Genesys Rules Authoring Tool Help

Genesys Rules System 9.0.0

1/10/2022

Table of Contents

Genesys Rules Authoring Tool Help

The Genesys Rules Authoring Tool (GRAT) is a browser-based application that enables you to create and edit business rules, and, in release 9.0, develop the templates that rules are based on. (These were formerly developed in Genesys Rules Development Tool, a separate Eclipse-based component.)

Business rule template developers use this tool to create the underlying templates on which rules and rule packages are built. Rule templates can also be developed externally and imported directly into GRAT.

Business rule authors use this tool to create, edit, or delete rules, and deploy them to either production or lab systems.

All the functionality described here is available only to users who have the relevant permissions configured using role-based access control (RBAC). Consult your system administrators if you do not have the permissions you need.

Quick links

Overviews, descriptions & summaries

Rules (linear & decision tables)
Rule packages
Business calendars
Rule template development
Rule testing

Working with linear rules

Creating Linear Rules
Updating Linear Rules
Copying Linear Rules
Importing Linear Rules
Exporting Linear Rules
Deleting Linear Rules

Working with decision tables

Creating Decision Tables
Updating Decision Tables
Copying Decision Tables

Working with rule packages 1

Creating rules packages
Editing rules packages
Importing rules packages

Importing Decision Tables

Exporting rules packages

Working with rule packages 2

Importing rules templates
Exporting rules templates
Check my permissions
Displaying package history
Audit trail

Working with business calendars

Creating business calendars
Copying business calendars
Deleting business calendars

Working with templates

Importing and exporting templates
Creating and editing templates
Publishing templates
Examples of template development
Rule language mapping
Using Drools 5

Working with template editors

Actions editor
Conditions editor
Enumerations editor
Fact Model editor
Functions editor
Parameters editor

Rule testing

Using split testing to compare business outcomes

New in GRAT release 9.0

The functionality of both Genesys Rules Authoring Tool and the previous Genesys Rules Development Tool are combined into a single component—GRAT. You can still use 8.1 GRDT to develop templates and import them into the new GRAT component, but 8.1 GRDT no longer ships with the new GRAT.

UI branding elements are brought into line with other Genesys software.

A new configuration option—**Disable Package Serialization**—appears on the **General** tab of a rule package. When this option is enabled (value is set to `false` (default)), low-use rule packages are serialized

out and can then be re-loaded quickly on demand.

- Undo/re-do buttons have been added to the user interface to assist in editing rules, calendars and test scenarios.
- Role-Based Access Control (RBAC) features are extended to the Rules Development functionality in GRAT that was formerly in GRDT.
- Known Limitation—Some functionality related to complex event processing and parameter mapping is not implemented in the intial 9.0 release. The result of this is that there is no provision for creating or editing templates that support rule testing scenarios and split testing. However, you can still import such templates that have been created in 8.1 GRDT and base rules and packages on them, but you can't edit or create the templates in the initial release.

Rules Overview

A business rule is a piece of logic that defines, on a small scale, what a business does. For the Genesys Rules System, a rule is an external piece of logic that can be customized by business analysts, and invoked by applications. This allows you to tune specific business behaviors as needed.

Types of Rule

GRAT allows you to configure two types of rules: linear and decision tables.

Linear rules

Linear rules follow the following basic format:

```
WHEN {condition} THEN {action}
```

When the condition is true, the action will occur. This form of rule is best for simple actions, such as assigning a value to return back to the application. Note, however, that linear rules can have multiple conditions and actions, or only actions with no conditions. The conditions and actions that are available depend upon the rule templates that are included in the rule package.

Example

Purpose
If a customer's age is within the range of 30-40 years, the customer's interaction will be routed to Agent Group 1.

In the Genesys Rules Authoring Tool, create a new linear rule. Enter the name, phase, and so on, as desired, and then add a condition and an action. The phases from which the rules author can select are dictated by the rule template that the rules author is using.

There is an enumeration called Phases within the `_GRS_Environment` fact, that will be created whenever a new rules template project is created in the Rules Development tab. If the Phases enumeration is not present, the rules author will simply see * in the **Phase** dropdown. In this case, Phase will not be considered when evaluating the rule package.

> **Important**
>
> The `_GRS_Environment` Fact must be provided for all rule evaluations. An empty `_GRS_Environment` fact must be provided for rules at the package level that do not use a Phase (that is, the Phase was defined to be *).

The Add Condition and Add Action drop-down lists are populated with all of the conditions and actions that were created in the rule templates that are included in the rule package. The drop-down lists

contain the language expressions that the rule developers used during creation of the components, and not the rule language mapping. This makes it possible to create rules without knowing the rule language mapping or being familiar with Drools.

The parameters that are contained in each condition and action are represented by the names that are entered for them. The business rule author must replace this name either by entering a value (such as for an age range) or by selecting an option from the drop-down list (such as for an Agent Group).

So, to create this rule, the rules author would select `Age Range` as the condition and enter `30` as the `{ageLow}` parameter and `40` as the `{ageHigh}` parameter. The action would be `Target Agent Group`, and `Agent Group 1` would be selected from the `{agentGroup}` drop-down list.

More information

The following topics explain how to work with linear rules in GRAT:

- **Creating Linear Rules**
- **Updating Linear Rules**
- **Copying Linear Rules**
- **Importing Linear Rules**
- **Exporting Linear Rules**
- **Deleting Linear Rules**

Decision tables

Decision tables allow you to join a number of Linear Rules with the same set of conditions (when) and actions (then) to be used for a complex (structured) business case. Use decision tables to avoid dozens of linear rules with identical structure in the system.

> ### Important
>
> 1. Choices in decision tables need to be mutually exclusive to avoid ambiguity. This ensures that there is only one outcome per evaluation. If the choices are not mutually exclusive, multiple rows can be executed, in no guaranteed order. The last row executed will determine the final result.
> 2. The maximum number of columns supported in Decision Tables is 50.
> 3. When you are editing rules, be careful not to clear your cookie data, as this might cause the rule to be lost. Consult the documentation for the browser that you are using for more information about how to prevent a user from clearing cookie data.

More information

The following topics explain how to work with Decision Tables in GRAT:

- **Creating Decision Tables**
- **Updating Decision Tables**
- **Copying Decision Tables**
- **Importing Decision Tables**
- **Exporting Decision Tables**
- **Deleting Decision Tables**

Order of Execution

You can configure rules for various business contexts (nodes representing the various elements in your business structure hierarchy), or, for global rules, at the rule package level. In the navigation panel, each business context within the configured business structure is represented at a different node level. The order of execution of rules within a rule package depends on the node level: rules execute first at package/global level, then at each level of the hierarchy in turn.

So if you have defined this hierarchy:

- Package
  - Sales Department
    - Finance

and during execution, you specify “Sales Department” / “Finance”, then the order of execution is:

1. Rules at Package level (according to priority).
2. Rules at Sales Department (according to priority).
3. Rules in Finance (according to priority).

Within a given node, you can modify the order of execution by using the up or down arrows on each rule.

Only rules on a particular node path are executed in any given rules run. The path of execution is determined by input to the Rules Engine on the execution request.

> **Important**
> The business structure is defined in Configuration Manager or Genesys Administrator.

> **Important**
> System administrators can configure rule execution to be "bottom-up" or "top-down".

> The **Rule Evaluation Order** indicator at the bottom of the screen shows you which of these is selected, and a ToolTip is available when you hover your cursor over this indicator. Any changes made to this configuration will apply dynamically, but only take effect after a restart or a browser refresh.

Audit Trail

The **Audit Trail** tab allows you to view the history of the individual rules, such as when they were updated or deployed, and by whom. When accessed within a business context (a node on the Explorer Tree), the **Audit Trail** tab lists the rules that exist for that business context.

Creating Linear Rules

Follow these steps to create a linear rule:

1. Navigate to the rule package to which the new rule will belong in the left navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Navigate to the correct node of the business structure under the rule package, which will define the node at which your linear rule will be created. If you create the linear rule at the rule package level, it will be a global rule. Select the node in the left navigation and click the **Rules** tab.
2. Click **New Linear Rule**.
3. In the **Rule Summary**, the **ID** field is populated automatically. It cannot be edited.
4. Enter a **Name** for the rule (for example, `Gold`).
5. Enter a brief **Description** for the rule (for example, If the customer is a Gold member, then increase the priority).
6. Select the **Phase** at which this rule will be applied (classification, prioritization, or archiving for iWD. Refer to the Genesys Rules System Deployment Guide for more information about phases).
7. Select the **Business Calendar** to use with this rule (optional).
8. The **Pending Snapshot** field is displayed with a tick symbol indicating that the contents of this rule have not yet been included in a package snapshot. See Deployment for details of how to work with snapshots.
9. Enter a **Start Date** and an **End Date** for the rule (optional). If the **End Date** is earlier than the current date, a warning indicate that the rule is out of date.
10. In the lower panel, fill in the **When** and **Then** rows.
    a. To add a Condition (When), click **Add Condition** and select from the list (for example, a condition for this scenario might be `When the customer is a Gold member`). The rule condition includes the name of the rule template from which the condition is derived.
    b. To add an Action (Then), click **Add Action** and select from the list (for example, an action for this scenario might be `Increase the priority by 100`). The rule action includes the name of the rule template from which the action is derived.

    > **Important**
    > The maximum number of segments (text plus variables) on Conditions or Actions in linear rules is 9. An error message is displayed if this limit is breached.

    c. Insert values for the parameters into the table under the **Condition** and **Action** columns. Depending on how the parameters were configured by the rule template developer, there may be constraints on the values that can be entered.
4. Click **Validate** to validate the syntax of the linear rule. The **Validate** option appears in the **More** menu.
5. Click **Save** to save your changes.

When editing rules, be careful not to clear your browsing history or cookie data, as unsaved changes could be lost.

Updating Linear Rules

Follow these steps to update an existing linear rule:

1. Navigate to the rule package to which the linear rule belongs in the left navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node in (at which the rule was defined) and click the **Rules** tab.

   > **Tip**
   > You can also use the **Search** feature to locate rules.

2. Locate the linear rule in the list and update the information as needed. Refer to **Creating Linear Rules** for details about the fields that can be updated. Any changes that you make to the **Rule Summary** are saved automatically. Click **Save** to save any changes made to the body of the rule. Provide a check-in comment that summarizes the changes you made. This will appear in the **Audit History**.

When you are editing rules, be careful not to clear your browsing history or cookie data, as unsaved changes could be lost.

Copying Linear Rules

You can copy a linear rule and paste that copy in the same rule package, either on the same or a different node. Follow these steps to copy a linear rule:

1. Navigate to the rule package to which the linear rule belongs in theleft navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node in the tree (the level at which the rule was defined) and click the **Rules** tab.
2. Locate the rule in the list, select it, and select **Copy Rule** from the **More** menu.
3. If you want the copy to be in the same node, click **Paste Rule**.
4. If you want the copy to be in a different node, select that node in the tree click the **Rules** tab, and click **Paste Rule**.

> **Important**
>
> If you wish to move the rule to another location, first copy, then paste, then go back and delete the original. The system will not allow you to paste a rule after it has been deleted from the repository.

5. Update the information as needed and click **Save**. Refer to **Creating Linear Rules** for details about the fields that can be updated.

When editing rules, be careful not to clear your browsing history or cookie data, as unsaved changes could be lost.

Importing Linear Rules

You can import linear rules in GRAT.

> **Important**
>
> You can import an entire rule package, containing the rule definitions, business calendars, and references to the templates and versions that the rule package is dependent on. Refer to **Importing Rule Packages** for details.

Follow these steps to import a linear rule:

1. Navigate to the rule package to which the linear rule will belong in left navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node (at which the rule will be defined) and click the **Rules** tab.
2. Click **Import Rule**.
3. Browse to the location of the linear rule file.
4. Enter a comment.
5. Click **Import**.
6. The linear rule will appear in the list of rules. Edit the fields as necessary. Refer to **Creating Linear Rules** for details about the fields that can be updated. Click **Save**.

When editing rules, be careful not to clear your browsing history or cookie data, as unsaved changes could be lost.

Exporting Linear Rules

You can export linear rules by exporting the rule package that contains the rule. You might want to export your rules to back them up or move them to another server, and then import them back in.

You can export an entire rule package containing the rule definitions, business calendars, and references to the templates and versions that the rule package is dependent on. Refer to **Exporting Rule Packages** for details.

To export a linear rule:

1. Navigate to the rule package to which the linear rule belongs in the left navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node (at which the rule was defined) and click the **General** tab.

   > ### Tip
   > You can also use the **Search** feature to locate rules.

2. Locate the rule in the list, select it, and select **Export Rule Package**. The exported rule can be saved as an .xml file only.

Deleting Linear Rules

Follow these steps to delete a linear rule:

1. Navigate to the rule package to which the linear rule belongs in the left navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node (the level at which the rule was defined) and click the **Rules** tab.

   > **Tip**
   > You can also use the **Search** feature to locate rules.

2. Locate the linear rule in the list and click the minus icon in the **Actions** column to the right.

Deleting Rules Created in Error

If you accidentally create a rule and wish to delete it, add the two mandatory values (**Name** and **Phase**) before attempting to delete it. You cannot delete a rule unless the mandatory values are supplied.

Creating Decision Tables

Follow these steps to create a new decision table:

1. Navigate to the rule package to which the new decision table will belong in the left navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Navigate to the correct node of the business structure under the rule package, which will define the node at which your decision table will be created. If you create the decision table at the rule package level, it will be a global rule. Select the node and click the **Rules** tab.
2. Click **New Decision Table**.
3. In the **Rule Summary**, the **ID** field is populated automatically. It cannot be edited.
4. Enter a **Name** for the decision table (for example, `Status`).
5. Enter a brief **Description** for the rule (for example, Adjust the priority, dependingupon the customer's status).
6. Select the **Phase** at which this rule will be applied (classification, prioritization, or archiving for iWD. Refer to the Genesys Rules System Deployment Guide for more information about phases).
7. Select the **Business Calendar** to use with this rule (optional).
8. Enter a **Start Date** and an **End Date** for the rule (optional). If the **End Date** is earlier than the current date, a warning is displayed to indicate that the rule is out of date.
9. Use the up and down arrows in the far right-hand column to control the ordering of the decision table rows. In some complex cases, rules can be designed so that multiple rows will evaluate as true. In this case, the order of the rows becomes important, so you can re-order the rows when creating and editing a decision table.

> ### Important
> System administrators can configure rule execution to be "bottom-up" or "top-down". The **Rule Evaluation Order** indicator at the bottom of the screen shows you which of these is selected, and a ToolTip is available when you hover your cursor over this indicator. Any changes made to this configuration will apply dynamically, but only take effect after a restart or a browser refresh.

10. Add **Conditions** and **Actions** in the lower panel.

> ### Important
> You can use a wildcard symbol (*) in row data in a decision table (if the feature is configured by administrators). The wild card indicates that, for this row, the value for the parameter where it is used is unimportant and not to be evaluated. A wildcard selection now appears at the top of all lists, regardless

of whether they are enumerations, business attributes, Configuration Server, database, and so on. In the case of numeric parameters, you must type in the wildcard value—GRAT now accepts that as a valid number field. For any condition that contains one or more wildcards, its evaluation will not be considered in the rule logic. There are some restrictions:

- The wildcard values will work only for strings and numeric fields—fields of type date, time and Boolean are not supported.
- Wildcard values are "all or nothing" for conditions with multiple parameters. For example:

```
Customer age is between 40 and 60
```

is ONE condition, and it will be excluded for that row if one or more of the fields contains a wildcard value.

a. Select one or more **Conditions** from the list (for example, a condition for this scenario might be named `Customer's age is ...`).
b. Select one or more **Actions** from the list (for example, an action for this scenario might be named `Increase priority by xxx`).
c. Insert values for the parameters into the table under the **Condition** and **Action** columns. Depending on how the parameters were configured by the rule template developer, there may be constraints on the values that can be entered.
d. Repeat Step c, adding more condition and action values.
e. Re-order the rows as appropriate.

6. Click **Validate** to validate the syntax of the linear rule.
7. Click **Save** to save your changes.

Important

When editing rules, be careful not to clear your browsing history or cookie data, as unsaved changes could be lost.

Important

The **Pending Snapshot** field indicates whether any snapshot of this rule has yet been created. See Deploying Rule Packages for information on snapshots.

Updating Decision Tables

When editing or updating a decision table rule, you can insert or remove condition or action columns only after all of the data in the table has been validated (no red lines are visible). This restriction prevents you from losing unsaved data that you have just entered. If you accidentally insert the wrong condition or action, or accidentally delete a condition or action column, you can click **Cancel** to revert the rule back to the last saved version.

> **Important**
>
> Clicking **Cancel** results in the loss of any data that has been entered into the table, but not saved.

When adding rows to a decision table rule, it is important to fix all of the validation errors on that row before attempting to add or delete any new condition or action columns, to prevent loss of data in the row on which you are working.

To update an existing decision table:

1. Navigate to the rule package to which the decision table belongs in the Explorer Tree (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node in the Explorer Tree (the level at which the rule was defined) and click the **Rules** tab.

   > **Tip**
   >
   > You can also use the **Search** feature to locate rules.

2. Locate the decision table in the list and update the information as needed. Refer to **Creating Decision Tables** for details about the fields that can be updated. Any changes that you make to the **Rule Summary** are saved automatically. Click **Save** to save any changes made to the body of the rule. Provide a check-in comment that summarizes the changes you made. This will appear in the **Audit History**.

When you are editing rules, be careful not to clear your browsing history or cookie data, as unsaved changes could be lost.

Copying Decision Tables

You can copy a decision table and paste that copy in the same rule package, either on the same or a different node. Follow these steps to copy a decision table:

1. Navigate to the rule package to which the decision table belongs in the left navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node (at which the rule was defined) and click the **Rules** tab.

> **Tip**
> You can also use the **Search** feature to locate rules.

2. Locate the decision table in the list, select it, and select **Copy Rule** from the **More** menu.
3. If you want the copy to be in the same node, click **Paste Rule**.
4. If you want the copy to be in a different node, select that node in the tree click the **Rules** tab, and click **Paste Rule**.

