

A numerical whose value varies from 0 to 100

Example

IF CERTAINYAT (tank.level.high, 60) > 90

THEN TEXT (“ tank level was definitely quite high 1 minute ago”, “Tank Info”)

Format

CHANGE (logical triplet, numeric triplet)

Description

This function adds to the value of the numerical keyword-triplet argument an amount dictated by the fuzzy set associated with the logical keyword argument.

Return Value

None.

Comments

The logical triplet argument must have a fuzzy set associated with it.

The numeric triplet is not changed TO the value in the fuzzy set, it is changed BY the value in the fuzzy set.

Example

```
IF reagent.tank.high
AND reagent.level.rising_fast
THEN control.action.open_relief_value is TRUE
THEN CHANGE (control.action.open_relief_value, relief_valve.setpoint.@f)
THEN TEXT (“The relief valve has been changed to !$ relief_valve.setpoint.@f$!”,
“Setpoint Change”)
```

In the above example the premise of the rule will be examined and if the combination of the certainties according to the Inference Facet of the current rule is above the system wide Confidence Level then the THENs will be executed. For example if reagent.tank.high has a cf=85% and reagent.level.rising_fast has a cf=55% and the inference facet for the rule has set AND expressions to combine using the AVERAGE confidence factor then the premise will be TRUE with a confidence factor of 70% (as long as the system wide Confidence Level is below 70%, otherwise the rule’s premise will be considered FALSE).

The CHANGE () function now changes the value of relief_valve.setpoint.@f by the amount dictated in the fuzzy-set associated with the triplet control.action.open_relief_valve. if this fuzzy set is:

Value	-0.4	-0.2	0.0	0.2	0.4
Rank	0	40	50	70	100

And the original value of relief_valve.setpoint.@f is 0.5 then the new value will be:

New value = original value + delta value (from fuzzy set)
 = 0.5 + 0.2
 = 0.7

and its certainty will be assigned 100%.

Format

CLASSAVG (numeric)

Description

This function computes the average value of the keyword-triplets, within a class, that share the same attribute name.

Return Value

A real numerical constant representing the average value.

Comments

The numeric argument must be a numerical class keyword-triplet. The class name must be enclosed in curly brackets.

Example

```
IF class_average.value.needed is TRUE  
THEN class_average.value.@f= CLASSAVG ({ tank).width.@f)
```

The function computes the average value of the keyword-triplets belonging to the class “tank” that have an attribute name “width” and a value name “@f”.

Format

CLASSMAX (numeric)

Description

This function computes the maximum value of the keyword-triplets, within a class, that share the same attribute name.

Return Value

A real numerical constant representing the maximum value.

Comments

The numeric argument must be a numerical class keyword-triplet. The class name must be enclosed in curly brackets.

Example

```
IF class_maximum.value.needed is TRUE  
THEN class_maximum.value.@f= CLASSMAX ({tank}.width.@f)
```

The function computes the maximum value among the keyword-triplets belonging to the class “tank” that have an attribute name “width” and a value name “@f”.

Format

CLASSMAXOBJECT (numeric class triplet)

Description

This function finds the object that has the maximum value.

Return Value

A string containing the object name.

Comments

If more than one object has the maximum value, only the first object encountered with the maximum value is returned

Example

```
IF best.student.@s IS CLASSMAXOBJECT ({student}.mark.@f)
THEN TEXT (“!$ best.student.@s $! Has the highest mark.”)
```

This example tests if the student name held within the triplet `best.student.@s` is the same name as the object name returned by the CLASSMAXOBJECT function. If so, then a text message is printed out indicating that the student with the best mark is `best.student.@s`.

Format

CLASSMIN (numeric)

Description

This function computes the minimum value of the keyword-triplets, within a class, that share the same attribute name.

Return Value

A real numerical constant representing the minimum value.

Comments

The numeric argument must be a numerical class keyword-triplet. The class name must be enclosed in curly brackets.

Example

```
IF class_minimum.value.needed is TRUE  
THEN class_minimum.value.@f= CLASSMIN ( {tank}.width.@f)
```

The function computes the minimum value among the keyword-triplets belong to the class “tank” that have an attribute name of “width” and a value name “@f”.

Format

CLASSMINOBJECT (numeric class triplet)

Description

This function finds the object that has the minimum value.

Return Value

A string containing the object name.

Comments

If more than one objects have the minimum value, only the first object encountered is returned.

Example

```
IF worst.student.@s is CLASSMINOBJECT ({student}.mark.@f)
THEN TEXT (“!$ worst.student.@s $! has the lowest mark.”)
```

This example tests if the student name held within the triplet [worst.student.@s](#) is the same name as the object name returned by the CLASSMINOBJECT function. If so, then a text message is printed out indicating that the student with the worst mark is [worst.student.@s](#).

Format

CLASSNUMOBJECT (string)

Description

This function returns the number of object in a class.

Return Value

An integer indicating the number of objects in the class.

Comments

This function only work on class keyword-triplets.
Enclose the class name in double quotes.

Example

```
IF TRUE  
THEN num.student.@i = CLASSNUMOBJECT ("Art_101")  
THEN TEXT ("There are !$ num.student.@i$! Student in Art 101.")
```

Format

CLASSSTDDEV (numeric)

Description

This function computes the standard deviation of the keyword-triplets, within a class, that share the same attribute name.

Return Value

A real numerical constant representing the standard deviation.

Comments

The numeric argument must be a numerical class keyword-triplet. The class name must be enclosed in curly brackets.

Example

```
IF class_stddev.value.needed is TRUE  
THEN class_stddev.value.@f= CLASSSTDDEV ({tank}.width.@f)
```

The function computes the standard deviation of the keyword-triplets belonging to the class “tank” that have an attribute name of “width” and a value name “@f”.

Format

CLASSTOLIST (string)

Description

This function returns the list of objects in a class.

Return Value

A dynamic list of objects.

Comments

This function only works on class keyword-triplets.
Enclose the class name in double quotes.

Example

```
IF TRUE  
THEN ?all_student = CLASSTOLIST ("Student")
```

This example creates the dynamic object all_student which contains a list of all the objects which belong to the class {Student}.

Format

CLASSTOTAL (numeric class triplet)

Description

This function sums the numeric values of the keyword-triplets, within a class, that share the same attribute name.

Return Value

A real number representing the total value.

Example

```
IF class_total.value.needed  
THEN class_total.value.@f= CLASSTOTAL ({tank}.width.@f)
```

Format

CONNECT (object-string, link-string)

Description

This function connects COMDALE/X to another application using the n Dynamic Data Exchange protocol.

Return Value

An integer holding the conversation ID tag.

Comments

Object-string is the name of the object which holds the return value from this function (name of triplet's object part to which this function is assigned)

Link-string contains the name of the application you are linking with and any optional parameters it may take.

SPECIAL DDE CONSIDERATIONS

CONNECT function cannot be called without assigning its return value to a numeric triplet. The object portion of the triplet assigned the DDE ID number can be anything BUT the associated triplet MUST contain:

1. an attribute/value: `type.@string` holding: "DDE"
2. an attribute/value: `mapfile.@string` holding: the name of the mapping file
3. an attribute/value: `id.@integer` holding: the return value from CONNECT

A unique mapping file is made for each DDE connection made using the Mapping File Editor provided with COMDEL/X.

A maximum of 10 DDE conversations can be open simultaneously.

The Link-string must hold the application's REGISTERED name, not filename. The following table lists the registered names of a few applications:

Table 15-1

Application Name	Registered Name
COMDALE/X Version 5.13	Comdx5
Microsoft Excel™	Excel
Microsoft Word for Windows™	Winword
Asymmetric's Toolbook™	Toolbook

™- All products are registered trademarks of their respective manufactures.

Example

```

IF communications.channel.needs_to_be_opened is TRUE
THEN my_dde_link.type.@string is "DDE"
THEN my_dde_link.mapfile.@string is "mapfile2.mpf"
THEN my_dde_link.id.@integer = CONNECT ("my_dde_link", "Excel sheet1.xls")

```

Format

COS (numeric)

Description

This function calculates the cosine of the numerical argument.

Return Value

A real numerical constant.

Comments

The numeric argument is in radians.

If the numeric argument is too large at 0 will be returned.

Example

IF variable.needs.calculation is TRUE

THEN `variable.value.@f`= COS (`other.value.@f`)

Format

COSH (numeric)

Description

This function calculates the hyperbolic cosine of the numeric argument.

Return Value

A real numerical constant.

Comments

The result from the hyperbolic cosine function increases rapidly as the absolute value of the argument becomes greater than one (1). If the hyperbolic cosine function is required to return a value given a large numeric argument the developer is advised to check that the return value does not exceed the limitations of the @float, @double or system specifications.

Example

```
IF integral.of_sinh.needed is TRUE  
THEN integral\_of.sinh\_a\_x\_dx.@f= COSH (constant.argument.@f*trig.argument.@f) /  
constant.@f
```

In this example the COSH function is used in calculating an integral.

