Use of Formulas

Formula Usage

A formula consists of one or more expressions that the scripting environment 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. (See the sections on variable usage, selection of targets by rules, and distribution of contacts to targets.)

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

Variable Usage

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 Service.
- The object-name is the name of an object contained in Unified ICM database, such as the name of a service (for example, BosSales).
- The variable-name is the name of an object that can hold a value, such as a call control variable (for example, (CallerEnteredDigits).
- 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:

Service.BosSales.ExpectedDelay: Examines the expected delay for the BosSales service.

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.

Business Hours Variables

Business Hours variables indicate whether a business hour is open or closed along with the ReasonCode. They can be used to decide the routing logic for an incoming contact. As a team lead or business user, you can open, close, or manage Business Hours from a simplified web interface.

Business Hours Variables

The following two variables are available in Script Editor for BusinessHours feature.

1. BusinessHourStatus
2. Reason Code

BusinessHourStatus

BusinessHourStatus variable indicates whether the BusinessHours is open or closed. The variable value 0 indicates the BusinessHours is closed and 1 indicates it is open.

You can use to take the routing decision for incoming call. The Syntax to use the variable is:

```
BusinessHours.<BusinessHoursName>.BusinessHourStatus == 1
```

The above expression will be successful if the BusinessHours status is open.

Reason Code

The Reason Code indicates the reason for the current status of Business Hours. You can use the **ReasonCode** to provide customized treatments for callers when the Business Hours are closed. The syntax to use this variable:

```
BusinessHours.<BusinessHoursName>.ReasonCode. == 1001
```

The above expression will be successful if the Reason code is 1001.

For more information on configuring the ReasonCode and associating with the Business Hours, see the following guides:

at http://www.cisco.com/c/en/us/support/customer-collaboration/unified-contact-center-enterprise/products-feature-guides-list.html

