Conversation Rules Templates Guide

Genesys Rules System 8.5.0

3/10/2022

Table of Contents

Conversation Rules Template Guide

These topics describe how to use the new Conversation Rules (CR) template that comes with Genesys Rules Authoring Tool (GRAT) 8.5.001.

The purpose of this CR template is to enable much closer integration between GRS and Context Services blocks without having to build a new template from scratch in Composer (create a Fact model, actions and conditions, and pass complex data structures between the Composer Business Rules block and the rules engine and re-evaluate the returned data, and so on).

[+] MORE

This new template does the following:

- Integrates with the appropriate UCS and Context Services blocks out of the box
- Provides useful and typical conditions and actions
- Provides common date/time-related functions that integrate with GRS business calendars (such as, `Today is a working day`)
- Provides a sample rule package that implements some common use case scenarios, and these are documented in this document along with Composer project screen shots and example projects illustrating usage.

> **Important**
>
> The workflows and strategies shown in this document are from Composer. You could equally use Genesys IRD workflows.

Both the template and the sample rule package are shipped as .xml files that can be easily imported directly into GRAT.

> **Important**
>
> Because the new CR template uses complex JSON objects and structures, it is not possible in the 8.5.001 release to use the Test Scenarios feature of GRAT on rule packages generated from the CR template. This compatibility is planned for a future release.

Overview

Working with the Business Rules Block

Use Cases

Conditions and Actions

Conversation Rules—Overview of Genesys Elements

Genesys Composer

Composer provides a set of function blocks that allow access to Context Services. These out-of-the box function blocks on the workflow diagram palette allow the developer to create applications that perform various actions, such as:

- Identify customers and update their profiles.
- Extend customer profiles with user-defined information.
- Query a customer's profile.
- Create, start, complete, and query customer services.
- Query customers' active services.
- Enter, complete, and query service states and specific tasks.
- Use a business rule block to request evaluation of business logic developed in Genesys Rules System by business analysts, and act on the result.

Orchestration Server

Execute the orchestration application. Orchestration Server has a function in Conversation Manager similar to the function of Universal Routing Server (URS) in Genesys voice and multimedia solutions. One of the main differences is that it operates based on business processes developed in State Chart XML (SCXML) rather than routing strategies written in IRL (Intelligent Routing Language, a Genesys proprietary language).

GVP

Executes the VoiceXML applications.

SCXML applications

SCXML applications can be written directly using any XML or plain text editor, or with Genesys Composer, an Eclipse-based development environment. They are published on an application server such as JBoss or another Java-based application server, and are executed on Orchestration Server.

Conversation Manager/UCS

Genesys Conversation Manager takes Genesys' core capability of routing and extends it, generalizes it, and integrates it more tightly with other Genesys products. Rather than the call (T-Server) or the interaction (eServices/Multimedia), Conversation Manager takes the service as the basic entity. It orchestrates the service process across channels and over time, using dynamic data and business rules to make decisions about operations. For example;

A bank customer calls a toll-free number inquiring about mortgage preapproval. An IVR prompts him to enter his account number, then transfers him to an agent, who fills in an application form for him and asks him to fax some supporting documents. After he faxes the documents, he receives an SMS message thanking him and informing him that he will receive a response within 48 hours. The next day he receives an e-mail congratulating him on the approval of his application.

This example involves voice, IVR, fax, SMS, and e-mail channels. Conversation Manager is able to treat the entire sequence as a single service.

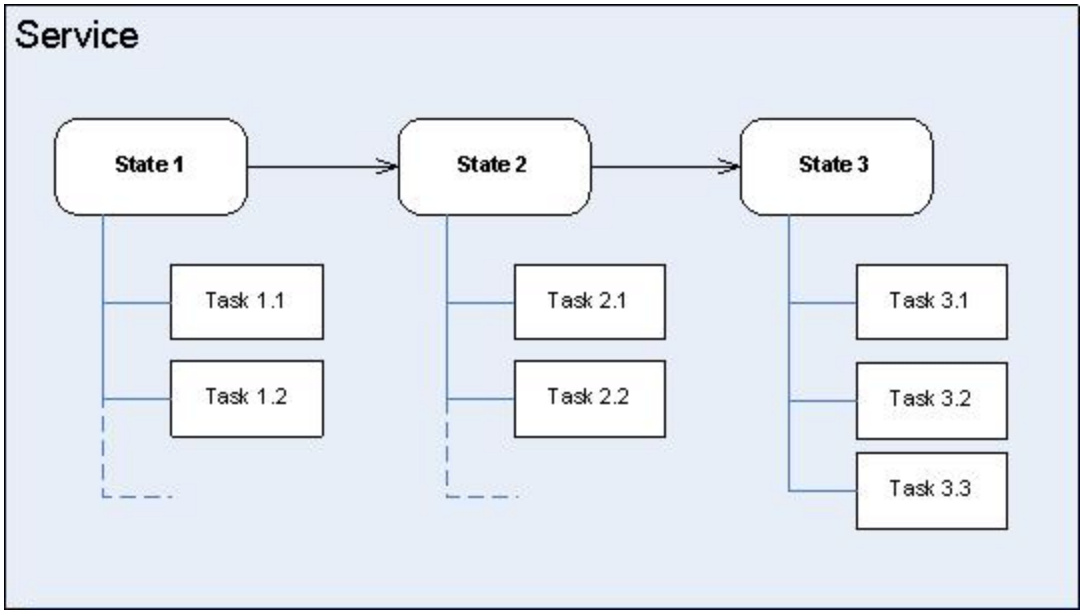
Service/State/Task Model

Conversation Manager adds to Genesys the concept of service, which may be defined as follows:

- It represents a business process, which in turn may be seen as a communication or series of communications between a customer and an enterprise, and possibly also between various parts of the enterprise.
- It can span multiple interactions.
- It may include interactions in various media.
- It has a temporal beginning and end.
- It may be subdivided into states, which in turn may be subdivided into tasks.

Services are composed of any number of States, and States can in turn be composed of any number of Tasks. This three-level structure provides a flexible vocabulary by which organizations store the history of the services that they provide to customers.

A Service may also be divided directly into tasks:



Services are defined by association to Service Types that you create as Business Attributes in the Configuration Layer. States may be used to represent components of customer service, such as:

- Customer identification
- Assigning a service agent (automated or live)
- Service identification
- Waiting for a service agent
- Offering another service while waiting for an agent
- Offering callback
- Waiting for customer input

Services, States and Tasks exist over some application-defined lifecycle. Upon completion, applications may specify a Disposition. For example, the offering of a new product or service might be recorded as a State of type `Offer another service`. The Disposition might be set to show whether the customer accepted or declined the offer. Information on past declined or accepted offers could then be used to calculate the likelihood that the customer might be interested in the offer at some point in the future.

> **Important**
>
> This Service Model can be used by any component that can access UCS/CS's HTTP interface. It is not limited to use in Conversation Manager.

Genesys Rules

Genesys Rules System provides the ability to develop, author, and evaluate business rules.

A **business rule** is a piece of logic defined by a business analyst. The rules in a rule package provide a set of functionality. The Genesys Rules Authoring Tool (GRAT) allows you to create, edit, and delete rules and rule packages.

Rule packages are bundles of rules. Rule packages are used to group, manage, and deploy rules. A rule package contains one or more rules plus the fact model that is needed to support the rules. The fact model is a description of the data. It contains field names and types which are grouped into tables/classes. Facts are input/output to rule execution and are instances of the tables/classes defined in the fact model. Rule packages also contain the rule definitions, business calendars, and also the templates that the rule package is dependent on. You deploy rule packages individually to the Rules Engine.

Rule packages also allow you to do the following:

- Partition rules and facts so that they are small, well-defined, and apply only to a particular application or use. This makes them easier to debug and understand.
- Isolate rule packages from one another when executing rules. This also improves performance because the Rules Engine has fewer candidates to examine during the evaluation.
- Update individual rule packages without affecting other deployed packages.
- Import and export an entire rule package containing the rule definitions, business calendars, and also the templates that the rule package is dependent on.

Genesys Reporting

Run reports to determine customer trends.

Configuration Prerequisites

Genesys Composer

The Conversation Rules template requires Genesys Composer release 8.1.300.89 at minimum.

Configuration Options

The following configuration options must be set in order to use the Conversation Rules template:

Genesys Rules Engine

- `json-hierarchical-driver` = `true`

Genesys Rules Authoring Tool

These options control how GRAT connects to the Context Services REST API.

- `context-services-rest-api-protocol`—`http` or `https`
- `context-services-rest-api-host`—Host name
- `context-services-rest-api-port`—Port number
- `context-services-rest-api-base-path`—The base path

Contact Server

- `map-names` = `true`

When a rule package is deployed and `map-name` is set to `true`, the business attribute name is encoded in the rule package. With value `false`, the DBID is encoded in the rule package.

If the `map-name` option is changed on UCS at any time from rule authoring to deployment to actual operation, existing rule packages based on the Conversation Manager template must be redeployed.

Getting Started

Importing the CM Template and Sample Rules Package

1. Install GRS as described in the **GRS Deployment Guide** (opens a new document).
2. Log into GRAT.
3. Navigate to the required solution in the left navigation pane.
4. Click the **Import Templates** button.
5. Browse to the template file—`cm_template.xml`—which will be in the **Examples** folder in the default installation directory unless you specified another location when you installed it. Click **Import**.
6. A prompt indicates whether or not the import succeeded. When the import is complete, you will see on the **Import Template** dialog a new template called **CM_Standard_Rules.**
7. From the **CM Examples Solution** folder, browse to the CM Sample Package file —`cm_sample.xml`. Click **Import**.
8. Give the sample rules package a suitable **Package Name** and **Business Name** for your purposes. See also importing a rules package.