Format

CYCLE ()

Arguments

None

Description

This function restarts the consultation and forgets the degrees of certainty and values of all instantiated keyword-triplets that are subgoals or goals.

Return Value

None.

Example

```
IF cycle.knowledge_base.needed is TRUE  
THEN CYCLE ( )
```

The function will forget the degrees of certainty and value of all instantiated keyword-triplets that are subgoals or goals and then restart the consultation at the beginning of the knowledge base.

Format

DATASOURCE_ENABLED ()

Description

This function tests to see if DataSource is enabled or not.

Return Value

TRUE if DataSource is enabled, FALSE otherwise.

Example

```
IF DATASOURCE_ENABLED ( )  
THEN DISABLEDATASOURCE (T)
```

Format

DAY (date)

Description

This function extracts the day from the date argument.

Return Value

An integer numerical constant representing the day of the argument.

Comments

The date argument must be a date keyword-triplet.

Example

```
IF day.value.needed is TRUE  
THEN day.value.@i = DAY (delivery.date.@date)
```

If `delivery.date.@date` is Jan 10, 1993, the function will return 10.

Format

DISABLEBACKWARDCHAIN (TRUE | FALSE)

Description

This function disables or enables backward chaining.

Return Value

TRUE if function is successful, FALSE otherwise.

Comments

Set the argument to TRUE will disable backward chaining.
Set the argument to FALSE will enable backward chaining.
Backward chaining is enabled by default.

Example

```
IF desable.backward.chain is TRUE  
THEN DISABLEBACKWARDCHAIN (T)  
ELSE DISABLEBACKWARDCHAIN (F)
```

Format

DISABLEDATASOURCE (TRUE | FALSE)

Description

This function disables or enables DataSource.

Return Value

TRUE if function is successful, FALSE otherwise.

Comments

Set the argument to TRUE will disable DataSource.
Set the argument to FALSE will enable DataSource.
DataSource is enabled by default.

Example

```
IF disable.DataSource.flag is TRUE  
THEN DISABLEDATASOURCE (T)  
THEN DISABLEDATASOURCE (F)
```

Format

DISABLEFORWARDCHAIN (TRUE | FALSE)

Description

This function disables or enables forward chaining.

Return Value

TRUE if function is successful, FALSE otherwise.

Comments

Set the argument to TRUE will disable forward chaining.
Set the argument to FALSE will enable forward chaining.
Forward chaining is enabled by default.

Example

```
IF disable.forward.chain is TRUE  
THEN DISABLEFORWARDCHAIN (T)  
THEN DISABLEFORWARDCHAIN (F)
```

Format

DISABLEIFCHANGE (TRUE | FALSE)

Description

This function disables or enables IfChange .

Return Value

TRUE if function is successful, FALSE otherwise.

Comments

Set the argument to TRUE will disable IfChange .

Set the argument to FALSE will enable IfChange.

IfChange is enabled by default.

Example

```
IF disable.IfChange.flag is TRUE  
THEN DISABLEIFCHANGE (T)  
THEN DISABLEIFCHANGE (F)
```

Format

DISCONNECT (ID-triplet)

Description

This function disconnects COMDALE/X from another application.

Return Value

TRUE if disconnected properly

Comments

ID-triplet is the triplet which was assigned the DDE conversation ID tag from the CONNECT function.

Example

```
IF disconnect.dde_communication_needed is TRUE  
THEN DISCONNECT (my\_dde\_link.id.@integer)
```

Format

DISPLAY (string, string)

Description

This function will display a topic (argument 2) of a document (argument 1).

Return Value

None.

Comments

The document must be compiled with the Comkale Hypertext utility (QNX: /Process Vision/hytext, DOD :hytext.exe). If the second argument is not defined the first page of the document will be presented.

Example

```
IF display.topic.needed is TRUE  
THEN DISPLAY ("tank", "material")
```

The function will display the topic named "material" of the document named "tank".

Format

DONOTHING ()

Description

This function takes no action.

Return Value

None.

Comments

This function is used to force the instantiation of keyword-triplets in the premise of a rule without concluding about a subgoal or goal.

Example

```
IF keyword_triplet.instantiation.needed is TRUE  
AND keyword\_triplet.value\_needed.@f  
THEN DONOTHING ( )
```

This function will return without taking any action. But the keyword-triplets in the premise will be instantiated.

Format

ELAPSEDTIME (time, time)

Description

This function computes the difference between the first time argument and the second time argument.

Return Value

An integer numerical constant representing the elapsed time in seconds

Comments

Both arguments must be valid time keyword-triplets.

Example

IF time.difference.needed is TRUE

THEN `time.difference.@i` = ELAPSEDTIME (`present.time.@time`, `delivery.time.@time`)

If present time is 10 hr 2min 16sec and delivery time is 9hr 1min 10sec, the function will return 3666 sec.

Format

EXP (numeric)

Description

The function calculates the exponential function of the numeric argument.

Return Value

A real numerical constant.

Comments

If the argument is too small the function will return a 0.

Example

```
IF exponential.value.needed is TRUE  
THEN exponential.value.@f= EXP (tank.width.@f)
```

Format

EXPORT (string, pattern, numeric, numeric)

Description

This function writes the value and degree of certainty of the triplets, as defined by the pattern, to the specified file.

Return Value

None.

Comments

The first argument must contain a string that describes the path and the filename of the file in which triplet information will be written. If a '+' sing is added to the end of the filename argument the export information will be appended, otherwise the file will be overwritten.

The ent must describe the keyword-triplets to be exported.

The numeric arguments must be the lower and upper bounds of the degree of certainty of the keyword-triplets to be exported. The first numeric argument should be less than the second numeric argument.

Example

```
IF exported.date.needed is TRUE
THEN EXPORT ("export.fil", "tank.*.*", 50.0, 100.0)
```

The function will export and write all the instantiated keyword-triplets, with an object name of "tank" that have a certainty between 50.0 and 100.0, into the file export.fil.

```
IF exported.data.needed is TRUE
THEN EXPORT ("export.fil +", "tank.*.*", 50.0, 100.0)
```

The function will export and append all the instantiated keyword-triplets, with an object name of "tank" that have a certainty between 50.0 and 100.0, into the file export.fil.

The format of the file is:

Keyword-triplet name	certainty	value
computer.status.on	45.900000	100.000000
Raw_mix.density.@f	100.000000	34.543241
Water.flow_rate.@f	100.000000	126.764
Kiln_output.last_hour.@i	100.000000	650.1
Times_output.last_hour.@i	100.000000	3
Out_of.spec.@time	100.000000	13:45:03

Format

FIND (pattern, ...)

Description

This function attempts to instantiate the keyword-triplets specified in the pattern.

Return Value

None.

Comments

The pattern argument must be a keyword-triplet pattern.

The maximum number of pattern arguments is 20.

Example

```
IF find.values.needed is TRUE  
THEN FIND ("tank.*.*", "valve.*.*")
```

The function will attempt to instantiate the keyword=triplets that have "tank" or "valve" as their object name.

Format

FIRSTTIME (t, condition)

only for use in Comdale/C and/or ProcessVision

Description

Returns the time in seconds that the condition first occurred. The check is done over the last t seconds or since time t.

Arguments

t is in seconds

condition is a conditional expression –see 1.3.3 Nomenclature

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database

Return Value

Time in seconds, or –1 if condition is never met.

Example

```
IF TRUE
THEN first.time.@f= FIRSTTIME (20, tank.level.@float < 1)
THEN TEXT (“The first time the condition was met was !$
First.time.@f$! seconds ago.”, “Tank Level Status”)
```

Format

FLOOR (numeric)

Description

The function rounds the numeric argument down to the next whole number.

Return Value

An integer constant.

Example

```
IF rounded_down.value.needed is TRUE  
THEN rounded_down.value.@i = FLOOR (tank.depth.@f)
```

Format

FORGET(pattern, ...)

Description

This function will un-instantiate all the keyword-triplets specified by the pattern.

Comments

By un-instantiating a keyword triplet the confidence factor in the triplet becomes NOTKNOWN. This effectively forces the system to re-establish its value and confidence factor.

Using the FORGET () function can prove beneficial when a change in the system requires a recalculation if certain triplet values and certainties.

The pattern argument must be a keyword-triplet pattern.

The maximum number of pattern arguments is 20.

Return Value

None

Example

```
IF forget.values.needed is TRUE  
THEN FORGET ("tank.level.high")
```

The function will un-instantiate the keyword-triplet tank.level.high.

```
IF forget.values.needed is TRUE  
THEN FORGET ("tank.*.*", "valve.*.*")
```

The function will un-instantiate the keyword-triplets that have "tank" or "valve" as their object name.

Format

FORM (string)

Description

This function will display a pre-designed form.

Return Value

None.

Comments

The string argument must contain the valid name of a form.
The form must be designed using the form editor.

Example

```
IF form.presentation.needed is TRUE  
THEN FORM ("tank.frm")
```

The function will display the form named "tank.frm". The system will wait for the user to input the required information if the form was designed to acquire information. If the form was designed to present information the system would wait for the user's acknowledgement.