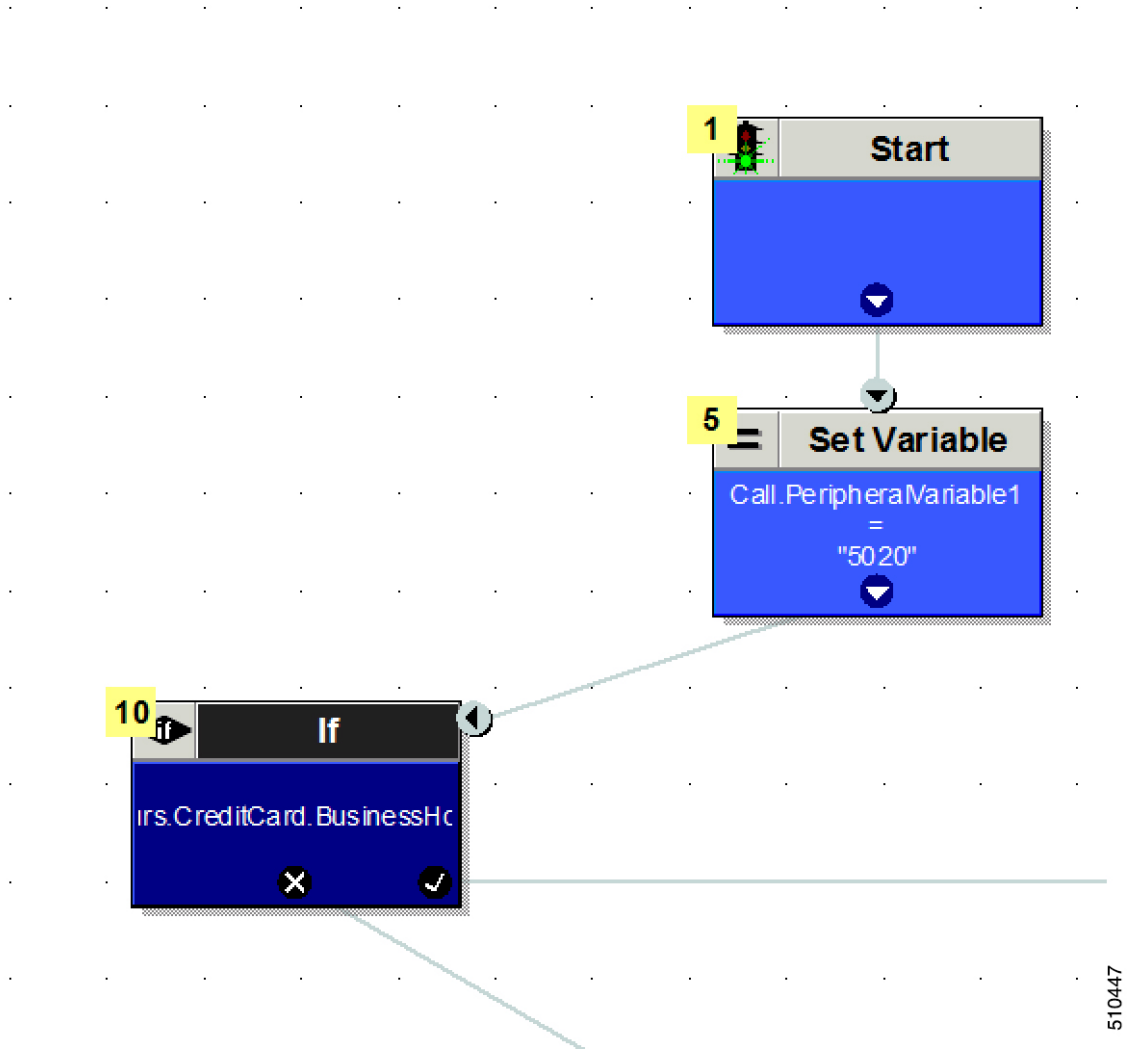
and

at https://www.cisco.com/c/en/us/support/customer-collaboration/unified-contact-center-enterprise/products-technical-reference-list.html.

The BusinessHours variables are available only in the IF node of Script Editor.

Business Hours in Scripts

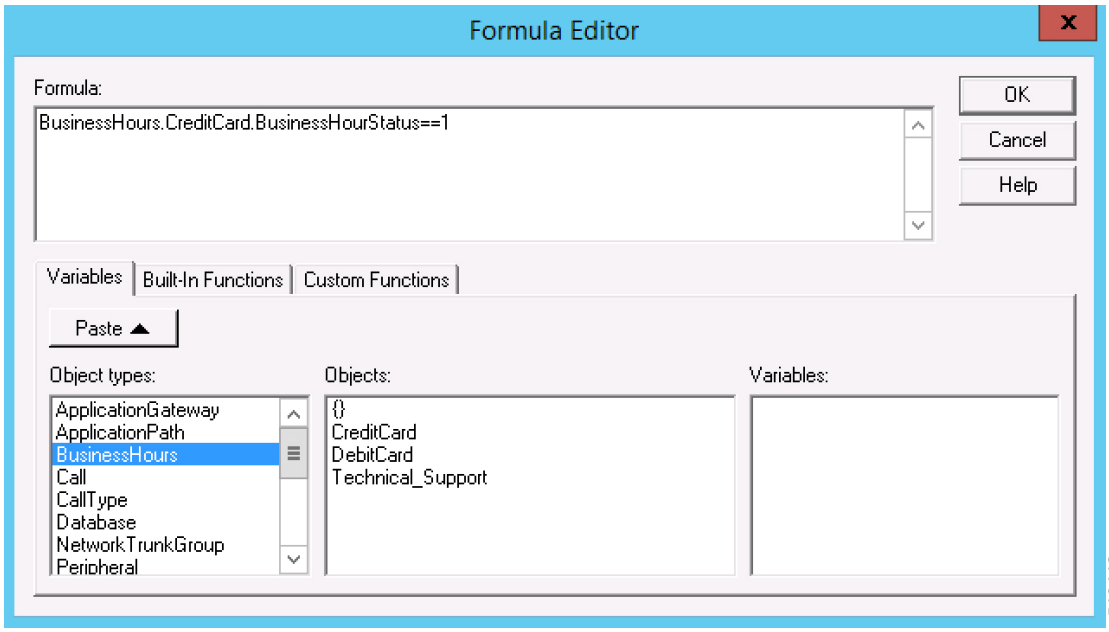
Using unified ICM script editor, you can configure BusinessHours in ICM script to control call treatment. You can create a simple expression in **IF** node which checks the Configured BusinessHours real time status and return true or false based on expression.




```
BusinessHours.CreditCard.BusinessHourStatus==1
```

The formula given above checks that the real time status of **BusinessHours** named **CreditCard** and returns through Success path of **IF node** when the **Business Hours** are open.

You can create above formula using **Formula Editor** in Script editor.



Note Business hour object type is available for selection only when there is at least one Business Hour that is configured in system using CCE Administration web page.