The template is now available for selection when you create a rules package, and the sample rules package is available to work with.

You now have available, via the drop-down menus in GRAT, a fully defined set of ready-made Conversation Manager-specific Conditions and Actions. Full detailed listings of these are provided in **Conditions** and **Actions**.

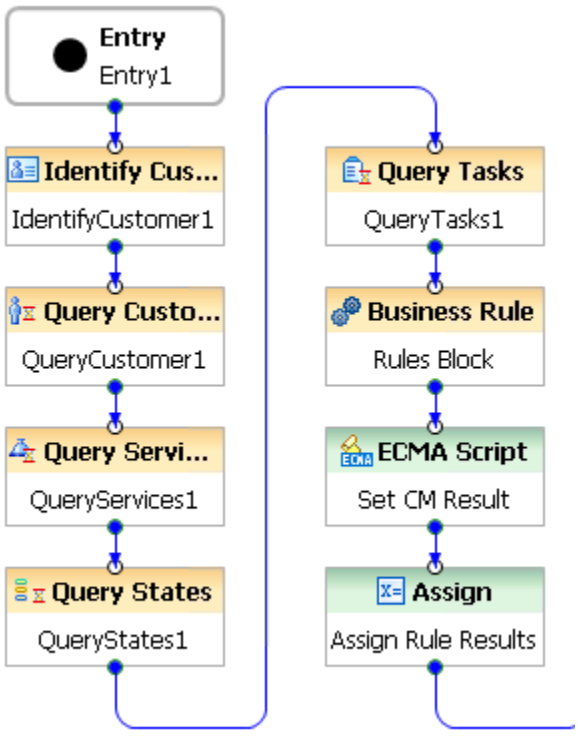
Working with Composer's Business Rule Block

Summary

Once the Rule Packages (created from Rule Templates) that you want to work with are deployed to the Genesys Rules Engine, you can use the Business Rule block on the Server Side palette to create voice and routing applications that use business rules.

Use this block to have Composer query the Genesys Rules Authoring Tool (GRAT) for deployed packages. For the Rule Package that you specify, Composer will query the GRAT for the Facts associated with the Rule Package. You can then set values for the Facts, call the Genesys Rules Engine for evaluation, and save the results in a variable.

Simple Workflow



In this typical workflow:

1. The **Identify Customer** block is used to identify the customer based on certain search criteria, such as ANI.
2. The **Query Customer** block is used to pull out the customer profile data.
3. The **Query Services** block is called to pull any related services for this customer.
4. The **Query States** block is called to pull the states related to a particular service.
5. The **Query Task** block is called to pull the tasks associated with a particular service or state.
6. The **Business Rule** block may be placed anywhere in the workflow/callflow, assuming the data needed for the rules called by the Business Rule block have already been fetched.
7. The **Assign** block enables the workflow to access all the different decisions of the CM rule package. The decision(s) requested by the Business Rule block and made by the Rules Engine are returned to the workflow/callflow which carries them out. The GRS does not actually execute the decisions (e.g. update the customer profile, transfer to agent, an so on).

User Variables in the Workflow

In this example workflow, the following user variables have been defined to retrieve data necessary for, as well as demonstrate the various decisions made by, the CM rules.

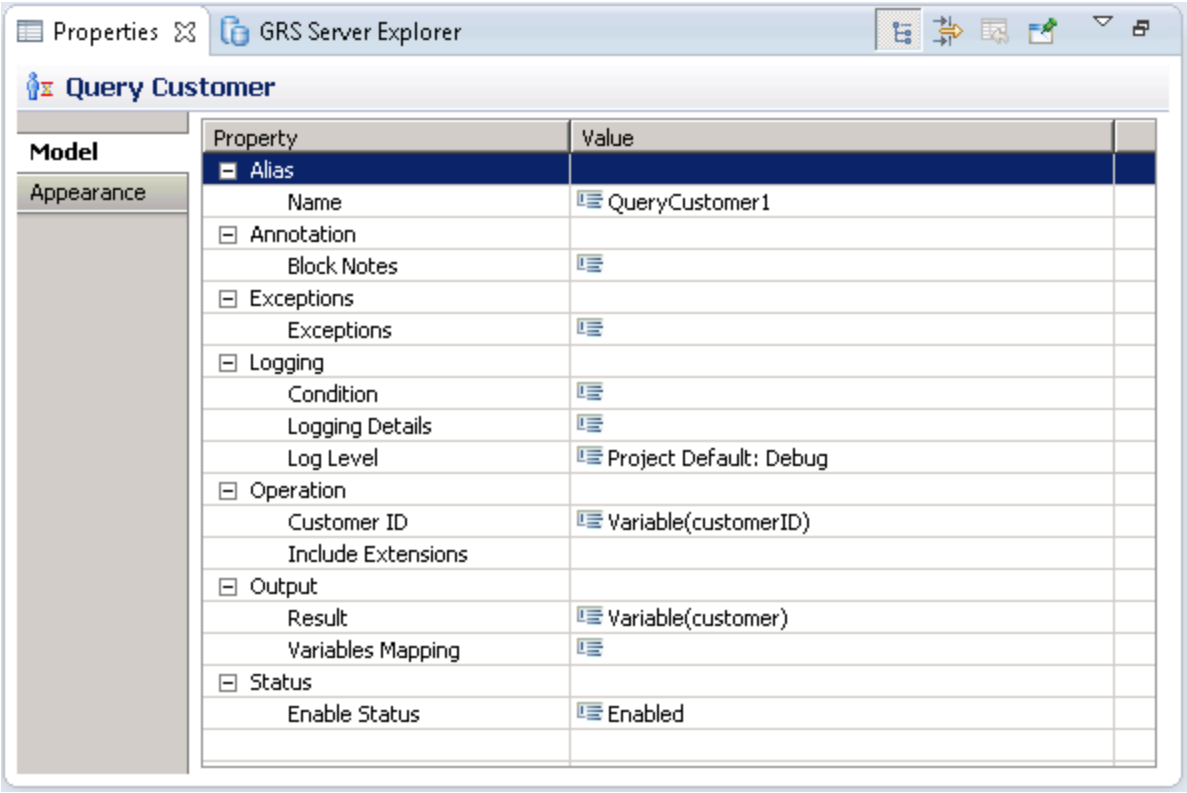
Variable Name	Default Value	Expected Type	Description
customer	undefined	JSON object	Customer profile data as returned by Query Customer block.
services	undefined	JSON array	Services of the customer as returned by Query Services block.
states	undefined	JSON array	States of a particular service as returned by Query States block.
tasks	undefined	JSON array	Tasks of a particular service or state as returned by Query Tasks block. (In Context Services, Tasks may be associated with service or state.)
contractEndDate	undefined	String	Contract end date timestamp string in the format of `2014-07-14T13:23:35.392Z`.
mediaType	undefined	String	Media type of the current interaction.
businessRulesResultObject	undefined	JSON object	Output of Business Rules block.
customerID	undefined	String	ID of the customer as returned by Identify Customer block.

Variable Name	Default Value	Expected Type	Description
serviceID	undefined	String	ID of a service of the customer to use in Query States and Query Tasks blocks.
stateID	undefined	String	ID of a state of a service of the customer to use in Query Task block. (Not used when querying tasks associated with service.)
cmResults	undefined	JSON object	Results from CM rules. This is extracted from businessRulesResultObject for easier access.
updatedFields	undefined	JSON object	CM Rules decision: Customer profile fields to update. Each key-value pair of this object correspond to a contact attribute. This is extracted from cmResults for easier access later in the workflow.
offerServiceResumption	false	Boolean	CM Rules decision: Whether to offer service resumption to customer. This is extracted from cmResults for easier access later in the workflow.
offerSurvey	false	Boolean	CM Rules decision: Whether to offer survey to customer. This is extracted from cmResults for easier access later in the workflow.
blockCommunication	false	Boolean	CM Rules decision: Whether to block further communication to customer. This is extracted from cmResults for easier access later in the workflow.
sendCommunication	undefined	String	CM Rules decision: Which media type to use for further communication with customer. This is

Variable Name	Default Value	Expected Type	Description
			extracted from cmResults for easier access later in the workflow.
requestedAgent	undefined	String	CM Rules decision: Which particular agent to route this customer to. This is extracted from cmResults for easier access later in the workflow.
requestedAgentGroup	undefined	String	CM Rules decision: Which agent group to route this customer to. This is extracted from cmResults for easier access later in the workflow.
requestedPlaceGroup	undefined	String	CM Rules decision: Which place group to route this customer to. This is extracted from cmResults for easier access later in the workflow.
requestedSkill	undefined	String	CM Rules decision: Which skill to route this customer to. This is extracted from cmResults for easier access later in the workflow.

