



This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

## Example - VXML Application

Genesys Rules System 8.1.4

# Table of Contents

<b>VXML Example</b>	<b>3</b>
Business Structure	4
Rule Template	5
Supporting Building Test Scenarios	8
Rule Package	11
Rule Evaluation	13

# VXML Example

This page illustrates how template development and rule authoring might be handled for a VXML application to call GRE for rule evaluation.

## Use Case

We want to create a VXML self-service application for our company, ACME Corporation. Within that application, we will collect information from the customer that will allow us to determine the customer's segment (that is, is the customer a Bronze, Silver, Gold, or Platinum customer), as well as the value of an order (in American dollars) that the customer has placed with our company.

Based on the values for the customer segment and the order, we will use predefined business rules to determine whether to play a prompt to the customer that offers them a special promotion. In other words, the logic that will determine whether the special offer should be made to the customer will be defined within the business rules themselves, and not within the VXML application.

This example does not describe how the logic would be created in the VXML application to collect information from the customer, look up related information in a customer database (for example, to establish the value of the customer's order), or play the prompt to the customer. It just demonstrates the use of business rules to supply the necessary information to the client application—in this case the VXML application—to allow the application to take the correct next step.

# Business Structure

The business structure of our organization is defined under our tenant in Configuration Server. It consists of a single entity that is called "ACME Solution."

Under this Solution there are two departments:

- Finance Department
- Sales Department

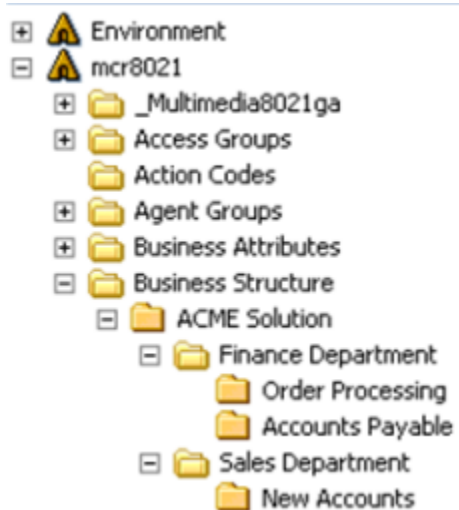
Under the Finance Department there are two processes:

- Order Processing
- Accounts Payable

Under the Sales Department there is a single process:

- New Accounts

The figure below shows the business structure as it appears in Configuration Manager. You can also manage the business structure in Genesys Administrator, although it does not appear as a hierarchical tree on the Administrator GUI.



Business Structure

# Rule Template

The rule template that is created for this example consists of two facts:

- `_GRS_Environment`
- `Customer`

The `_GRS_Environment` fact is a mandatory fact for all Genesys Rules Systems rule templates. It is used to establish two important fact properties:

- `businessContext_Level*`—Used in the request to the Rules Engine, to determine the node(s) of the business structure at which to evaluate rules
- `phase`—Used within the request to the Rules Engine, to determine which rules to evaluate.

Each rule that you create in Genesys Rules System must have a rule phase defined. The list of rule phases can be modified by changing the values of the enumeration that is called Phases, in the rule template. In this example, the phase that is selected is called segmentation, so we can assume that the values for the Phases enumeration contains at least one value called segmentation, and possibly others.

The `Customer` fact contains three properties that we will use in our business rule:

- `segment`
- `order`
- `offer`

Our rule template contains two conditions and one action, as well as the necessary parameters that are used within these conditions and actions. See the two following tables for details of these parameters.

## Rule Language Mapping

**Rule Language Mapping Parameters**

Name	Language Expression	Rule Language Mapping
Segment	Customer segment is {customerSegment}	Customer(segment=='{customerSegment}')
OrderValue	Order value is greater than {orderValue}	Customer(order>{orderValue})
SpecialOffer	Offer special promotion {specialOffer}	\$Customer.offer='{specialOffer}'

**Language Expression Details**

Name	Type	Comments
customerSegment	Enumeration	An enumeration must be created in the rule template that contains the values for Customer Segment from which the rules author will be able to select (for example, Bronze, Silver, and so on). Note that there are two properties that you must provide for each value of the enumeration: Name and Label. The Label is what will appear to the business rules author when the business rules author is using a rule condition or action that includes a parameter that references this enumeration. The Name is what is used in the request/response to/from the Rules Engine; therefore, case is important. For example, you may want to use uppercase for the labels of these enumeration values, and lowercase for the names. shows an example of how that might appear in the Genesys Rules Development Tool:
orderValue	Input Value (Numeric)	Optionally, you can supply upper and lower bounds for this parameter. If these are supplied in the template, the rules author will be constrained as to the values the rules author can provide in the rule condition that uses this parameter.
specialOffer	Input Value (Boolean)	Because the parameter type is Boolean, this will present a checkbox to the rules author when this parameter is used in the rule action.