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
CLIDRestricted	Integer	If 1, CLID presentation should be restricted. If 0, CLID presentation should not be restricted.	Set in Unified ICM Configuration Manager. Open Tools > Miscellaneous Tools > System Information. Check Enabled in the CLID Masking section of the screen to turn on.
CustomerProvidedDigits	String	Digits to be passed to the routing client for forwarding to the call recipient.	Yes
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
NetworkTransferEnabled	Integer	If 1, network transfer is enabled. If 0, network transfer is not enabled.	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 Unified ICM processes the call.	No
RouterCallKey	Integer	A value that is unique among all calls Unified ICM 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 For a Post-Routing® request from an Aspect ACD, PeripheralVariable1 through PeripheralVariable5 map to the Aspect variables A through E. The Aspect routing client passes these variables to the Unified ICM as part of the request and the Unified ICM returns them with the response. Other routing clients might use some of these variables for other purposes. The values of these variables are also stored in the Route_Call_Detail table of the database.

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 Context (ECC) Variables

Expanded call context (ECC) variables store values associated with the contact.

ECC 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 VRU, a NIC, CTI, and so on. This applies to null values as well as non-null values.

If an ECC 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 for expanded call context variables and peripheral call variables is supported when used with Unified CVP, Cisco Finesse, and Cisco SocialMiner, among others.

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

ECC 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. If it does not involve translation routes or the Unified CCE, and is outside the original peripheral, then the solution cannot propagate ECC variables, like all call variables.

The solution comes with some predefined ECC variables. You can create others througn the Configuration Manager.

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. The system does this because it automatically adds all ECC variables to the default ECC payload if that is the only ECC payload.

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.

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.

ECC Payload Use by Interface

This table summarizes the use of ECC payloads in various operations:

Condition	ECC Payload That Is Used
Routing to VRU	Default payload If an ECC payload is specified in the configuration of that VRU, it overrides the Default payload.
Routing to Application Gateway	ECC payload that currently has scope in the script
Routing to Agent PG (including the Unified CM PG and Avaya PG)	ECC payload that currently has scope in the script
Routing to Media Routing PG	Default payload If an ECC payload is specified in the configuration of the VRU for that MR PG, it overrides the Default payload.
Routing to pre-12.0 PG	Always Default payload
Routing to System PG (Agent or VRU)	Always Default payload
Routing to Avaya Aura Symposium PG	Always Default payload
Routing to Aspect PG	Always Default payload

Condition	ECC Payload That Is Used
Contact Director to target Unified CCE	ECC payload that currently has scope in the script
Routing to INCRP NIC	ECC payload that currently has scope in the script
Pre-route to Gateway PG on Parent in Parent/Child	Always Default payload

Note If you do not create another ECC payload, the solution uses the Default payload for everything.

ECC Payloads in Scripts

The solution uses the Default payload unless you override it.

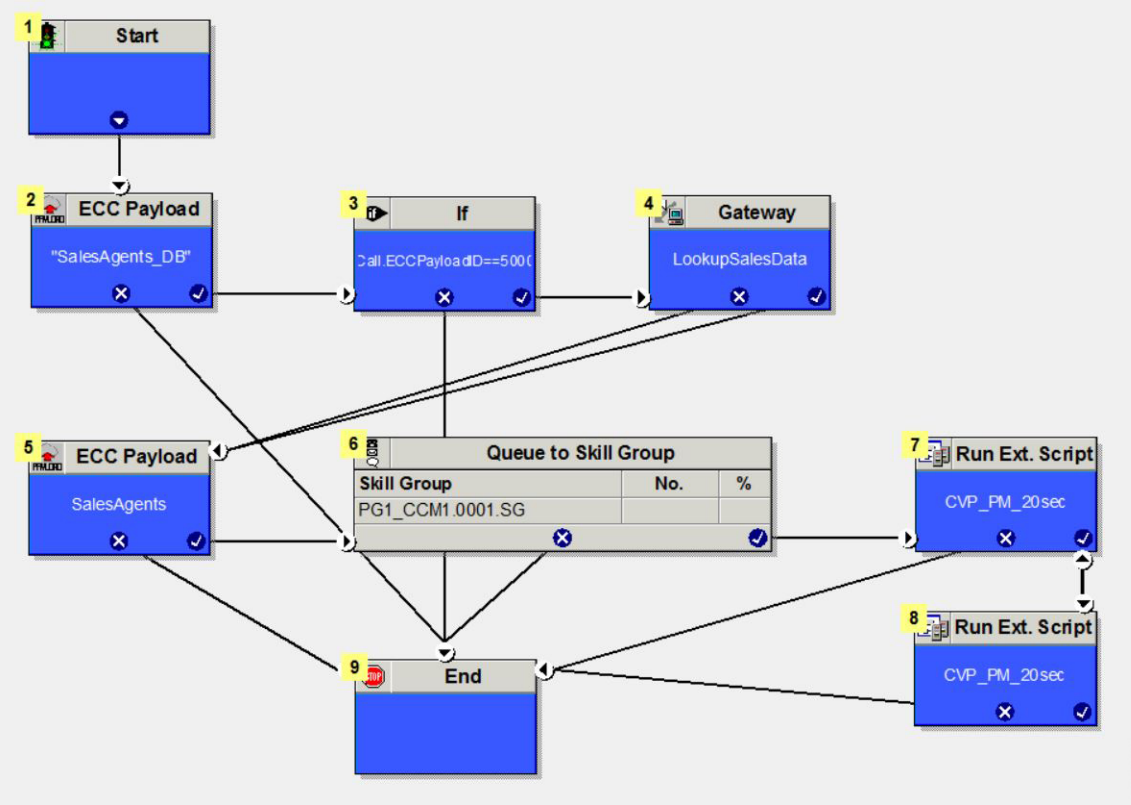
When a call or task begins a routing script, it always starts with the Default payload. If the Default payload contains the ECC variables that the call or task requires, there is no need to change ECC payloads. The ECC Payload node sets which ECC payload currently has scope in the scripting environment. When you set an ECC payload to have scope, it retains scope until you change to another ECC payload. Any scripting node that sends messages externally uses the ECC payload that currently has scope.

