Use of Formulas

- Formula Usage, on page 1
- Formula Example, on page 1
- Variables, on page 1
- Operators, on page 10
- Built-in Functions, on page 13
- Custom Functions, on page 16

Formula Usage

You can use formulas in many routing nodes to both categorize contacts and select routing targets.

A formula consists of one or more expressions that Packaged CCE evaluates to produce a value that it can use for subsequent script processing. You define expressions—made up of variables, constants, operators, and functions—as part of custom selection rules or distribution criteria in scripts.

Formula Example

This is an example of a simple formula:

```
CallerEnteredDigits == 1
```

In this example:

- The left value, *CallerEnteredDigits*, is a variable. More specifically, it is a call control variable.
- The operator is the "Equal To" equality operator.
- The right value is the number 1.

If the value of *CallerEnteredDigits* is 1, the formula returns true; otherwise, the formula returns false.

Variables

A variable is a named object that holds a value. You use variables in formulas to select targets and help in call tracking.

Variable Syntax

Following is the syntax for using a variable in a formula:

object-type.object-name.variable-name

Where:

- The object-type is an object category, such as SkillGroup.
- The object-name is the name of an object contained in Packaged CCE database, such as the name of a skill group (for example, BosSales).
- The variable-name is the name of an object that can hold a value, such as call information for the skill group; for example, (CallsInProgress).
- Each component in the variable is separated by a period (.).

Note Passing of internationalized characters through Media Routing interface is not supported. The application that interacts with ICM through the Media Routing interface must send any call related data in English only.

Single-Target Variables

A single-target variable examines data for one specified routing target. For example, the variable:

SkillGroup.BosSales.CallsInProgress: Examines the number of calls in progress for the BosSales skill group.

Multiple-Target Variables

A multiple-target variable examines data across multiple routing targets. For example, the function:

Max(SkillGroup..LongestAvailable)*: Finds the skill group, from all skill groups defined in the target set for the script node that calls the function, with the longest available agent.

You use an asterisk (*) as the object-name value to indicate that the variable is to examine data across multiple targets.

Call Control Variables

Call control variables provide information about the current contact that is being routed by the script. Call control variables include information about where the route request came from, contact classification data, and data to be passed to the peripheral that receives the contact.

Variable	Data Type	Description	Can be Set by the User
CallerEnteredDigits	String	Digits caller entered in response to prompts.	Yes
CallingLineID	String	Billing telephone number of the caller.	No
DialedNumberString	String	Telephone number dialed by the caller.	No

Variable	Data Type	Description	Can be Set by the User
ExpCallVarName	String	Expanded Call Context (ECC) variable value assigned in scripts and passed with contact.	Yes
PeripheralVariable1- PeripheralVariable10	String	Values passed to and from the peripheral.	Yes
RequeryStatus	Integer	Provides the ability to test the error path of the Label, Queue, RouteSelect, and Select nodes to determine the specific network cause of failure and conditionally retry the attempt as necessary.	No
RouterCallDay	Integer	An encoded value that indicates the date on which Packaged CCE processes the call.	No
RouterCallKey	Integer	A value that is unique among all calls Packaged CCE has processed since midnight. RouterCallDay and RouterCallKey combine to form a unique call identifier.	No
RoutingClient	String	The name of the routing client that made the route request.	No
TimeInQueue	Integer	Number of seconds a call has been queued.	No
UserToUserInfo	String	ISDN private network User to User information	Yes
VruStatus	Integer	Indicates the result of a previous VRU node.	No
CallGUID	varchar (32)	Globally unique call identifier.	No
LocationParamName	varchar(50)	Location name.	No
PstnTrunkGroupID	varchar(32)	The Trunk Group ID on which the call arrived on IOS Gateway.	No
PstnTrunkGroupChannelNumber	Integer	The Trunk Group Channel Number on which the call arrived on IOS Gateway.	No
SIPHeader	varchar(255)	Specific header information extracted from a SIP call that arrives at Unified CVP (or VRU).	Yes

Note The Call Variables can be used in a "SET" node in an Admin Script as temporary placeholders for complex calculation. However, because any call context is only existent as long as the call itself, the Variables cease to exist after the Route Request (a.k.a Call) is complete (be it by virtue of a successful Routing Script Execute Completion or an Administrative Script Execute Completion). They cannot be used to store values, so as to be re-used in Routing Scripts, as the Routing Scripts themselves will have a new set of CallVariables created for the Route Request.