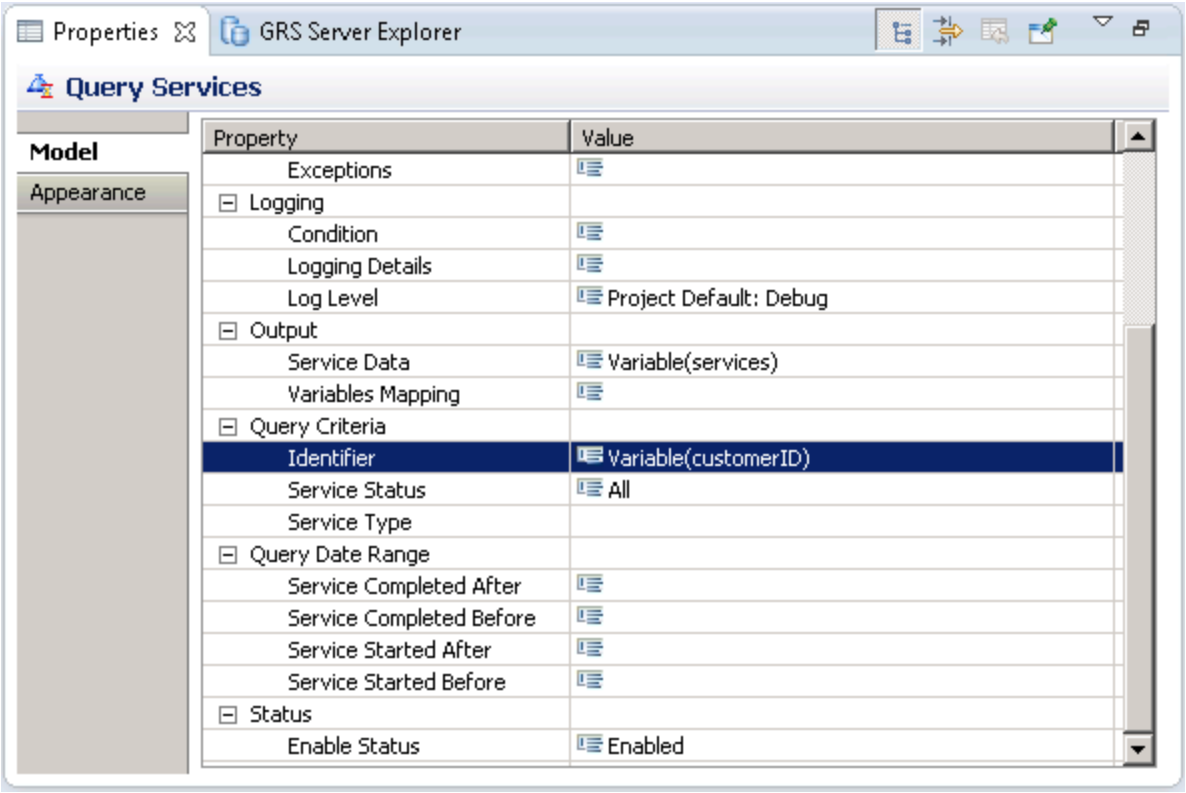
Query Customer Block

This is the Query Customer block. Note the Output: Result is stored in the user variable customer.



Query Services Block

The following is the **Query Services** block.

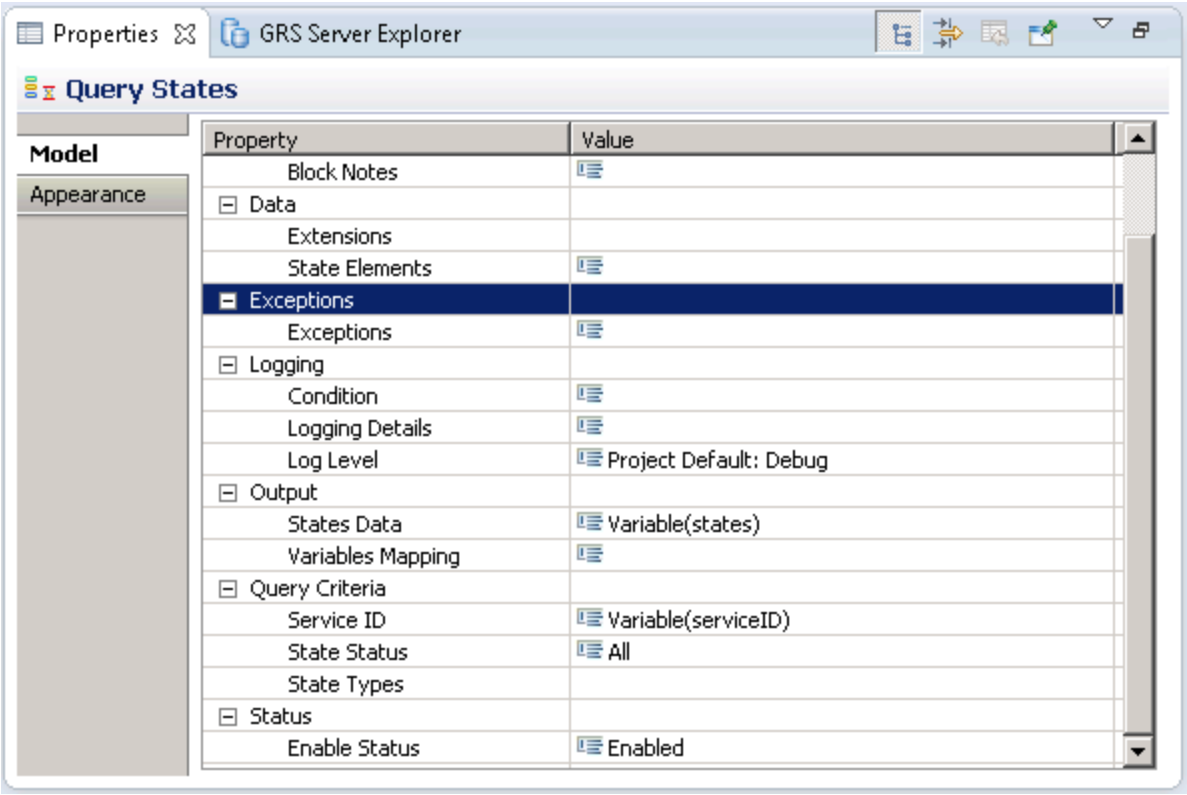


Notes:

- The result Output—`Service Data` is stored in user variable services.
- Query Criteria—`Identifier` is set to `customerID`, meaning we are querying services of the identified customer.
- Query Criteria—`Service Status` is set to `All` to ensure both conditions for active services and completed services can be correctly checked.
- Query Criteria—`Service Type` is not set, ensuring services of all service types are returned for service type-related rule conditions.
- No Query Date Range is set, which is needed for date checking service conditions.

Query States Block

The following is the **Query States** block.

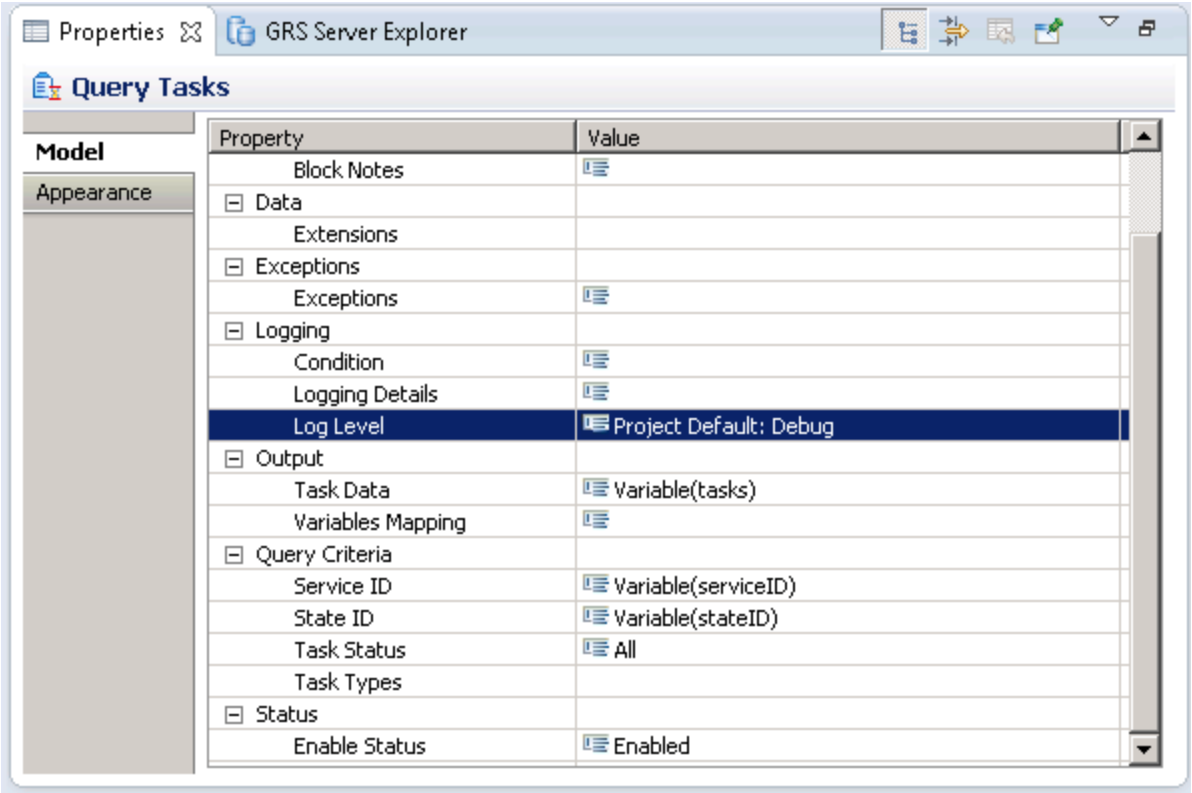


Notes:

- The result Output—`States Data` is stored in user variable states.
- Query Criteria—`Service ID` is set to `serviceID`, which should be previously extracted from **Query Services** results.
- Query Criteria—`State Status` is set to `All` to ensure both conditions for active states and completed states can be correctly checked.
- Query Criteria—`State Types` is not set, ensuring states of all state types are returned for state type-related rule conditions.
- Since Query States may only fetch states of a single service, the current CM template expects all states supplied are associated with a single service. If states of multiple services are aggregated to a CM rule package, the rules may not behave as expected.