However, you can specify an ECC payload in the configuration of a VRU. The VRU configuration includes a field for specifying an ECC payload to use with that VRU. If that field is set, the scripting environment always uses the specified ECC payload for that VRU. If you did not set the field for a particular VRU, the scripting environment always uses the Default payload.

Note The same behavior occurs with the Type 2 VRU that is associated with the MR PG.

For example, take the following script. When the routing script begins, the Default payload has scope. Before sending a request to the Application Gateway, you use the ECC Payload node to change to the “SalesAgents_DB” payload. That ECC payload's members contain the data that the Application Gateway uses in a database lookup for details that the agent needs for this call. On returning, you use the ECC Payload node to give scope to the “SalesAgents” payload. This ECC payload contains the ECC variables that the Agent PG in the PG1_CCM1.0001.SG skill group needs. If the call queues at the VRU, the scope switches to the specified ECC payload for the VRU, if configured, or else to the Default payload.

The *Call.ECCPayloadID* variable contains the ID of the ECC payload which currently has scope. When the script follows the success path from the ECC Payload node, the node updates *Call.ECCPayloadID*. In case of any error while setting the ECC payload, this script takes the failure path from the ECC Payload node. The ECC payload that last had scope continues to have scope in the failure path case.



Transfers and Conferences with ECC Payloads

In releases before the introduction of ECC payloads, the solution merged the ECC variables from all the calls during a transfer or conference. Now, with ECC payloads, the retained data depends on the ECC payload that you use for each call.

Using the same ECC payload for all the transfer or conference is simplest. This method ensures that all members are available for reporting and on the desktop. When the merge occurs, the system keeps the value from the latest call that has a value for that member.

If your call flow swaps ECC payloads during a conference or transfer, only ECC variables from the final ECC payload appear in the reporting and on the desktop. Ensure that you include any member that you need for reporting or on the desktop in all the ECC payloads that the call flow uses.

ECC Payloads with Contact Director or ICM-to-ICM

The Contact Director and its targets (and ICM-to-ICM solutions) can use different payloads across their interface. When the Contact Director sends a message to the target Unified CCE, the script that runs on the Contact Director Router governs the ECC payload that is sent. When the target Unified CCE sends a message to the Contact Director, the script that runs on the target Unified CCE Router governs the ECC payload that is sent.

If you translation route the call to a VRU on the target Unified CCE, the initial connect message goes up to the Contact Director. The script that runs on the target Unified CCE Router governs the ECC payload that is sent to the Contact Director. By comparison, when the target Router exchanges a RunScript message with its own VRU, it uses the ECC payload that is configured for that VRU.

During initialization, the INCRP NIC maps all ECC variables that the Contact Director and target Unified CCE use between the two systems. The INCRP NIC then merges the ECC variables that are sent by each system into the destination environment by the target system.