Note When comparing two Call Variables of Numeric string, you must use the Built-In Function "value()" in the IF Node to perform Numeric comparison, otherwise there is a String comparison. Example: value(Call.PeripheralVariable1)>=value(Call.PeripheralVariable2) where Call.PeripheralVariable1 and Call.PeripheralVariable2 are given as Numeric string.

Expanded Call Variables

Expanded call variables store values associated with the contact.

Expanded call values are written to Termination Call Detail records only if, and when, an ECC value is explicitly set. You can set the variables in several ways, such as using a script, a Unified CVP, CTI, and so on. This applies to null values as well as non-null values.

If an expanded call variable is defined, but never assigned a value, it does not have a row in the Termination Call Variable table when a Termination Call Detail record is written.

The Latin 1 Character set is supported for expanded call context variables and peripheral call variables when used with Unified CVP, Cisco Finesse, and Cisco SocialMiner, among others.

The use of multi-byte character sets is also supported in limited usage for ECC and peripheral call variables, when:

- Setting them in the Script Editor using double quotes.
- Stored in Termination Call Variables with an appropriate SQL collation.
- Setting and receiving them through CTI OS desktops.

Expanded call values are generally passed from leg to leg on the call. After a value is assigned, the value is recorded in the Termination Call Variable for every Termination Call Detail Segment. However, this depends on how each new call segment is created.

The solution comes with some predefined expanded call variables. You can create others througn the Unified CCE Administration tool.

ECC Payloads

You can define as many ECC variables as necessary. But, you can only pass 2000 bytes of ECC variables on a specific interface at any one time. To aid you in organizing ECC variables for specific purposes, the solution has *ECC payloads*.

An ECC payload is a defined set of ECC variables with a maximum size of 2000 bytes. You can create ECC payloads to suit the necessary information for a given operation. You can include a specific ECC variable in multiple ECC payloads.The particular ECC variables in a given ECC payload are called its *members*.

Note For ECC payloads to a CTI client, the size limit is 2000 bytes plus an extra 500 bytes for the ECC variable names. Unlike other interfaces, the CTI message includes ECC variable names.

In certain cases, mainly when using APIs, you might create an ECC payload that exceeds the CTI Server message size limit. If you use such an ECC payload in a client request, the CTI Server rejects the request. For an OPC message with such an ECC payload, the CTI Server sends the message without the ECC data. In this case, the following event is logged, "CTI Server was unable to forward ECC variables due to an overflow condition."

You can use several ECC payloads in the same call flow, but only one ECC payload has scope at a given moment. TCDs and RCDs record the ID of the ECC payload that had scope during that leg of the call. The *Call.ECCPayloadID* variable contains the ID of the ECC payload which currently has scope.

In solutions that only use the default ECC payload, the system does not create an ECC variable that exceeds the 2000-byte limit for an ECC payload or the 2500-byte CTI Message Size limit.

Note Packaged CCE 2000 Agent deployment allows you to use only the default ECC payload for the Network VRU.

If you create another ECC payload, the system no longer checks the 2000-byte limit when creating ECC variables. The system creates the ECC variables without assigning them to an ECC payload. Assign the new ECC variable to an appropriate ECC payload yourself through the ECC Payload Tool.

You can create and modify ECC payloads in the **Configuration Manager** > **List Tools** > **Expanded Call Variable Payload List** tool. In Packaged CCE 4000 Agent and 12000 Agent deployments, you can assign an ECC payload to Network VRU using the Network VRU Explorer tool in Configuration Manager.

Default ECC Payload

The solution includes an ECC payload named "Default" for backward compatibility. If your solution does not require more ECC variable space, you only need the Default payload. The solution uses the Default payload unless you override it.

If your solution only has the Default payload, the solution automatically adds any new ECC variables to the Default payload until it reaches the 2000-byte limit.

Note You cannot delete the Default payload. But, you can change its members.

Important During upgrades, when the system first migrates your existing ECC variables to the Default payload, it does not check the CTI message size limit. The member names might exceed the extra 500 bytes that is allocated for ECC payloads to a CTI client. Manually check the **CTI Message Size** counter in the **Expanded Call Variable Payload List** tool to ensure that the Default payload does not exceed the limit. If the Default payload exceeds the limit, modify it to meet the limit.