Format

FORMAT (triplet, format string)

Description

This function formats the value of a keyword triplet.

Return Value

A formatted string.

Comments

The format string is a 'C' format string.

Formatting is only allowed on string (@s) or numeric keyword-triplets (@i, @long, @f, @double).

Use:

%s for string

%ld for @integer and @long

%lf for @float and @double

An integer placed between the % sign and the 's', 'ld', or 'lf' will act as a minimum-field width specification. This pads the output string with blanks on the left to ensure a minimum length. If the result is longer than the minimum-length specify and it is a number then it will be displayed in full. To pad with zeros other than the default specify a zero before the minimum-field width specify. For example, to pad with zeros instead of spaces use %05ld. This will pad a number or string which is less than five characters with zeros so that its total length is five.

To specify the number of characters after the decimal point for a floating point number place a decimal point and a number of characters desired after the field width specify. For example %12.6lf will display a number at least 12 characters wide with 6 decimal places. When this format is applied to integers or strings it is interpreted as being the maximum field width specify. For example %4.8s will display a string of at least 4 characters but no longer than 8. If the string is longer than eight characters it will be truncated to fit.

The default output is right-justified, to over-ride this use a minus sign. For example % -11.3lf will left-justify a floating-point number with three decimal places in an eleven character field.

Example

```
IF TRUE
THEN formatted.string.@s is FORMAT (unformatted. String.@s,
  "This is the string : %15.25s")
```

```
IF TRUE
THEN TEXT ("!$ FORMAT (tank.level.@f "Tank level = %.2lf") $!")
```

The first example will add the prefix "This is the string:" to the unformatted string and will ensure a minimum of 15 characters in the final string but not more than 25.

The second example will return a string which contains the value of the tank.level.@f with a two character decimal point.

Format

FORWARDCHAIN_ENABLED ()

Description

This function tests to see if forward chaining is enabled or not.

Return Value

Is TRUE if forward chaining is enabled, FALSE otherwise.

Example

```
IF FORWARDCHAIN_ENABLED ( )  
THEN DISABLEFORWARDCHAIN (T)
```

Format

FREERULE (\$constant, pattern)

Description

This function frees up rules specified by the pattern that were previously executed, and makes them available again for evaluation (execution) during the same consultation session.

Return Value

None.

Comments

The \$constant argument specifies how the pattern should be interpreted.

If the \$constant argument is \$RULE the pattern will be interpreted as a rule pattern and all rules with rule names matching the pattern will be made available to be fired again.

If the \$constant argument is \$ALLPREM the pattern will be interpreted as a keyword-triplet pattern and all rules with keyword-triplets in their premises that match the pattern will be made available to be fired again.

If the \$constant argument is \$ALLCONCL the pattern will be interpreted as a keyword-triplet pattern and all rules with keyword-triplets in their conclusions that match the pattern will be made available to be fired again.

****IMPORTANT:** The FREERULE function is essential in allowing procedures to repetitively call rules when using a .knw file developed for use with cc_batch.

Example

```
IF free.rules.needed is TRUE
THEN FREERULE ($RULE, "rule1")
THEN FREERULE ($RULE, "setpoint*")
```

The function will free up the rule with the name "rule", also all rules with names beginning with 'setpoint' will be freed.

```
IF free.rules.needed is TRUE
THEN FREERULE ($ALLPREM, "tank.*.*")
```

The function will free up rules containing keyword-triplets with the object name "tank" in their premises.

```
IF free.rules.needed is TRUE
THEN FREERULE ($ALLCONCL, "tank.*.*")
```

The function will free up rules containing keyword-triplets with the object name "tank" in their conclusions.

Format

GETPASSWORD ()

Description

This function returns the password of the knowledge base.

Return Value

A string containing the password.

Comments

Password is 8 characters or less.

If password does not exist, an empty string (“”) is returned.

Example

```
IF STRCOPY (current.password.@s, GETPASSWORD ( ))  
THEN TEXT ( “Current password is !$ current.password.@s $!”)
```

This example uses the STRCOPY () function to copy the string returned by the GETPASSWORD () function into a string type keyword triplet. If the string copy is successful then a test message showing the password is shown.

Format

GETPOINTDATA (pattern)

Description

This function reads the current value, string or certainty of a triplet from the point database.

Return Value

TRUE (100)

Comments

Can be used to force a rule to look for a value in the point database before backchaining to find the value.

This function is similar to ASKCC () in that it queries the point database for the current value of a triplet, but is different in that it accepts a triplet pattern as an argument.

Example

```
IF GETPOINTDATA ("a_line.temperature.*")  
AND a_line.temperature.reading_taken is TRUE  
THEN RUN_PROCEDURE ("report_proc")
```

Format

GETREPORTPROCEDURE ()

Description

This function returns the name of the report procedure.

Return Value

A string containing the name of the report procedure.

Comments

If report procedure does not exist, an empty string (“”) is returned.

Example

```
IF GETREPORTPROCEDURE ( ) is “”  
THEN SETREPORTPROCEDURE (“report_proc”)
```

If report procedure is not set, then set it to report_proc.

Format

GETSTARTPROCEDURE ()

Description

This function returns the name of the start procedure.

Return Value

A string containing the name of the start procedure.

Comments

If start procedure does not exist, an empty string (“”) is returned.

Example

```
IF GETSTARTPROCEDURE ( ) is “Start_proc”  
ELSE SETSTARTPROCEDURE (“Start_proc”)
```

This rule makes sure the start procedure is set to “Start_proc”.

Format

GETSTARTRULE ()

Description

This function returns the name of the start rule.

Return Value

A string containing the name of the start rule.

Comments

If start rule does not exist, an empty string (“ ”) is returned.

Example

```
IF GETSTARTRULE ( ) is “Rule_1”  
THEN SETSTARTRULE (“Rule_2”)
```

Format

GETSTARTRULESET ()

Description

This function returns the name of the start rule set.

Return Value

A string containing the name of the start rule set.

Comments

“root” is a default rule set that contains all the rules in a knowledge base.

Example

```
IF GETSTARTRULESET ( ) is_not “root”  
THEN SETSTARTRULESET (“root”)
```

Format

GOTO (string)

Description

This function will fire the rule with the name specified by the string argument. (It does not return to the rule containing the function).

Return Value

None.

Example

```
IF rule.firing.needed is TRUE  
THEN GOTO (“integralrule1”)
```

The function will fire the rule named “integralrule1”.

Format

HALT ()

Description

This function terminates the consultation session and displays the conclusions.

Return Value

None.

Comments

Using this function in Comdale/C causes the controller module to exit from memory

Example

```
IF stop.session.needed is TRUE  
THEN HALT ( )
```

The system will halt and display all conclusions.

Format

HOUR (time)

Description

This function computes the hour from the time argument.

Return Value

An integer numerical constant representing the hour of the time argument.

Comments

The time argument must be a time keyword-triplet.

Example

```
IF hour.value.needed is TRUE  
THEN hour.value.@i = HOUR (delivery.time.@time)
```

If `delivery.time.@time` is 32hr 5min 10sec, the function will return 23.

Format

HOWLONG (triplet)

only for use in Comdale/C and/or ProcessVision

Description

Returns the time in seconds since this triplet last changed its value.

Arguments

t is in seconds

Return Value

Time in seconds.

Example

```
IF TRUE
THEN length_of.time.@integer = HOWLONG (tank.level.@float)
THEN TEXT ("The tank level has been at its current level for !$
Length_of.time.@integer $! Seconds", "Tank Status")
```

Format

IFCHANGE_ENABLED ()

Description

This function tests to see if IfCange is enabled or not.

Return Value

TRUE if IfChange is enabled, FALSE otherwise.

Example

```
IF IFCHANGE_ENABLED ( )  
THEN DISABLEIFCHANGE (T)
```

Format

IGNORE (\$constant, pattern)

Description

This function disallows the firing of rules specified by the pattern.

Return Value

None.

Comments

The \$constant argument specifies how the pattern should be interpreted.

If the \$constant argument is \$RULE the pattern will be interpreted as a rule pattern and all rules with rule names matching the pattern will be disallowed from firing.

If the \$constant argument is \$ALLPREM the pattern will be interpreted as a keyword-triplet pattern and all rules with keyword-triplets in their premises that match the pattern will be disallowed from firing.

If the \$constant argument is \$ALLCONCL the pattern will be interpreted as a keyword-triplet pattern and all rules with keyword-triplets in their conclusions that match the pattern will be disallowed from firing.

Example

```
IF disallow.rules.needed is TRUE
THEN IGNORE ($RULE, "rulw1")
THEN IGNORE ($RULE, "upset*")
```

The function will disallow the rule with the name "rule1" from firing during the consultation session, also all rules that have names beginning with "upset" will be disallowed from firing.

```
IF disallow.rules.needed is TRUE
THEN IGNORE ($ALLPREM, "tank.*.*")
```

The function will disallow all rules containing keyword-triplets with the object name "tank" in their premises.

```
IF disallow.rules.needed is TRUE
THEN IGNORE ($ALLCONCL, "tank.*.*")
```

The function will disallow all rules containing keyword-triplets with the object name "tank" in their conclusions.