Query Tasks Block

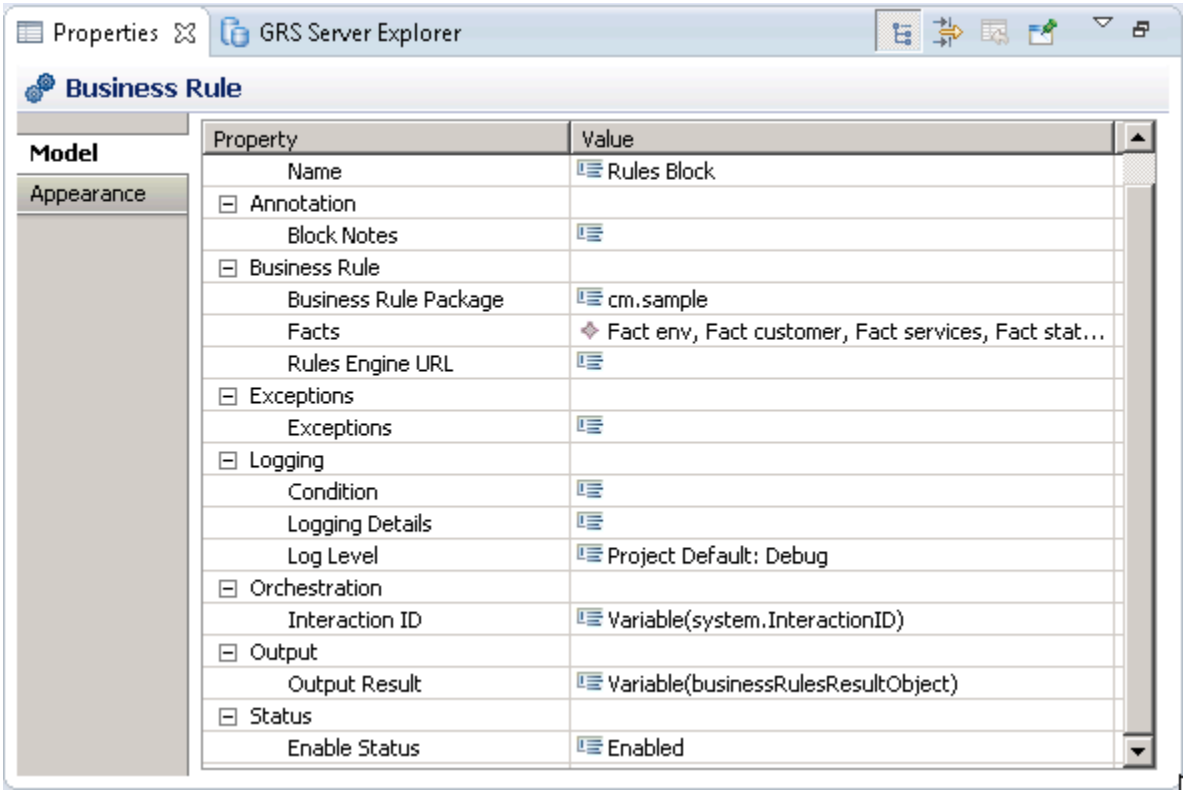
The following is the **Query Tasks** block.



Notes:

- The result Output—`Task Data` is stored in user variable tasks.
- Query Criteria—`Service ID` is set to `serviceID`, which should be previously extracted from **Query Services** results.
- Query Criteria—`State ID` is set to `stateID`, which should be previously extracted from **Query States** results. This field may be omitted when querying tasks associated with a service.
- Query Criteria—`Task Status` is set to `All` to ensure both conditions for active tasks and completed tasks can be correctly checked.
- Query Criteria—`Task Types` is not set, ensuring tasks of all task types are returned for task type-related rule conditions.
- Since **Query Tasks** may only fetch tasks of a single service or state, the current CM template expects all tasks supplied are associated with a single service or state. If tasks of multiple services/states are aggregated to a CM rule package, the rules may not behave as expected.

Business Rules Block

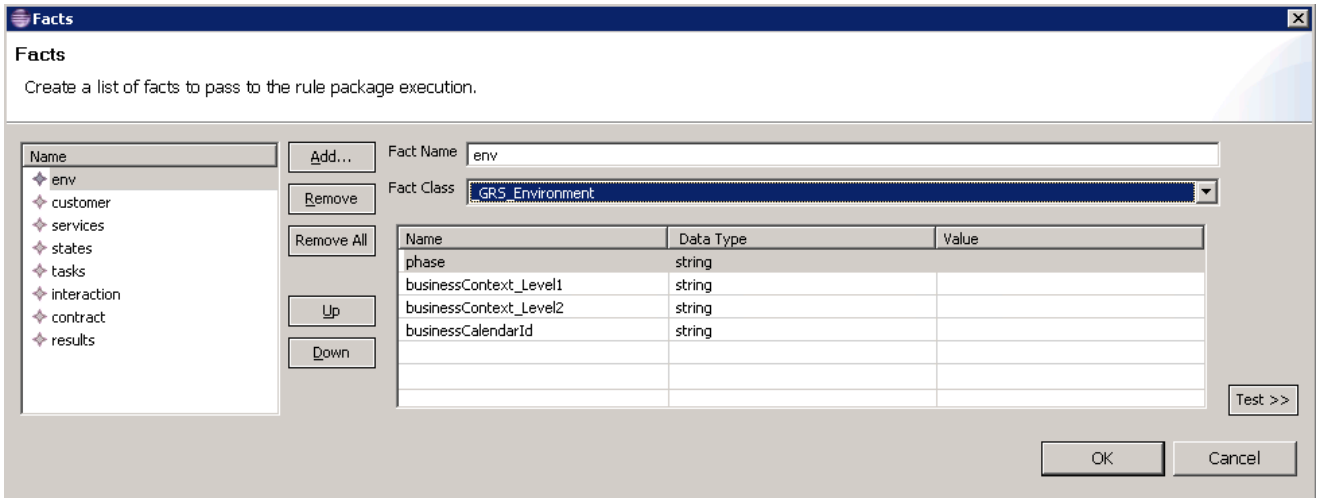


Notes:

- The **Business Rule** block is where GRS is invoked in the workflow/callflow. Before configuring this block, a rule package using the CM template should already be authored and deployed to GRE.
- `Business Rule Package` is the name of the rule package. This is chosen from a list of all deployed rule packages on the GRE specified in **Windows -> Preferences**.
- Output Result is copied to user variable `businessRulesResultObject`.
- Facts brings up a list of all facts to send to GRE. See below for further detail.

Facts

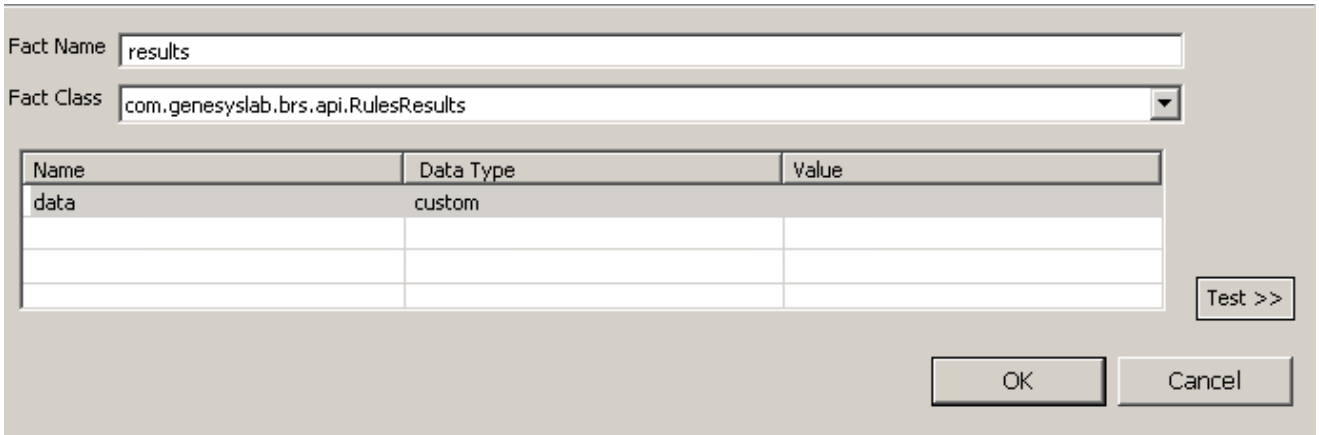
_GRS_Environment



The `_GRS_Environment` fact class is available to rules packages of all types, this fact class is for storing global environment variables.

com.genesys.brs.api.RulesResults

Fact class `com.genesys.brs.api.RulesResults` is a placeholder for storing results from GRE. It is required for CM rule packages. No value should be set for the data field.

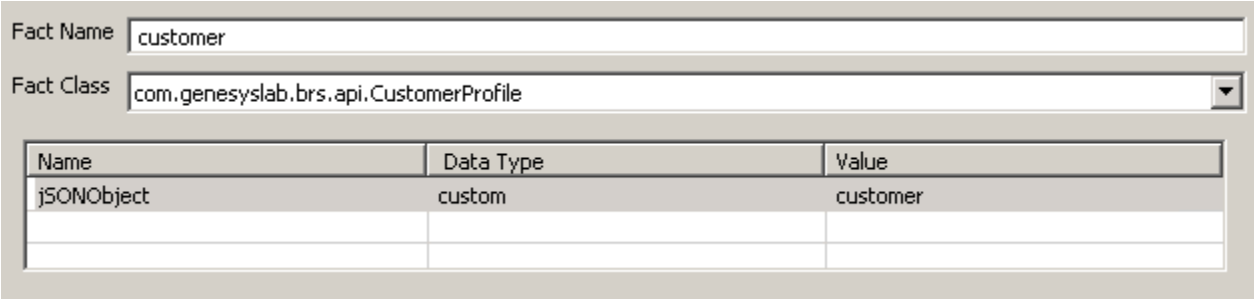


com.genesyslab.brs.api.CustomerProfile

Fact class `com.genesyslab.brs.api.CustomerProfile` provides the customer profile from **Query Customer** block to GRE. It is required only if customer-related conditions are used; otherwise it is ignored. If it is not provided, all customer-related conditions are considered failed.