> **Important**
> If you wish to move the rule to another location, first copy, then paste, then go back and delete the original. The system will not allow you to paste a rule after it has been deleted from the repository.

5. Update the information as needed and click **Save**. Refer to **Creating Decision Tables** for details about the fields that can be updated.

When editing rules, be careful not to clear your browsing history or cookie data, as unsaved changes could be lost.

Importing Decision Tables

You can import decision tables in GRAT.

> **Important**
>
> You can import an entire rule package, containing the rule definitions, business calendars, and references to the templates and versions that the rule package is dependent on. Refer to **Importing Rule Packages** for details.

Follow these steps to import a decision table:

1. Navigate to the rule package to which the decision table will belong in the left navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node (at which the rule will be defined) and click the **Rules** tab.
2. Click **Import Rule**.
3. Browse to the location of the decision table file. Decision table files can be in either `.xml` or `.xls` format.
4. Enter a comment.
5. Click **Import**.
6. The Decision Table will appear in the list of rules. Edit fields as necessary. Refer to **Creating Decision Tables** for details about the fields that can be updated. Click **Save**.

When you are editing rules, be careful not to clear your browsing history or cookie data, as unsaved changes could be lost.

You can import an entire rule package, containing the rule definitions, business calendars, and references to the templates and versions that the rule package is dependent on. Refer to **Importing Rule Packages** for details.

Exporting Decision Tables

You can export decision tables in GRAT. For example, you might want to export your rules to back them up or move them to another server, and then import them back in. You can also export the file as a spreadsheet (.xls) format. This can be useful if you want a decision table with hundreds of rows, where it might be easier to work in Excel, replicating rows and making minor changes, and then import it back into the Rules Authoring Tool, rather than creating hundreds of rows in the tool.

> **Important**
>
> You can export an entire rule package containing the rule definitions, business calendars, and references to the templates and versions on which the rule package is dependent. Refer to **Exporting Rule Packages** for details.

To export a decision table:

1. Navigate to the rule package to which the decision table belongs in the left navigation(verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node (at which the rule was defined) and click the **Rules** tab.

   > **Tip**
   >
   > You can also use the **Search** feature to locate rules.

2. Locate the decision table in the list and click **Export Rule**. There are two options for exporting decision tables: `.xml` or `.xls`. Select the format you prefer.
3. You can either **Open** or **Save** the exported rule file.

Deleting Decision Tables

Follow these steps to delete a decision table:

1. Navigate to the rule package to which the decision table belongs in the left navigation (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select the correct node (at which the rule was defined) and click the **Rules** tab.

   > **Tip**
   > You can also use the Search feature to locate rules.

2. Locate the decision table in the list and click the minus icon in the **Actions** column to the right.

Rules Package Overview

Rule packages are bundles of rules. Rule packages are used to group, manage, and deploy rules. The rules in a rule package provide a set of functionality (like an iWD solution). The Genesys Rules Authoring Tool (GRAT) allows you to create, edit, and delete rule packages.

Rule packages provide the following capabilities:

- The ability to 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. 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.
- The ability to 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.
- The ability to update individual rule packages without affecting other deployed packages.
- The ability to import and export an entire rule package containing the rule definitions, business calendars, and also the templates that the rule package is dependent on.
- A rule package contains one or more rules plus the fact model that is needed to support the rules. You deploy rule packages individually to the Rules Engine.

When you select an existing rule package in the left navigation, four tabs are displayed:

- The **General** tab displays the basic information for the rule package, such as name, type, and the associated templates.
- The **Rules** tab allows you to create, edit, and view rules. When you click the rule package node and then the Rules tab, you can create, edit and view rules at the global level for that package. Clicking on the other nodes (which represent various business contexts) enables you to modify the rules defined for that specific business context.
- The **Audit** Trail tab allows you to view the history of the individual rules, such as when they were updated or deployed, and by whom.
- The **Package History** tab allows you to view the history of a package and its versions and snapshots, including changes to rules, templates, calendars, test scenarios, imports/exports and deployments. History for all packages across one tenant can also be displayed at the tenant level.

> ### Important
> When a package is deleted, all references to it in Package History are also deleted.

Check My Permissions

To check all the permissions granted to you under role-based access control:

1. In the left navigation, select the **Tenant** tab.
2. Click the **Check My Permissions** button.

This page shows you:

- Which tenants you have access to
- Which role permissions you have
- Which templates you can access
- Which role permissions you have for the selected rule package. If you don't see the rule package name, you cannot view or edit it. If the rule package is not mapped to a role, then the permissions shown here will match those at the GRAT level.

To see how permissions are allocated and maintained, click **here** (new document).

Creating Rules Packages

Follow these steps to create a new rule package:

1. Select the node in the business hierarchy to which this rule package will belong from the drop-down list. Rule packages can belong to any node in the hierarchy.

> **Important**
>
> Package names must be unique across nodes or tenants. Package names should follow a naming convention such as including the node/tenant name, or company name, in their package names to avoid conflict.

2. In the left navigation, select **New Rules Package** under the appropriate node or **Solution**. You must have appropriate permissions for this option to display.
3. In the **Details Panel**, enter a name property for the new rule package.

> **Important**
>
> There are two name properties for a rule package: **Package Name** and **Business Name**.
>
> **Package Name** must conform to Java package naming conventions. Generally speaking, the package name should be in all lower case, can contain digits but must not start with a digit, and "." should be used as a separator, not spaces. For example, **my.rules** and **myrules1** are both valid names, but **My Rules** and **1my.rules** are not valid package names. Each organization should establish its own naming conventions to avoid name collision. Additionally, Java keywords must be avoided in package names. For example, **my.package** or **new.rules** are not valid package names. A list of Java keywords can be found **here**.
>
> **Business Name** allows you to provide a user-friendly name for the rule package, as it appears in the GRAT navigation tree. For example, **Acme Rules** is not a valid rule package name, but you could use **acme** as the **Package Name** and **ACME Rules** as the Business Name.

4. Select which type of rule package you are creating. The drop-down list shows which types are already in the repository for the selected tenant. As you change the type, the list of templates for that type will be displayed.
5. Enter a description for the rule package. The available rule templates (that were created for the node/Tenant and match the type selected in Step 4) will appear in the table. Templates prefixed with "(*)" are templates that were created in the **Environment Tenant** and can be used by all **Tenants**. Rule developers create rule templates and publish them to the rules repository by using the Template

Development module.

> **Important**
>
> The access permissions configured in Configuration Server can also affect which templates are displayed.

> **Important**
>
> GRAT users can select between multiple versions of templates, which are displayed on the enhanced **Template Selection** dialog along with version comments created by the template developer to help identify differences between the versions. The number of versions of a template that are displayed is configured in Genesys Administrator.

6. If required, check the **Disable Package Serialization** checkbox to turn serialization off for this rule package, if it is enabled. (Package serialization can be enabled/disabled globally in GRE. iWD customer should always have serialization turned off.)
7. Select the template(s) you want to include and click **Save**.
8. The new rule package will appear in the left navigation. Expand the new rule package, and the following options (subject to the permissions set for your user ID) will appear under the rule package folder:
    - Split Test Configuration—Use this node to create rules that enable you to control how Split Testing applies to rule at the rule package level.
    - Business Calendars
    - Test Scenarios
    - Deploy Rules
    - Search

    You will also see the business structure nodes to which you have access permission.
9. You can now create rules for your rule package.

Editing Rules Packages

Follow these steps to edit an existing rule package:

1. Select the node to which the rule package belongs from the drop-down list.
2. In the left navigation, select the name of the rule package.
3. In the **Details Panel**, you can modify the **Description** field, and update which template(s) to include in the rule package. You cannot edit the **Package Name** and type, but you can edit the **Business Name** of the rule package. Click **Save**.
4. If required, check the **Disable Package Serialization** checkbox to turn serialization off for this rule package, if it is enabled. (Package serialization can be enabled/disabled globally in GRE. iWD customer should always have serialization turned off.)

> ### Important
> If you want to update the rule package by selecting a newer version of a template that is already associated with the package, you must deselect the current version of that template before saving your changes. You can only have one version of a particular rule template associated with a package at any time.

> ### Warning
> Be careful when changing templates or template versions as it could affect existing rules. For example, an existing rule might use a condition that does not exist in a different version of the template. Consult with the rule template developer to ensure that you are choosing the correct templates/versions for your application. Multiple versions of templates are available for selection.

5. To edit the Global rules that are configured for the rule package, select the name of the package in the left navigation, and then click the **Rules** tab. To edit the rules created for a specific business context, select the node in the left navigation.
6. You can also edit the **Business Calendars** that are configured for your rule package.

Deleting Rules Packages

> **Important**
> When a package is deleted, all references to it in **Package History** are also deleted.

Follow these steps to delete an existing rule package:

1. Select the node to which the rule package belongs from the drop-down list.
2. In the Explorer Tree, select the name of the rule package.
3. In the **Details Panel**, click **Delete**. A confirmation dialog box will appear.
4. Click **OK** to confirm the deletion of the rule package, and its associated rules and business calendars.

> **Warning**
> If you attempt to delete a rule package that is still deployed, you are warned before committing the delete action. This gives you the opportunity to undeploy the rule package. Once the rule package is deleted from GRAT, you can no longer use the Undeploy feature to undeploy it. In that scenario, you would have to manually remove the files from all GREs.

Deploying / Undeploying Rules Packages

Summary

In order for rules to be invoked by Genesys applications, you must deploy the rule package to one or more Genesys Rules Engines (or for Genesys Web Engagement, to the GWEB backend server). The deployment process (whether you choose to deploy immediately or to schedule the deployment for later) attempts to compile the rule package and informs you of the result via the **Deployment Pending** pop-up message. You can check on the status of your deployment by looking at the **Deployment History** tab, which shows the status **Pending**. When a deployment is in pending status, you will not be able to cancel or undo it.

This process enables you to correct any errors before deployment. In addition, if you attempt a deployment that would duplicate either;

- An already scheduled deployment or;
- An attribute of an already scheduled deployment, such as;
  - The same rule package
  - For the same snapshot
  - For the same destination server/cluster

an appropriate message is displayed. You can then either change the attributes of your deployment, or go to **Deployment History** and change/delete the scheduled deployment.

> **Important**
>
> If your GRAT instance is part of a GRAT cluster, you can also view, edit, delete or re-schedule deployments that have been scheduled by other members of the same GRAT cluster (the **Deployment History** tab now has a **Deployed From** field showing which GRAT last scheduled the deployment). As soon as any GRAT instance that did not originally schedule a deployment makes any changes to a scheduled deployment, it takes over responsibility for the deployment.

To use the deployment screen, you must have deploy permissions set up in Genesys Administrator.

To deploy a rule package:

1. Select the Tenant to which the rule package belongs from the drop-down list.
2. In the left navigation, select the name of the rule package.

3. Under the rule package, select **Deploy Rules**. (The number of rules as yet not included in a snapshot appears in parentheses.) The **Details Panel** contains two tabs:

- The **Outstanding Deployments** tab allows you to select from a list of snapshots of the package including the LATEST version of the package (if configured by an administrator), create a new snapshot, export a snapshot (as an XML file downloadable to the user's local file system), delete a snapshot, deploy the rule package, schedule a deployment to occur at a future time, and show the source of the package. (**Show Package Source** displays the actual contents of the package snapshot you are deploying. The fact model, calendar definitions, and rule definitions will be coded into the rule language and displayed.)

> ### Important
>
> When you create a snapshot, you can choose to check the **Run as Background Task** option. For very large rule packages, it can take a long time to create a snapshot. When this option is checked, this operation will be completed in the background. This allows you to do other things or log off. When the snapshot is complete, it appears under **Package Snapshots**.
>
> Even if **Run as Background Task** is checked, the package will first be built and validated to ensure there are no errors. Once the validation is successful, the snapshot will be queued to a background task.
>
> You cannot delete the LATEST snapshot, and you cannot delete a snapshot for which there is a scheduled deployment.

- The **Deployment History** tab shows details about when the package snapshot was deployed in the past, and by whom. Failed deployments also appear in the list. In addition, the **Deployment History** displays scheduled deployments, and allows you to cancel or change the schedule of upcoming deployments.

To deploy the package immediately:

1. Select the package snapshot, or the LATEST version (if available).

> ### Important
>
> The LATEST version is available only if configured in Genesys Administrator. Your organization may choose not to make it available because its contents may vary over time, for example between scheduled deployments.

2. Click **Deploy Now** in the **Outstanding Deployments** tab.
3. Select the **Location** to which the package snapshot will be deployed. Locations can include standard application clusters configured in Genesys Administrator, special smart clusters based on the `Genesys_Rules_Engine_Application_Cluster` application template, or the `GWEB` backend server for Genesys Web Engagement.
4. Enter some comments about the deployment (these will appear in the Deployment History).

5. Click **Deploy**.

A message will be displayed indicating whether the deployment was successful, failed, or partial. A partial deployment means that not all nodes in the cluster successfully received the deployed rules package.

To deploy the package later:

1. Click **Schedule Deployment** in the **Outstanding Deployments** tab.
2. Select the **Location** (the name of the Rules Engine application or application cluster, or the GWEB backend server for Genesys Web Engagement) to which the package snapshot will be deployed.
3. Enter the date and time you would like the package snapshot to be deployed.
4. Enter some comments about the deployment (these will appear in the **Deployment History**).
5. Click **Schedule**.

A message will be displayed indicating whether the deployment was successfully scheduled.

If you wish to reschedule a previously scheduled deployment, or wish to cancel a scheduled deployment, you may do so from the **Deployment History** tab.

To refresh the display of a deployment history, click the **Refresh** button, or click in the relevant node in the Explorer Tree.

To display details of a deployment to a cluster:

If you are deploying to a cluster, you can now display a detailed report of the deployment, whether it was successful, failed or partial. This gives useful information on how a deployment has progressed: you can click the **Deployment Status** result to see, for example, whether a server connection was temporarily down at a critical moment, or whether a server timeout setting might need to be changed. Where a deployment shows as partial, you can click the **Partial** link in the **Deployment Status** panel to display details of individual GREs, whether and when they have auto-synchronized subsequently.

If partial deployment is NOT configured

When deploying to a cluster, GRAT uses a two-phase commit protocol to ensure that all GRE nodes running in the cluster are running the same version of the deployed rule package. If any of the nodes in the cluster fails during Phase 1, the Phase 2 is not committed.

- Phase 1 - (Deploy) All GREs in the cluster are notified about the new rule package. Each GRE downloads the new rule package and compiles it.
- Phase 2 - (Commit) Once all GREs have successfully completed Phase 1, GRAT notifies each GRE to activate and commit the new rule package.

The Deployment Status shows the detail of each node in the cluster and whether or not any errors

occurred.

If partial deployment IS configured

GRAT attempts to deploy the rules package to all GRE nodes running in the cluster. If any nodes are down, or disconnected, or the deployment fails for any reason, the rules package is still deployed to the remaining nodes in the cluster. The GREs in the cluster can be configured to auto-synchronize when disconnected nodes are reconnected, or when new nodes are added to the cluster.

GRAT still uses a two-phase commit protocol. The only difference is that in a partial deployment case, we continue to Phase 2 for GREs which successfully complete Phase 1. Overall Status is set to `Partial` when 1 or more (but not all) of the GREs in the cluster fails the deployment.

- Phase 1 - (Deploy) All GREs in the cluster are notified about the new rule package. If any GRE fails to respond successfully, the overall deployment status is set to `Partial`.
- Phase 2 - (Commit) For GREs that have successfully completed Phase 1, GRAT notifies each GRE to activate and commit the new rule package.

To show the deployment report:

1. Click the **Failed/Successful/Partial** link in the **Status** column.
2. The details of each deploy action to each server in the cluster are displayed, including:

- The GRE Server Name
- The server status
- The success or error message generated by the server
- The Phase 1 and Phase 2 deployment times in seconds
- Whether and when the GRE was auto-synchronized and from which other member of the cluster the rules package data was received (if the auto-synchronization feature is configured).

> **Important**
>
> The time zone for scheduled deployments is always the time zone of the server on which the Genesys Rules Authoring Tool is installed.

Undeploying a rules package

For users with the correct privileges, an **Undeploy** button now displays on the **Outstanding Deployments** tab. This button lets you undeploy a rules package from a single GRE or cluster (but not from a GWE back-end rules engine or cluster).

To undeploy a rules package:

1. Click the **Undeploy** button. This displays the **Undeploy** dialog.
2. Select the single GRE or cluster from which to undeploy the rules package and click **Undeploy**.
3. If partial undeployment is enabled, the details in the **Deployment History** tab might show where a partial undeployment has taken place. Click the **Failed/Successful/Partial** link in the **Status** column to display the Undeployment report. **Partial** status indicates that one or more GRE nodes were offline when the rule package was undeployed. When those nodes come back online, and if auto-synchronization is enabled, they will automatically synchronize with the other GRE nodes and undeploy the package.

> ### Important
> If you try to undeploy a package that has a pending deployment, a warning message displays. Either cancel the undeployment or wait until the deployment has completed before attempting another undeployment.

If partial undeployment IS enabled:

GRAT attempts to undeploy the rules package from all GRE nodes running in the cluster. If any nodes are down, or disconnected, or the undeployment fails for any reason, the rules package is still undeployed from the remaining nodes in the cluster. The GREs in the cluster can be configured to auto-synchronize when disconnected nodes are reconnected, or when new nodes are added to the cluster.

Overall Status is set to `Partial` when 1 or more (but not all) of the GREs in the cluster fails the undeployment.

If partial undeployment is NOT enabled

When undeploying from a cluster, GRAT only undeploys the rules package if all the members of the cluster are active. If any node is inactive the undeployment fails and the rules package remains deployed at all nodes in the cluster.

Audit Trail

The **Audit Trail** tab allows you to view the history of the individual rules, such as when they were updated or deployed, and by whom.

The **Audit Trail** tab lists the rules that exist for the selected rule package, or for the selected business context (node), depending on where you access the Audit Trail. The **Audit Trail** tab shows the history of the currently selected rule.