Format

IMPORT (string, numeric, numeric)

Description

This function imports the value and degree of certainty of the triplets from the specified file and instantiates them in the knowledge system.

Return Value

None.

Comments

The first argument must contain a string that describes the path and the filename of the file from which triplet information will be read.

The numeric arguments must be the lower and upper bounds of the degree of certainty of the keyword-triplets to be imported. The first numeric argument should be less than the second numeric argument.

The format for the file is that which is identical to that created by the EXPORT function.

Example

```
IF imported.values.needed is TRUE  
THEN IMPORT ("import.fil", 50.0, 100.0)
```

Format

INHERITPARENTVALUE (triplet)

Description

This function allows the given triplet to inherit value from its parent classes.

Return Value

None.

Comments

The triplet can either be a normal keyword-triplet.

Upon successful inheritance, the keyword-triplet will be instantiated with the inherited value and certainty.

Example

INHERITPARENTVALUE ([smalltank.level.@float](#))

If smalltank belongs to a class 'tank' and [{tank}.level.@float](#) is instantiated with 89.0 and degree of certainty of 100.0. Upon execution, [smalltank.level.@float](#) will be instantiated with 89.0 and certainty of 100.0.

Format

JUSTAFTER (t, condition)

only for use in Comdale/C and/or ProcessVision

Description

Tests if a condition occurred only at times between t to present.

Arguments

t is in seconds

condition is a conditional expression –see Nomenclature

Return Value

TRUE (100), FALSE (0)

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

ExampleIF JUSTAFTER (100, “2 < [tank.level.@float](#)”)

THEN TEXT (“The tank level has been greater than 2 since 100 seconds ago”, “Tank Status”)

Format

JUSTBEFORE (t1, t2, condition) only for use in Comdale/C and/or ProcessVision

Description

Tests if a condition occurred only at times t2. The check is done as far to the present as time t1.

Arguments

t1 is in seconds

t2 is in seconds

condition is a conditional expression –see Nomenclature

If t1 is 0 then the check is done to the present.

Return Value

TRUE (100), FALSE (0)

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database

Example

IF JUSTBEFORE (10, 20, “tank.level.@float >

[tank.height_setpoint.@float](#)”)

THEN TEXT (“The tank level has been above the setpoint value only before 10 seconds ago”,
“Tank level status”)

Format

JUSTDURING (t1, t2, condition) only for use in Comdale/C and/or ProcessVision

Description

Tests if a condition occurred only at times during the period t1 to t2.

Arguments

t1 is in seconds

t2 is in seconds

condition is a conditional expression –see Nomenclature

Return Value

TRUE (100), FALSE (0)

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database

Example

IF JUSTDURING (20, 50, “tank.level.@float >=
[tank.height_setpoint.@float](#)”)

THEN TEXT (“The tank level has been above the setpoint value only during 20 to 50 seconds ago”, “Tank level status”)

Format

KNOWN (triplet)

Description

This function finds if the triplet is known.

Return Value

If the triplet is known the function returns true otherwise it returns false.

Comments

This function makes no attempt to instantiate keyword-triplets.

Example

```
IF keyword-triplet.status.needed is TRUE  
THEN keyword_triplet.status.instantiated = KNOWN (tank.level.high)
```

If tank.level.high has been instantiated, the function will return true.

Format

LISTADDOBJECT (dynamic list, string)

Description

This function adds an object to a dynamic list.

Return Value

A dynamic list with the new object added to the end of the list.

Comments

The object name must appear in double quotes.

Object will only be added if it is not in the list already.

Example

```
IF TRUE
```

```
THEN ?going = LISTADDOBJECT (?going, "John")
```

Format

LISTALLOBJECT (dynamic list, string, string)

Description

This function outputs the object names in a dynamic list into a string.

Return Value

A string containing the object names.

Comments

The original dynamic list is not modified.

Object names are joined by the first string.

If the dynamic list is empty, the second string is returned.

Example

```
IF TRUE
```

```
THEN ?X = ANY ({person}.go_to.BBQ is TRUE)
```

```
THEN whose.going.@s is LISTALLOBJECT (?x, “ and “, “Nobody”)
```

```
THEN TEXT (“!$ whose.going.@s $! is going to the BBQ.”)
```

Format

LISTCLEAR (dynamic list)

Description

This function empties a dynamic list.

Return Value

TRUE if list is emptied, FALSE otherwise.

Example

```
IF TRUE  
THEN LISTCLEAR (?list)
```

Format

LISTCOMMONOBJECT (dynamic list, dynamic list)

Description

This function returns a dynamic list containing the common objects in the two lists.

Return Value

A dynamic list of object names.

Comments

The original dynamic lists are not modified.

Example

```
IF ?large_vol = ANY ({tank}.volume.@f > 100)
AND ?tall_tank = ANY ({tank}.level.@f > 50)
THEN ?big_tank = LISTCOMMONOBJECT (?large_vol, ?tall_tank)
THEN big.tank.@s is LISTALLOBJECT (?big_tank, "", "No big tank.")
```

Format

LISTFIRSTOBJECT (dynamic list)

Description

A string containing an object name.

Comments

The original dynamic list is not modified.

If the list is empty, an empty string (“”) is returned.

Example

```
IF ?large_tank = ANY ({tank}.volume.@f > 100)
THEN first.large_tank.@s is LISTFIRSTOBJECT (?large_tank)
THEN TEXT (“The first large tank is !$ first.large_tank.@s $!”)
```

Format

LISTHASOBJECT (dynamic list, string)

Description

This function tests to see if an object is in a dynamic list.

Return Value

TRUE, if object is in the list.

FALSE, otherwise.

Comments

The original dynamic list is not modified.

Put the object name to test for in double quotes.

Example

```
IF ?going = ANY ({person}.go_to.BBQ is TRUE)
AND LISTHASOBJECT (?going, "John")
THEN TEXT ("John is going to the BBQ.")
ELSE TEXT ("John is not going to the BBQ.")
```

Format

LISTLASTOBJECT (dynamic list)

Description

This function returns the last object name in a dynamic list.

Return Value

A string containing an object name. If the list is empty, an empty string (“”) is returned.

Comments

The original dynamic list is not modified.

Example

```
IF ?large_tank = ANY ( {tank}.volume.@f > 100)
THEN last.large_tank.@s is LISTLASTOBJECT (?large_tank)
THEN TEXT (“The last large tank is !$ last.large_tank.@s $!”)
```

Format

LISTMERGE (dynamic list, dynamic list)

Description

This function merges two lists together to form a combined list.

Return Value

A dynamic list.

Comments

The original dynamic list is not modified.

Example

```
IF ?red = ANY ({car}.color.@s is "red")  
AND ?green = ANY ({car}.color.@s is "green")  
THEN ?new_list is LISTMERGE (?green, ?red)
```

Format

LISTNTHOBJECT (dynamic list, integer)

Description

This function returns the n-th object name in a dynamic list.

Return Value

A string containing an object name.

Comments

The original dynamic list is not modified.

If the list is empty or invalid index is given, an empty string (“”) is returned.

Example

```
IF ?student = LISTALLOBJECT (({ART_101}.name.@s)
THEN ?list_1 = LISTSORTVALUE (?student.mark.@f)
THEN ?list_2 = LISTREVERSE (?list_1)
THEN TEXT (“1st place = !$ LISTNTHOBJECT?list_2, 1) $!”)
THEN TEXT (“2nd place = !$ LISTNTHOBJECT?list_2, 2) $!”)
THEN TEXT (“3rd place = !$ LISTNTHOBJECT?list_2, 3) $!”)
```

Format

LISTNUMOBJECT (dynamic list)

Description

This function returns the number of objects in a dynamic list.

Return Value

An integer indicating the number of objects in the list.

Comments

The original dynamic list is not modified.

Example

```
IF ?going = ANY ({person}.go_to.BBQ is TRUE)
THEN number.going\_to\_BBQ.@i= LISTNUMOBJECT (?going)
```

Format

LISTREMOVEOBJECT (dynamic list, string)

Description

This function removes an object from a dynamic list.

Return Value

A dynamic list with the object removed.

Comments

The original dynamic list is not modified.

Example

```
IF ?going = ANY ({person}.go_to.BBQ is TRUE)
AND LISTHASOBJECT (?going, "John")
THEN TEXT ("John is going to the BBQ.")
THEN ?going = LISTREMOVEOBJECT (?going, "John")
THEN TEXT ("No, he is not!")
```

Format

LISTREVERSE (dynamic list)

Description

This function reverses the objects in a dynamic list.

Return Value

A dynamic list.

Comments

The original dynamic list is not modified.

Example

```
IF ?tank = LISTALLOBJECT ({tank}.level.@f)
THEN ?small_to_big_tank = LISTSORTVALUE (?tank.level.@f)
THEN ?big_to_small_tank = LISTREVERSE (?small_to_big_tank)
```

Format

LISTSORTNAME (dynamic list)

Description

This function sorts the objects in a dynamic list into alphabetical order.