In a fresh install, the Default payload includes the predefined system ECC variables. In an upgrade, the Default payload's contents depend on whether the starting release supports ECC payloads:

- **ECC payloads not supported**—During the upgrade, a script adds your existing ECC variables to the Default payload.
- **ECC payloads are supported**—The upgrade brings forward the existing definition of your Default payload.

Note If your solution includes PGs from a previous release that does not support ECC payloads, the Router always sends the Default payload to those PGs. Those PGs can properly handle the Default payload.

ECC Payload Node

The **ECC Payload** node is available from the **General** tab on the **Object Palette**:

Figure 1: Payload icon

Use this node to change the ECC payload that has scope for the following part of your script. Once you select an ECC payload, it has scope for all non-VRU operations until changed. You can select the ECC payload either statically or dynamically by the payload's EnterpriseName or ID.

Persistent vs. Non-persistent Call Variables

When Packaged CCE writes call data records to its historical database, it can store the values of all call variables. Storing excessive call variable data can degrade historical database performance. When you define a call variable (using the Expanded Call Variables gadget in the Unified CCE Administration web tool), you can tag it as either *persistent* or *non-persistent*. Only persistent call variables are written to the historical database. You can use non-persistent variables in routing scripts, but they are not written to the database.

User Variables

User variables are variables you create to serve as temporary storage for values you can test with an **If** node. For example, you could create a user variable called usertemp to serve as a temporary storage area for a string value used by an If node.

You create user variables through the Unified CCE Administration tool.

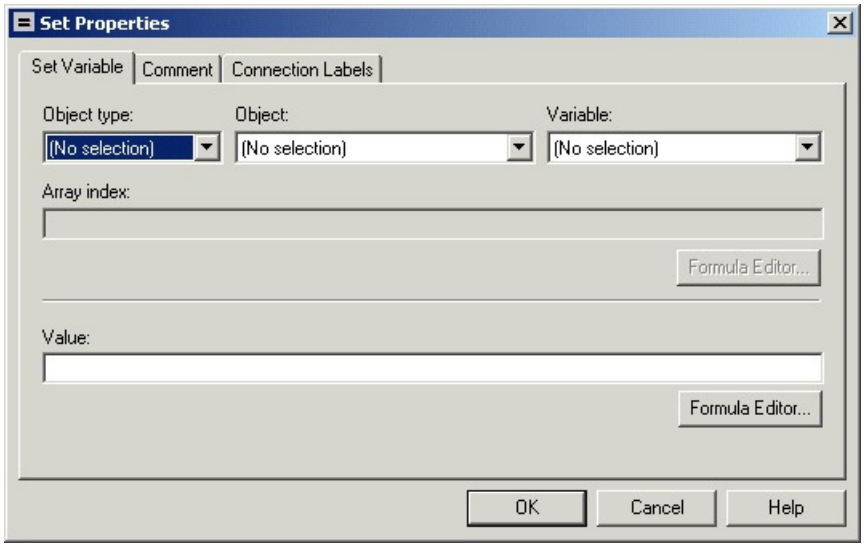
Each user variable must:

- Have a name that begins with user.
- Be associated with an object type, for example, skill group. (This enables the Packaged CCE to maintain an instance of that variable for each object of that type in the system.)
- Store a value up to 40 characters long.

After you have define a variable, you can use the Formula Editor to access the variable and reference it in expressions, just as you would with a built-in variable.

Set Variable Node Usage

Figure 2: Set Properties Window



You can set the value of a variable with the Set Variable node:

- Object type - Choose the type of object the variable is associated with.
- Object - Choose the specific object the variable is associated with.

Note If you choose Call as the Object Type, this field does not apply.

- Variable - The specific variable you want to set.

Note The variables that are available are determined by the value you choose in the Object Type field.

Note Define all integer fields in tables accessed by a Set Variables node as NOT NULL.

- Array index - Enter an integer or an expression that evaluates to an integer. For example, if the Array Index expression evaluates to 2, then the Set Variable node sets the second element of the variable array.

Note This field is only available if you select an array variable in the Variable field.

- Value - Enter the value to assign to the variable. The value can be:
  - A constant
  - A reference to another variable
  - An expression

SkillGroup.Avail and SkillGroup.ICMAvailable Variables

When the Packaged CCE system includes only the voice channel, the value of the SkillGroup.Avail variable is the number of agents in the available state, meaning that the agents are able to accept new calls.