You can select the **Rule ID/Name** drop-down to select another rule. For each rule you can view the history of the rule, including different versions that have been saved, and the configured actions, conditions, parameters and comments.

If a particular revision of a rule was saved as part of a snapshot, the snapshot name appears in the **Last Snapshot Name** column. This enables you to determine the content of the rule when the snapshot was taken. You can filter the list of rule versions by **Last Snapshot Name**, **Action** (**Created**, **Modified**, and so on), and by the user name of the person who made the changes (**Taken By**). You can sort the list by clicking a column name, displaying the results in ascending or descending order by the selected column.

You can export the history of the rule into a file (spreadsheet format). Select the rule from the list, and click **Export Rule History**. You can choose to open the file that is created, or save it.

You can revert back to a previous version of a specific rule: select the version to which you would like to revert, and click **Revert**. The revert operation will create a new version of the rule containing the same contents as the older version that you selected. The original versions and audit history will be preserved. Revert may also be used to restore a previously deleted rule. To do this, select the **Rule ID/Name** from the drop down and then revert the deleted version.

Displaying Package History

GRAT tracks all changes made to a package and displays them in a tab called **Package History**. This tab displays the package history at either individual package level or across a Tenant. At the Tenant level, the tab displays a history of all changes to all packages for that Tenant.

You can filter by package name, snapshot name, description of change or by the user who made the change, and you can sort by any column, either the entire results or within a filtered subset.

Every saved change made to a package causes a new package version to be generated. This package version is displayed on the **Package History** tab, as well as on the **Deployment History** view.

Notes

- The **Business Hierarchy** column displays the business hierarchy node that the rule package or rule package item is related to—for example, **Houses > Sales > Closing**.
- Package history shows only changes in the business structure nodes to which the user has access.
- The **Change By** column is visible only to users with relevant role privileges.
- The **Snapshot Name** column shows the name of the snapshot in which a change was made.

Importing Rules Templates

> **Important**
> A template exported with a pre-8.1.2 version of GRAT cannot be imported with release 9.0.0. You must republish such a template from 8.1.3 GRDT once GRAT 9.0.0 is running.

You can import rule templates from an .XML file. Rule templates are stored in the repository as separate assets, so they can be used by multiple rule packages. The rule templates are not part of the rule package themselves; the rule package refers to the rule templates that it needs.

If it is necessary to import the rule templates, you should import them prior to importing the rule packages, since the rule packages make references to the templates that they use.

It is not necessary to import the rule templates if you are importing from the same system (for example, backing up or restoring a rule package) or from an equivalent system (for example, a lab versus a production environment). However, if you are moving a rule package to a new system or sending it to Genesys for service, you should import both the rule templates and the rule packages so that, when imported, all referenced templates are available in the target system.

Refer to **Importing Rules Packages** for details on how to import rule packages.

> **Important**
> To import a rule package template, you must have **Create Template** permission.

To import a rule package template:

1. In the left navigation, select the **Tenant** tab.
2. Click **Import Rule Templates**. A dialog box opens in which you select the `.xml` file to import.

> **Important**
> Clicking **Replace existing templates in repository** clears the repository of any previous versions of each template before the new ones are imported into the target system. If this option is not enabled and an existing template with the same name is found in the repository, an error message appears and the import is terminated.

Warning

Be careful when changing templates or template versions as it could affect existing rules. For example, an existing rule might use a condition that does not exist in a different version of the template. Consult with the rule template developer to ensure that you are choosing the correct templates and versions for your application.

Importing Rules Packages

You can import an entire rule package containing the rule definitions, business calendars and test scenarios for that rule package, from an .XML file.

If it is necessary to import the rule templates, you should import them prior to importing the rule packages, since the rule packages make references to the templates that they use.

It is not necessary to import the rule templates if you are importing or exporting from the same system (for example, backing up or restoring a rule package) or from an equivalent system (for example, a lab versus a production environment). However, if you are importing the rule package to a new system or sending it to Genesys for service, you should export both the rule templates and the rule packages so that, when imported, all referenced templates are available in the target system.

> **Important**
>
> The **last-modified-by** date and all rule audit history are not part of the rule package (or rule) export. So, when re-importing an exported rule package, the user doing the importing becomes the new owner of each rule created. Package and rule history are not maintained. The imported rules is considered a new rule package with new history starting from the point of import.

Refer to Importing Rule Templates for details on how to import rule templates.

Importing rule packages enables you to do the following:

- Copy an entire rules configuration from a test environment to a production environment.
- Perform a backup of the entire rules configuration before performing a Genesys Rules System upgrade

> **Important**
>
> To import a rule package, you must have Create Package and Create Business Calendar permissions.

To import a rule package:

1. Select the Tenant to which the rule package belongs from the drop-down list.
2. In the left navigation, select **New Rules Package** under the appropriate Solution.
3. Click **Import Rule Package**. A dialog box opens in which you to enter the **Package Name** and the **Business Name**, and select the `.xml` file to be imported.

4. Check **Auto-save each rule** to auto-save each rule on import. This option should only be used if the rule package is known to be valid on the target system, such as when copying between two identical systems (a lab versus a production environment). Auto-save commits each rule in the package without validating that it matches the underlying templates. If you do not use this option, each rule is imported in the draft state and must be saved manually. This method shows any validation errors and gives the rule author the opportunity to fix them before deployment.
5. **If your business hierarchy is non-nested,** check **Auto-create business hierarchy during import** to tell GRAT to automatically create any missing nodes in your business hierarchy for rules that are contained within the .xml file. For example, if this option is selected, during the import if there is a rule that is associated with the "Widget Sales" department, but no such department is defined in the business hierarchy, GRAT will attempt to create it during the import operation. The GRAT user who is performing the rule package import must have permission to create this folder. If the box is not checked and there are rules associated with missing nodes, the import will fail.

   If your business hierarchy is nested, and you select the **Auto-create business hierarchy during import** during the import process, GRAT ensures that both business structures are compatible, and prevents an import if it they are not, and displays an error message informing you that the business hierarchy is not compatible with the imported rule package.

   > **Important**
   >
   > Even if the **Auto-create business hierarchy during import** button is selected, GRAT prevents the same node name from being created anywhere in the hierarchy—uniqueness of business node names across the entire hierarchy is still enforced.

6. Click **Import**.

Exporting Rules Templates

You can export rule templates to an .xml file. Rule templates are stored in the repository as separate assets so they can be used by multiple rule packages. The rule templates are not part of the rule package themselves; the rule package refers to the rule templates that it needs.

It is not necessary to import or export the rule templates if you are importing or exporting to the same system (for example, backing up or restoring a rule package) or to an equivalent system (for example, a lab versus a production environment). However, if you are moving the rule package to a new system or sending it to Genesys for service, you should export both the rule templates and the rule packages so that, when imported, all referenced templates are available in the target system.

Refer to **Exporting Rule Packages** for details on how to export rule packages.

> ### Important
> To export a rule template, you must have **View Template** permission and **Read** access for the Script objects that represent the templates being exported.

To export a rule template

1. In the left navigation, select the **Tenant** tab.
2. Click **Export Rule Templates**.
3. Select the individual template or templates that you wish to export. All versions of the selected templates will be exported in order to maintain the version sequence at the target system.
4. The .xml file is generated.

Exporting Rules Packages

You can export an entire rule package containing the rule definitions, business calendars, and references to the templates and versions that the rule package is dependent on, to an .XML file.

It is not necessary to export the rule templates if you are exporting to the same system (for example, backing up or restoring a rule package) or to an equivalent system (for example, a lab versus a production environment). However, if you are moving the rule package to a new system or sending it to Genesys for service, you should export both the rule templates and the rule packages so that, when imported, all referenced templates are available in the target system.

> **Important**
>
> The **last-modified-by** date and all rule audit history are not part of the rule package (or rule) export. Package and rule history are not maintained.

Refer to **Exporting Rule Templates** for details on how to export rule templates.

Exporting rule packages enables you to do the following:

- Copy an entire rules configuration from a test environment to a production environment.
- Perform a backup of the entire rules configuration before performing a Genesys Rules System upgrade.

> **Important**
>
> In order to export a rule package, you need to have **View Rule** and **View Business Calendar** permissions.

To export a rule package:

1. Select the **Tenant** to which the rule package belongs from the drop-down list.
2. In the left navigation, select the correct node (the level at which the rule package was defined).
3. Click **Export Rule Package**. The selected rule package is exported to a single `.xml` file.

> **Important**
>
> This .xml file contains overall package information (name, type, description, and list of templates and versions), a list of rules (decision tables and linear rules), and a list of

business calendars and calendar rules associated with the rule package. This `.xml` file does not contain the template contents, but does contain a reference to the template names and versions used.

Business Calendars Overview

Rule packages can contain one or more Business Calendars. Business calendars define the working days and hours of the organization. They can also be associated with any rule in the package.

Calendars are out-of-the-box classes available in the Fact Model that can be used by Rules. A calendar contains:

- Name
- Time zone (the list of available timezones is defined in the Java runtime)

> **Important**
>
> Business calendars can be configured to allow the timezone to be provided at rule-evaluation time.
>
> When the GRAT user configures a business calendar, a timezone is chosen along with the other attributes of the calendar (normal work week, exceptions, holidays). Timezones that respect Daylight Saving display with a "*" suffix.
>
> You can also use standard methods that can be accessed from within the rule template to allow the timezone ID to be passed in at rule evaluation time by the application that is requesting rule evaluation. If the timezone ID is not passed in in this way, then the "saved" timezone is used. If the timezone ID is passed in, then it overrides the saved timezone and the calculations will be done using the provided timezone. See **Business Calendar Enhancements** (Best Practice/User Guide).

- Week start day and time
- Week end day and time
- Holidays (one or more)
- Time Change (one or more)

A **holiday** is fixed, relative, or annual.

- A fixed holiday contains the date of the holiday, including day, month, and year, such as 01/01/2015.
- A relative holiday contains the month and weekday of the holiday and whether it is on the first, second, third, fourth, or last day of that month, such as the third Thursday of November.
- An annual holiday contains the month and day of the holiday, such as July 4.

A **time change** indicates how the work hours can be adjusted on particular days; for example, defining a half day on a particular day of the work week. Like a holiday, a time change is fixed, relative, or annual and contains the same date definition as the corresponding holiday definition. In addition, the time change contains the start and end time for the defined date.

Business calendars are needed to be able to define rules based on work hours. For instance:

`WHEN Task is idle for more than` *3 Working Days* `THEN increase Priority by 20`

`WHEN` *Today is a holiday* `AND Task is urgent THEN Agent Group is “Urgent Care”`

The italicized portions of the above examples use business calendar information.

The following topics explain how to work with Business Calendars in GRAT:

- **Creating Business Calendars**
- **Copying Business Calendars**
- **Deleting Business Calendars**

Creating Business Calendars

Follow these steps to create a new business calendar:

1. Navigate in the left navigation to the rule package to which the business calendar will belong (verify that you have selected the correct Tenant from the **Tenant** drop-down list).
2. Select **Business Calendars** under the rule package.
3. Click **New Calendar**.
4. The **ID** field is populated automatically. This is not editable.
5. Enter a **Name** for the business calendar. Use something descriptive that will make it easier to identify the rule, such as Regular Work Week.
6. Select which day of the week the week starts on (such as Monday).
7. Select the ending day of the week (such as Friday).
8. Enter the start time (such as 9:00 AM).
9. Enter the end time (such as 5:00 PM). If the end time is earlier than the start time, it is assumed that the workday spans midnight.
10. Select the default timezone that applies to this business calendar.

> **Important**
>
> 1. Timezones that respect Daylight Saving display with a "*" suffix.
>
> 2. The timezone selected can be overridden if a different timezone ID is passed in at rule evaluation time by the application requesting an evaluation by the rules engine.

11. If necessary, you can configure business calendar rules for your new business calendar.
    a. In the lower pane, click the **Add** button .
    b. A new row will appear in the **Business Calendar Rules** panel. Enter a **Name** for the rule, such as **New Year's Day**.
    c. Select the **Entry Type** for the rule, such as **Holiday**.
    d. Select the **Calendar Placement**, such as **Annual** for New Year's Day, or **Relative** for Memorial Day. You might also need to configure a Fixed holiday, for example, if the holiday will be observed on a different day one year, because the actual holiday falls on a non-working day.
    e. Enter the parameters for the rule, such as the specific date (January 1, for New Year's Day), or the x day of a specific month (such as the third Monday in May, for Memorial Day).
    f. Configure any time change exceptions for this business calendar. A time change indicates how the work hours can be adjusted on particular days; for example, defining a half day on a particular day of the work week. Like a holiday, a time change is fixed, relative, or annual and contains the same date definition as the corresponding holiday definition. In addition, the time change contains the start and end time for the defined date.
    If you have configured more than one holiday or time change exception, provided that the

calendar is not locked by another user you can use the far right-hand column to:

- Adjust the order in which they are processed. This lets you avoid setting up clashing exceptions. If there is a conflict, the highest entry takes priority.
- Make copies of existing exceptions and adjust them.
- Create new exceptions.

g. Click **Save**.

Copying Business Calendar

You can copy a business calendar and paste that copy in either the same, or a different, rule package. Follow these steps to copy a business calendar:

1. Navigate to the rule package to which the business calendar belongs in the Explorer Tree (verify that you have selected the correct Tenant from the **Tenant** drop-down list). Select **Business Calendars** under the rule package in the Explorer Tree.
2. Locate the business calendar in the list and click **Copy Calendar**.
3. If you want the copy to be in the same rule package, click **Paste Calendar**. Enter a name for the new business calendar.
4. If you want the copy to be in a different rule package, locate that rule package and select **Business Calendars** under that rule package. Click **Paste Calendar**. Enter a name for the new business calendar.
5. Update the information as needed. Click **Save**. Refer to **Creating Business Calendars** for information about the various fields and configuring business calendar rules.

Deleting Business Calendars

Follow these steps to delete a business calendar:

1. Navigate to the rule package to which the business calendar belongs (verify that you have selected the correct Tenant from the **Tenant** drop-down list).
2. Select **Business Calendars** under the rule package.
3. Locate the business calendar in the list and click the Delete button.

Rule Testing Overview

> **Important**
>
> Certain features on which testing scenarios depends were not implemented in the initial 9.0.00.13 release of GRAT (see the Release Note). This functionality is implemented in release 9.0.000.17.

Before deploying a rules package to the Genesys Rules Engine, subject to having the relevant permissions in Genesys Administrator, you can :

- Create, modify and run one or more test scenarios for each rule package
- Add input data, business context and phase data and expected results
- Review the test outcomes in plain language
- Import and export the test scenarios in the same way as with rules packages

These test features allow rules authors to test any changes made to existing rules packages before deploying them, in order to ensure that no errors are introduced.

The rule testing functionality is available via a node in the navigation tree called **Test Scenarios**. Click this node to use the rule testing features.

- **Creating rule test scenarios**
- **Running rule test scenarios**
- **Test scenario results**
- **Importing and exporting rule scenarios**

Split Test functionality that allows you to test alternative business rule outcomes is available via a node in the navigation tree called Split Test Configuration.

- **Split Testing**

Using Split Testing to Compare Business Rule Outcomes

> **Important**
> Certain features on which Split Testing depends were not implemented in the initial 9.0.000.13 release of GRAT. This functionality is implemented in release 9.0.000.17.

Overview

What problem does A/B/C Split Testing solve?

If your organization has a complex set of rules to define business decision-making, and you want to be able to compare alternative outcomes and scenarios, then rather than just changing a condition and hoping for the best you probably want to phase changes in slowly in order to measure the impact over time.

A/B/C Split Testing (hereafter, **Split Testing**) allows you to compare the business outcomes of alternative rule scenarios before rolling out significant changes to the way you make your business decisions. With Split Testing you can make, test and review changes incrementally to test their effects before committing to a particular change set.

For example, you might want to test several subtle changes in the way applications for credit cards are treated, in order to identify which has the best business outcome. You could split test for one income level against another, or one age range against another, or between different customer segments, or indeed any conditions in your rule package.

Flexible Approaches to Split Testing

Split Testing offers several approaches. For example, you can choose to:

- Unconditionally force either path A or B or C.
- Tie Split Testing to certain conditions (whether the customer is from a particular city, of a certain age, not a Gold customer, and so on).
- Weight the paths for A, B and C by percentage. This allows you to take a more statistical approach and use larger test data sets.
- Tie Split Testing to a business calendar (for example only do split testing on Fridays after 5 PM or Saturday morning 9 AM - 12 PM).

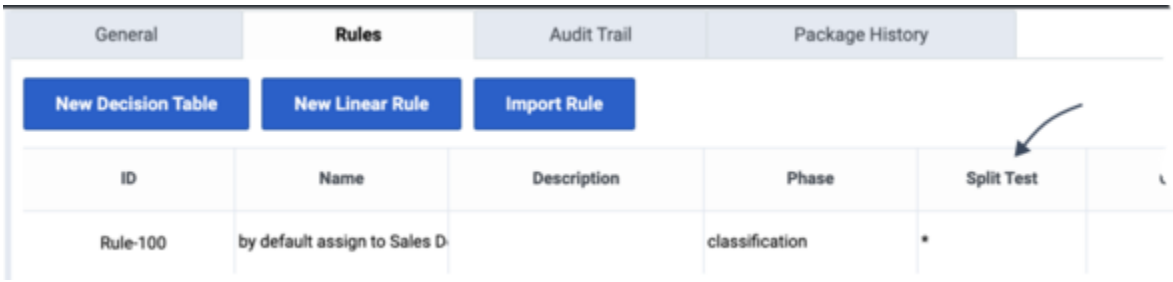
Split Testing Template

The GRSSplitTest_template.xml template provides some basic Facts (Conditions and Actions) for the Split Testing feature. To implement the feature, GRAT users must import this template (from the **Samples** folder) and attach it to all rules packages for which the Split Test functionality is required.

> **Important**
>
> The template—`GRSSplitTest_template.xml`—is shipped with type `samples`. To use this template with other rule package types, you can import the template into GRAT, change the name (for example, `GRSSplitTestForMyType`) and the type (to match your rule package type) and publish. Then you can use it with another package type.