Return Value

A sorted dynamic list.

Comments

The original dynamic list is not modified.

Example

```
IF ?student = CLASSTOLIST ({student}.name.@s)
THEN ?alphabetical_order = LISTSORTNAME (?student)
```

Format

LISTUNCOMMONOBJECT (dynamic list, dynamic list)

Description

This function returns a dynamic list containing the object names that are either in the first list or the second list, but not in both.

Return Value

A dynamic list of object names.

Comments

The original dynamic list is not modified.

Example

```
IF ?visa = ANY ({person}.has.visa is TRUE)
AND ?amex = ANY ({person}.has.amex is TRUE)
THEN ?one_card = LISTUNCOMMONOBJECT (?visa, ?amex)
```

The list ?one_card will contains the names of people who do not own both VISA and AMERICAN EXPRESS card.

Format

LOAD (string)

Description

This function loads the knowledge base specified by the string.

Return Value

None.

Comments

The string argument must contain a valid knowledge base name.

This function is usually used in conjunction with the run() and run_continue() functions.

Example

```
IF tank.knowledge_base.needed is TRUE  
THEN LOAD ("tank.knw")
```

The function will load the knowledge base named "tank.knw" and it will become the current knowledge base.

Format

LOG (numeric)

Description

This function calculates the natural logarithm of the numeric argument.

Return Value

A real numerical constant.

Comments

The function will return a 0 if the argument is less than or equal to 0.

Example

```
IF natural.log.needed is TRUE  
THEN log.value.@f= LOG (tank.width.@f)
```

If `tank.width.@f` is 2.7182818, the function will return 1.000000.

Format

LOG10 (numeric)

Description

This function calculates the base 10 logarithm of the numeric argument.

Return Value

A real numerical constant.

Comments

The function will return a 0 if the argument is less than or equal to 0.

Example

```
IF base10.log.needed is TRUE  
THEN log.value.@f= LOG10 (tank.width.@f)
```

If `tank.width.@f` is 10.0, the function will return 1.0.

Format

MACRO (pattern)

Description

The function will fire all the rules with names specified by the pattern argument, and returns to the rule containing the function.

Return Value

None.

Comments

The pattern argument must be a rule name pattern.

Example

```
IF rules.fired.needed is TRUE  
THEN MACRO ("rule1*")
```

The function will fire all the rules with a name that start with "rule1".

Format

MAX (numeric, . . .)

Description

This function returns the maximum value of the numeric arguments.

Return Value

A real numerical constant.

Comments

The maximum number of numerical arguments for the function is 20.

Example

```
IF maximum.value.needed is TRUE  
THEN maximum.value.@f= MAX (tank.depth.@f, tank.height.@f,  
tank.width.@i)
```

Format

MIN (numeric, . . .)

Description

This function returns the minimum value of the numeric arguments.

Return Value

A real numerical constant.

Comments

The maximum number of numerical arguments for the function is 20.

Example

```
IF minimum.value.needed is TRUE  
THEN minimum.value.@f= MIN (tank.depth.@f, tank.heigh.@f,  
tank.width.@i)
```

Format

MINUTE (time)

Description

This function computes the hour from the time argument.

Return Value

An Integer numerical constant representing the minutes of the argument.

Comments

The time argument must be a time keyword-triplet.

Example

```
IF minute.value.needed is TRUE  
THEN minute.value.@i= MINUTE (delivery.time.@time)
```

If `delivery.time.@time` is 23hr 5min 10sec, the function will return 5.

Format

MODULUS (integer1, integer2)

Description

This function returns the remainder when integer1 is divided by integer2.

Return Value

An integer value representing the remainder.

Comments

integer2 cannot be zero.

Example

```
IF random.num.@long = RAND()  
THEN number.from_1_to_9.@i = MODULUS(random.num.@1, 9) + 1
```

This example generates a random number between 1 and 9.

Format

MONTH (date)

Description

This function extracts the month from the date argument.

Return Value

An Integer numerical constant representing the month of the argument.

Comments

The date argument must be a date keyword-triplet.

Example

```
IF month.value.needed is TRUE  
THEN month.value.@i = MONTH (delivery.date.@date)
```

If `delivery.date.@date` is Oct 7, 1991, the function will return 10.

Format

NOT (logical value)

Description

This function returns the logical negation of a logical value.

Return Value

The logical negation of a logical value. The return value is a logical value between 0(False) to 100(Ture).

Comments

FALSE is returned if input argument is invalid.

Example

IF both.possibilities to be examined is TRUE
THEN operation.process.not_ok is NOT (operation.process.ok)

Format

NOTKNOWN (triplet)

Description

This function tests to see if a keyword_triplet is notknown.

Return Value

TRUE if triplet is notknown, FALSE otherwise.

Comments

The function makes no attempt to instantiate the keyword-triplet.

Example

```
IF NOTKNOWN (tank.level.@f)  
THEN tank.level.@f= 13.45
```

Format

OBJ (triplet)

Description

This function extracts the object name of the keyword-triplet.

Return Value

A string constant containing the object name of the keyword-triplet.

Comments

OBJ () will be most useful when used in the embedded text of the ‘ask’ facet. If it is used in a facet, \$OAV can be used in place of the triplet.

Example

```
IF object.manipulation.needed is TRUE  
THEN object.name.@s = OBJ (tank.width.@f)
```

In this rule the OBJ function is passed the keyword triplet `“tank.width.@f”`. It returns the object name `“tank”` which is stored in the keyword-triplet `object.name.@s`.

Format

PAINT (string, string, T | F)

Description

This function will display a bitmap file.

Return Value

None.

Comments

The first argument is a string containing the bitmap file name.

The second argument is a string containing the picture's title.

The third argument is the user-acknowledge flag. Set this flag to TRUE if you want user's acknowledgement (by clicking on "REMOVE") before leaving this function. Set this flag to FALSE if user's acknowledgement is not required.

Example

```
IF graphic.display.needed is TRUE  
THEN PAINT ("tank1.bmp", "Tank #1", T)  
THEN PAINT ("tank2.bmp", "Tank #2", F)  
THEN PAINT ("tank3.bmp", "Tank #3", F)
```

The system will display the image stored in tank1.bmp and wait for the user's acknowledgement. Then the system will display the image stored in tank2.bmp and tank3.bmp without waiting for the user's acknowledgement.

Format

PAUSE (string)

Description

This function pauses the consultation session until the user acknowledges the dialog box.

Return Value

None.

Comments

The string argument will be displayed in the dialog box.

Example

```
IF user.acknowledgemnet.needed is TRUE  
THEN PAUSE (" Hit enter to continue")
```

The system will display the string argument in the dialog box and wait for the user's acknowledgement.

Format

POW (numeric, numeric)

Description

This function calculates the value of the first numeric argument raised to the power of the second numeric argument.

Return Value

A real numerical constant.

Comments

If the first argument is nonzero and the second argument is 0 the function

Example

If the first argument is 0 and the second argument is less than 0 the if the first argument is 0 and the second argument is 0 the function will return a 0.

If the first argument is less than 0 and the second argument is not an integer the function will return a 0.

IF power.value.needed is TRUE

THEN `power.value.@f`= POW (`tank.width.@f`, power.variable.@i)

If `tank.width.@f` is 30.0 and `power.variable.@i` is 2, the function will return 900.0.

Format

PUTPOINTDATA (pattern, numeric, numeric) only for use in Comdale/C and/or ProcessVision

Description

This function puts the value of the triplet into the point database with its associated degree of belief.

Return Value

TRUE (100) or FALSE (0)

Comments

The first argument is a pattern of the triplet variables to be updated in the point database.

These can include numeric, string or logical type triplets.

The second field contains the value or string to be assign to the triplet variable in the point data base.

The third string contains the certainty factor to be assigned to the triplet variable.

Example

```
IF TRUE  
THEN PUTPOINTDATA ("A1.value.@float", 25.00, 100)
```

For numeric triplets, a value and confidence factor must be specified.

```
IF TRUE  
THEN PUTPOINTDATA ("A1.string.@string", "No problem", 100)
```

For string triplets, a string and confidence factor must be specified.

```
IF TRUE  
THEN PUTPOINTDATA ("A1.value.steady", 100)
```

For logical triplets, a confidence factor must be specified.

Format

RAND ()

Description

The function generates a random integer in the range 0 to 32767.

Return Value

An integer numerical constant.

Comments

No argument is required.

Example

```
IF random.value.needed is TRUE  
THEN random.value.@i= RAND ( )
```

The function will generate an integer numerical constant.

Format

RATEOFCHANGE (triplet, t1, t2) only for use in Comdale/C and/or ProcessVision

Description

Calculates the rate of change of a variable between t1 and t2 seconds ago.