The figure below shows how the enumeration is configured.

**Enumeration Details**

Name:

Description:

**Values**

Name	Label
bronze	Bronze
gold	Gold
platinum	Platinum
silver	Silver

Enumeration Details

Note that for this template, because the `orderValue` parameter is numeric, when it is used in a rule condition, there are no single quotation marks (") surrounding it in the rule language mapping, whereas there are single quotation marks surrounding the `customerSegment` string parameter.

With the Drools language you cannot set the value of a Fact property by referring to the Fact's name. In the condition section you must first declare a variable and associate this variable with a Fact object. Once this association has been made within a condition then the variable can be used in actions (and other conditions) to reference fields contained within the fact. A period (".") is used to access the fields on a fact. Use a colon (":") when you want to create a variable in a condition. So, in the preceding example, the "." is used in the rule language mapping for the action (`$Customer:Customer()` in the condition, `$Customer.offer` in the action).

Beginning with the 8.1.2 release, conditions are automatically added to declare variables which are referenced in actions. For a variable to be automatically declared, the variable name must be the name of the fact preceded by a '\$' sign. So in this example, `$Customer` is referenced in an action so the condition `$Customer:Customer()` will automatically be added to the rule.

Prior to 8.1.2, variables must be declared by a condition within the template and added to the rule by the rule developer. With this example the following generic rule condition needs to be defined within the template and the rules developer would add this condition to any rule that referenced the `$Customer` variable. Note: A variable cannot be declared twice.

Language Expression: Customer exists  
Rule Language Mapping: `$Customer:Customer()`

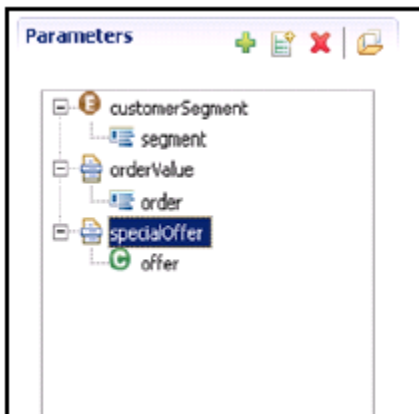
# Supporting Building Test Scenarios

To support building test scenarios from release 8.1.2, the rule developer should provide a mapping between the parameters (which the rule author is familiar with) and the underlying fact model. In this way, when the rule author provides a sample value for say, `orderValue`, GRAT will know how to build the appropriate Fact object to run the test.

In this case, it would create a Customer fact and set the order field to the specified value. For this example, we will map the parameters to the Fact model in the following way:

- `customerSegment` -> `Customer.segment`
- `orderValue` -> `Customer.order`
- `specialOffer` -> `Customer.offer`

In GRDT, right-click on each parameter and choose **Associate Property**. Then choose the appropriate Fact and field from the pop-up window.



Mapping Parameters Popup Window

## Mapping Parameters Window

Navigate to the **Test Scenarios** tab and create a test scenario to test our decision table rule at the Finance Department node. Select test values `customerSegment` and `orderValue` from the **Add Given** drop-down. Then select `specialOffer` from the **Add Expectation** drop-down.

Now, insert rows of data. In these rows you can put some test values and also choose what your predicted or expected result should be.

When you click on the **Run Test Scenario** button, these test values will be passed into the rule package and the result will be compared to your expectations. If they match, you will see a green check mark in the **Results** column.



Note that we purposely passed in data that we predicted would return a positive result (for example, the customer gets the special offer) as well as a negative result (for example, the customer does not qualify). These test scenarios are then saved and can be executed in the future when rules are added or modified.

