GRS Best Practice Guide

Genesys Rules System 8.5.1

2/25/2022

Table of Contents

GRS Best Practice Guide

This document describes how, once GRS is installed, to:

- Develop templates
- Author, test, deploy rules packages
- Work with the Composer Business Rules Block
- Work with supplied solution-specific templates
- Understand use cases and worked examples

Please read the **GRS Overview** document if you are unfamiliar with GRS.

To install Genesys Rules System, please refer to the **Deployment Guide**.

Genesys Rules System 101 Worked Example

Genesys Rules System 101 - Video Topics

Solution-Specific Templates

Solution-Specific Templates

Other Documents

Deployment Guide

GRS Overview

New Features by Release

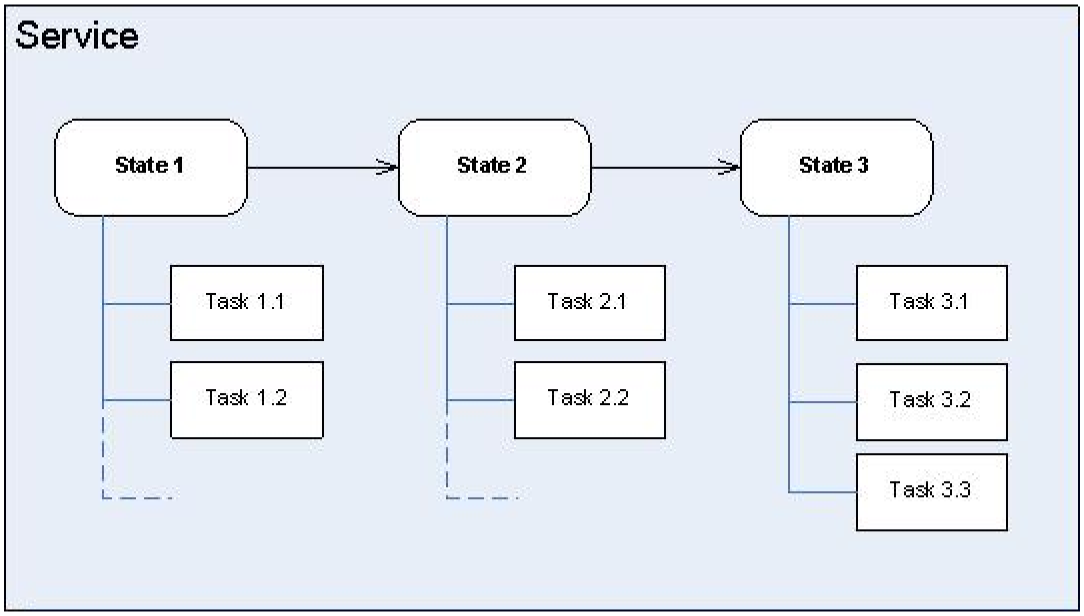
New in 8.5.1

Support for Conversation Manager Templates in Test Scenarios

In the initial 8.5.001 release of GRS, the Test Scenario feature did not support rules that were created using the Conversation Manager (CM) template. This is because the Test Scenario feature in release 8.5.001 works by taking the input data (a set of one or more facts with different fields) that is configured by the user and building the appropriate Fact model, then running the rules under GRAT using that set of data. In release 8.5.1, the Test Scenario feature now supports rules based on the CM template.

Data Structure in CM

With Conversation Manager, the data is in a hierarchical JSON format of **Customer -> Service -> State -> Task**. Any given **Customer** may have one or more **Services**. Each **Service** may be in at most one **State** at a time. Each **State** may have one or more **Tasks**. **Tasks** may also be associated directly with **Services**.