Arguments

t1 and t2 are in seconds

Triplet is a float type keyword triplet

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Return Value

A numerical value

Example

```
IF RATEOFCHANGE (tank.level.@float, 200, 400) > 0.2.  
THEN TEXT ("all is ok", "ok")
```

Format

READ (ID-triplet, triplet-string)

Description

Reads the value or certainty of the triplet contained in the triplet-string from the application associated with the numeric triplet via DDE (set using CONNECT).

Arguments

ID-triplet an @integer type triassociatedng the conversation identification number
Between the two tasks communicating via DDE (set using CONNECT function)
Triplet-string a triplet whose value is to be read from the connected application via DDE.
To read the external value as the certainty of the triplet prefix the triplet's name with: (CF)

Return Value

TRUE if successful

Example

```
IF local_update.of _DDE_link.is_required is TRUE  
THEN READ (my_link.id.@integer, "flow.rate.@float")
```

```
IF read.cf.from_DDE is TRUE  
THEN READ (my_dde_link.id.@integer, "(CF) flow.rate.@float")
```

In the first example the value of `flow.flow.rate.@f` is read from the application connected through the ID held in `DDE_conversation.id.@integer`.

In the second example the certainty of the triplet `flow.rate.@f` is read from the connected application through the ID held in `DDE_conversation.id.@integer`.

***NOTE:** before any READ () function is done the DDE link must be established using the CONNECT function.

Format

REDO ()

Description

This function will restart the consultation and forget the degree of certainty and value of each of the instantiated keyword-triplets.

Return Value

None

Example

```
IF consultation.restart.needed is TRUE  
THEN REDO ( )
```

The function will forget the degree of certainty and value of each of the instantiated keyword-triplets and restart the consultation.

Format

RELATIVECHANGE (triplet, t1, t2)

only for use in Comdale/C and/or ProcessVision

Description

Calculates the relative change of a variable between t1 and t2 seconds ago. The relative rate of change is the rate of change divided by the value of the variable at the start of the period.

Arguments

t1 and t2 are in seconds

Triplet is a float type keyword triplet.

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Return Value

A numerical value

Example

```
IF RELATIVECHANGE (tank.level.@float, 200, 400) > 0.02.  
THEN TEXT ("all is ok", "ok")
```

Format

RUN (string)

Description

This function runs a knowledge base that has already been loaded.

Comments

The string argument must contain a knowledge base name that has already been loaded.

The name cannot be the same name as the knowledge base being run.

You may use the 'load' function to load a knowledge base.

Return Value

None

Example

```
IF run.new.knowledge_base is TRUE
AND new.knowledge_base.loaded is TRUE
THEN RUN ("tank.knw")
```

```
IF run.new.knowledge_base is TRUE
AND new.knowledge_base.loaded is FALSE
THEN LOAD ("tank.knw")
THEN RUN ("tank.knw")
```

In the first rule the RUN function will run the knowledge base named "tank.knw"

In the second rule the LOAD function will load the knowledge base and then the RUN function will run the knowledge base named "tank.knw"

Format

RUN_CONTINUE (string)

Description

This function runs a previously loaded knowledge base and then returns to the location from where it was called.

Return Value

None.

Comments

The string argument must contain a previously loaded knowledge base name.
The knowledge base to be run cannot be the knowledge base currently being run.
Use the 'load' function to load a knowledge base.

Example

```
IF run.new.knowledge_base is TRUE  
AND new.knowledge_base.loaded is TRUE  
THEN RUN_CONTINUE ("tank.knw")
```

```
IF run.new.knowledge_base is TRUE  
AND new.knowledge_base.loaded is FALSE  
THEN LOAD ("tank.knw")  
THEN RUN_CONTINUE ("tank.knw")
```

In the first rule the RUN_CONTINUE function will run the knowledge base named "tank.knw".

In the second rule the LOAD function will load the knowledge base and then the RUN_CONTINUE function will run the knowledge base named "tank.knw".

Upon termination of the consultation with tank.knw, the function will return to the calling knowledge base.

Format

RUN_PROCEDURE (string)

Description

This function runs the procedure specified by the string argument.

Return Value

None.

Comments

The string argument must contain a valid procedure name.

Example

```
IF run.procedure.needed is TRUE  
THEN RUN_PROCEDURE ("proc1")
```

The function will run the procedure named 'proc1'.

Format

SAVESESSION (string)

Description

This function will create a record of questions asked, and responses given, during the consultation.

Return Value

None.

Comments

This string argument specifies the file to be created for recording the session.

Example

```
IF session.needs.to_be_saved is TRUE  
THEN SAVESESSION ("tank.sav")
```

The consultation session will be recorded and written to the file named 'tank.sav'.

Format

SECOND (time)

Description

This function computes the seconds from the time argument.

Return Value

An integer numerical constant representing the seconds of the argument.

Comments

The time argument must be a time keyword-triplet

Example

```
IF seconds.value.needed is TRUE  
THEN seconds.value.@i = SECOND (delivery.time.@time)
```

If [delivery.time.@time](#) is 23hr 5min 10sec, the function will return 10.

Format

SETPASSWORD (string)

Description

This function sets the password of a knowledge base.

Return Value

TRUE if password is set, FALSE otherwise.

Comments

Password must be 8 characters or less.

Put the password in double quotes.

To remove the password, use an empty string (“”).

Example

```
IF new.password.required is TRUE  
AND SETPASSWORD (new.password.@s)  
THEN TEXT (“New password is set.”, “Message”)  
ELSE TEXT (“Password remains unchanged.”, “Message”)
```

Format

SETREPORTPROCEDURE (procedure name)

Description

This function sets the report procedure to be the given procedure.

Return Value

TRUE if procedure is set, FALSE otherwise.

Comments

False is returned if the given procedure dose not exist.

Put the procedure's name in double quoted.

To remove the report procedure, use an empty string (“”).

Example

```
IF GETREPORTPROCEDURE ( ) is “ “  
THEN SETREPORTPROCEDURE (“report_proc”)
```

If report procedure is not set, then set it to report_proc.

Format

SETSTARTPROCEDURE (procedure name)

Description

This function sets the start procedure to be the given procedure.

Return Value

TRUE if procedure is set, FALSE otherwise.

Comments

False is returned if the given procedure dose not exist.

Put the procedure's name in double quoted.

To remove the start procedure, use an empty string (“”).

Example

```
IF TRUE  
THEN SETSTARTPROCEDURE (“start_proc_2”)  
THEN CYCLE ()  
THEN number.value.@f= STRING_TO_NUMBER (tank.width.@s)
```

If [tank.width.@s](#) is “10.0” (where “10.0” is a string constant), the function will return 10.0.

Format

SETSTARTRULE (rule name)

Description

This function sets the start rule to be the given rule.

Return Value

TRUE if rule is set, FALSE otherwise.

Comments

False is returned if the given rule dose not exist.

Put the rule's name in double quoted.

To remove start rule, use an empty string ("").

Example

```
IF TRUE  
THEN SETSTARTRULE ("Rule_2")  
THEN CYCLE ( )
```

Format

SETSTARTRULESET (ruleset name)

Description

This function sets the start rule set to be the given rule set.

Return Value

TRUE if rule set is set, FALSE otherwise.

Comments

“root” is a default rule set that contains all the rules in a knowledge base.

False is returned if the given ruleset dose not exist.

Put the ruleset’s name in double quoted.

To remove start ruleset, use an empty string (“”).

Example

```
IF GETSTARTRULESET ( ) is_not “root”  
THEN SETSTARTRULESET (“root”)
```

Format

SIN (numeric)

Description

This function calculates the sine of the numeric argument.

Return Value

A real numerical constant.

Comments

The numeric argument must be in radians.

Example

```
IF integral.of_cos.needed is TRUE  
THEN trig.argument.@f= ASIN (sin_of.trig_argument.@f)  
THEN integral_of.cos_a_x_dx.@f= SIN (constant.argument.@f*  
Trig.argument.@f) / constant.argument.@f
```

The keyword-triple `sin_of.trig_argument.@f` contains the sine of a variable. The ASIN function is used to obtain the angle in radians which is then assigned to the keyword-triplet “trig.argument.@f”. This keyword-triplet multiplied by a constant variable is used in the SIN function in the calculation of the integral.

Format

SINH (numeric)

Description

This function calculates the hyperbolic sine of the numeric argument.

Return Value

A real numerical constant.

Comments

The numeric argument must be in radians.

Example

```
IF integral_of_cosh_needed is TRUE  
THEN integral\_of\_cosh\_a\_x\_dx.@f = SINH (constant.argument.@f*  
Trig.argument.@f / constant.argument.@f)
```

The SINH function is used in calculating the integral.

Format

SLEEP (numeric)

Description

This function causes the inference engine to suspend for a period of time.

Return Value

None.

Comments

The numerical argument is the number of seconds you want to suspend the inference engine. Under multi-tasking OS , other tasks can still be active while the COMDALE's inference engine is suspended.

Example

```
IF how.long.@i
THEN TEXT ("I will sleep for !$ how.long.@i! Seconds now.")
THEN SLEEP (how.long.@i)
THEN TEXT ("I am awake.")
```

Format

SQRT (numeric)

Description

This function calculates the square root of the numerical argument.