Test Scenarios								
ID	Name	Description	Phase	Business Hierarchy	Simulated Date	Simulated Time	Time Zone	Result
TS-108	Finance	Finance Rules	segmentation	Finance Department			Greenwich Mean Time	
<div>  New Test Scenario            Run Test Scenario            Run All            Import         </div> <div>           Add Given ▼ Add Expectation ▼         </div>								
ID	Name	Results	{customerSegment}	{orderValue}	{specialOffer}			
TSR-114			Platinum	4125	<input checked="" type="checkbox"/>			
TSR-109			Gold	5555	<input checked="" type="checkbox"/>			
TSR-110			Silver	5555	<input type="checkbox"/>			
TSR-115			Bronze	9000	<input type="checkbox"/>			

Test Scenario Tab 1








Note, the 4th row of the table, shows our Bronze customer with an order value of 9000 NOT receiving a special offer. This is because the test was run against the Finance Department node of the hierarchy. In our example, we added a linear rule to the Accounts Payable department which addresses the Bronze customer.

We can now create a new test scenario which targets the Accounts Payable department and validates that, in this case, the Bronze customer gets an offer. In our new test scenario (TS-116), we set the Business Hierarchy to the Finance Department > Accounts Payable department. We copy the same test data and when we run it, notice that the Bronze customer shows an unsuccessful result when our expectation is that they do NOT receive an offer (Figure 24).

Test Scenarios							
ID	Name	Description	Phase	Business Hierarchy	Simulated Date	Simulated Time	Time Zone
TS-108	Finance	Finance Rules	segmentation	Finance Department			Greenwich Mean
 TS-116	Finance	Finance Rules	segmentation	Finance Department > Accounts Payable			Greenwich Mean
<div>     </div> <div> Add Given ▼ Add Expectation ▼ </div>							
ID	Name	Results	{customerSegment} -	{orderValue} -	{specialOffer} -		
TSR-117		✓	Platinum	4125	<input checked="" type="checkbox"/>	  	
TSR-118		✓	Gold	5555	<input checked="" type="checkbox"/>		
TSR-119		✓	Silver	5555	<input type="checkbox"/>		
TSR-120		✗	Bronze	9000	<input type="checkbox"/>		

Test Scenario Tab 2

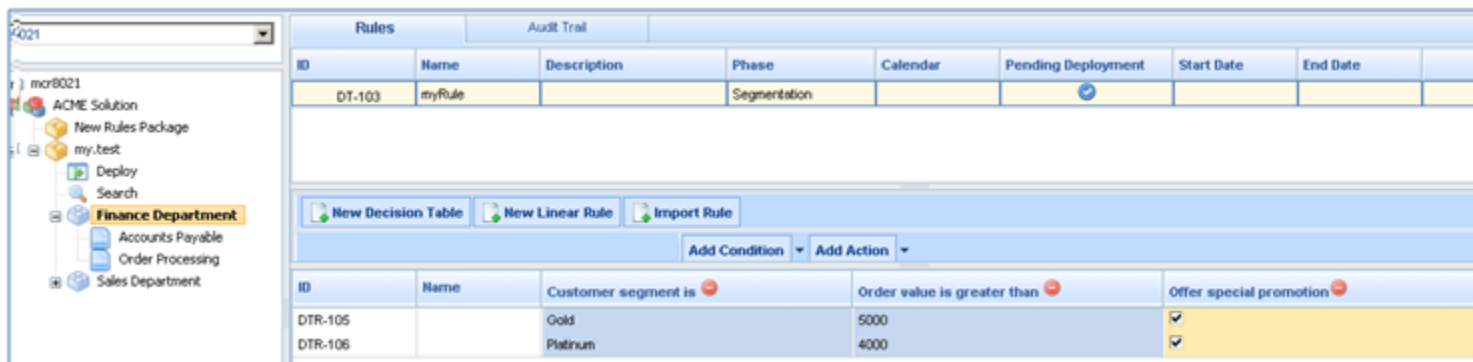
We simply adjust the test scenario so that we now expect an offer for this customer by checking the specialOffer box. We now get a successful result when running the test.

Test Scenarios							
ID	Name	Description	Phase	Business Hierarchy	Simulated Date	Simulated Time	Time Zone
TS-108	Finance	Finance Rules	segmentation	Finance Department			Greenwich Mean
TS-116	Finance	Finance Rules	segmentation	Finance Department > Accounts Payable			Greenwich Mean
<div>     </div> <div> Add Given ▼ Add Expectation ▼ </div>							
ID	Name	Results	{customerSegment} -	{orderValue} -	{specialOffer} -		
TSR-117		✓	Platinum	4125	<input checked="" type="checkbox"/>	  	
TSR-118		✓	Gold	5555	<input checked="" type="checkbox"/>		
TSR-119		✓	Silver	5555	<input type="checkbox"/>		
TSR-120		✓	Bronze	9000	<input checked="" type="checkbox"/>		

Test Scenario Tab 3

# Rule Package

The rule package that is created for this example is called `my.test`. Three rules are defined within the package. Two of the rules are defined as two rows of a single Decision Table, which is created at the Finance Department node of the business structure.