Split Test Column in Rule Packages

A Split Test column displays at the rule package level.



When the Split Test template has been imported into GRAT, this drop-down field will show the values imported from the template (`A`, `B` and `C` are the default values in the shipped template). If the template has not been imported, only the wildcard (*) symbol will be available.

Duplicate (copy and paste) the rule change that you wish to test. On the duplicated rule, make your logic change (for example, adjust the income requirement needed to offer a credit card).

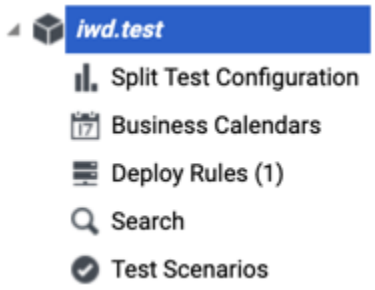
Mark the original rule as “A” and the new rule as “B” by selecting from the “Split Test” column.

Once “A”, or “B” is selected, the rows color-code for easier viewing. The wildcard symbol (*) means that the Split Test condition is ignored, so this rule will always fire.

The rules marked as “A” or “B” (or “C”) will fire according to the logic in the Split Test Configuration. For example, you can weight the distribution such that “A” fires 75% of the time and “B” fires 25% of the time, etc. See the next section on Split Test Configuration for more options.

Split Test Configuration Node

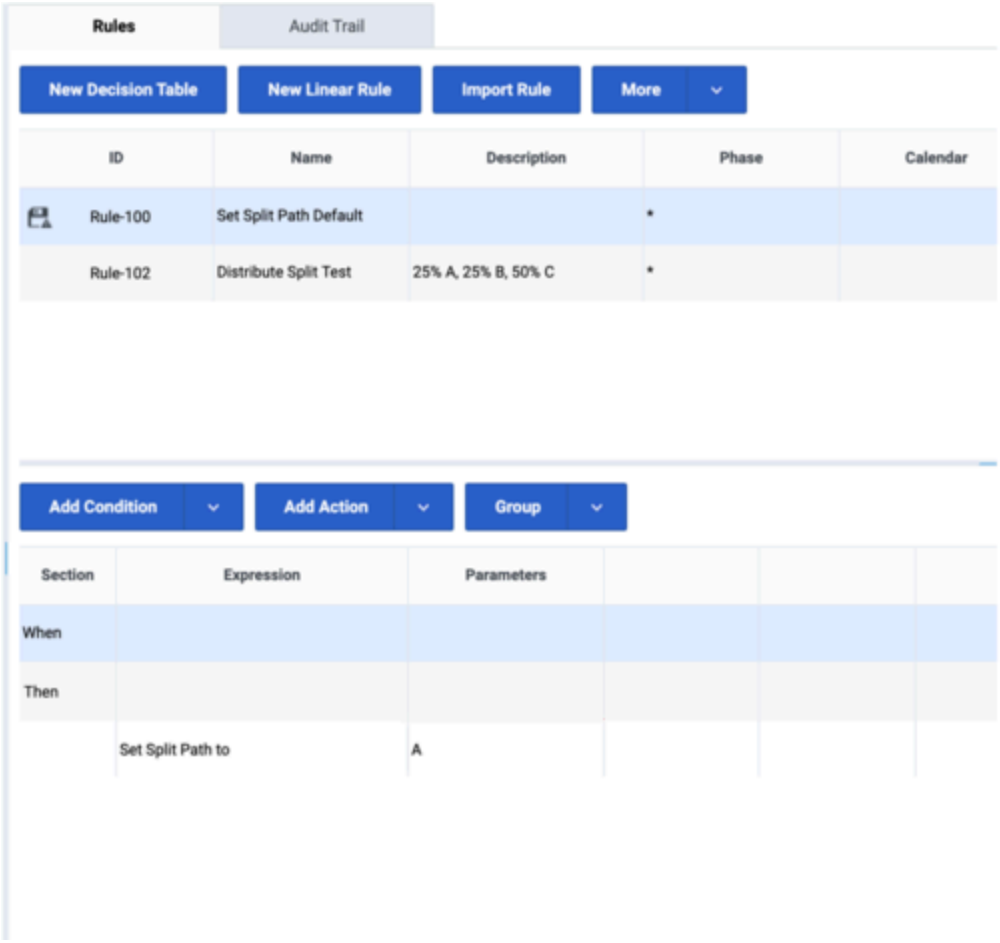
A Split Test Configuration node appears in the left navigation tree. You can use this node to create (and display existing) Split Test rules that will apply at the rule package level. In rules created under this node, the **Split Test** column is hidden to avoid confusion.



What Can I Do with Split Test Rules?

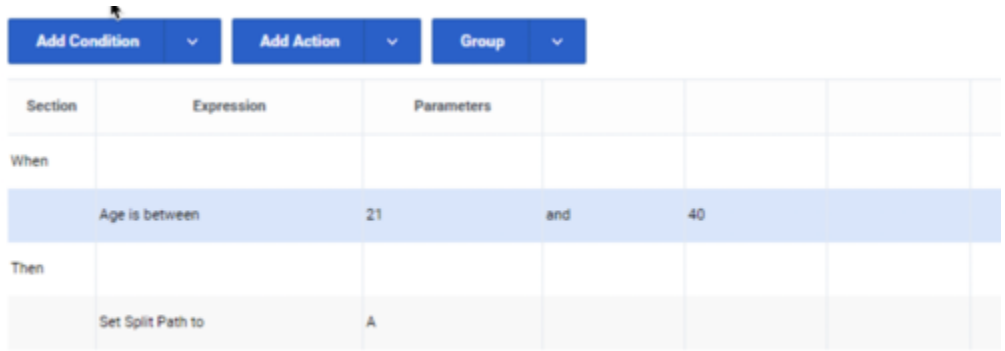
Unconditionally force either path A or B or C

In rules created under this node, the `splitTestValue` can be selected as a Parameter for the rule. You can use this simple rule to force the Split Test path to any of the values in the template (the default shipped values in the template are `A`, `B` and `C`, but you can change these in the template if required).



Tie Split Testing to a Condition

To add more flexibility and precision to your use of Split Test rules, you can add to your rule any other condition available from the template(s) attached to the rule package. The example below has lower and upper age limits:



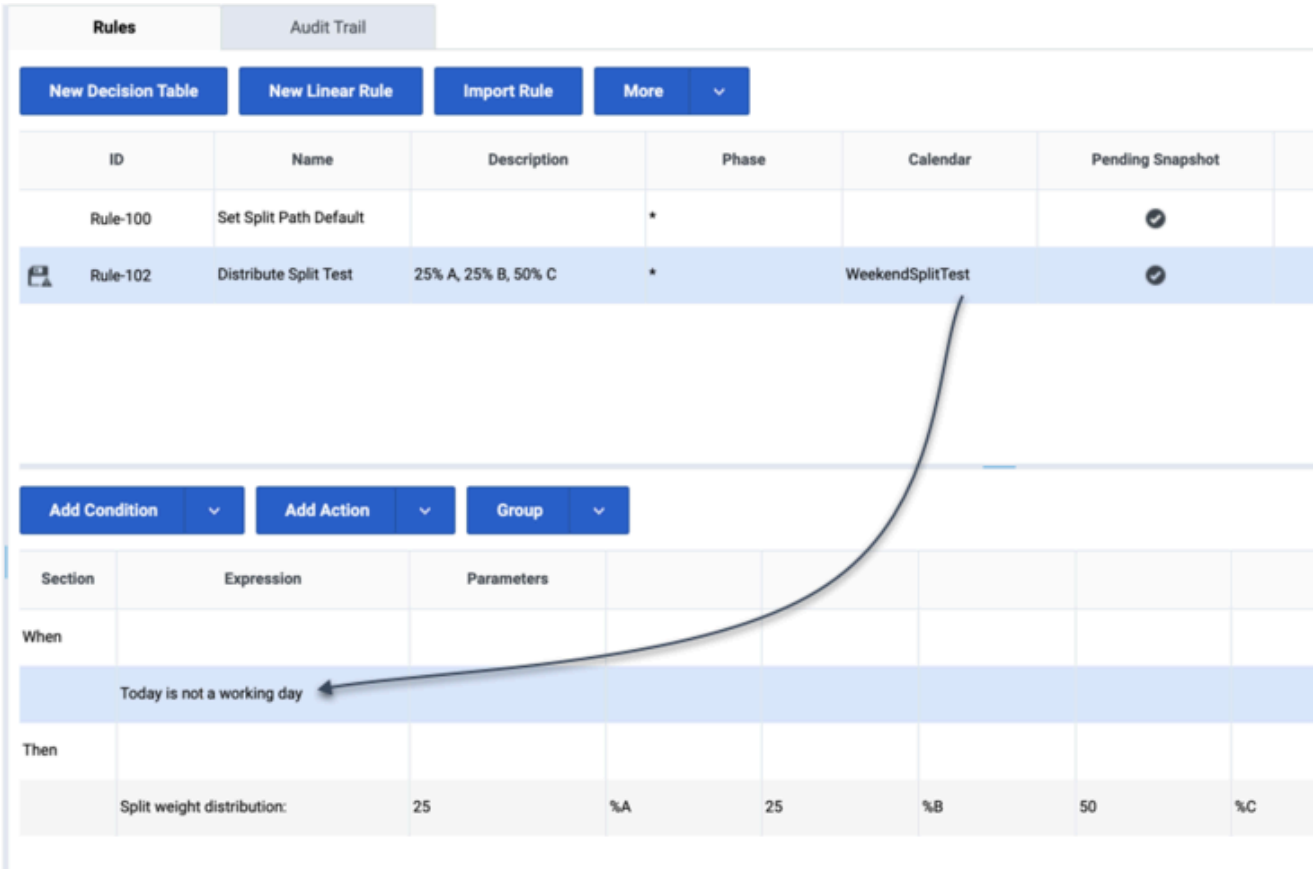
Weight the paths for A, B and C by percentage

You could add a condition to perform Weighted Distribution Split Testing. In this example the Split Test rule distributes the incoming rule evaluation requests across the A, B and C paths in the percentage proportions 25/25/50. This should look something like this:

Tie Split Testing to a Business Calendar

You can also tie Split Testing to a Business Calendar. For example, you might prefer to keep Split Testing outside normal business hours, or maybe run Split Testing in response to specific customer offers or other commercial events. To do this you can either use an existing Business Calendar or create a new one for your specific purpose. You then add this calendar to the rule to which you want to apply Split Testing.

So, when the parameters set for the Business Calendar apply, the weighted Split Testing begins and ends automatically.

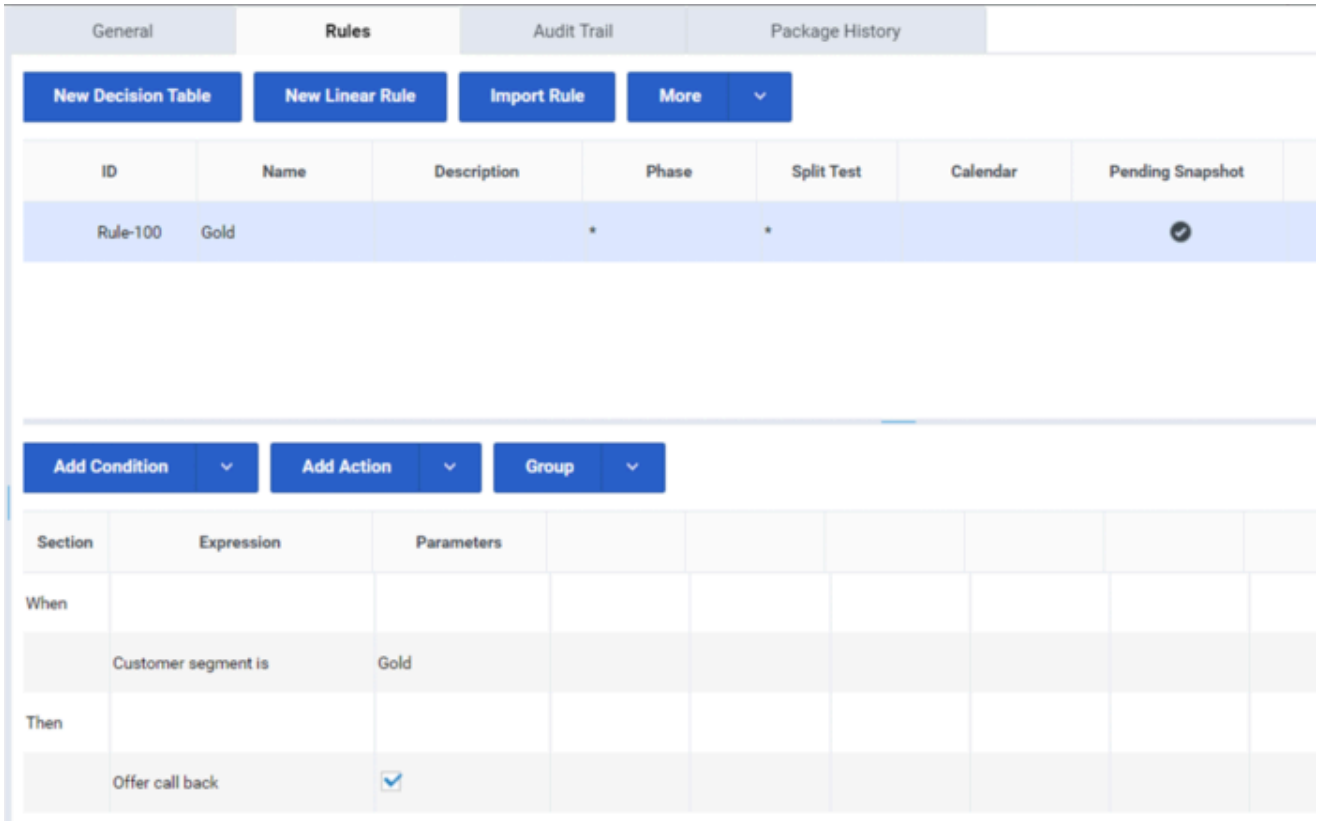


Using Test Scenarios to Check Your Split Test Logic

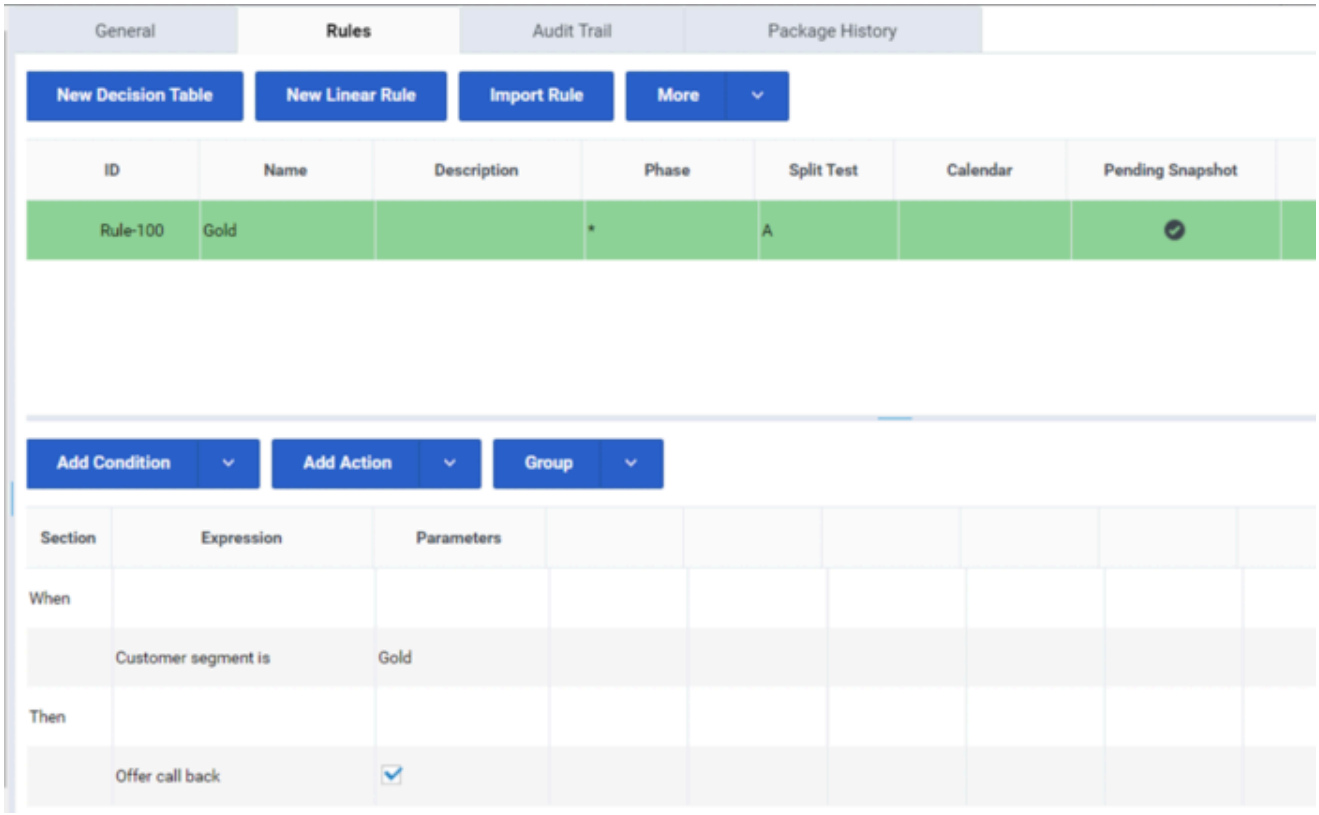
GRAT's Test Scenario feature allows you to run pre-deployment checks on the logic of any rule package that you want to deploy. Because A/B/C Split Testing enables an additional level of logic to be built into a rule package, it's advisable to run more complex Split Test logic through a Test Scenario before deploying it to a production rules engine.