Return Value

A real numerical constant.

Comments

The numeric argument must be positive.

If the numeric argument is negative the function will return a 0.

Example

```
IF square_root.value.needed is TRUE  
THEN square.root.@f= SQRT (tank.width.@f)
```

If `tank.width.@f` is 100.0, the function will return 10.0.

Format

STDDEV (numeric, ...)

Description

This function returns the standard deviation of the numeric arguments.

Return Value

A real numerical constant.

Comments

The maximum number of numeric arguments for the function is 20.

Example

```
IF standard.deviation.needed is TRUE  
THEN standard.deviation.@f= STDDEV (tank.depth.@double,  
tank.height.@float, tank.width.@integer)
```

If [tank.depth.@double](#) is 30.0, [tank.level.@float](#) is 10.0 and [tank.width.@integer](#) is 50, the function will return 20.

Format

STRCOMPARE (string1, string2)

Description

This function compares string1 and string2 lexicographically and returns a value indicating their relationship.

Return Value

a negative value if string1 < string2,
0 if string1 = string2,
or a positive value if string1 > string2.

Comments

strcompare is case-sensitive.
strcompare function is case-insensitive version of strcmp.

Example

```
IF 0 = strcmp (user.name.@s, master.name.@s)
THEN TEXT ("Same")
ELSE TEXT ("Different")
```

Format

STRCONCAT (string, string)

Description

This function concatenates two strings together.

Return Value

A concatenated string.

Comments

Original strings are not modified.

An empty (“”) string is returned in cases of error.

Example

IF TRUE

THEN `user.name.@s` is STRCONCAT (`first.name.@s`, “ “)

THEN `user.name.@s` is STRCONCAT (`user.name.@s`, `last.name.@s`)

Format

STRCOPY (string triplet, string)

Description

This function copies a string into a string keyword-triplet.

Return Value

TRUE if string is copied, FALSE otherwise.

Comments

The string argument can be anything that evaluates to a string, such as a string keyword-triplet, or functions that return a string.

Example

```
IF STRCOPY (temp.name.@s, user_1.name.@s)
AND STRCOPY (user_1.name.@s, user_2.name.@s)
AND STRCOPY (user_2.name.@s, temp.name.@s)
THEN TEXT ("OK")
ELES TEXT ("Not OK")
```

Format

STRICOMPARE (string1, string2)

Description

This function compares string1 and string2 lexicographically and returns a value indicating their relationship.

Return Value

a negative value if string1 < string2,
0 if string1 = string2,
or a positive value if string1 > string2.

Comments

stricmpare is case-insensitive.
stricmpare function is case-sensitive version of strcmpare.

Example

```
IF 0 = STRICOMPARE (user.name.@s, master.name.@s)
THEN TEXT ("Same")
ELSE TEXT ("Different")
```

Format

STRING_TO_NUMBER (string)

Description

This function will convert the string to a number.

Return Value

A real numerical constant.

Comments

The function will return a 0 if the string argument is not a valid numerical string.

Example

```
IF number.from_string.needed is TRUE  
THEN the.number.@f= STRING_TO_NUMBER (number.in_string.@s).
```

Format

STRLEN (string)

Description

This function will return the length of the string argument.

Return Value

An integer numerical constant representing the length of the string.

Example

```
IF string.length.needed is TRUE  
THEN string.length.@f= STRLEN (tank.color.@s)
```

If [tank.color.@s](#) is “red”, the function will return 3.

Format

STRLOWERCASE (string)

Description

This function returns a lower case version of a string.

Return Value

A lower case string.

Comments

The original string is not modified.

Example

IF convert_name.to.lower_case is TRUE
THEN [user.name.@s](#) is STRLOWERCASE ([user.name.@s](#))

Format

STRNCOMPARE (string1, string2, integer n)

Description

This function compares, at most, the first n characters of string1 and string2 lexicographically and returns a value indicating the relationship.

Return Value

a negative value if string1 < string2,
0 if string1 == string2,
or a positive value if string1 > string2.

Comments

strcmp is case-sensitive.
strcmp function is case-insensitive version of strcmp.

Example

```
IF 0 = STRNCOMPARE (user.name.@s, master.name.@s, 3)
THEN TEXT ("First 3 characters are the same")
ELSE TEXT ("First 3 characters are different")
```

Format

STRNCOPY (string triplet, string, integer)

Description

This function copies up to N characters from a string into a string keyword-triplet.

Return Value

TRUE if string is copied, FALSE otherwise.

Comments

The string argument can be anything that evaluates to a string, such as a string keyword-triplet, or functions that return a string.

Example

```
IF copy.n_characters.@i  
THEN STRNCOPY ( a.string.@s, "ABCDEF", copy.n_characters.@i)
```

Format

STRNICOMPARE (string1, string2, integer n)

Description

This function compares, at most, the first n characters of string1 and string2 lexicographically and returns a value indicating their relationship between the substrings.

Return Value

a negative value if string1 < string2,
0 if string1 = string2,
or a positive value if string1 > string2.

Comments

strcmp is case-insensitive.
strcmp function is case-sensitive version of strcmp.

Example

```
IF 0 = STRNICOMPARE (user.name.@s, master.name.@s, 5)
THEN TEXT ("First 5 characters are the same")
ELSE TEXT ("First 5 characters are different")
```

Format

STRUPPERCASE (string)

Description

This function returns a upper case version of a string.

Return Value

An upper case string.

Comments

The original string is not modified.

Example

```
IF convert_name.to.upper_case is TRUE  
THEN user.name.@s is STRUPPERCASE (user.name.@s)
```

Format

TAN (numeric)

Description

This function calculates the tangent of the numeric argument.

Return Value

A real numerical constant.

Comments

The numeric argument must be in radians.

If the function fails to compute the tangent of the numeric argument it will return a 0.

Example

```
IF integral.of_one_over_cos_squared.needed is TRUE
THEN trig.argument.@f= ATAN (tan_of.trib_argument.@f)
THEN integral_of.one_over_cos_squared_a_x_dx.@f= TAN
(constant.argument.@f* trig.argument.@f) / constant.argument.@f
```

The keyword-triplet `tan_of.trig_argument.@f` contains the tangent of a variable. The ATAN function is used to obtain the angle in radians which is then assigned to the keyword-triplet “trig.argument.@f”. This keyword-triplet multiplied by a constant variable is used in the TAN function to calculate the integral.

Format

TANH (numeric)

Description

This function calculates the hyperbolic tangent of the numerical argument.

Return Value

A real numerical constant.

Comments

The numeric argument must be in radians.

Example

```
IF integral.of_one_over_cosh_squared.needed is TRUE  
THEN integral_of.one_over_cosh_squared_a_x_dx.@f= TANH  
(constant.argument.@f* trig.argument.@f) / constant.argument.@f
```

The TANH function is used in calculating the integral.

Format

TEXT (string, string)

Description

This function display a text message in the text window.

Return Value

None.

Comments

The first string argument contains the text message.

In Comdale/X the second string argument contains the title of the text message. This argument is optional.

In Comdale/C and Process Vision the second argument is the message class through which the text message will be passed. This class must be configured in a .msg file which is loaded with mg_admin.

Example

```
Comdale/X
IF text.message.needed is TRUE
THEN TEXT ("tank is red in color", "tank")
```

The system will display the text message with the text title labeled "tank".

```
Comdale/C, Process Vision
TEXT ("Warning : Tank level # 5 too high – current readings show
!$ tank5.level_pv.@float $! ", alarm1")
```

This example puts out an alarm message through message class alarm1. All operator stations which are configured to receive message class alarm will see this message. This message also embeds a triplet value within !\$ \$! Delimiters – the contents of which will be evaluated at the time the message is passed.

Format

THRESHOLD ()

Description

This function returns the current confidence level of the system.

Return Value

A real numerical constant representing the system threshold.

Comments

No argument is required.

Example

```
IF confidence.level.needed is TRUE  
THEN confidence.level.@f= THRESHOLD ( )
```

If the system confidence level is 80.0, the function will return 80.0.

Format

TIMEAVERAGE (triplet, t1, t2) only for use in Comdale/C and/or ProcessVision

Description

Calculates the average value of a variable between t1 and t2 seconds ago.

Arguments

t1 and t2 are in seconds
triplet is a float type keyword triplet.

Return Value

A numerical value.

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Example

```
IF TIMEAVERAGE (tank.level.@float, 200, 400) > 20.6  
THEN TEXT ("all is ok", "ok")
```

Format

TIMEHIGHEST (triplet, t1, t2) only for use in Comdale/C and/or ProcessVision

Description

Returns the highest value of a variable between t1 and t2 seconds ago.

Arguments

t1 and t2 are in seconds
triplet is a float type keyword triplet.

Return Value

A numerical value.

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Example

```
IF TIMEHIGHEST (tank.level.@float, 200, 400) > 29.0  
THEN TEXT ("all is ok", "ok")
```

Format

TIMELOWEST (triplet, t1, t2) only for use in Comdale/C and/or ProcessVision

Description

Returns the lowest value of a variable between t1 and t2 seconds ago.