ECC Payloads with Parent/Child

In general, Parent/Child solutions use the Default payload for all call flows. The Router always uses the Default payload for these operations:

- Local calls on a System PG.
- Calls queued on a Type 9 IP-IVR. (Ignores any ECC payload that is configured on the Network VRU.)
- Calls pre-routed on the Gateway PG on the Parent.

The CVP at the Parent uses the ECC payload that is configured for the corresponding Network VRU.

Persistent vs. Non-persistent Call Variables

When the Unified CCE/Unified ICM 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 (in the Configuration Manager), 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.

Expanded Call Context Variables for Web Callback

You must create several ECC variables if you intend to use Enterprise Chat and Email and/or Voice Media Routing Domains to route Delayed Callback requests. The ECC variables are:

- user.ece.activity.id - used to send the CIM activity ID to Unified ICM.
- user.ece.customer.name - used to pass the customer name from the entry point to Unified ICM.

Warning Do not overwrite reserved ECC variables in a script. Overwriting these variables may cause the application to route tasks to agents in an incorrect manner.

For more information about configuring ECC variables, see the *Configuration Guide for Cisco Unified ICM/Contact Center Enterprise*.

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 Configuration Manager. For more information, see *Configuration Guide for Cisco Unified ICM/Contact Center Enterprise*.

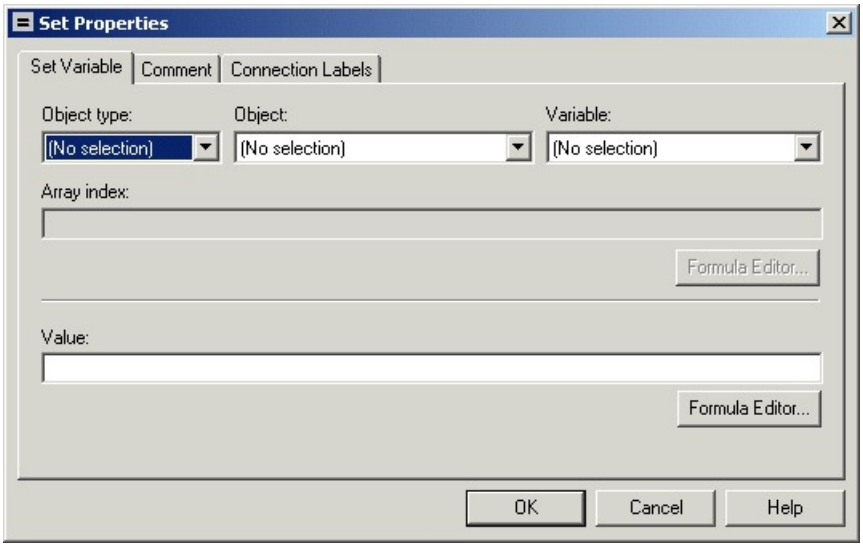
Each user variable must:

- Have a name that begins with user.
- Be associated with an object type, for example, service. (This enables the Unified ICM 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 Unified ICM 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.

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	,

Operators

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 Unified ICM 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. Unified ICM 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.

Function	Data Type	Return Value/Example
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.
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.

Function	Data Type	Return Value/Example
sqrt(n)	Floating Point or Integer	Returns the square root of n. (The operand n must be numeric and non-negative).
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.

Function	Data Type	Return Value/Example
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.
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.
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.
		Note Use the following formula for the Enterprise Skillgroup that may not contain any Skillgroups:
		`ValidValue(EntSkill.`
		`Default\EnterpriseSkill`
		`groupPri.Loggedon,0)`

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.

Note When you import the custom functions, the pattern #[0-9]+# is replaced with corresponding object name. The digits between "#" characters represent object ID. The # characters which are not the part of this pattern are not removed or replaced.

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.

Dynamic Formula for PQ

You can pass the PQ name or ID dynamically while creating a formula in the IF node. Whenever a call encounters the IF node after you create a formula, the router evaluates the formula based on the PQ name or ID that is given in the Peripheral or ECC variable. This feature is implemented to check the real time statistics of the PQ before the call is queued to that PQ.

For example: Consider a formula `PQ.{Call.PeripheralVariable1}.LoggedOn > 0`. In this formula, `{Call.PeripheralVariable1}` is the variable that picks the PQ based on the PQ Name or ID dynamically. When a call comes in, the router evaluates the formula and determines the numbers of agents that are logged in the particular PQ that is given in the Peripheral or ECC variable . After the formula is evaluated, and if an agent is logged in, the call is routed to the agent.