Example

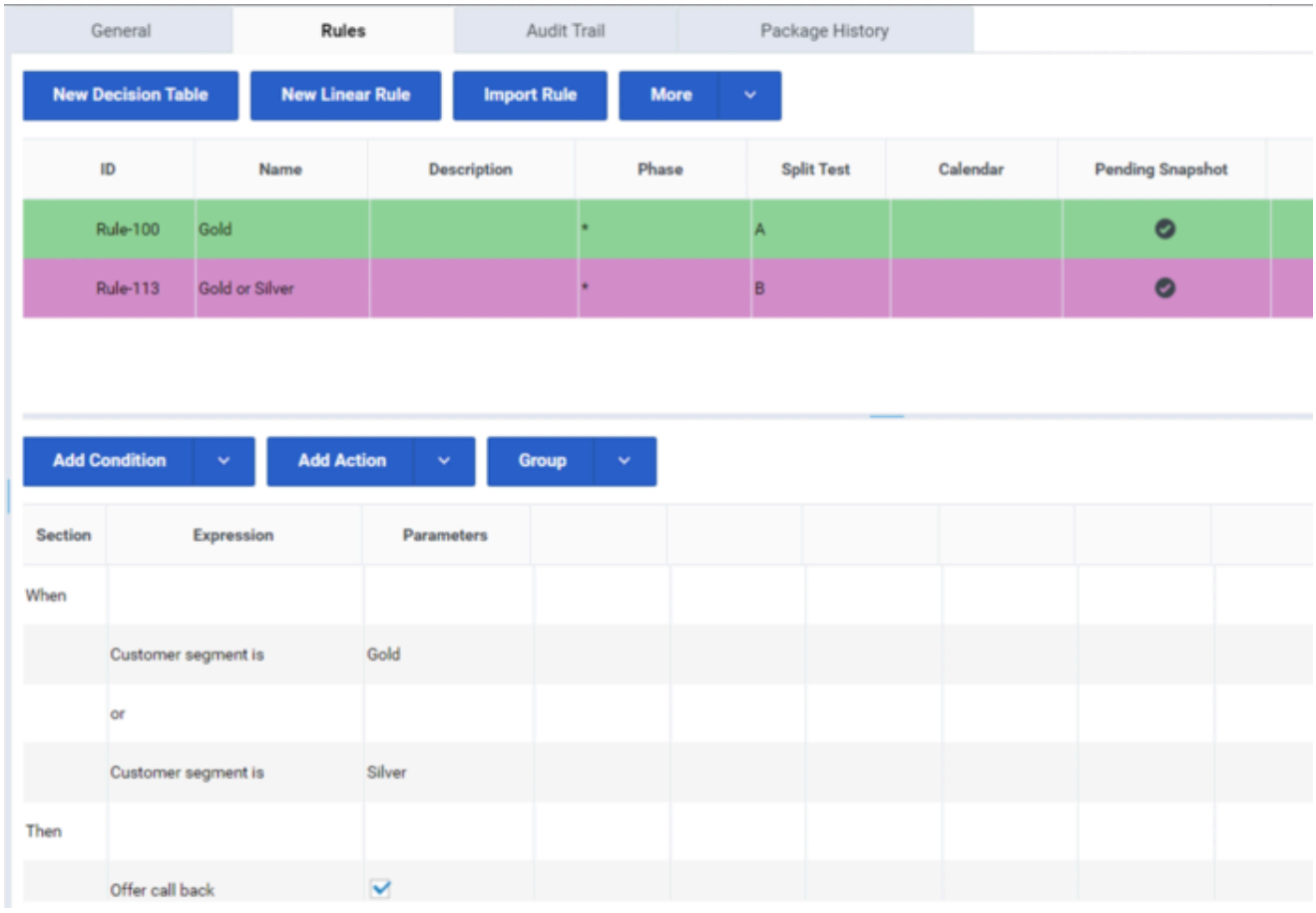
1. You have a simple Call Treatment rule which states that only Gold customers will be offered a callback.



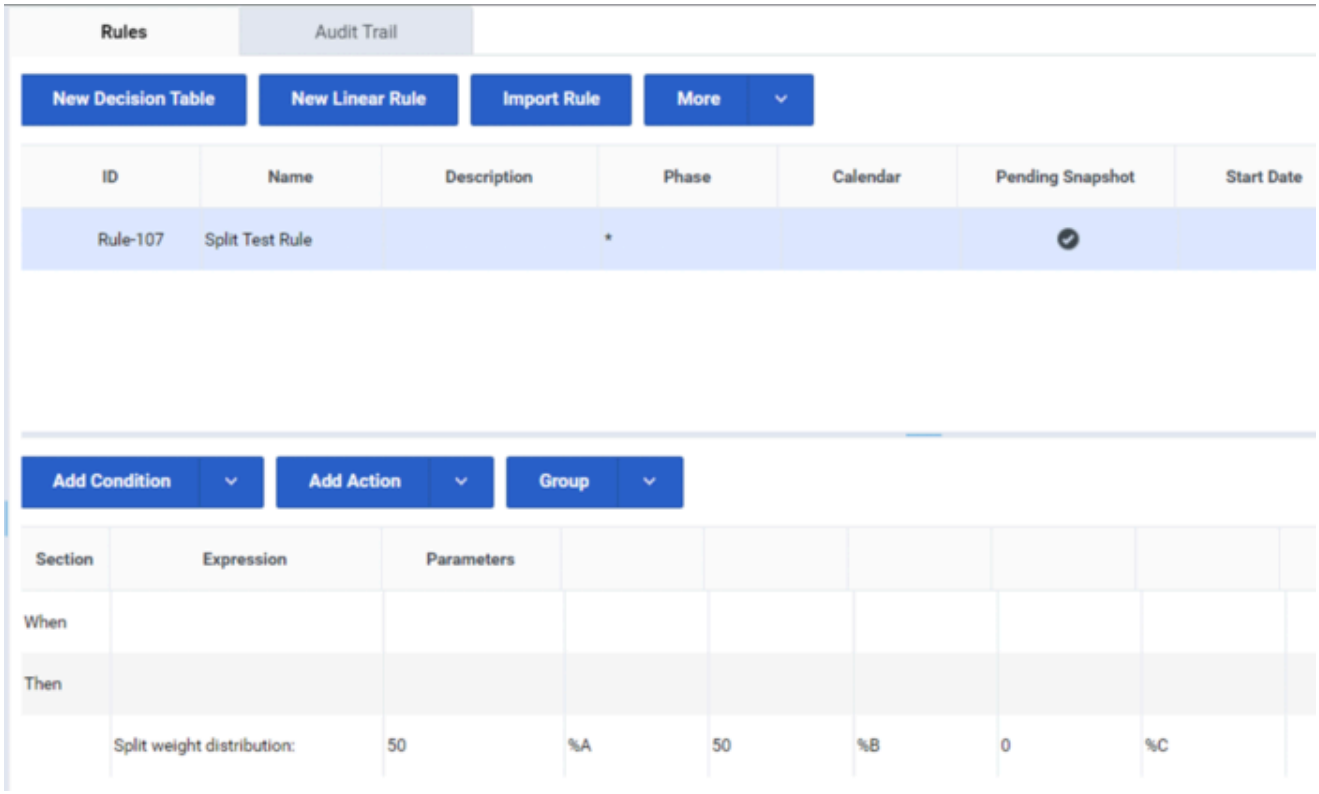
2. You want to test what would happen if you started offering callback to Gold and Silver customers. You decide you want to test this change by integrating it in for only 50% of the calls. So, you make the standard rule split test **A**:



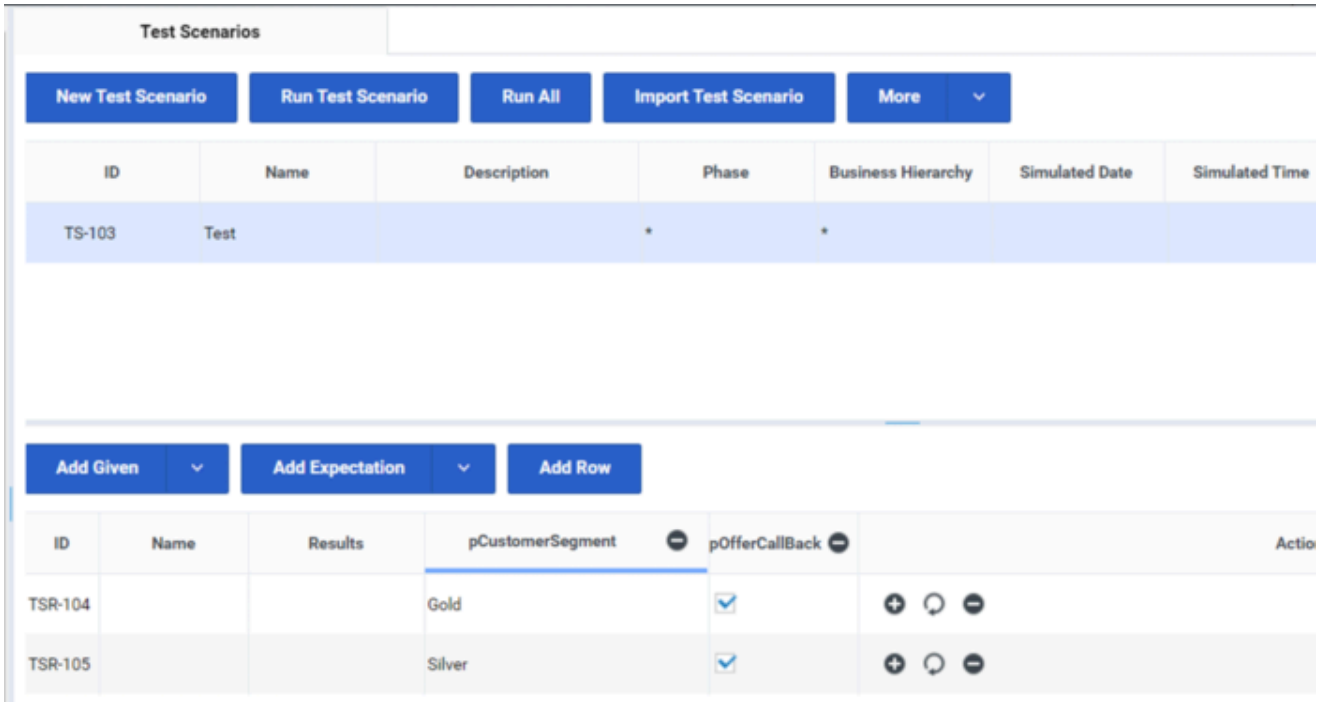
3. The next step is to copy and paste the rule. Make the second rule split test **B**, then change the rule logic to allow **Gold** or **Silver**:



4. Next, create a simple Split Test rule, which will determine when to use rule **A** and when to use rule **B**. In this use case, you want this distributed 50% **A** and 50% **B**. So, select the action `Split weight distribution 50% A, 50% B and 0% C`. This action is provided in the Split Test example template.

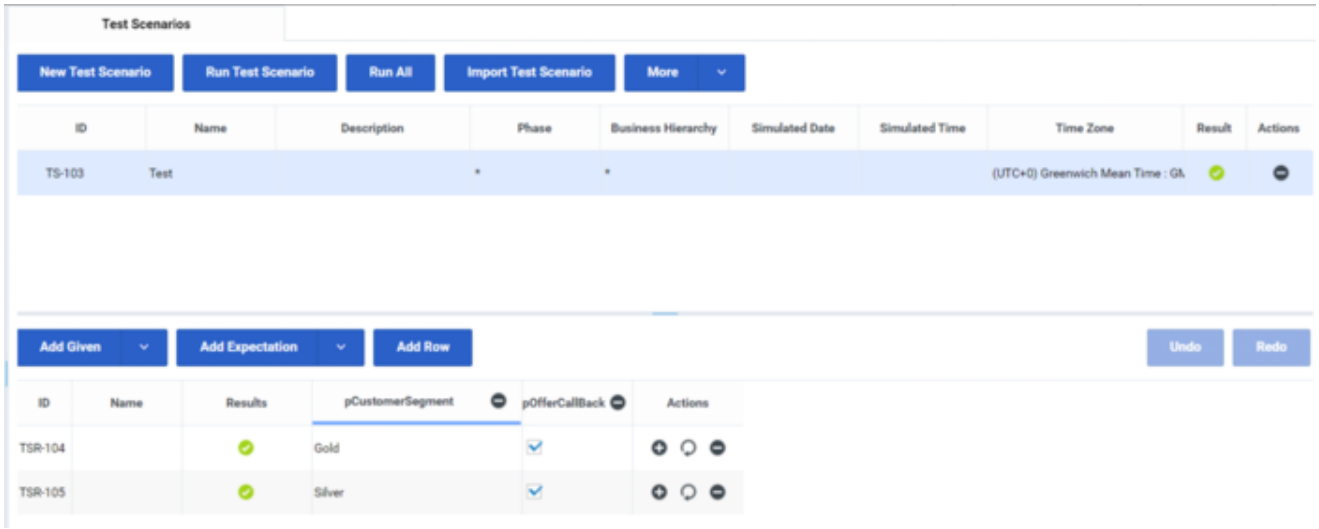


5. Now, create a test scenario to test the changes. In this case, if the customer is **Gold**, you expect to always offer a callback. If the customer is **Silver**, you expect to offer a callback 50% of the time:

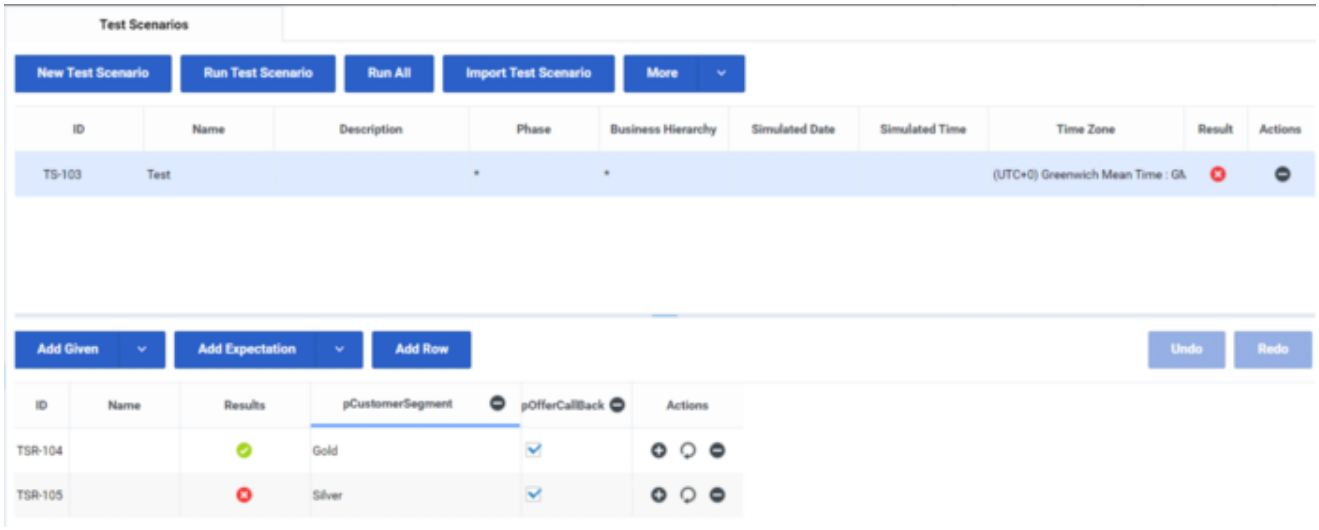


6. If you now run the test scenario multiple times, you will see that 50% of the time you get a green light on the Silver (indicating that you offered a callback) and 50% of the time you get a red light (indicating that you did not offer a callback):

First run



Second run



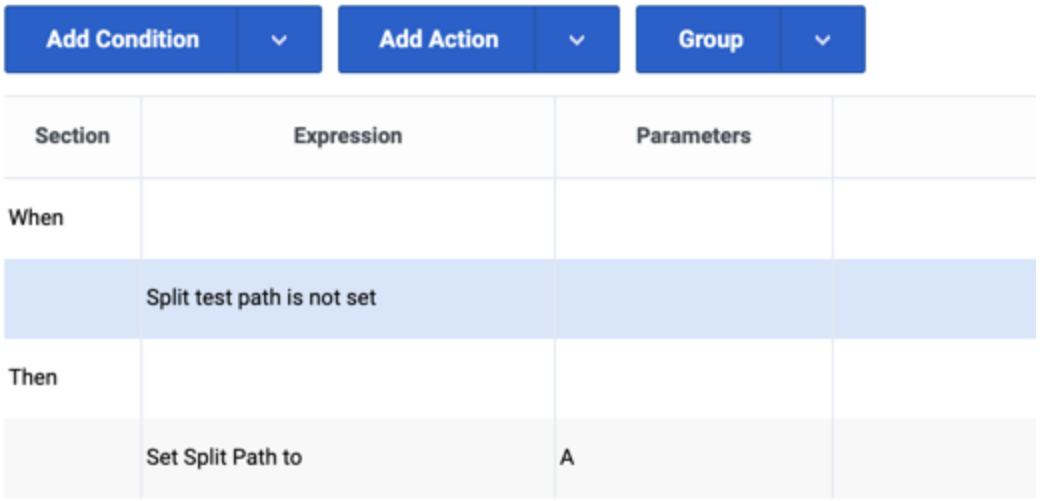
7. After testing that the A/B/C split logic is correct, you can deploy your rule package. At this point, the A/B/C split logic will run as you have it configured during normal rule evaluation. To adjust the A/B/C configuration (for example, make the **A** path 25% instead of 50% and the **B** path 75% instead of 25% and so on), you must make the changes in GRAT and then redeploy the rule package.

Allowing an Application to Override Split Test Configuration Rules

In addition to setting the A/B/C path in the Split Test Configuration section, it is possible for the invoking application (GVP, IVR, ORS and so on) to set the value. This can either be used as a default, or can be used to override the logic in the Split Test Configuration.

The application can pass in the split test value in the **_GRS_Environment** fact (field name: **splitTestPath**).

If you want the invoking application to be able to override the normal logic in Split Test Configuration, then you can simply add the `Split Test Path is not set` condition to your rule.