However, when the web or e-mail channel is used with non-voice Media Routing Domains and agents log in to multiple domains, the value of the SkillGroup.Avail variable is calculated differently. There is also a SkillGroup.ICMAvail variable.

The following table describes the difference between the SkillGroup.Avail and the SkillGroup.ICMAvail variables:

Case	SkillGroup.Avail	SkillGroup.ICMAvailable
Only voice domain is used	Number of agents in the Available state.	Same
Multiple Domains are used	Number of agents in the Available state, regardless of what they may be doing in this or other domains.	Number of agents who can actually handle an additional task or call in the domain.

SkillGroup.ICMAvailable Variable

The value of the SkillGroup.ICMAvailable variable is the actual number of agents logged in to the skill group who can take new calls or tasks. Such agents must meet all the following criteria:

1. They are routable in the domain.
2. The agent's state in the domain is something other than "Not-Ready".
3. The agent is below the maximum task limit.

Note For most domains (that is, if the agent is not a Enterprise Chat and Email Multi-session agent), the maximum task limit is 1, and an agent is below the maximum only when the agent is not working on any call or task.

4. The agent is not working on another task in a non-interruptible domain.

SkillGroup.Avail Variable

SkillGroup.Avail is the number of agents in the skill group who are not doing anything in the domain. An agent who is logged in to two domains can be counted as Avail in one domain even though that agent is handling a task in another non-interruptible domain. An agent in a domain that handles multiple tasks (such

as chat) is not counted as Avail if that agent is handling a task, even though the agent has additional capacity for more tasks.

The following table shows some possible values for these variables. Assume three agents are logged in to a voice skill group, and the same three agents are also logged in to another non-interruptible domain, such as a chat domain. This table shows the voice skill group states and the number of agents available in that state.

Case	SkillGroup.Avail	SkillGroup.ICMAvailable
Initial state	3	3
First agent handles a call	2	2
Second agent handles a chat session	2 (because there are two agents doing nothing in the domain)	1 (because there is only one agent left to handle voice calls)
Voice call ends	3	2
Chat ends	3	3

If a routing script needs to check the number of available agents, using SkillGroup.Avail produces effective results as it uses an extrapolation mechanism in determining the available agent.

Following is another example showing agents handling non-interruptible chat tasks. Assume three agents are logged in to a chat skill group, each allowed to handle two chats. This table shows states for the chat skill group.

Case	SkillGroup.Avail	SkillGroup.TalkingIn	SkillGroup.ICMAvailable
Initial state	3	0	3
First agent handles a chat session	2 (because the agent is now in the talking state)	1	3 (because all three agents can still handle additional chats)
Second agent handles a chat session	1	2	3
Third agent handles a chat session	0	3	3
First agent handles second chat session	0	3 (even though a total of 4 chats are in progress, only 3 agents are doing the work)	2 (because only the second and third agents can handle an additional chat)

By default, Script Editor shows the ICMAvailable value instead of Avail value when displaying skill group real-time data.

Closed Variables

Closed variables are available for use for skill groups, peripherals, and Media Routing Domains. Closed variables allow administration scripts to turn dequeuing to these objects on and off. The Closed variables

default to 0, meaning that the object is open. A script (usually an administration script) can change the state of the Closed variables.

If a Closed flag is set to a non-zero integer, then calls are not dequeued to affected agents, regardless of their state.

When closed variables are set to zero, the queued calls do not go to the available agents immediately, and continue to be in the queue. When the agent state changes from "Not Ready" to "Ready" state, the new calls are sent to the available agents (agents in the "Ready" state) only, and not the queued calls.

Operators

Operator Precedence

The following table shows the order in which operators are evaluated.

Note The operators with priority 1 are evaluated first, then those with priority 2, and so on. The order of evaluation within each priority level can also be important. Prefix operators are evaluated from right-to-left in an expression. Assignment operators are also evaluated from right-to-left. In all other cases where operators have equal priority, they are evaluated left-to-right.