Decision Table

The rule checks two conditions:

- The value of the segment property of the Customer fact
- The value of the order property of the Customer fact

If the conditions are all true, the rule will fire a single action, which is to set the value of the offer property of the Customer fact to 1.

The third rule is defined as a linear rule and has been created at the Accounts Payable node of the business structure.

The screenshot displays the Genesys Rules Engine interface. On the left is a tree view of the business structure, including nodes like 'my.test', 'Finance Department', 'Accounts Payable', 'Order Processing', and 'Sales Department'. The main area is titled 'Rules' and contains a table with columns: ID, Name, Description, Phase, Calendar, Pending Deployment, Start Date, and End Date. A single rule is listed: 'Rule-117' with name 'myRule2', description 'Segmentation', and phase 'Segmentation'. Below the table are buttons for 'New Decision Table', 'New Linear Rule', and 'Import Rule'. Further down are buttons for 'Add Condition', 'Add Action', and 'Group'. At the bottom is a table for rule conditions and actions.

ID	Name	Description	Phase	Calendar	Pending Deployment	Start Date	End Date
Rule-117	myRule2	Segmentation					

Section	Expression	Parameters
When	Customer segment is	Bronze
	Order value is greater than	8000
Then	Offer special promotion	<input checked="" type="checkbox"/>

Linear Rule

This rule checks the same conditions—and has the same action—as the rules that are defined at the Finance Department node. Normally, you might expect this rule to be a third row in the earlier decision table, at the Finance Department node. It is included here only to demonstrate how the rules at different nodes in the business structure can be evaluated.

The my.test rule package is deployed to the Genesys Rules Engine or, in release 8.1.2, an application cluster. When two or more conditions are listed, there is an implied “and” between them. So, this rule is saying that “when the customer segment is bronze and the order value is greater than 8000, then offer special promotion”. The rule author can also choose other logical operators, such as OR, NOT, AND NOT, and so on.

# Rule Evaluation

We want to call the GRE from our client (VXML) application. For the rule to be evaluated properly, we will have to populate the fact properties of the `_GRS_Environment` and `Customer` facts correctly.

To test this rule evaluation, you can use a REST client, such as the free Firefox REST Client add-on, or you can test the rule by using Composer's Business Rule Block, which has a built in Test feature that provides sample values to the rule and evaluates the results.

In most cases, you will use Genesys Composer to build applications that will invoke the GRE. However, to simplify rule testing, it might be more convenient to use a REST client in the manner that is described here.

The request to the Rules Engine will be a POST request. The URL we will use to make the POST request will be constructed as follows:

```
http://[server:port]/genesys-rules-engine/knowledgebase/[package]
```

where: `server` is the IP address or host name of the application server on which the rules engine is running `port` is the listening port of the application server. For example, 8080 is the default Tomcat port. `package` is the name of the rule package to evaluate. In this example it is `my.test`.

So, the URL might look like this:

```
http://myserver:8080/genesys-rules-engine/knowledgebase/my.test
```

We have to populate the request body with the request, in XML format. In the request body we specify the two fact classes, both of which are prefixed with the package name; for example, `my.test._GRS_Environment` and `my.test.Customer`, respectively.

For the `_GRS_Environment` fact, we have to provide values for the fact properties `phase` and `businessContext_Level*`. Note that your request can include multiple values for the `businessContext_Level*` fact property, depending on the node(s) of the business structure at which you want the Rules Engine to evaluate rules.

In our case, let us assume that in this request, we want the Rules Engine to evaluate the rules at both the Finance Department level and the Accounts Payable level. In this case, in our request we will populate fact properties that specify both of these levels (`businessContext_Level1` and `businessContext_Level2`). Alternatively, if we omitted `businessContext_Level2` from the request, we could ask the Rules Engine to evaluate only the rules at the Finance Department level, which is

businessContext\_Level1.

Note also that if you had any rules configured at the “global” level (which are configured for the rule package itself by selecting the name of the package in the navigation tree, and then selecting the Rules tab), they will always be evaluated for every request, without having to specify anything explicitly in the `_GRS_Environment` fact property.