This condition will check whether the value has already been passed in by the invoking application. If it has been passed in, then the Split Test logic will be bypassed. If not, then the normal Split Test rule logic will apply.

Rule Template Development Overview

Rules templates allow the business rules developer to create rule templates that define the conditions and actions that will be used by the business rule author. The developer creates the plain-language statements that the business author will see and maps them to the rule language statements that the rules engine will execute. For each rule condition and action, the developer decides what kind of data the rules author will be providing. Some examples include whether the input should be an integer value, a non-integer numeric value, a string, a selection from a pre-defined list, or a selection from a dynamic list. Rule templates are used by rules authors to build rules for task classification and prioritization at the Global, Department, and Processes levels of the business structure of Genesys solution.

Quick links—Working with Templates

Importing and exporting templates
Creating and editing templates
Publishing templates
Examples of template development
Rule language mapping
Using Drools 5

Quick Links—Working with Editors

Actions Editor
Conditions Editor
Enumerations Editor
Fact Model Editor
Functions Editor
Parameters Editor

Template contents

Rule templates are made up of several elements:

- Actions
- Conditions
- Enumerations
- Fact models
- Events
- Functions
- Parameters

Actions and Conditions

Actions and conditions define WHEN/THEN scenarios, such as WHEN a customer is a Gold customer,

THEN target the GoldAgentGroup. The WHEN statement is the condition, and the THEN statement is the action. A rule may have zero or more conditions, and one or more actions. This example also includes parameters: the status of the customer (Gold) and the name of the Agent Group (GoldAgentGroup).

Whenever a condition contains a rule language mapping that begins with `eval(...)`, you must enclose the entire expression in parenthesis, as follows:

```
( eval(.... ) )
```

This will ensure it will compile properly when used with the NOT operator.

See Actions editor and Conditions editor.

Enumerations

Enumerations are used to define lists of possible choices that will be displayed to the business rule author, when the author is creating rules that are based on the rule template. In some cases, the list of possible choices will be selected dynamically from Genesys Configuration Server objects or from external data sources. For WFM Activities and Multi-Site Activities, the list of possible choices is retrieved dynamically from the Genesys WFM Server. Thus, enumerations are used during definition of a discrete list of choices that will not change dynamically.

See Enumerations editor

Fact models

All rule templates include a fact model with one or more facts. A fact model structures basic knowledge about business operations from a business perspective. A fact model focuses on logical connections (called facts) between core concepts of the business. It indicates what you need to know about the business operations in order to support (or actually do) those operations.

A good fact model tells you how to structure your basic thinking (or knowledge) about the business process based on a standard vocabulary. By using standard, business-focused vocabulary, it ensures that the business rules themselves can be well-understood by key stakeholders such as business analysts. For example, in your business you may have a Fact that represents a Customer, and another Fact that represents an Order.

The Customer could have fields such as name, age, location, credit rating, and preferred language. The Order may have fields such as order amount and order date. A rule could be constructed using these values such as:

When Customer is at least 21 years old and his order is > 100.00 then invite customer to participate in survey.

See Fact model editor

Events

In order to support Complex Event Processing, template developers need to be able to designate certain facts as events, and rules authors need to change the way that the DRL is generated when a fact is designated as an event.

So the fact model includes include events, and the fact model dialog now includes a Create Event button. An event has the following fields:

- Name
- Description
- An optional list of Properties.
- User-defined expiration metadata for the event

In GRAT, the `@role` meta-data tag determines whether we are dealing with a fact or an event. The `@role meta-data` tag can accept two possible values:

- `fact`—Assigning the fact role declares the type is to be handled as a regular fact. Fact is the default role.
- `event`—Assigning the event role declares the type is to be handled as an event.

Functions

Functions are used to define elements other than Conditions and Actions, for example, when parsing of timestamps is required. The Functions editor enables you to write specific Java functions for different purposes for use in rule templates. The specified functions may then be used in the rule language mappings (see Rule Language Mapping).

When the rule templates are created, the rule developer publishes them to the Rule Repository, making them available in the GRAT for business users to create rules.

Actions and conditions can contain parameters. Various types of parameters are supported.

Certain dynamic parameter types that refer to external data sources require a Profile to be selected. The Profile is defined as a Script object of Data Collection type, and it provides connection information that enables the GRAT to retrieve this dynamic data from the external data source. The next sections describe how to configure Profiles for database, Web Service, and Workforce Management parameters.

See Functions editor.

Parameters

Database, Web Service and Workforce Management parameters are used in the actions and conditions.

Database Parameters

Database Parameter Properties

Property	Mandatory/optional	Description
driver	Mandatory	The name of the jdbc driver to be used. For example,

Property	Mandatory/optional	Description
		`com.mysql.jdbc.Driver`
url	Mandatory	The url for the database in the correct format for the jdbc driver to be used.
username	Mandatory	A valid username to connect to the database.
password	Mandatory	The password needed for the user to connect to the database.
initSize	Optional	The initial size of the connection pool. The default is `5`.
maxSize	Optional	The maximum size of the connection pool. The default is `30`.
waitTime	Optional	The maximum time (in milliseconds) to wait for obtaining a connection. The default is `5000`.

In general, the optional values do not need to be set or changed.

You can only configure database parameters with an SQL SELECT statement. Any other type of statement will fail when configured.

Web Service Parameters

In Configuration Server, Web Service Scripts must have a section called webservice. The table below lists the properties that you can specify for web service parameters.

Web Service Parameter Properties

Property	Mandatory/optional	Description
host	Mandatory	The host for the service.
base-path	Mandatory	The base path to access the service.
protocol	Optional	The default is `http`.
port	Optional	The default is `80`.
headers	Optional	Any custom HTTP headers that are needed for the service.
parameters	Optional	Any custom HTTP settings that are needed to tune the connection.

In general, the parameters values do not need to be set or changed. Headers and parameters are lists in the following format:

```
key:value[,key:value]
```

Warning:	You cannot specify headers or parameters that contain "," in the value. Warning: If you are sending a message to the service, it is expected that `Content-Type` is specified in the header since it defines the overall message interaction with the server. An optional charset can be included. For example, `Content-Type:applicaton/json;charset=UTF-8`.

You have to completely define the message to be sent and it must be constant. No variable substitution is done. The XPath Query is used to pull values out of the response from the server. The response must be in XML or JSON, otherwise this will not work. A valid XPath query for the response must be specified. This depends entirely on the service you interface with.

Note:	The message is sent to the server only once per session. This is done both for performance reasons and because the values in the response are expected to be relatively constant.

The path for the parameter is added to the `base_path` in the script.

For example:

If the Script contains:

```
host = api.wunderground.com
base_path = /auto/wui/geo/ForecastXML/
```

and Template Development specifies:

```
query type = List
XPath Query = //high/fahrenheit/text()
HTTP Method = GET
path = index.xml?query=66062
message (not set)
```

then the message that is sent is:

```
GET /auto/wui/geo/ForecastXML/index.xml?query=66062 HTTP/1.1
```

This will return the week's highs in Fahrenheit:

```
81
77
81
81
83
85
```

Workforce Management Parameters

In Configuration Server, Workforce Management Scripts must have a section called wfm. Table 4 lists the properties that you can specify for Workforce Management parameters.

Workforce Management Parameter Properties

Property	Mandatory/optional	Description
wfmCfgServerApplName	Mandatory	Configuration Server application name for the WFM server.
wfmCfgServerUserName	Mandatory	Configuration Server user name.
wfmCfgServerPassword	Mandatory	Configuration Server password.
wfmServerUrl	Mandatory	URL of WFM Server.

When configuring a new parameter of type “Workforce Management”, simply name the parameter and choose the WFM profile (script object just created) from the drop-down list. When the author is using this parameter, the GRAT will fetch the current list of WFM Activities from the WFM Server and display them to the rule author.

Support for User-Defined Template Types

GRAT automatically displays the list of template types published to it, and template designers can select these user-defined template types or define new ones, according to their own needs.

Template versions

Each time a rule template is published, a new version is created in the repository. The rule author will be able to select any version of the template from the Template Selection dialog when creating a rule package. This dialog shows the last N versions of a template, where N is a value configured by using configuration option display-n-template-versions in Genesys Administrator.

When you are publishing newer versions of the rule template, be aware that certain changes could affect rules that already have been created using the earlier version of the template. Be careful not to make changes that could void existing rules, unless these changes are communicated to the rule author.

For example, if Rule Template version 1 contains a condition that is removed later in version 2, then if a rule were already built using that condition, it will no longer compile if the rule author upgrades to Rule Template version 2.

For example, if the configuration were set to show the last 3 versions of a template, the currently selected template is `GRS Template` version 2, and there are 5 versions in the repository, we would show `GRS Template` versions 5, 4 and 3, as well as `GRS Template` version 2. Users could choose between versions 3, 4, or 5.

Version Comment

In order to provide details about the differences between template versions, rules template developers can publish a version comment that describes specific changes made to individual template versions. This version comment appears in GRAT in the Template Selection table, and can

be edited by the rule author in GRAT.

Importing and exporting templates

Import a template project

Use this panel to import previously exported templates—for example, sample templates or templates exported from another system.

1. Select the **Templates** node in the left-hand navigation tree.
2. Select the **Import Templates** button.
3. Select the **Choose file** button, browse to your file system, and select the template project to import.
4. Clicking **Replace existing templates in repository** clears the repository of any previous versions of each template before the new ones are imported into the target system. If this option is not enabled and an existing template with the same name is found in the repository, an error message appears and the import is terminated.
5. Click **Finish**.

> ### Warning
> Be careful when changing templates or template versions as it could affect existing rules. For example, an existing rule might use a condition that does not exist in a different version of the template. Consult with the rule template developer to ensure that you are choosing the correct templates and versions for your application.

Export a template project

1. Select the **Templates** node in the left-hand navigation.
2. Select the template project(s) that you want to export.
3. Name the export file and location to which the template project will be exported.
4. Click **Finish** to complete the export.

> ### Important
> It is up to the rule template developer to ensure that the templates that they develop do not have issues with name collision. For example, function names, Java method signatures, and facts should have different names even if they are in different templates, because a rule author can create rules based on multiple templates.

> Names should not be duplicated and it must be communicated to the rules author which templates/versions to use, and in which combination.

Creating and Editing Templates

Rule Templates are created as Projects in the **Template Projects** node on the **Template Development** tab of GRAT.

Create a new Template Project

1. Enter a name for the new template project. Template names must be unique within a tenant. Click **Next**.
2. Select the type of template you are creating from the drop-down list, or enter the name of a new template type to create. To create an iWD template, select `iWD`.
3. Click **Finish**. The new template project will now appear.

Editing and Configuring Rule Templates

Once it is created, the rule template appears in the left navigation pane. Expanding the template displays a list of components that can be configured. Double-click the component type to open the appropriate Editor and begin configuring components.

Renaming Rule Templates

To rename a rule template, just edit its name. A copy of the previously named template remains in the repository.

> ### Important
> Duplicate template names are not allowed within tenants, but are allowed in different tenants. Creating such a duplicate name will rename the project, but the name as published in GRAT is set via **Project/Properties/Template** Properties**.**

Deleting Rule Templates

Rule templates can be deleted provided that:

- The user has rule template delete permissions, and;

- The rule is not used in any rule package.

Publishing Templates

You must publish the template in order for it to be available to business users to create rules. Publishing is also the preferred mechanism for sharing the templates so other template developers can edit or modify the templates, if necessary. The visibility of the template is determined by access permissions. These permissions are defined in Configuration Manager or Genesys Administrator by an Administrator. Each template has a corresponding Script object in Genesys Configuration Server for which access control can be configured.

Rule authors can select prior versions of published rule templates. You can optionally publish a version comment for a specific template in order to inform rule authors about the specific differences between individual versions of a template.

Publish the template to the repository

1. Select the template in the left-hand navigation and click **Publish**.
2. Add a version comment.
3. Click **Publish** to publish the template.

Examples of template development

This section provides some examples of what a rule developer might configure in the Template Development tab. For specific information about how rule templates are configured to be used with the Genesys intelligent Workload Distribution (iWD) solution, refer to the *iWD and Genesys Rules System* guide.

Example 1: Condition and Action

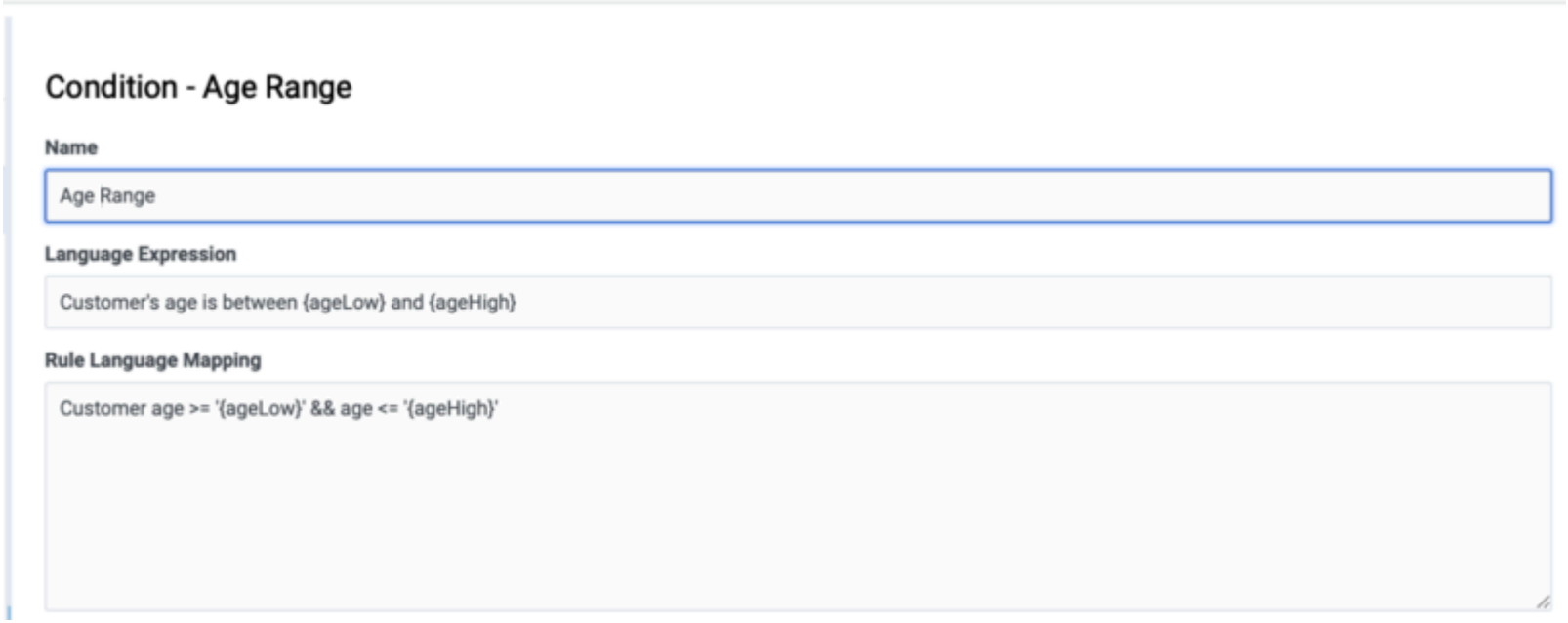
Age Range Condition

If a customer's age is within a specific range, a specific Agent Group will be targeted. In this scenario, the Condition is whether the customer's age falls within the range. In the Genesys Rules Development Tool, the conditions would be configured as follows:

```
Name: Age Range
Language Expression: Customer's age is between {ageLow} and {ageHigh}
Rule Language Mapping: Customer(age >= '{ageLow}' && age <= '{ageHigh}')
```

Do not use the word 'end' in rule language expressions. This causes rule parsing errors.

The figure below shows how this condition would appear in GRAT Template Development.



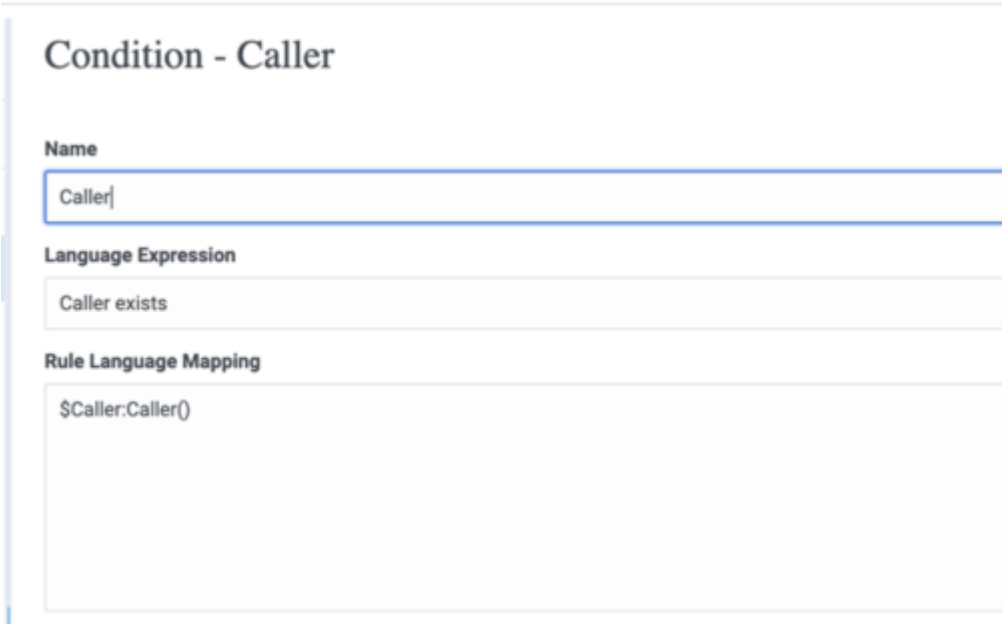
Caller Condition

In addition to testing that the Caller exists, the next condition also creates the `$Caller` variable which is used by actions to modify the Caller fact. The modified Caller will be returned in the results of the evaluation request.