Priority	Operator type	Operators
1	Prefix (unary)	+ - ! ~
2	Multiplication and division	* /
3	Addition and subtraction	+ -
4	Shift right and shift left	>> <<
5	Relational	< > <= >=
6	Equality	== !=
7	Bitwise And	&
8	Bitwise exclusive Or	^
9	Bitwise inclusive Or	|
10	And	&&
11	Or	||
12	Conditional	?
13	Sequential	,

Prefix Operators

The Prefix Operators in the following table take a single operand:

Operator	Meaning	Comments/Examples
+	Positive	Numeric values are positive by default, so the positive operator (+) is optional. Example: 2 and +2 represent the same value.
-	Negative	The negative operator (-) changes the sign of a value. Example: 2 represents a positive value; -2 represents a negative value.
!	Logical negation	A logical expression is any expression that evaluates to true or false. The logical negation operator (!) changes the value of a logical expression. Note: Numerically, a false value equates to 0 and a true value equates to a non-zero value. Example: If the current value of SkillGroup.Sales.Avail is 3, then SkillGroup.Sales.Avail > 0 is true and (SkillGroup.Sales.Avail > 0) is false.
~	One's complement	Operates on a bit value, changing each 1 bit to 0 and each 0 bit to 1. Note: This operator is rarely used.

Arithmetic Operators

The Arithmetic Operators in the following table take two operands:

Operator	Meaning	Comments/Examples
*	Multiplication	Arithmetic operators perform the basic operations of addition, subtraction, multiplication and division. You can use them in making calculations for a skill group, service, or route. Note: Multiplication (*) and division (/) operators are evaluated before addition (+) and subtraction (-) operators.
/	Division	
+	Addition	
-	Subtraction	

Equality Operators

The Equality Operators in the following table take two operands:

Operator	Meaning	Comments/Examples
==	Equal to	Equality operators allow you to determine whether two values are equivalent or not.
!=	Not Equal To	

Relational Operators

The Relational Operators in the following table take two operands:

Operator	Meaning	Comments/Examples
>	Greater than	Relational operators allow you to perform a more sophisticated comparison than the equality operators.
<	Less than	
>=	Greater Than or Equal To	
<=	Less Than or Equal To	

Logical Operators

The Logical Operators in the following table take two operands. Logical operators examine the values of different logical expressions:

Operator	Meaning	Comments/Examples
&&	And	The expression is true if both of the operands are true. If either is false, the overall expression is false.
||	Or	The expression is true if either or both of the operands is true. If both are false, the overall expression is false.

Note The equality (==) and relational (>) operators are evaluated before the logical operators (&& and ||).

Bitwise Operators

The Bitwise Operators in the following table take two operands.

Operator	Meaning	Comments/Examples
&	And	The & Bitwise Operator turns specific bits in a value on or off.
|	Inclusive Or	Inclusive Or and Exclusive Or differ in the way they handle the case where bits in both values are 1: Inclusive Or evaluates the result as true and sets a 1 bit in the result. Exclusive Or evaluates the result as false and sets a 0 bit in the result. (An Exclusive Or applies the rule "one or the other, but not both").
^	Exclusive Or	

Miscellaneous Operators

The following table lists miscellaneous operators:

Operator	Meaning	Comments/Examples
?	Conditional	The conditional operator (?) takes three operands and its syntax is as follows: The Packaged CCE evaluates the expression by first examining the logical expression condition and then tests the following condition: If the result is true, then the overall expression evaluates to the value of the expression true-result. If the result is false, then the overall expression evaluates to the expression false-result.
&	Concatenation	The concatenation operator (&) joins two strings end-to-end. returns the value.
,	Sequential	The sequential or comma operator (,) takes two operands, each of which is an expression. Packaged CCE evaluates the left expression first and then the right expression. The value of the overall expression is the value of the right expression. The first expression typically affects the valuation of the second.
<< >>	Shift left Shift right	The shift left (<<) and shift right (>>) operators shift the bits within a value.

Built-in Functions

Date and Time Functions

The following table lists the built-in date and time functions:

Function	Data Type	Return Value/Example
date [(date)]	Integer	Returns the current system date or the date portion of a given date-time value. The given date can be a floating point value (as returned by the now function), a string of the form mm/dd/yy, or three integers: yyyy, mm, dd. date (with no arguments) returns the current date. For example, = date (2001, 7, 15) tests whether the current date is July 15, 2001. **Note** Do not use the slash (/) character in defining a date function. Because it is the division operator, the function would not return the results you are looking for. You can enclose the argument within a string.
day [(date)]	Integer	Returns the day of month (1-31) for the current date or a given date. The given date must be an integer or a floating-point value, as returned by the date or now function.
hour [(time)]	Integer	Returns the hour (0-23) of the current time or a given time. The given time must be a floating-point value, as returned by the now function.
minute [(time)]	Integer	Returns the minutes (0-59) of the current time or a given time. The given time must be a floating-point value as returned by the time function.