In this example, the value of `jSONObject` is set to the user variable customer, which was previously

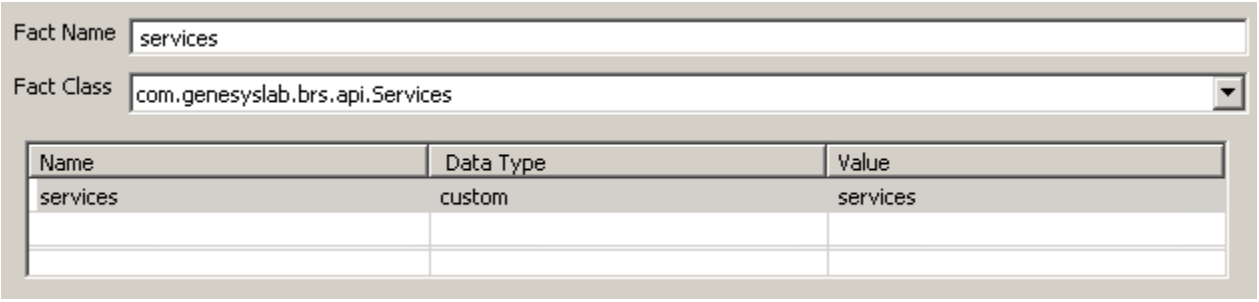
populated by **Query Customer** block.



com.genesyslab.brs.api.Services

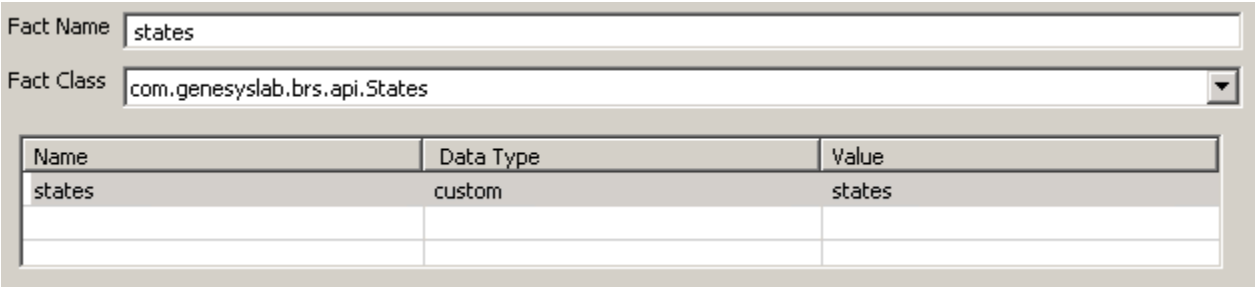
Fact class `com.genesyslab.brs.api.Services` provides services from **Query Services** block to GRE. It is only required if service-related conditions are used; otherwise it is ignored. If it is not provided, all service-related conditions are considered failed.

In this example, the services field is set to the user variable services, which was previously populated by **Query Services** block.



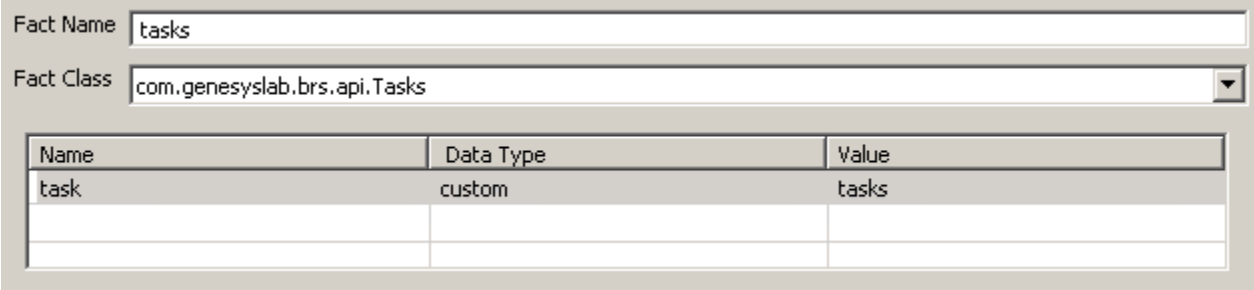
com.genesyslab.brs.api.States

Fact class `com.genesyslab.brs.api.States` provides the states from **Query States** block to GRE. It is required only if state-related conditions are used; otherwise it is ignored. If it is not provided, all state-related conditions are considered failed. In this example, the states field is set to the user variable states, which was previously populated by **Query States** block.



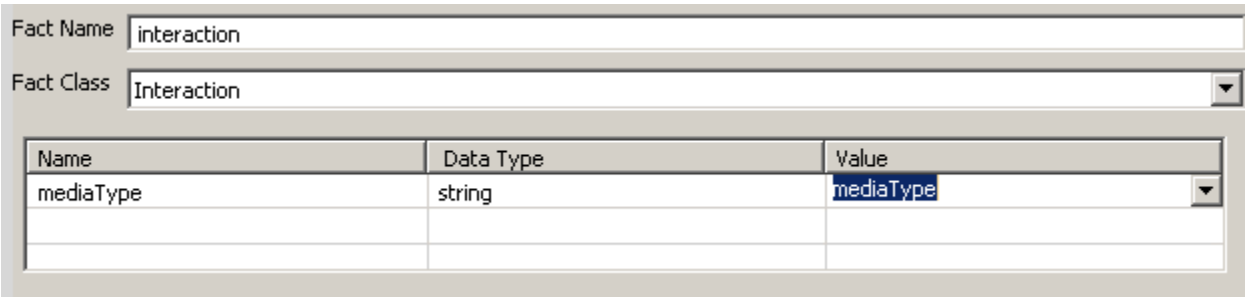
com.genesyslab.brs.api.Tasks

Fact class `com.genesyslab.brs.api.Tasks` provides the states from **Query Tasks** block to GRE. It is required only if task-related conditions are used; otherwise it is ignored. If it is not provided, all task-related conditions are considered failed. In this example, the tasks field is set to the user variable tasks, which was previously populated by **Query Tasks** block.



Interaction

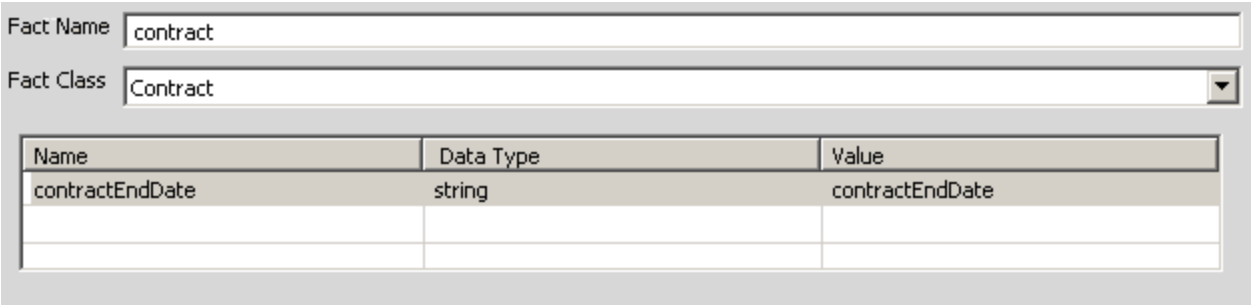
Fact class `Interaction` provides the media type of the current interaction to GRE. It is required only if media type-related conditions are used; otherwise it is ignored. If it is not provided, all media type-related conditions are considered failed. In this example, the `mediaType` field is set to the user variable `mediaType`, which is not actually populated in previous blocks in the current workflow. In normal usage, the user variable for providing media type to this fact should either be pre-populated in a previous block, or hard-coded based on the workflow (for example, if this workflow is only executed for voice).



Contract

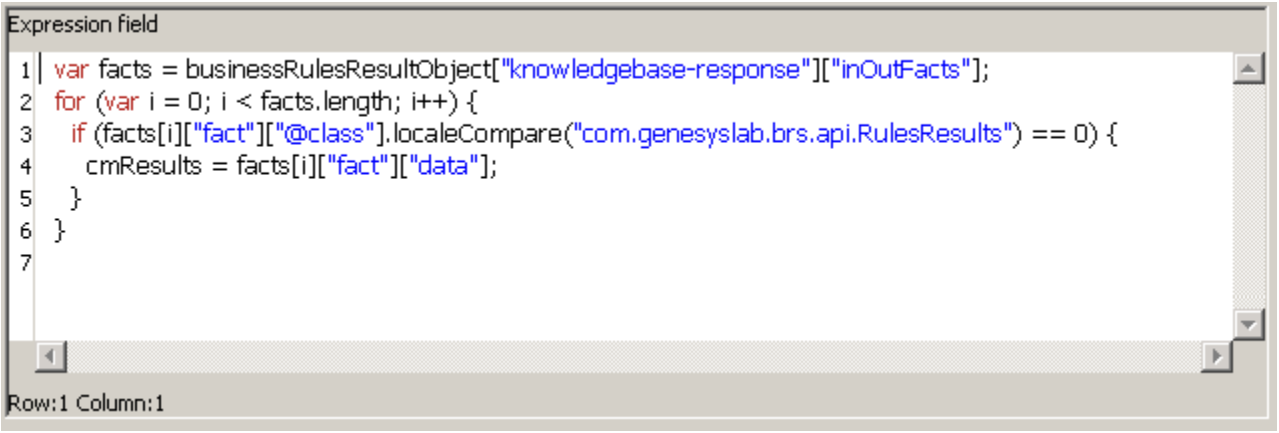
Fact class **Contract** provides the contract end date of the user to GRE. It is only required if contract-related conditions are used; otherwise it is ignored. If it is not provided, all contract-related conditions are considered failed.