You cannot create a variable more than once within a rule, and you cannot use variables in actions if the variables have not been defined in the condition.

```
Name: Caller
Language Expression:  Caller exists
Rule Language Mapping: $Caller:Caller
```

The figure below shows how this condition would appear in GRAT Rules Development.



Target Agent Group Action

The action would be configured as follows:

```
Name: Route to Agent Group
Language Expression: Route to agent group {agentGroup}
Rule Language Mapping: $Caller.targetAgentGroup='{agentgroup}'
```

The figure below shows how this action would appear in GRAT Rules Development.

Parameters

The condition in this example has two parameters:

- `{ageLow}`
- `{ageHigh}`

The action has the `{agentGroup}` parameter. The Parameters Editor screenshot shows a sample `{ageHigh}` parameter.

How it works

The way the preceding example would work is as follows:

1. The rule developer creates a fact model (or the fact model could be included as part of a rule template that comes out of the box with a particular Genesys solution). The fact model describes the properties of the `Customer fact` and the `Caller` fact. In this case we can see that the `Customer` fact has a property called age (probably an integer) and the `Caller` fact has a property called `targetAgentGroup` (most likely a string).
2. The rule developer creates the `ageLow` and `ageHigh` parameters, which will become editable fields that the business user will fill in when they are

authoring a business rule that uses this rule template. These parameters would be of type `Input Value` where the `Value Type` would likely be integer. The rule developer optionally can constrain the possible values that the business user will be able to enter by entering a minimum and/or a maximum.

3. The rule developer also creates the `agentGroup` parameter, which will likely be a selectable list whereby the business user would be presented with a drop-down list of values that are pulled from Genesys Configuration Server or from an external data source. The behavior of this parameter depends on the parameter type that is selected by the rule developer.
4. The rule developer creates a rule action and rule condition as previously described. The action and condition include rule language mappings that instruct the Rules Engine as to which facts to use or update based on information that is passed into the Rules Engine as part (of the rule evaluation request coming from a client application such as an SCXML application).
5. The rule developer publishes the rule template to the Rules Repository.
6. The rules author uses this rule template to create one or more business rules that utilize the conditions and actions in the combinations that are required to describe the business logic that the rules author wants to enforce. In this case, the previously described conditions and action above likely would be used together in a single rule, but the conditions and action could also be combined with other available conditions and actions to create different business policies.
7. The rules author deploys the rule package to the Rules Engine application server.
8. A client application such as a VXML or SCXML application invokes the Rules Engine and specifies the rule package to be evaluated. The request to the Rules Engine will include the input and output parameters for the fact model. In this example, it would have to include the age property of the Customer fact. This age might have been collected through GVP or extracted from a customer database prior the Rules Engine being called. Based on the value of the `Customer.age` fact property that is passed into the Rules Engine as part of the rules evaluation request, the Rules Engine will evaluate a particular set of the rules that have been deployed. In this example, it will evaluate whether `Customer.age` falls between the lower and upper boundaries that the rules author specified in the rule.
9. If the rule evaluates as true by the Rules Engine, the `targetAgentGroup` property of the `Caller` fact will be updated with the name of the Agent Group that was selected by the business rules author when the rule was written. The value of the `Caller.targetAgentGroup` property will be passed back to the client application for further processing. In this example, perhaps the value of `Caller.targetAgentGroup` will be mapped to a Composer application variable which will then be passed into the Target block to ask the Genesys Universal Routing Server to target that Agent Group.

Example 2: Function

Functions are used for more complex elements and are written in Java. In this example, the function is used to compare dates. It would be configured as follows:

```
Name: compareDates
```

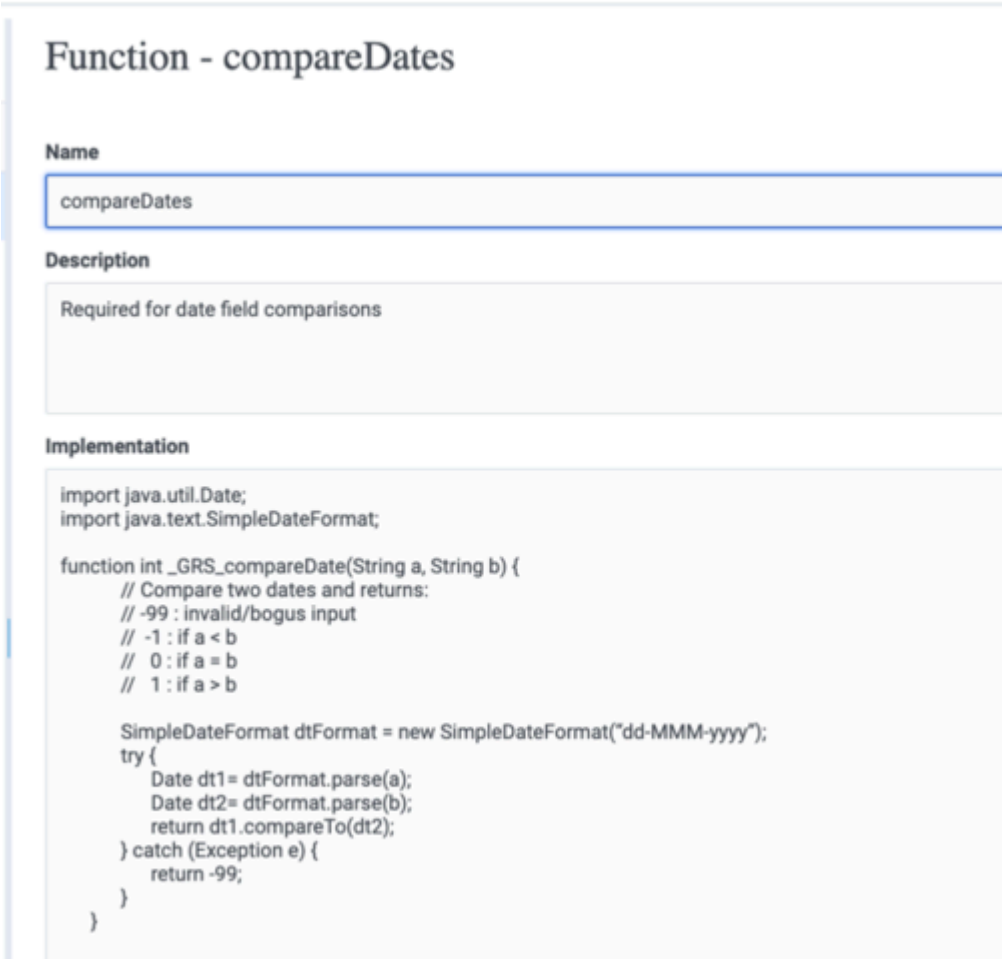
```
Description: This function is required to compare dates.
Implementation:
import java.util.Date;
import java.text.SimpleDateFormat;

function int _GRS_compareDate(String a, String b) {
            // Compare two dates and returns:
            // -99 : invalid/bogus input
            //  -1 : if a < b
            //   0 : if a = b
            //   1 : if a > b

            SimpleDateFormat dtFormat = new SimpleDateFormat(“dd-MMM-yyyy”);
            try {
                  Date dt1= dtFormat.parse(a);
                  Date dt2= dtFormat.parse(b);
                  return dt1.compareTo(dt2);
            } catch (Exception e) {
                  return -99;
            }
      }
```

For user-supplied classes, the .jar file must be in the CLASSPATH for both the GRAT and the GRE.

The figure below shows how this function would appear in GRAT Rules Development.



Function - compareDates

Name

compareDates

Description

Required for date field comparisons

Implementation

```
import java.util.Date;
import java.text.SimpleDateFormat;

function int _GRS_compareDate(String a, String b) {
        // Compare two dates and returns:
        // -99 : invalid/bogus input
        //  -1 : if a < b
        //   0 : if a = b
        //   1 : if a > b

        SimpleDateFormat dtFormat = new SimpleDateFormat("dd-MMM-yyyy");
        try {
            Date dt1= dtFormat.parse(a);
            Date dt2= dtFormat.parse(b);
            return dt1.compareTo(dt2);
        } catch (Exception e) {
            return -99;
        }
    }
```

Example 3: Using a JSON Object

Template developers can create templates that enable client applications to pass Facts to GRE as JSON objects without having to map each field to the fact model explicitly.

> **Important**
>
> Rules based on templates that use this functionality do not support the creation of test scenarios at present.

This example shows how to create a template containing a class (called `MyJson`) for passing a JSON object.

Start

1. Create the following class and import it into a rule template:

```
package simple;
 import org.json.JSONObject;
 import org.apache.log4j.Logger;

 public class MyJson {
         private static final Logger LOG = Logger.getLogger(MyJson.class);
         private JSONObject jsonObject = null;

         public String getString( String key) {
                         try {
                                         if ( jsonObject != null)
                                                         return jsonObject.getString( key);
                         } catch (Exception e) {
                         }
                         LOG.debug("Oops, jsonObect null ");
                         return null;
         }

         public void put( String key, String value) {
                         try {
                         if (jsonObject == null) {
```

```
                                jsonObject = new JSONObject();
                        }
                        jsonObject.put( key, value);
                        } catch (Exception e) {
                        }
        }
}
```

2. Create a dummy fact object with the same name (`MyJson`) in the template.
3. Add the `MyJson.class` to the class path of both GRAT and GRE.
4. Create the following condition and action:

```
Is JSON string "{key}" equal "{value}"       eval($MyJson.getString("{key}").equals("{value}"))
 Set JSON string "{key}" to "{value}"          $MyJson.put("{key}", "{value}");
```

5. Use this condition and action in a rule within the `json.test` package. The following will be generated:

```
rule "Rule-100 Rule 1"
 salience 100000
   agenda-group "level0"
   dialect "mvel"
   when
          $MyJson:MyJson()
          and (
          eval($MyJson.getString("category").equals("test"))
          )
   then
          $MyJson.put("newKey", "newValue");
 end
```

6. Deploy the `json.test` package to GRE.
7. Run the following execution request from the RESTClient:

```
{"knowledgebase-request":{
 "inOutFacts":{"anon-fact":{"fact":{"@class":"simple.MyJson", "jsonObject":
 {"map":{"entry":[{"string":["category","test"]},{"string":["anotherKey","anotherValue"]}]}}}}}}}
```

8. The following response is generated:

```
{"knowledgebase-response":{"inOutFacts":{"anon-fact":[{"fact":{"@class":"simple.MyJson","jsonObject":
```

```
{"map":{"entry":[{"string":["category","test"]},{"string":["newKey","newValue"]},
{"string":["anotherKey","anotherValue"]}]}}}}],
"executionResult":{"rulesApplied":{"string":["Rule-100 Rule 1"]}}}}}
```


End

Example 4: Developing Templates to Enable Test Scenarios

> **Important**
>
> Creation and editing of events is not supported in the initial 9.0.0 release of GRAT, so templates that support test scenarios cannot be developed. However, templates supporting events/test scenarios created in the Genesys Rules Development Tool in 8.5 can still be developed in GRDT, imported to GRAT 9.0 and used to build rules packages that support events/test scenarios.

For more information on this topic, please refer to *Developing templates to enable test scenarios* in the 8.1.3 GRDT Help.

Mapping Multiple Instances of a Rule Parameter to a Single Parameter Definition

At the point of creating parameters, instead of creating the `ageLow` and `ageHigh` parameters the rule template developer can create a single `{age}` parameter and use the underscore notation shown in the example below to create indices of it for scenarios in which multiple instances of parameter with the same type `(age)` are required (most commonly used with ranges). For example: `{age_1}, {age_2}....{age_n}` These will become editable fields. This feature is most typically used for defining ranges more efficiently.

Fact/Condition

Facts can be referenced in conditions and actions by prefixing the fact name by a `$` sign. For example, the fact `Caller` can be referenced by the name `$Caller`. GRS will implicitly generate a condition that associates the variable `$Caller` to the fact `Caller` (that is, `$Caller:Caller()`).

The condition `$Caller:Caller()` requires a `Caller` object as input to rules execution for this condition to evaluate to true.

Rule Language Mapping

When rule developers create the conditions or actions in a rule template, they enter the rule language mapping. With Genesys Rules System 9.0, use the 5.5 versions of the Drools Rule Language, found here:

```
http://downloads.jboss.com/drools/docs/5.5.FINAL/drools-expert/html/ch04.html
```

Because URLs change frequently, search the Drools web site for the Drools Expert User Guide, and then look at the table of contents of that guide for the information on the Drools Rule Language.

The rule language mapping is not visible to the business user when they are authoring rules in the Genesys Rules Authoring Tool. Instead, the rule authors will see the Language Expression that the rule template developer enters. The language expression is a plain-language description that uses terminology that is relevant to the business user, instead of low-level code. Rule language mapping is provided in the examples in the following section.

Language Expressions

When you are building a rule template, the Language Expression cannot use the open or closed parenthesis character. For example, the expression:

```
More than {parCallLimit} calls within {parDayLimit} day(s)
```

will result in an error when you try to save the rule in GRAT. But if you want the business user to see a parenthesis in GRAT, you can use backslash characters in your Language Expression. For example:

```
More than {parCallLimit} calls within {parDayLimit} day\(s\).
```

HTML Constructs

For security reasons, GRAT does not allow any HTML commands to be entered as parameters of a rule. For example, if a condition is:

```
Customer requests a callback on {day}
```

and {day} is defined as a string, we would not allow a rule author to enter the string:

```
Customer requests a callback on ‹b›Tuesday‹/b›.
```

All HTML constructs will be removed from the string. This applies to string parameters as well as dynamic list parameters such as business attributes, database or web service.

Using Drools 5

Drools 5 introduces the concept of hard and soft keywords.

Hard Keywords

Hard keywords are reserved—you cannot use any hard keyword when naming domain objects, properties, methods, functions and other elements that are used in the rule text. The following list of hard keywords must be avoided as identifiers when writing rules:

- true
- false
- null

Soft Keywords

Soft keywords are just recognized in their context, enabling you to use these words in any other place if you wish, although Genesys recommends avoiding them if possible to prevent confusion. The list of soft keywords is:

- lock-on-active
- date-effective
- date-expires
- no-loop
- auto-focus
- activation-group
- agenda-group
- ruleflow-group
- entry-point
- duration
- package
- import
- dialect
- salience
- enabled
- attributes
- rule
- extend
- when
- then
- template
- query
- declare
- function
- global
- eval
- not
- in
- or
- and
- exists
- forall
- accumulate
- collect
- from
- action
- reverse
- result
- end
- over
- init

You can use these (hard and soft) words as part of a method name in camel case, for example `notSomething()` or `accumulateSomething()` without any issues.

Escaping Hard Keywords

Although the three hard keywords above are unlikely to be used in your existing domain models, if you absolutely need to use them as identifiers instead of keywords, the DRL language provides the ability to escape hard keywords on rule text. To escape a word, simply enclose it in grave accents, like this:

```
Holiday( `true` == "yes" ) //
```

Please note that Drools will resolve that reference to the method:

```
Holiday.isTrue()
```

Actions Editor

The Actions Editor allows you to create and edit rule actions. Each action contains the same fields:

- **Name**—The name of the action.
- **Language Expression**—The plain language description of the action that the rule author sees when constructing a business rule in the Rules Authoring Tool.
- **Rule Language Mapping**—The action expressed in code. See Rule Language Mapping for more information.

When configuring actions, parameters can be used in the Rule Language Mapping and Language Expression.

For example, the action **Target Agent** may be configured as follows:

- **Name**—Target Agent
- **Language Expression**—Target specific agent "{agent}"
- **Rule Language Mapping**— $Caller.targetAgent='{agent}'

In this example, `{agent}` is a parameter.

> **Important**
>
> The above example also assumes that there is a fact called **Caller** with a field called **targetAgent**.

Actions in Linear Rules

For a linear rule, there is a maximum limit 6 columns of parameter data (including static text labels). So, for example, if your expression is:

```
Set customer data to: {parm1} and {parm2} and
{parm3} and {parm4}
```

In this case, `{parm4}`, as the 7th parameter, will not be displayed. Reword your actions to fit within these boundaries.

> **Warning**
>
> When configuring actions for an “operational parameter”, it is important that you *do*

> *not* enclose the variable in either double or single quotes in either the language expression or the rule language mapping. See the example here.

Conditions Editor

The Conditions Editor allows you to create and edit rule conditions. Each condition contains the same fields:

- **Name**—The name of the condition.
- **Language Expression**—The plain language expression of the condition that the rule author sees when constructing a business rule in the Rule Authoring tab.
- **Rule Language Mapping**—The condition expressed in code. See Rule Language Mapping for more information.

When configuring conditions, parameters can be used in the Rule Language Mapping and Language Expression. For example, the Condition Age Range may be configured as follows:

- **Name**—Age Range
- **Language Expression**—Customer's age is between `"{ageLow}"` and `"{ageHigh}"`
- **Rule Language Mapping**—`Customer(age >= '{ageLow}' && age <= '{ageHigh}')`

In this example, `{ageLow}` and `{ageHigh}` are parameters.

Conditions in Linear Rules

For a linear rule, there is a maximum limit 6 columns of parameter data (including static text labels). So, for example, if your expression is:

```
Set customer data to: {parm1} and {parm2} and {parm3} and {parm4}
```

In this case, `{parm4}`, as the 7th parameter, will not be displayed. Reword your conditions to fit within these boundaries.

> ### Warning
> When configuring a condition for an “operational parameter”, it is important that you *do not* enclose the variable in either double or single quotes in either the language expression or the rule language mapping. See the example here.

Enumerations Editor

The Enumerations (Enums) Editor allows you to create and edit enumerations. An enumeration is a data type that consists of a set of named values that represent constants. Enumerations are useful for parameters that have a small number of possible values. For example, an enumeration of **CustomerValue** might be gold, silver, or bronze.

Each enumeration contains the same fields:

- **Name**—A name for the enumeration.
- **Description**—A brief description of the enumeration.
- **Values**—A list of possible values. Click **Add**, **Edit**, and **Remove** to update the list.

The value provided for the **Name** property corresponds to the value of the **Fact** property that must be provided in the Rules Engine as part of a rules evaluation request. The **Name** is case-sensitive.