Arguments

t1 and t2 are in seconds
triplet is a float type keyword triplet.

Return Value

A numerical value.

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Example

```
IF TIMELOWEST (tank.level.@float, 200, 400) > 29.0  
THEN TEXT ("all is ok", "ok")
```

Format

UNKNOWN (triplet)

Description

This function tests to see if a keyword-triplet is unknown.

Return Value

TRUE if triplet is unknown, FALSE otherwise.

Comments

This function makes no attempt to instantiate the keyword-triplet.

Example

```
IF UNKNOWN (tank.level.@f)  
THEN TEXT (“tank.level.@f is unknown.”)
```

Format

UNLOAD (string)

Description

This function unloads the loaded knowledge base specified by the string.

Return Value

None.

Comments

The string argument must contain a previously loaded knowledge base name.
This frees memory for COMDALE/X and other applications.

Example

```
IF knowledge_base.unloading_needed is TRUE  
THEN UNLOAD ("tank.knw")
```

The function will unload the knowledge base named "tank.knw".

Format

USERLEVEL ()

Description

This function returns the current userlevel.

Return Value

An integer numerical constant representing the system userlevel.

Comments

No argument is required.

Example

```
IF user.level.needed is TRUE  
THEN user.level.@f = USERLEVEL ( )
```

If the system userlevel is 5, the function will return 5.

Format

VALUE (triplet)

Description

This function will extract the value token name of the keyword-triplet.

Return Value

An string constant that is the value token name of the triplet.

Comments

VALUE () will be most useful when used in the embedded text of the 'ask' facet. If used in a facet, \$OAV can be used in place of the keyword-triplet.

Example

```
IF value_name.manipulation.needed is TRUE  
THEN value.name.@s = VALUE (tank.width.@f)
```

In this rule the VALUE function is passed a keyword triplet "tank.width.@f". It returns the value name "@f" which is then stored in the keyword-triplet "value.name.@s".

Format

VALUEAT (triplet, t) only for use in Comdale/C and/or ProcessVision

Description

Query the value of the triplet at time t second ago.

Arguments

t is in seconds
triplet is a numerical triplet

Return Value

A numerical value of the triplet at time t second ago

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Example

```
IF VALUEAT (tank.level.@f, 60 ) > 1000  
THEN TEXT ("tank level is greater than 1000 1 minute ago", "tank")
```

Format

VARIANCE (numeric, ...)

Description

This function returns the variance of the numeric arguments.

Return Value

A real numerical constant.

Comments

The maximum number of numeric arguments for the function is 20.

Example

```
IF variance.value.needed is TRUE  
THEN variance.value.@f= VARIANCE (tank.depth.@double, tank.height.@float,  
tank.width.@integer)
```

If `tank.depth.@double` is 30.0, `tank.level.@float` is 10.0 and `tank.width.@integer` is 50, the function will return 400.

Format

WAIT (\$constant, pattern, numeric)

Description

This function will disallow a rule from firing or keyword-triplet(s) from being instantiated for a specified period of time.

Return Value

None.

Comments

The \$constant argument specifies how the pattern should be interpreted.

If the \$constant argument is #RULE the pattern will be interpreted as a rule pattern and all rules with rule names matching the pattern will be disallowed from being fired for the specified period of time.

If the \$constant argument is \$ALLPREM the pattern will be interpreted as a keyword-triplet pattern and all rules with keyword-triplets in their premises that match the pattern will be disallowed from being fired for the specified period of time.

If the \$constant argument is \$ALLCONCL the pattern will be interpreted as a keyword-triplet pattern and all rules with keyword-triplets in their conclusions that match the pattern will be disallowed from being fired for the specified period of time.

The pattern argument contains either a rule or keyword-triplet pattern.

The numeric argument must be in units of seconds.

Example

```
IF wait.rules.needed is TRUE
THEN WAIT ($RULE, "rule 1", 10)
THEN WAIT ($RULE, "change*", 20)
```

The function will disallow the rule with the name "rule" from firing for 10 seconds, also all rules that have names beginning with "change" will be disallowed for 20 seconds.

```
IF wait.rules.needed is TRUE
THEN WAIT ($ALLPREM, "tank.*.*", 10)
```

The function will disallow rules with keyword-triplets, have an object name "tank", in their premise from firing for 10 seconds.

```
IF wait.rules.needed is TRUE
THEN WAIT ($ALLCONCL, "tank.*.*", 10)
```

The function will disallow rules with keyword-triplets, having an object name "tank", in their conclusion from firing for 10 seconds.

Remember that the value of the wait time can also be any valid numeric triplet (@float, @integer, etc.)

Format

WEEKDAY (date)

Description

This function computes the weekday from the date argument.

Return Value

An integer numerical constant representing the weekday of the argument.

A return value of 1 represents Monday, 2 represents Tuesday and so on.

Comments

The date argument must be a date keyword-triplet.

Example

IF weekday.value.needed is TRUE

THEN `weekday.value.@i` = WEEKDAY (`delivery.date.@date`)

If `delivery.date@date` is Feb 10, 1991 (a Sunday), the function will return 7.

Format

WHEN (triplet)

only for use in Comdale/C and/or ProcessVision

Description

Returns the absolute time (same as a time variable, @time) when this triplet last changed its value.

Arguments

Triplet is a keyword-triplet.

Return Value

A numeric value in seconds which is should be mapped to a variable of type @time.

Comments

To use this function correctly you must have a historical database (hd_dbase) running and have all the points in the conditional configured in that database.

Example

```
IF WHEN (tank.level.@float) > 0  
THEN TEXT ("tank level is ok", "ok")
```

Format

WRITE (numeric triplet, string)

Description

Writes the value or certainty of the triplet contained in the string to the application associated with the numeric triplet (set using CONNECT).

Arguments

numeric triplet	an @integer type triplet holding the conversation identification number between the two tasks communicating via DDE.
String	a triplet whose value is to be sent to the connected application via DDE. To send the certainty of the triplet prefix the triplets name with : (CF)

Return Value

TRUE if successful

Example

```
IF update.of DDE_link.is_required is TRUE  
THEN WRITE (DDE_conversation.id.@integer, "flow.rate.@float")
```

```
IF write.cf.to_DDE is TRUE  
THEN WRTIE (DDE_conversation.id.@integer," (CF) flow.rate.@float")
```

In the first example the value of `flow.rate.@f` is written to the application connected through the ID held in `DDE_conversation.id.@integer`.

In the second example the certainty of the triplet `flow.rate.@f` is written to the connected through the ID held in `DDE_conversation.id.@integer`.

*NOTE: Before any WRITE () function is done the DDE link must be established using the CONNECT function.

Format

YEAR (date)

Description

This function extracts the year from the date argument.

Return Value

An integer numerical constant representing the year of the argument.

Comments

The date argument must be a date keyword-triplet.

Example

```
IF year.value.needed is TRUE  
THEN year.value.@i= YEAR (delivery.date.@date)
```

If `delivery.date.@date` is Feb 10, 1995, the function will return 1995.

Format

YEARDAY (date)

Description

This function computes the year-day from the date argument.

Return Value

An integer numerical constant representing the year-day of the argument.

Comments

The date argument must be a date keyword-triplet.

Example

```
IF year.day.needed is TRUE  
THEN year.day.@i = YEARDAY (delivery.date.@date)
```

If [delivery.date.@date](#) is Feb 10,1995, the function will return 95.

15.13 SYSTEM VARIABLES

System Variables are reserved objects which have special meaning to the inference engine. There variables are usually used in expressions when building a knowledge base.

List of System Variables

The following is a list of all system variables.

\$Agoal	- internal use.
\$Allconcl	- used in wait () and ignore () functions
\$AllPremc	- used in wait () and ignore () functions
\$AllRules	- internal use
\$AlphaOrder	- internal use
\$Any	- internal use
\$AnyGoal	- internal use
\$Avg	- internal use
\$BackwardChain	- internal use
\$BestGoal	- internal use
\$BreathFirst	- internal use
\$CaseSensitive	- internal use
\$Cf	- returns the certainty factor of a conclusion statement
\$Current_date	- returns the current date.
\$Current_time	- returns the current time.
\$DateSource	- internal use
\$DepthFirst	- internal use
\$Disable	- internal use
\$Dt	- returns the degree of truth of the current rule premise
\$Enable	- internal use
\$First	- internal use
\$ForwardChain	- internal use
\$HightCf	- internal use
\$IfChange	- internal use
\$LowCf	- internal use
\$LowPri	- internal use
\$Max	- internal use
\$Min	- internal use
\$Ndt	- returns the net degree of truth of the current rule
\$NoNewGoal	- internal use
\$NoPossibleChaining	- internal use
\$NoPossibleGoal	- internal use
\$OAV	- represents the current keyword triplet used in facets to substitute for the current triplet
\$Rule	- internal use
\$SpecificRule	- internal use
\$SpecificTriplets	- internal use
\$Status	- internal use
\$Trip	- internal use