The other `_GRS_Environment` fact property that we must populate in the request is the phase. In our example, all rules were written for the segmentation phase.

For the Customer fact, we must provide values for the fact properties segment and order. We can provide whatever values we want, in order to test the results of the rule evaluation. Note that the value that you provide for the segment fact property is case-sensitive, as is the value for the phase fact property. See the description of the customerSegment enumeration in **Rule Template**.

The following is an example of the request body:

```
<knowledgebase-request>
  <inOutFacts>
    <named-fact>
      <id>env</id>
      <fact class="my.test._GRS_Environment">
        <phase>segmentation</phase>
        <businessContext_Level1>Finance
Department</businessContext_Level1>
        <businessContext_Level2>Accounts
Payable</businessContext_Level2>
      </fact>
    </named-fact>
    <named-fact>
      <id>customer</id>
      <fact class="my.test.Customer">
        <segment>gold</segment>
        <order>6345.32</order>
      </fact>
    </named-fact>
  </inOutFacts>
</knowledgebase-request>
```

Based on our rule configuration, we would expect that the Rule Engine would return a value of 1 for the offer property of the Caller fact, indicating that under these conditions (customer is Gold and the customer’s order value is greater than \$5,000.00), we want to offer them a special promotion. This is because the parameter (specialOffer) that is being used in the rule action is a Boolean type. In this case, the response body will look like the following:

```
<knowledgebase-response>
  <inOutFacts>
```

```

        <named-fact>
            <id>env</id>
            <fact class="my.test._GRS_Environment">
                <businessContext__Level2>Accounts
Payable</businessContext__Level2>
                <businessContext__Level1>Finance
Department</businessContext__Level1>
                <phase>segmentation</phase>
            </fact>
        </named-fact>
        <named-fact>
            <id>customer</id>
            <fact class="my.test.Customer">
                <order>6345.32</order>
                <segment>gold</segment>
                <offer>1</offer>
            </fact>
        </named-fact>
    </inOutFacts>
    <executionResult>
        <rulesApplied>
            <string>Row 1 DT-103 myRule</string>
        </rulesApplied>
    </executionResult>
</knowledgebase-response>

```

If you pass in values in your request that the Rules Engine will not evaluate to true, based on all of the rules that you have deployed, no value for the offer fact property will be returned in the result. For example, if you set the value of order to 2345.32, the response body will look like the following:

```

<knowledgebase-response>
    <inOutFacts>
        <named-fact>
            <id>env</id>
            <fact class="my.test._GRS_Environment">
                <businessContext__Level2>Accounts
Payable</businessContext__Level2>
                <businessContext__Level1>Finance
Department</businessContext__Level1>
                <phase>segmentation</phase>
            </fact>
        </named-fact>
        <named-fact>
            <id>customer</id>
            <fact class="my.test.Customer">
                <order>2345.32</order>
                <segment>gold</segment>
            </fact>
        </named-fact>
    </inOutFacts>
    <executionResult>
        <rulesApplied>
        </rulesApplied>
    </executionResult>
</knowledgebase-response>

```

Note that this is not the same as the value of offer being 0. In this example, because all of the conditions in the rules were not met (evaluated as true by the Rules Engine), the action was not fired. Thus, offer has no value populated in the result. If you wanted the value of offer to be set to 0, you would have to have a rule that included a rule action whereby the value of offer was unchecked

(remember that it is a Boolean parameter so it is either checked or unchecked by the rules author). If all of the conditions of such a rule were evaluated as true by the Rules Engine, the result would set offer to 0.

If you want the response to include the offer fact property, with no value, it must be included in the request (even if no value is provided). In this case the `my.test.Customer` fact class would look like the following in the request:

```
<named-fact>
  <id>customer</id>
  <fact class="my.test.Customer">
    <segment>gold</segment>
    <order>2345.32</order>
    <offer></offer>
  </fact>
</named-fact>
```

And the response body would include the following section:

```
<named-fact>
  <id>customer</id>
  <fact class="my.test.Customer">
    <order>2345.32</order>
    <segment>gold</segment>
    <offer></offer>
  </fact>
</named-fact>
```

You can also try populating the request with values that will be relevant to the rule at the Accounts Payable level of the business structure—for example, `segment = bronze` and `order = 9345.33`. In this case, you should also see the value of `order` set to 1 in the response body.