The value provided for the **Label** property corresponds to what the rule author will see when they use a rule condition or action that includes a parameter of type **Input Value**, with an enumerated list of values.

For example, an enumerated list called **CustomerSegment** could be defined as follows:

Name	Label
101	Bronze
102	Silver
103	Gold

In this example the **Name** consists of digits only, so case-sensitivity is not an issue. For example, if the **Name** properties were bronze, silver, and gold then you must ensure that you enter in the exact value `Bronze` as the **Fact** property in order for the rule to be evaluated as expected. If you enter in `bronze`, the rule will not return the result you are expecting.

Fact Model Editor

Use the Fact Model Editor to create and edit Fact Models for the template. The Fact Model Editor is arranged as a standard master/details view. The master component displays defined facts and child properties for each fact.

> **Important**
> In the initial release of GRAT 9.0, there is no support for Events in the Fact Model.

Facts

To create a new fact, click in the `Facts` pane of the Fact Model Editor. Each *fact* that is created in the editor contains the following fields:

- **Name**—A name for the fact. **Important note:** Fact names must be written in English characters only. No other character set is supported.
- **Description**—A brief description of the fact.
- **Sensitive**—Determines whether the fact is logged when the Rules Engine evaluates facts. If this field is selected, the fact is not logged. If this field is cleared, the fact is logged.
- **Properties**—A list of properties values. Click Add, Edit, and Remove to update the list. Clicking Add or Edit opens a dialog box. This dialog box contains the following fields:
  - **Name**—A name for the property.
  - **Type**—The type of property. The property types are presented in a drop-down list. Each property type has an icon to indicate the type. This icon is displayed in the properties list beside the property name.
  - **Description**—A description of the property.
  - **Sensitive**—Determines whether the fact property is logged when the Rules Engine evaluates the fact. If selected, the fact property is not logged. If cleared, the fact property is logged.

Functions Editor

Use the Functions Editor to develop Java methods that can be called from rule actions and conditions.

Each function contains the same fields:

Function Name—A name for the function.

Description—A brief description of the function.

Implementation—Either Java or Groovy.

Example

This function is used to compare dates. It would be configured as follows:

```
Name: compareDates
Description: This function is required to compare dates.
Implementation:
import java.util.Date;
import java.text.SimpleDateFormat;

function int _GRS_compareDate(String a, String b) {
            // Compare two dates and returns:
            // -99 : invalid/bogus input
            //  -1 : if a < b
            //   0 : if a = b
            //   1 : if a > b

            SimpleDateFormat dtFormat = new SimpleDateFormat(“dd-MMM-yyyy”);
            try {
                  Date dt1= dtFormat.parse(a);
                  Date dt2= dtFormat.parse(b);
                  return dt1.compareTo(dt2);
            } catch (Exception e) {
                  return -99;
            }
      }
```

For user-supplied classes, the .jar file must be in the CLASSPATH for both GRAT and the GRE.

Parameters Editor

The Parameters Editor allows you to create rule parameters, which are optionally used in rule conditions and actions.

> **Important**
> In release 9.0.0, mapping of parameters to fact models is not supported.

Each parameter contains the same fields in the `Details` section:

Name—A name for the parameter. **Description**—A brief description of the parameter. **Type**—The Parameter type. Full information on types is in the following topics.

Parameter Names

The underscore (_) character in parameter names has a special meaning when building rule templates. It is used to specify an index of the parameter in circumstances where the rule expression requires additional instances of the parameter. The most common example is a range definition.

For example, suppose you need to create a condition which has to check if the task's due date is in either the date1 to date2 range, or in the date3 to date4 range. You could create a condition such as:

```
Due is in "{dueDT1}" to "{dueDT2}" or in "{dueDT3}"  to "{dueDT4}")
```

But this requires the definition of 4 parameters with type `InputDate` in the `Parameters` section. This approach can become inefficient, especially if there is more than one occurrence of the condition/action.

A better solution is to use the underscore and index approach:

```
Due is in "{dueDT_1}" to "{dueDT_2}" or in "{dueDT_3}" to "{dueDT_4}"
```

Using this approach, you need to specify only one parameter, with name dueDT and type InputDate.

Categories of Parameter

The **Configuration** section contains information that is dependent on the parameter type. When a type is selected from the drop-down list, different fields are displayed that relate to that type.

There are eight main categories of parameters:

- String

- Integer
- Numeric
- Date
- Time
- Boolean
- Calendar
- Configuration Server

Input values

Boolean, Integer, Numeric, String, Date and Time are simply parameters for which the rules author can provide a value based on the defined parameter type. These parameters can also be constrained. For instance, an integer value can be constrained to be within a defined range.

Matching Patterns

For Input Value parameters of type `String`, you can enter a matching pattern that must be followed. Enter a Javascript regular expression to define the matching pattern. For example, a Zip Code parameter might have the matching pattern:

```
>^\d{5}$|^\d{5}-\d{4}$
```

which represents a 5-digit zip code. A Phone Number parameter might have the matching pattern:

```
^\(?(\d{3})\)?[- ]?(\d{3})[- ]?(\d{4})$
```

which represents a 10-digit phone number in the format (xxx)-xxx-xxxx.

Custom Tooltips

Use Custom Tooltip allows you to enter useful tooltip text when defining all Input Value parameters (except for the Boolean parameter type, which does not need a tooltip). If you check **Use Custom Tooltip**, the text that you enter in the tooltip field is displayed in GRAT when this parameter is used in a rule condition or action. If you do not check **Use Custom Tooltip**, GRAT displays an automatically generated tooltip; for example, `{age} is an integer between 1 and 99`.

The regex pattern supported should conform to the browser's Javascript engine and may vary slightly depending on the browser version.

Calendars

Calendar parameters indicate to GRAT that it should display a drop-down list of business calendars associated with the rule package being edited. The rule author can then choose one of the calendars.

Example: Calendar parameters can be used in a rule to dynamically assign a calendar as follows:

```
Assign business calendar "{businessCalendar}"
```

When defining a Calendar parameter, the template designer only needs to provide the parameter name and select a type of Calendar. There is no other configuration required.

Configuration Server

Configuration Server parameters give the rule author the ability to choose a single value from a drop-down list of values. For example, a Configuration Server parameter may be configured to pull a list of Agent Groups from the Configuration Server database. The list is populated from Configuration Server. Configuration Server parameters require you to select the Object type:

- Agent
- Agent Groups
- Agent Skills
- Business attribute
- Business context
- Extension
- External Route Point
- Interaction Queue
- Media Type
- Place
- Place Groups
- Route Point
- Switch
- T-Server
- Virtual Route Points

Selecting **Business Attribute** prompts you to select the name of the Business Attribute from a list defined in Configuration Manager.

Selecting **Business Context** prompts you to enter the level of the Business Context that is of interest for this parameter. Here, Business Context refers to the level of the hierarchy under the **Business Structure** folder in Configuration Server.

Database

Database parameters give the rule author the ability to choose a single value from a drop-down list of values. For example, a Database parameter may be configured to pull a list of Order Types from a database. The list is populated by a database query. Database parameters require the Profile Name (the name of the Configuration Server Script object that contains your database connection

information), Query Type (Single Value or List, depending on what you want to show up in GRAT), and the SQL Query to be executed.

Note: The list of values is fetched at the time the rules author logs into GRAT. If any values are updated from the external system after the user has logged into GRAT, the user must click the Logoff button and then log back in again to see any changes.

Example

To use a Database parameter, a parameter profile must previously have been configured for the Tenant in Configuration Server. This is a Script object that specifies the JDBC driver, as well as the database url, username, and password required to perform the query. Refer to the Genesys Rules System Deployment Guide for information on configuring these profiles. The name of this Script object is used as the profile name for the database parameter.

To obtain values from the database, a valid SQL Select statement must be specified. For instance, to get all the values of a column use a select statement of the following form:

```
SELECT column_name FROM table_name
```

For dynamic Database parameters, you can configure the parameter so that both a name (an internal value that is included with a rule evaluation request) and a label (the information that is displayed to a rules author when authoring a rule that uses this parameter) can be retrieved from two different database columns.

Database Profile Parameter Properties

Property	Mandatory/optional	Description
driver	Mandatory	The name of the jdbc driver to be used. For example, `com.mysql.jdbc.Driver`
url	Mandatory	The url for the database in the correct format for the jdbc driver to be used.
username	Mandatory	A valid username to connect to the database.
password	Mandatory	The password needed for the user to connect to the database.
initSize	Optional	The initial size of the connection pool. The default is 5.
maxSize	Optional	The maximum size of the connection pool. The default is `30`.
waitTime	Optional	The maximum time (in milliseconds) to wait for obtaining a connection. The default is `5000`.

In general, the optional values do not need to be set or changed.

In template development, you can only configure database parameters with an SQL SELECT

statement. Any other type of statement will fail when configured.

Operational

Operational parameters are created by users through Genesys Administrator Extension and, when deployed, are stored as options of Transaction objects of the type `List` in the Genesys Configuration Server Database. At rule execution time, when the Rules Engine evaluates a rule that contains an Operational parameter, it obtains the current value of the associated Transaction object option from Configuration Server. The rule developer determines from which Transaction object, and which option of that object, the value of the Operational parameter is fetched, and the rule author uses this parameter as part of a condition or action.

Example An Operational parameter called **waitTimeThreshold** may be defined. If a caller is waiting longer than this threshold for an available agent, some other action may be performed.

Instead of specifying a value for the threshold in the rule like the following:

```
When

Caller’s wait time is greater than 30 seconds

Then

Offer a callback
```

the rule author could specify:

```
When

Caller’s wait time is greater than {waitTimeThreshold}

Then

Offer a callback
```

The value of `{waitTimeThreshold}` can be changed at any time by a user using Genesys Administrator Extension and will have an immediate effect without having to modify and redeploy a rule package.

For example, use the following condition when you define the mapping:

```
Queue(waitTime >  {waitTimeThreshold} )
```

To configure an Operational parameter you need two IDs:

- The **List ID**, which corresponds to the name of the Transaction object in which the Operational parameter is stored
- The **Parameter ID**, which corresponds to the name of an option of that Transaction object.

The option value contains the actual value of the Operational parameter that is retrieved by the Rules Engine when the rule is evaluated. Operational parameters are always stored as Transaction objects of the type `List`, but the precise configuration of the options within that List object varies depending on how the Operational parameter was configured.

Warning

When configuring an "operational parameter", it is important that you *do not* enclose the variable in either double or single quotes in either the language expression or the rule language mapping. For example, a condition that uses an operational parameter `{opParmEwtThreshold}` should be configured like this:

- **Language Expression**—`Estimated wait time is greater than {opParmEwtThreshold}`
- **Rule Language Mapping**—`CallInfo ( ewt > Integer.parseInt({opParmEwtThreshold}))`

Note that there are no single or double quotes around `{opParmEwtThreshold}`

For additional examples, see the "Operational Parms" sample template and rule package shipped with GRAT (in the **examples** folder).

Refer to the Genesys Administrator Extension Help for general information about Operational parameters.

Select Enumeration

Select Enumeration parameters are linked with an Enumeration. This provides the rules author with a specific list from which to select.

Web Service

Web Service parameters give the rule author the ability to choose a single value from a drop-down list of values. For example, a Web Service parameter may be configured to pull a list of stock symbols from an external web service. The list is populated by a Web Service query. Web Service parameters require the Profile Name (the name of the Configuration Server Script object that contains your web service connection information), Query Type (Single Value or List), and the XPath Query to be executed. In addition, Web Service parameters require that some Protocol Settings be entered, specifically the HTTP Method, Path, and the Message Body.

Note: The list of values is fetched at the time the rules author logs into GRAT. If any values are updated from the external system after the user has logged into GRAT, the user must click the Logoff button and then log back in again to see any changes.

Example

Similar to a Database parameter, a parameter profile must also have been previously created. This profile will contain information such as the server's address (host and port), the path to the service, and any other necessary HTTP settings. Refer to the Genesys Rules System Deployment Guide for information about configuring these profiles.

To obtain values from the service, a valid message for the service must be specified. This message must be constant. In other words, no variable substitution will occur.

Note: No message can be sent for HTTP GET requests. All the information in the request is provided through the query string and/or headers.

For instance, to obtain the weather forecast for San Francisco, the following request can be made to the Weather Underground REST service:

http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=94129

However, this is the complete request. The host (api.wunderground.com) and the base path (/auto/wui/geo/ForecastXML/), must be specified in the profile.

To define a parameter to make this request, the profile name must reference the correct information described above. In addition, the Protocol Settings must specify GET as the method, along with `index.xml?query=94129` as the path. No message is needed for this request.

To obtain values from the result, a valid XPath expression must be specified. The Web Service must return results in XML or JSON. Please see the XPath specification for more information on XPath expressions.

For example, to get the forecast highs from the previously described request, the following XPath expression may be used:

```
//high/fahrenheit/text()
```

In Configuration Server, Web Service Scripts must have a section called webservice. The table below lists the properties that you can specify for web service parameters.

Web Service Profile Parameter Properties

Property	Mandatory/optional	Description
host	Mandatory	The host for the service.
base_path	Mandatory	The base path to access the service.
protocol	Optional	The default is `http`.
port	Optional	The default is `80`.
headers	Optional	Any custom HTTP headers that are needed for the service.
parameters	Optional	Any custom HTTP settings that are needed to tune the connection.

In general, the parameters values do not need to be set or changed. Headers and parameters are lists in the following format:

```
key:value[,key:value]
```

You cannot specify headers or parameters that contain "," in the value. If you are sending a message to the service, it is expected that `Content-Type` is specified in the header since it defines the overall message interaction with the server. An optional charset can be included. For example, `Content-Type:applicaton/json;charset=UTF-8.`

> **Important**
>
> In Template Development you have to completely define the message to be sent and it must be constant. No variable substitution is done. The XPath Query is used to pull values out of the response from the server. The response must be in XML or JSON, otherwise this will not work. A valid XPath query for the response must be specified. This depends entirely on the service you interface with.

> **Important**
>
> The message is sent to the server only once per session. This is done both for performance reasons and because the values in the response are expected to be relatively constant.

In Template Development, the path for the parameter is added to the `base_path` in the script. For example, if the Script contains:

```
host = api.wunderground.com
base_path = /auto/wui/geo/ForecastXML/
```

and the GRDT specifies:

```
query type = List
XPath Query = //high/fahrenheit/text()
HTTP Method = GET
path = index.xml?query=66062
message (not set)
```

then the message that is sent is:

```
GET /auto/wui/geo/ForecastXML/index.xml?query=66062 HTTP/1.1
```

This will return the week's highs in Fahrenheit:

```
81
77
81
81
83
85
```

Workforce Management

Workforce Management (WFM) parameters enable the rules author to select a value from a drop-down list of activities (a WFM database object that represents contact center tasks in which agents can be engaged) and multi-site activities (a collection of activities performed at multiple physical sites) that is dynamically retrieved from the Genesys Workforce Management Server. Workforce Management parameters require the WFM Profile (the Configuration Server Script object of type Data Collection).

> **Important**
>
> The list of values is fetched at the time the rules author logs into GRAT. If any values are updated from the external system after the user has logged into GRAT, the user must click the Logoff button and then log back in again to see any changes.

Example

An activity is the main planning object that is used when building forecasts and schedules. An activity can be associated with an individual WFM Site object, or multi-site activities can be created at the WFM Business Unit level, which aggregates information from multiple "child" activities across multiple WFM Sites. So, when providing the rules author a list of WFM activities that are dynamically fetched from the WFM Server, the name of the WFM activity or multi-site activity is prefixed with the name of the WFM Site or the WFM Business Unit, respectively.

For example, if the WFM configuration has the following structure:

Business Unit with the name ‘ACME’

- Site with the name ‘San Francisco’
- Activity with the name ‘Disputes’
- Activity with the name ‘Billing Inquiries’
- Site with the name ‘Chicago’
- Activity with the name ‘Disputes’
- Activity with the name ‘Address Changes’
- Multi-Site Activity with the name ‘Billing’ (comprised of ‘Billing Inquiries’ from the San Francisco site and ‘Address Changes’' from the Chicago site)
- Multi-Site Activity with the name ‘Disputes’ (comprised of ‘Disputes’ from the San Francisco site and ‘Disputes’ from the Chicago site)

The rules author will see the following items in the drop-down list when using the rule action Assign WFM Activity in a rule:

`B.U. ACME: Billing` `B.U. ACME: Disputes` `Site Chicago: Address Changes` `Site Chicago: Disputes` `Site San Francisco: Billing Inquiries` `Site San Francisco: Disputes`

> **Important**
>
> The names of the Business Units and Sites are pre-fixed with ‘B.U.’ and ‘Site’ respectively, to help the rules author understand the context.

In Configuration Server, Workforce Management Scripts must have a section called `wfm`. The table below lists the properties that you can specify for Workforce Management parameters.

Workforce Management Profile Parameter Properties

Property	Mandatory/optional	Description
wfmCfgServerApplName	Mandatory	Configuration Server application name for the WFM server.
wfmCfgServerUserName	Mandatory	Configuration Server user name.
wfmCfgServerPassword	Mandatory	Configuration Server password.
wfmServerUrl	Mandatory	URL of WFM Server.

When configuring a new parameter of type “Workforce Management” in Template Development, simply name the parameter and choose the WFM profile (script object just created) from the drop-down list. When the author is using this parameter, GRAT will fetch the current list of WFM Activities from the WFM Server and display them to the rule author.

Search

GRAT includes a Search feature. You can search for existing rules to more easily locate them for editing.

Some of the fields that can be searched are:

- Rule ID
- Rule Name (the full name, "starts with", or "includes")
- Last Modified By: the user name of the person who last updated the rule
- Creation Date Range (any rule created between x and y)
- Business Calendar (calendar name)
- Rules pending snapshot
- Rules that contain a specific parameter in a rule condition.
- Rules that contain a specific parameter in a rule action.

> **Important**
> After the search results are presented, you can click the rules to see the contents, modify the rules, or delete the rules from the search screen. You can also click the navigate icon to locate the context in which the rule was originally defined.