In this example, the `contractEndDate` field is set to the user variable `contractEndDate`, which is not actually populated in previous blocks in the current workflow. In normal usage, the user variable for providing contract end date to this fact should should be pre-populated in a previous block (for example, calculated based on service start date and pre-configured contract length, fetched from external services).



ECMA Script Block

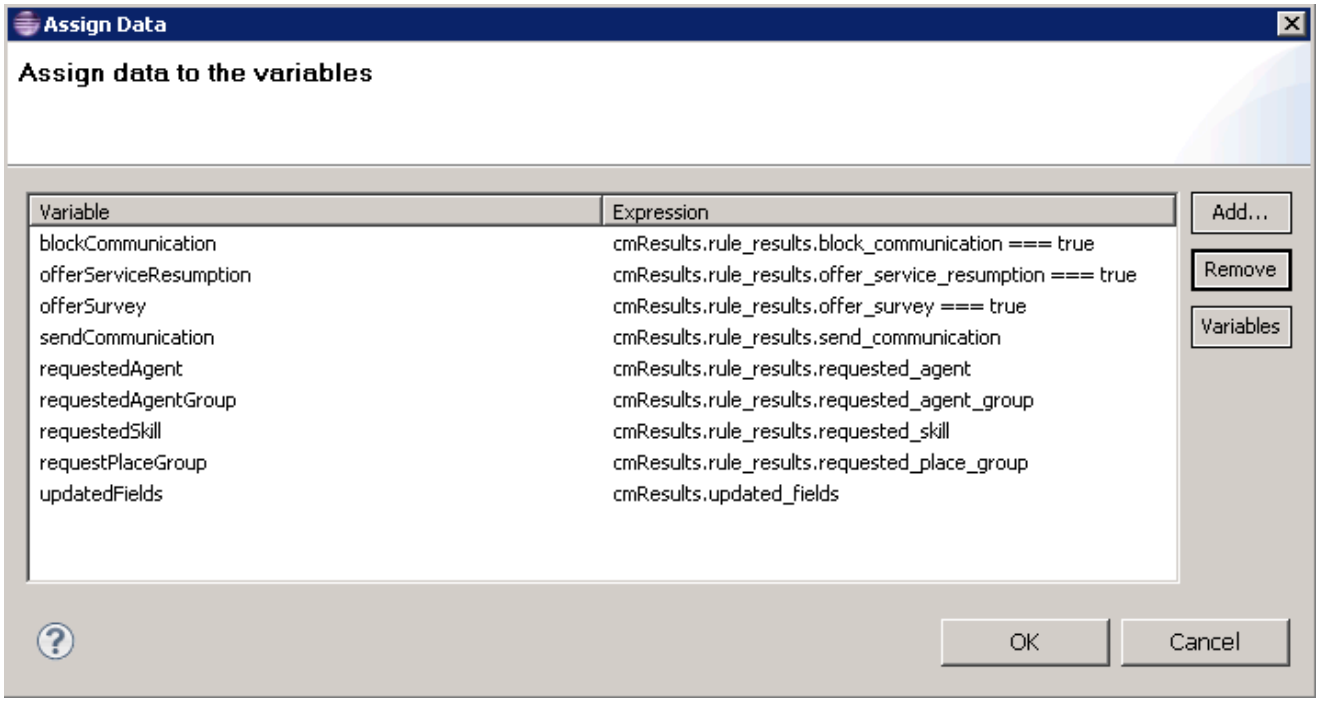
The purpose of this ECMA Script block is to extract the decisions/recommendations of the CM rule package from `businessRulesResultObject` (output of Business Rule block) into the user variable `cmResults`. It is not absolutely necessary, but as `businessRulesResultObject` contains all the input to the Business Rule block, and actual results of the CM rule package are buried in deep.



Assign Block

The purpose of this block is to demonstrate how to access all the different decisions of the CM rule package. Notice all results are in the `rule_results` field of `cmResults`, and the `updated_fields` field of `cmResults` stores any customer profile updates.

In this example, the Boolean values are assigned either true or false even if the rule package did not make any decision. If the "=== true" part is removed, the value would be undefined if no decision was made by the rule package one way or another, which, depending on the situation, could be a valid branch for a **Branching** block.


Assign Data
Assign data to the variables
Variable
Expression
blockCommunication
cmResults.rule_results.block_communication === true
offerServiceResumption
cmResults.rule_results.offer_service_resumption === true
offerSurvey
cmResults.rule_results.offer_survey === true
sendCommunication
cmResults.rule_results.send_communication
requestedAgent
cmResults.rule_results.requested_agent
requestedAgentGroup
cmResults.rule_results.requested_agent_group
requestedSkill
cmResults.rule_results.requested_skill
requestPlaceGroup
cmResults.rule_results.requested_place_group
updatedFields
cmResults.updated_fields
Add...
Remove
Variables
OK
Cancel

Use Case—Frequent Caller Interceptor

Scenario

Jane is a Contact Center Manager. She is responsible for achieving a First Call Resolution rate of X%. In order to accomplish that, Jane needs to:

- Know what the current First Call Resolution Rate is.
- Be able to make changes that will positively impact the rate.

Solution

The proposed Genesys solution is to implement the Conversation Manager Use Case for Frequent Caller Interceptor. This use case will provide a rules package (in GRS) that:

1. Call the IVR to check whether the calling customer has previously called within a specified timeframe; and, if so;
2. Check for a good probability that this call is for the same reason.

The solution will allow Jane to determine what treatment to provide to customers that meet these conditions, such as `Route to Supervisor`, `Route to Proactive Survey`, or to send a specialized, focused Survey at the end of the interaction to find out why the customers had to call back more than once to resolve their issue. Conversation Manager Reports will keep Jane aware of her progress to increase First Call Resolutions.

Creating the Rule in GRAT—Linear Rule Example

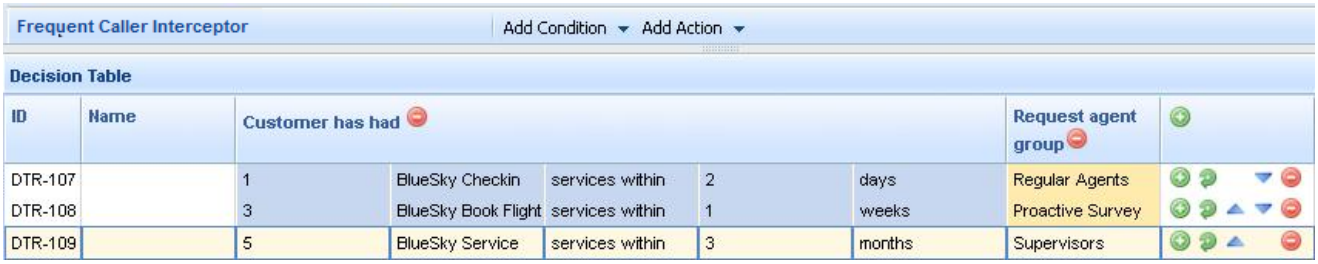
Frequent Caller Interceptor					
Section	**Expression**	**Parameters**			
When					
	Customer has at least	3	services completed within the last	2	weeks
Then					
	Request skill	Customer Care			
	Offer survey to customer	☑			

Narrative

This linear rule is Frequent Caller Interceptor. This is a simpler rule testing only one condition and action.

When the condition "Customer has had 3 services completed within the last 2 weeks" evaluates as `true`, then they are routed to the Customer Care skill group and are offered a survey.

Creating the Rule in GRAT—Decision Table Example



Narrative

This decision table is Frequent Caller Interceptor. It is more complex and has three potential outcomes. It consists of a table of 3 decision table rows: DTR-107, DTR-108 and DTR-109.

- In DTR-107, if the condition "Customer has had 1 BlueSky Check-in service within the last 2 days" evaluates `true` the customer is routed to the `Regular Agents` agent group. If not, DTR-108 is evaluated.
- In DTR-108, if the condition "Customer has had 3 BlueSky Book Flight services within 1 week" evaluates `true`, the customer is routed to the `Proactive Survey` agent group. If not DTR-109 is evaluated.
- In DTR-109, if the condition "Customer has had 5 of any BlueSky service within 3 months" evaluates `true`, the customer is routed to the `Supervisors` agent group. If not, the customer does not meet the Frequent Caller conditions described in the table.

Use Case—Contract Renewal

Scenario

Mobile Operator X is the leading mobile phone services provider in their region. However, recently, more competitors have entered the market and are starting to take customers away. The reasons:

- Newer devices.
- Better pricing.
- Customer satisfaction.

The Conversation Manager Contract Renewal application can be used to mitigate loss of customers to these issues, as much as possible. The Mobile company has a new line of devices and special rate plans to existing customers that may churn. Implementing this application to recognize the customers that have:

- A contract end date of Y days/weeks/months away.
- A specific device.
- Eligibility for the new rate plans.

will enable a special group of agents skilled in "Customer Saves" to know who the customer is, what plan and device they have, know that they are close to contract end, and what offers they can provide to ensure continuity of service and revenue.

A standard set of rules comes with the solution and the Mobile operator needs to make only minimal changes to start using the solution immediately. Conversation Manager Reports show the number of customers identified as Contract Renewal candidates, the treatment provided, and the effectiveness of renewals.