Function	Data Type	Return Value/Example
month [(date)]	Integer	Returns the month (1-12) of the current month or a given date. The given date must be a floating-point value, as returned by the date or now function.
now	Float	Returns the current date and time, with the date represented as an integer and the time represented as a fraction. Note: You can use the date or time functions without any arguments to return just the current date or time. This function is useful for comparing the current date and time to a specific point in time.
second [(time)]	Integer	Returns the seconds (0-59) of the current time or a given time. The given time must be a floating-point value, as returned by the time function.
time [(time)]	Float	Returns the current system time or the time portion of a date-time value. The given time can be a floating point value, a string of the form hh:mm:ss, or two or three numeric values: hh, mm [, ss]. (with no arguments) returns the current time. For example, = time (20:05:00) tests whether the current time is 08:05:00
weekday [(date)]	Integer	Returns the current day of week (Sunday=1, Monday=2, etc.) of the current date or given date. The given date must be an integer or floating-point value, as returned by the date or now function.
year [(date)]	And	Returns the year of the current year or given date. The given date must be a floating-point value, as returned by the date or now function.

Mathematical Functions

The following table lists the built-in mathematical functions:

Function	Data Type	Return Value/Example
abs(n)	Floating Point or Integer	Returns the absolute value of n (the number with no sign).
max(n1, n2 [,n3] . . .)	Floating Point or Integer	Returns the largest of the operands. Each operand must be numeric.
min(n1, n2 [,n3] . . .)	Integer	Returns the smallest of the operands. Each operand must be numeric.
mod(n1,n2)	Floating Point or Integer	Returns the integer remainder of n1 divided by n2.
random()	Floating Point or Integer	Returns a random value between 0 and 1.
sqrt(n)	Floating Point or Integer	Returns the square root of n. (The operand n must be numeric and non-negative).

Function	Data Type	Return Value/Example
trunc(n)	Floating Point or Integer	Returns the value of n truncated to an integer.

Miscellaneous Functions

The following table lists the built-in miscellaneous functions:

Function	Data Type	Return Value/Example
after(string1,string2)	String	That portion of string2 following the first occurrence of string1. If string1 does not occur in string2, the null string is returned. If string1 is the null string, string2 is returned.
before(string1,string2)	String	That portion of string2 that precedes the first occurrence of string1. If string1 does not occur in string2, string2 is returned. If string1 is the null string, the null string is returned.
CallTypeSurvey	String	Returns the format required for CVP to run the post call survey.
ClidInRegion	Logical	Indicates whether the CLID for the current contact is in the geographical region specified by string. The value string must be the name of a defined region. You can use the Name variable of a region to avoid entering a literal value.
concatenate(string1,string2, . . .)	String	Returns the concatenation of the arguments. The function takes up to eight arguments.
EstimatedWaitTime	Integer	Returns the minimum estimated wait time for each of the queues against which the call is queued (skill group(s) or precision queue(s)). Queue to Agent(s) is not supported. If no data is available, returns -1. The estimated wait time is calculated once, when the call enters the queue. The default estimated wait time algorithm is based on a running five minute window of the rate of calls leaving the queue. Any calls which are routed or abandoned during the previous 5 minutes are taken into account as part of the rate leaving queue. For precision queues, the rate leaving queue represents the rate at which calls are delivered or abandoned from the entire precision queue, not any individual precision queue steps.
find(string1, string2 [,index])	Integer	Returns the starting location of string1 within string2. If you specify an index value, searching starts with the specified character of string2.