Note Dynamic formula for PQ is only supported in the IF node.

Note Only Peripheral Variables and User defined ECC Variables are allowed within the curly brackets of a Dynamic Expression.

Note Evaluation of dynamic expression is done first by using name and then by ID of the PQ. If no PQ is found in system based on name specified in **Set** variable, then PQ is searched based on ID.

Ensure PQ names are unique and doesn't match with other PQ's ID configured in system. In case ID is used for dynamic expression and it matched with any other name, then wrong PQ is picked up during expression evaluation.

Procedure

Step 1 Place an **If object** in the workspace; right-click to open the Properties dialog box.

Step 2 Click **Formula Editor**.

Step 3 Select the **Variables** tab, and in the Object types list, select **PQ**.
The variables that can be selected for the object are listed.

Step 4 In the Objects list, select **{}**.
The variables that can be selected for the object are listed.

Step 5 From the **Variables** list, select the variable that you want to include in the formula.

Step 6 Click **Paste** to view the selected elements in the Formula box.

Step 7 To add a Peripheral or an ECC variable, place the cursor inside the curly brackets.

The options in the Object types and Variables lists change.

Step 8 In the Object types list, select **Call**

Step 9 In the Variables list, select the variable that you want to include in the formula.

Step 10 Optional: Add a relational operator to complete the expression

Step 11 Click **OK** to close the Formula Editor dialog box.

Step 12 Click **OK** to close the IF Properties dialog box.

Step 13 Click **Save**.

Dynamic Formula for Business Hours

You can pass the Business Hours name or ID dynamically while creating a formula in the **IF** node. Whenever a call encounters the **IF** node after you create a formula, the router evaluates the formula based on the Business Hours name or ID that is given in the Peripheral or ECC variable. This feature is implemented to check the real-time statistics of the Business Hours.

Note Dynamic formula for Business Hours is only supported in the **IF** node.

Note Only Peripheral and User defined ECC Variables are allowed within the curly brackets of a Dynamic Expression.

You cannot create multiple formulas using dynamic expressions.

For example:

```
BusinessHours.{Call.PeripheralVariable1}.BusinessHourStatus==1 &&
BusinessHours.{Call.PeripheralVariable1}.BusinessHourStatus==1
BusinessHours.{Call.PeripheralVariable1}.BusinessHourStatus==1 || BusinessHours.test
BusinessHours.BusinessHourStatus==0
```

.

You can pass the Business Hours name or ID dynamically while creating a formula in the **IF** node. Whenever a call encounters the **IF** node after you create a formula, the router evaluates the formula based on the Business Hours name/ID that is given in the Peripheral/ECC variable.

For example:

```
BusinessHours.{Call.PeripheralVariable1}.BusinessHourStatus==1.
```

In this formula, `{Call.PeripheralVariable1}` is the variable that dynamically picks the Business Hours based on the Business Hours Name or ID. When a call comes, the router evaluates the formula and determines the real-time status of that Business Hours that is given in the Peripheral or ECC variable. After the formula is evaluated, and if BusinessHours's status is 1, the call takes the success path of IF node.

Note Dynamic expression is first evaluated by using name and then by ID of the BusinessHours. If no BusinessHours is found in system based on name specified in **Set** variable, then BusinessHours is searched based on ID.

Ensure BusinessHours' names are unique and doesn't match with other BusinessHours' ID configured in system. In case ID is used for dynamic expression and it matched with any other BusinessHours' name, then wrong BusinessHours is picked up during expression evaluation.

Procedure

Step 1 Place an **If object** in the workspace; right-click to open the Properties dialog box.

Step 2 Click **Formula Editor**.

Step 3 Select the **Variables** tab, and in the Object types list, select **Business Hours**.
The variables that can be selected for the object are listed.

Step 4 In the Objects list, select **{}**.
The variables that can be selected for the object are listed.

Step 5 From the **Variables** list, select the variable that you want to include in the formula.

Step 6 Click **Paste** to view the selected elements in the Formula box.

Step 7 To add a Peripheral or an ECC variable, place the cursor inside the curly brackets.
The options in the Object types and Variables lists change.

Step 8 In the Object types list, select **Call**

Step 9 In the Variables list, select the variable that you want to include in the formula.

Step 10 Optional: Add a relational operator to complete the expression

Step 11 Click **OK** to close the Formula Editor dialog box.

Step 12 Click **OK** to close the IF Properties dialog box.

Step 13 Click **Save**.