Creating the Rule

Contract Renewal

Sectio	Expression	Parameters						
When								
	Contract will expire within	3	months					
Then								
	Request skill	Sales						

Narrative

This is linear rule Contract Renewal.

When the "Contract end date will expire within 3 months" condition evaluates true, the customer is routed to the Customer Care skill group.

Technical Detail

In order to implement this use case, the customer will have to fetch their `Contract End Date` from their business's back-end database. They can do this using Orchestration Server via Composer's Database, Web Service and other blocks. The `Contract End Date` can be passed into the rules block in our pre-defined Fact called `Contract`. This enables the condition:

```
Contract end date is within "{time}" "{timePeriod}"
```

Use Case—Integrate Data and Decision-Making for Developers

Scenario

A developer who is responsible for ensuring that data and decisions are shared across customer communication channels, needs to perform two or three series of data manipulation in his application in order to move customer information from Context Services, to the Rules System, to the application, and back to Context Services. Digital channels need to have a single approach to handling cross-channel data and central decisions with regard to data handling.

Solution

We can use GRAT to create a decision table checking for various combinations of customer segment, media type, and active services.

Creating the Rule

Integrate Data and Decision Making

ID	Name	Customer segment is	Media type is	Customer has at least one active service of type	Request skill
DTR-120		Gold	chat	BlueSky Book Flight	Sales
DTR-121		Gold	email	BlueSky Book Flight	Internet
DTR-122		Gold	voice	BlueSky Book Flight	Phone
DTR-123		Gold	(*)	BlueSky Checkin	Service
DTR-124		(*)	(*)	(*)	Support

Narrative

This is decision table Integrate Data and Decision Making with 5 rows; DTR-120, to DTR-124. The wildcard values indicate a parameter which is diregarded for evaluation purposes.

- In DTR-120 if the "Customer is Gold and contacts via chat and has an active service type of `BlueSky Book Flight`" condition evaluates `true`, then route them to the `Sales` skill group. If not, evaluate DTR-121.
- In DTR-121, if the "Customer is Gold and contacts via email and has an active service type of `BlueSky Book Flight`" condition evaluates `true`, then route them to the `Internet` skill group. If not, evaluate DTR-122.

- In DTR-122, if the "Customer is Gold and contacts via voice and has an active service type of `BlueSky Book Flight`" condition evaluates `true`, then route them to the Phone skill group. If not, evaluate DTR-123.
- In DTR-123, if the "Customer is Gold and contacts via **any channel** and has an active service type of `BlueSky CheckIn`" condition evaluates `true`, then route them to the `Service` skill group. If not, evaluate DTR-124.
- In DTR-124, if the "Customer is of any type and contacts via any channel and has any active service " condition evaluates `true`, then route them to the `Support` skill group.

Conditions

Condition	Example Usage	Parameters	Description
Time Sensitive			
Today is a work day	Today is a work day	N/A	To use this condition, a business calendar must be associated with the rule. Based on the definition of the business calendar, this condition evaluates true if the current day is a "work day".
It is currently during business hours	It is currently during business hours	N/A	To use this condition, a business calendar must be associated with the rule. Based on the definition of the business calendar, this condition evaluates true if the current time is during business hours (without regard to the day)
Today is a work day and it is currently during business hours	Today is a work day and it is currently during business hours	N/A	To use this condition, a business calendar must be associated with the rule. Based on the definition of the business calendar, this condition evaluates true if the current day and time are on a business day and during business hours
Media-Related			
Media type is `"{mediaType}"`	Media type is `"voice"` Media type is `"chat"` Media type is `"email"`	`mediaType` • fetched from Business Attributes -> Media Type	To use this condition, the media type needs to be passed in a separate fact field, as this value is not passed in on the standard JSON structure from a Query Customer Profile block. The media type can be extracted from the interaction.
Customer-Related			
Usage Notes: To use the following customer-related conditions, the Composer application must use the Query Customer Profile block to retrieve details about the customer, and the result of Query Customer Profile must be assigned to the `CustomerProfile` fact.			

Condition	Example Usage	Parameters	Description
Customer "{contactAttribute}" "{stringOperator}" "{stringValue}"	• Customer Last Name starts with Sh • Customer Zip Code equals 27613 • Customer Country is Canada	contactAttribute • fetched from Business Attributes -> Contact Attributes stringOperator • contains • ends with • equal to • equal to ignore case • starts with stringValue • any string	The rule author can choose any of the defined string fields from the drop-down list. This list is populated from Configuration Server, so will contain any new fields added for the solution. The rule author can choose any of the operators from a drop-down list, and will then type in a value to compare against. The rule author can negate any of the conditions by using the not operator in GRAT, and can group related conditions together using the grouping feature. The Composer application can pass in the result from Query Customer or Identify Customer blocks as a single variable to the rule. This condition will extract the field from the input variable and compare it to the stringValue using the operator specified in stringOperator.
Customer (numeric) "{contactAttribute}" "{operator}" "{numericValue}"	Customer (numeric) "age" is greater than 50 Customer (numeric) "credit score" is less than 500 Customer (numeric) "weight" is less than or equal to 150	contactAttribute • fetched from Business Attributes -> Contact Attributes operator • not equal to • equal to • greater than • greater than or equal to • less than • less than or equal to numericValue • any number	The rule author can choose any of the defined string fields" from the drop-down list. This list is populated from Configuration Server, so will contain any new fields added for the solution. The rule author can choose any of the operators from a drop-down list, and will then type in a value to compare against. The Composer application can pass in the result from Query Customer or Identify Customer blocks as a single variable to the rule. This condition will extract the field from the input variable and compare it to the intValue using the operator specified in operator.
Customer segment is "{customerSegment}"	Customer segment is "Gold"	customerSegment	The rule author can choose any of the

Condition	Example Usage	Parameters	Description
		• Fetched from Business Attributes -> CustomerSegment	defined segments from the drop-down list. This list is populated from Configuration Server, so will contain any new fields added for the solution.
Service-Related			
Usage Notes To use the following Service-Related conditions, the Composer application must: 1) Use the "Query Services" block for all services associated with the customer: a)Identifier = customer ID. b)Service Status = all c) Service Type (unset). 2) Populate the result of "Query Services" into "Services" fact **Note: Certain conditions may not require querying all service status and/or all service types. However, querying all services provide the most flexibility in rules authoring should the business decision changes.**			
Customer has at least one active service	Customer has at least one active service		If the customer has at least one service that is active, this condition evaluates true.
Customer has at least one active service of type "`{serviceType}`"	Customer has at least one active service of type "`Reservation`" Customer has at least one active service of type "`Merchandise Return`"	`serviceType` • Fetched from Business Attributes -> ContextManagementService	If the customer has at least one active service of the type specified, the condition evaluates true.
Customer has at least one service of type "`{serviceType}`" that has completed.	Customer has at least one service of type "`Reservation`" that has completed. Customer has at least one service of type "`Merchandise Return`" that has completed.	`serviceType` • Fetched from Business Attributes -> ContextManagementService	If the customer has at least one service of the type specified that is in completed state, the condition evaluates true.

Customer has at least `{numberOfServices}` services currently active and started within `{time}` "`{timeUnit}`"	Customer has at least 3 services currently active and started within 1 week
	Customer has at least 2 services currently active and started within 24 hours
		`numberOfServices` - integer value > 0
		`time - integer > 0`
		`timeUnit`
		• hours
		• days
		• weeks
		• months
			If the customer has at least the specified number of services currently active, that were all started within the time specified, the condition evaluates true.

Customer has at least `{numberOfServices}` services that completed within the last `{time}` "`{timeUnit}`"	Customer has at least 3 services that completed within the last 2 weeks Customer has at least 2	`numberOfServices` - integer value > 0 `time - integer > 0`	If the customer has at least the specified number of services that all completed within the time specified, the

Condition	Example Usage	Parameters	Description
	services that completed within the last 5 days	`timeUnit` • hours • days • weeks • months	condition evaluates true.

Customer has at least `{numberOfServices}` services of type “`{serviceType}`” currently active and started within `{time}` “`{timeUnit}`”	Customer has at least 3 services of type “`Reservation`" currently active and started within 1 month
	Customer has at least 2 services of type "`Product Defect`" currently active and started within 3 days.
	Customer has at least 2 services of type "`Complaints`" currently active and started within 90 days
		`numberOfServices` - integer value > 0
		`serviceType`
		• Fetched from Business Attributes -> ContextManagementService
		`time - integer > 0`
		`timeUnit`
		• hours
		• days
		• weeks
		• months
			If the customer has the specified number of services, of the given type, currently active and all started within the time specified, the condition evaluates true.

Customer has at least `{numberOfServices}` services of type “`{serviceType}`” completed within the last `{time}` “`{timeUnit}`”	Customer has at least 2 services of type "`Airline Reservation`" completed within the last 2 months
	Customer has at least 5 services of type "`Complaint`" completed within the last 180 days
		`numberOfServices` - integer value > 0
		`serviceType`
		• Fetched from Business Attributes -> ContextManagementService
		`time - integer > 0`
		`timeUnit`
		• hours
		• days
		• weeks
		• months
			If the customer has the specified number of services, of the given type, that all completed within the time specified, the condition evaluates true.

Customer had last completed "`{serviceType}`" service occur within `{time}` "`{timeUnit}`"	Customer had last completed "`Complaint`" service occur within 7 days	`serviceType` • Fetched from Business Attributes ->	If the customer's last completed service occurred on or before the time specified, the condition evaluates true.