Function	Data Type	Return Value/Example
if(condition,true-value,false-value)	Logical	Returns a value of true-value if the condition is true; false-value if the condition is false. Returns the current hour in 12-hour format rather than 24-hour format.
isPickPullRequest()	Logical	Whether the current service requested is for pick or pull type.
isPickPullRequest()	Logical	Whether the current service requested is for pick or pull type.
left(string,n)	String	Returns the left-most n characters of the string.
len(string)	Integer	Returns the number of characters in the string.
mid(string,start,length)	String	Returns a substring of the string, beginning with the specified start character and continuing for the specified number of characters.
result	Floating Point or Integer	Returns the result of the current Select node. (This function is valid only in a Select node.) If you are using the LAA rule in the Select node, the result function returns the number of seconds the selected agent has been available.
right(string,n)	String	Returns the right-most n characters of the string.
substr(string,start [, length])	String	Returns a substring of the string, beginning with the specified start character and continuing for the specified number of characters.
text(n)	String	Converts a numeric value into a string.
valid(variable)	Logical	Returns whether the variable has a valid value.
ValidValue(variable,value)	String	If the variable has a valid value, returns that value; otherwise, returns "value". Returns either a name from the database or the string value None.
value(string)	Floating Point or Integer	Converts a string into a numeric value.

Custom Functions

Custom functions are those functions you create for use within scripts, as opposed to built-in functions.

Add Custom Functions

Procedure

Step 1 In Script Editor, from the **Script** menu, choose **Custom Functions**. The Custom Functions dialog box opens, listing all the custom functions currently defined.

Step 2 Click **Add** to open the Add Custom Function dialog box.

Step 3 Specify the following:

a) Function name. All custom function names must begin with user.

b) Number of Parameters. The number of parameters to be passed to the function. A function may take 0, 1, or more parameters.

c) Function definition. The expression to be evaluated when the function is called. When entering the function definition, keep the following in mind:

The parameters to a function are numbered beginning with 1. To reference a parameter within the expression, surround it with percent signs (%). For example, %3% is a reference to the third parameter.

The lower portion of the dialog box is just like the Formula Editor. You can use it to help build the expression.

Step 4 When finished, click **Test**. The Test Function dialog box opens.

Step 5 Test the function by entering an example of how you might reference the function. Include a specific value for each parameter.

Step 6 Click **Evaluate** to see how the Script Editor interprets the function call and click **Close** to return to the Add Custom Function dialog box.

Step 7 Use one of the Validate buttons to validate the scripts that reference a selection function. (The Validate All button lets you validate all the scripts that reference any custom function.)

Step 8 When finished, click **OK** to apply changes and to close the dialog box.

Import Custom Functions

Procedure

Step 1 In Script Editor, from the **Script** menu, choose **Custom Functions**. The Custom Functions dialog box opens, listing all the custom functions currently defined.

Step 2 Click **Import**. The Import Custom Function dialog box opens.

Step 3 Choose a file name with an ICMF extension (.ICMF) and click **Open**. The Script Editor examines the file for naming conflicts. If a conflict is found, a dialog box appears listing options for resolving the conflict.

Step 4 Choose one of the options and click **OK**.

Note If you choose to rename the function, the new name must begin with user.

The Script Editor performs automapping and the following happens:

- If all imported objects were successfully auto-mapped, a message window appears prompting you to review the mappings. Click **OK** to access the Object Mapping dialog box.
- If some imported objects were not successfully auto-mapped, the Object Mapping dialog box appears, with all unmapped objects labeled Unmapped.

The Object Mapping dialog box contains three columns:

- Object Types. The type of imported objects.
  - Imported Object. Name of imported object.
  - Mapped To. What this imported object will be mapped to.
- (Optional.) Click an Imported Object value. The Mapped To column displays all the valid objects on the target system.
- (Optional.) Choose an object from the Mapped To columns drop-down list on the target system that you want to map the imported object to.

Note Multiple objects may be mapped to the same target. Objects may be left unmapped; however, the resulting custom function are not valid until all objects are mapped.

When the mapping is complete, click **Apply** and **Finish**.

Export Custom Functions

Procedure

Step 1 In Script Editor, from the **Script** menu, choose **Custom Functions**. The Custom Functions dialog box opens, listing all the custom functions currently defined.

Step 2 Choose the custom function(s) from the list and click **Export**. The Export Custom Function dialog box opens.

Note If you selected a single function, that functions name appears in the File Name field. If you selected more than one function, the File Name field is blank.

Step 3 (Optional.) Change the File Name.

Note You cannot change the file type; you can save the script only in .ICMF format.

Step 4 Click **Save**. If the file name already exists, the system prompts you to confirm the save.

Step 5 If prompted, click **OK**. The custom function(s) are saved to the specified file in text format.