Condition	Example Usage	Parameters	Description

	Customer had last completed "Reservation" service occur within 1 month	ContextManagementService
		`time - integer > 0`
		`timeUnit`
		• hours
		• days
		• weeks
		• months	

The number of active services associated with this customer is "{operator}"{numberOfServices}.	The number of active services associated with this customer is greater than 5
	The number of active services associated with this customer is less than 3
		`operator`
		• not equal to
		• equal to
		• greater than
		• greater than or equal to
		• less than
		• less than or equal to
		`numberOfServices integer >= 0`
			If the number of active services associated with this customer matches the condition specified (eg, "greater than 5", "equal to 10"), the condition evaluates true.

The number of completed services associated with this customer is “{operator}” {numberOfServices}.	The number of completed services associated with this customer is greater than 5.
	The number of completed services associated with this customer is less than 3.
		`operator`
		• not equal to
		• equal to
		• greater than
		• greater than or equal to
		• less than
		• less than or equal to
		`numberOfServices integer >= 0`
			If the number of completed services associated with this customer matches the condition specified (eg, "greater than 5", "equal to 10"), the condition evaluates true.

The total number of services associated with this customer is “{operator}” {numberOfServices}	The total number of services associated with this customer is less than 3	`operator` • not equal to • to • greater than • greater than or equal to	If the number services (active or completed) associated with this customer matches the condition specified (eg, "greater than 5", "equal to 10"), the condition evaluates true.

Condition	Example Usage	Parameters	Description
		• less than • less than or equal to `numberOfServices` integer >= 0	
State-Related			
Usage Notes: To use the following State-Related conditions, the Composer application must: 1) Use the “Query States” block for all services associated with the service: a) Service ID = service in question; b)State Status = all; c) State Type (unset); 2) Populate the result of “Query States” into “States" fact of Rule Block. Note: Certain conditions do not require querying all state statuses and/or all state types. However, querying all states provides the most flexibility in rules authoring should the business decision changes.			
Service is currently in "`{state}`" state.	Service is currently in "`Offering Callback`" state Service is currently in "`Collection`" state	`stateType` • Fetched from Business Attributes -> ContextManagementState	This condition will examine the active state related to the Service object and compare it to the selected value. If it matches, the condition will evaluate true. Example: Service "Travel Reservation" could have states Query Airfares (completed) Reserve flights (completed) Make payment (active) The following condition would evaluate true: Service is currently in "Make payment" state.

Service has completed state "`{state}`" within `{time}` "`{timeUnit}`"	Service has completed state "Delivering Callback" within 5 days.
	Service has completed state "Payment" within 24 hours
		`stateType`
		• Fetched from Business Attributes -> ContextManagementState
		`time - integer > 0`
		`timeUnit`
		• hours
		• days
		• weeks
		• months
			This condition will examine the list of "states" that are provided in the Service object. If there is at least one state of the specified "type" that has completed within the specified time range, it will evaluate true.

Service has been in "`{state}`" state for at	Service has been in "`Pending Payment`"	`stateType`	This condition will examine the list of

Condition	Example Usage	Parameters	Description
least {time} "{timeUnit}"	state for at least 1 week. Service has been in "Confirm Reservation" state for at least 24 hours	• Fetched from Business Attributes -> ContextManagementState time - integer > 0 timeUnit • hours • days • weeks • months	"states" that are provided in the Service object. If the specified state type has been active for at least the specified time range, it will evaluate true.
Task-Related			
"Usage Notes: To use the following Task-Related conditions, the Composer application must: 1) Use the Query Tasks block for all tasks associated with the service/state - this means a)Service ID = service in question: b)State ID = state in question, if tasks are associated with state; c) Task Status = all; d) Task Type (unset); 2)Populate the result of Query Tasks into “Tasks" fact of the Rule Block. **Note: Certain conditions do not require querying all task statuses and/or all task types. However, querying all tasks provides the most flexibility in rules authoring should the business decision change.**			
Task "{task}" is active	Task "Make Payment" is active Task "Pay Taxes" is active	taskType • Fetched from Business Attributes -> ContextManagementTasks	This condition will examine the list of active "tasks" that are provided in the State object. If there is an active task of the specified "type", then the condition will evaluate true.
Task "{task}" has been completed	Task "Make Payment" has been completed Task "Pay Taxes" has been completed	taskType • Fetched from Business Attributes -> ContextManagementTasks	This condition will examine the list of completed "tasks" that are provided in the State object. If there is a completed task of the specified "type", then the condition will evaluate true.
Task "{task}" has been completed within {time} "{timeUnit}"	Task "Call Customer" has been completed within 1 day Task "Process Payment" has been completed within 1 week	taskType • Fetched from Business Attributes -> ContextManagementTasks time - integer > 0 timeUnit	This condition will examine the list of completed "tasks" that are provided in the State object. If there is a completed task of the specified "type" that completed within the specified time range, it will evaluate true.

Condition	Example Usage	Parameters	Description
		• hours • days • weeks • months	
Task "{task}" has been active for at least {time} "{timeUnit}"	Task "Call Customer" has been active for at least 8 hours Task "Mail check" has been active for at least 6 months	taskType • Fetched from Business Attributes -> ContextManagementTask time - integer > 0 timeUnit • hours • days • weeks • months	This condition will examine the list of tasks" that are provided in the State object. If there is at least one task of the specified "type" that has been active for at least the specified time range, it will evaluate true.
Miscellaneous			
Usage Notes: The concept of a "Contract" is abstract and will vary for different customers using the CM Templates. If the user wants to test for contract expiration, they can retrieve the actual contract via the Orchestration application (for example, database fetch, web services, and so on) and then pass in the end date in the "Contract" fact. This allows the rule author to test the end date and integrate this condition in with others (Customer, Service, State, Task related).			
Contract will expire within {time} "{timeUnit}"	Contract will expire within 7 days Contract will expire within 3 months	time - integer > 0 timeUnit • hours • days • weeks • months	"Contract" will have to be defined as a separate fact. The customer will have to map their actual contract object end date (obtained from their back-end databases using Orchestration or other techniques) to the Contract fact and pass it in. We can then examine the end date passed in and determine if it is within the range specified.

Actions

Action	Example Usage	Parameters	Description
Update Customer Profile "{contactAttribute}" to "{stringValue}"	Update Customer Profile "City" to "Raleigh" Update Customer Profile "Country" to "USA"	contactAttribute • fetched from Business Attributes -> Contact Attributes stringValue • any string	Allows rule to pass back a new value for one or more Customer Profile fields to the invoking application. The invoking application perform the update using "Update Customer Profile". The updated fields are returned in the following structure: "rule_results" : { "updated_fields" : "["City", "Raleigh", "Country", "USA"] }
Update Customer Profile (numeric) "{contactAttribute}" to {numericValue}	Update Customer Profile (numeric) "age" to 55 Update Customer Profile (numeric) "credit score" to 500	contactAttribute • fetched from Business Attributes -> Contact Attributes numericValue • any number	Allows rule to pass back a new value for one or more Customer Profile fields to the invoking application. The invoking application perform the update using Update Customer Profile. The updated fields are returned in the following structure: "rule_results" : { "updated_fields" : "["age", 55, "credit score", 500] }
Request specific agent "{agent}"	Request specific agent "Fred Flintstone" Request specific agent "Betty Rubble"	agent • List of agents fetched from Configuration Server	Allows rule to pass back a specific agent to the invoking application for processing. The requested agent is returned in the following structure: "rule_results" : { "requested_agent", "Betty Rubble }
Request agent group	Request agent group	agentGroup	Allows rule to pass back

Action	Example Usage	Parameters	Description
"{agentGroup}"	"Customer Retention" Request agent group "Widget Service"	• List of agent groups fetched from Configuration Server	a specific agent group to the invoking application for processing. The requested agent group is returned in the following structure: "rule_results" : { "requested_agent_group", "Customer Retention" }
Request place group "{placeGroup}"	Request place group "Widget Sales" Request place group "San Francisco Office"	placeGroup • List of place groups fetched from Configuration Server	Allows rule to pass back a specific place group to the invoking application for processing. The requested place group is returned in the following structure: "rule_results" : { "requested_place_group", "Widget Sales" }
Request skill "{skill}"	Request skill "Spanish" Request skill "Installations"	skill • List of skills fetched from Configuration Server	Allows rule to pass back a specifically requested skill to the invoking application for processing. The requested skill is returned in the following structure: "rule_results" : { "requested_skill", "Spanish" }
Send communication to customer via "{mediaType}"	Send communication to customer via "Email" Send communication to customer via "Voice"		Allows rule to pass back an indication that further communication with the customer is permissible. The result is returned in the following structure: "rule_results" : { "send_communication", "Email" }
Block communication to customer	Block communication to customer		Allows rule to pass back an indication that further communication with the customer should be blocked. The result is returned in the

Action	Example Usage	Parameters	Description
			following structure: `"rule_results" : { "block_communication", "true" }`

Offer Service Resumption `{offerToResume}`	Offer Service Resumption `"true"`
	
	Offer Service Resumption `"false"`
	
	Note: The GUI will render a "checkbox" which can be checked or unchecked by the user
		`offerToResume`
		
		• boolean value
			Allows rule to pass back an indication that the customer should be offered an option to resume an open/existing service or not.
			
			`"rule_results" : { "offer_resumption", "true" }`
			
			or...
			
			`"rule_results" : { "offer_resumption", "false" }`

Offer Survey to Customer `{offerToSurvey}`	Offer Survey to Customer `"true"`
	
	Offer Survey to Customer `"false"`
	
	Note: The GUI will render a "checkbox" which can be checked or unchecked by the user
		`offerToSurvey`
		
		• boolean value
			Allows rule to pass back an indication that the customer should be offered a survey or not.
			
			`"rule_results" : { "offer_survey", "true" }`
			
			or...
			
			`"rule_results" : { "offer_survey", "false" }`

<disqus> </disqus>