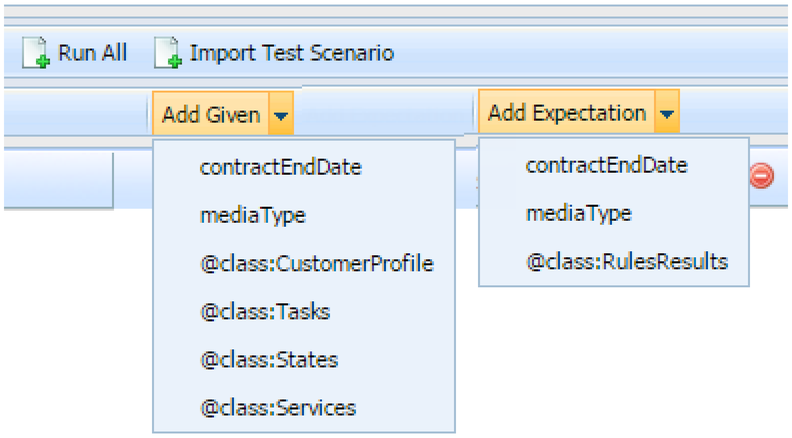


So the `Customer`, `Services`, `States` and `Tasks` Facts have now been added the lists of Facts that can be defined as **Given** fields, and the `RulesResults` Fact has been added to the list of Facts that can be defined as an **Expectation**.

> **Important**
> The current CM Template is only interested in the Type, Start Time, and Completion Time (if any) of Services, States, and Tasks.

Each of the new values is represented by a JSON string which will be the value for that field.

Now, when the type of rule for which you want to create a test scenario is a Conversation Manager rule (based on the Conversation Manager template), a series of different values for the **Given** and **Expectation** elements that reflect these more complex data structures are available. In the example below you can see the **Customer > Service > State > Task** structure is reflected by the four **@class** entries in the drop-down list of Givens and the **@class:RulesResults** entry in the drop-down list of Expectations.



When you select an **@class** entry, a new column is added. Click on a grid cell under the new column to bring up the edit dialog for that entry. The additional data listed below can be selected as either a **Given** or an **Expectation**.

Additional CM Template Objects

Givens

The list below shows the additional provided data.

- Available by selecting one of the **@class** entries:
  - `Add Customer Attribute`
  - `Add Service`

    - `Add Service Type`
    - `Add Service Start Time`
    - `Add Service Completion Time`
    - `Add State`
    - `Add State Type`
    - `Add State Start Time`
    - `Add State Completion Time`
    - `Add Task`
    - `Add Task Type`
    - `Add Task Start Time`
    - `Add Task Completion Time`
- Available for direct selection from **Givens**:
    - `Add Interaction Media Type`
    - `Add Contract End Date`

Expectations

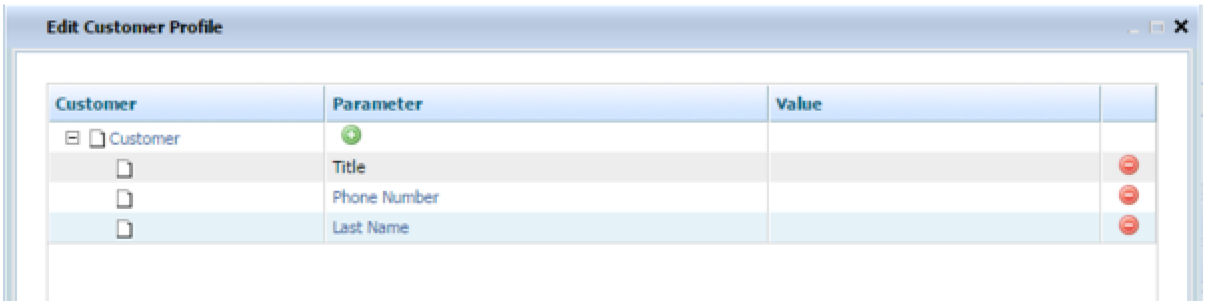
The list below shows the additional expected results:

- `Update Customer Attribute`
- `Request Specific Agent`
- `Request Agent Group`
- `Request Place Group`
- `Request Skill`
- `Send Communication to Customer`
- `Block Communication to Customer`
- `Offer Service Resumption`
- `Offer Survey to Customer`

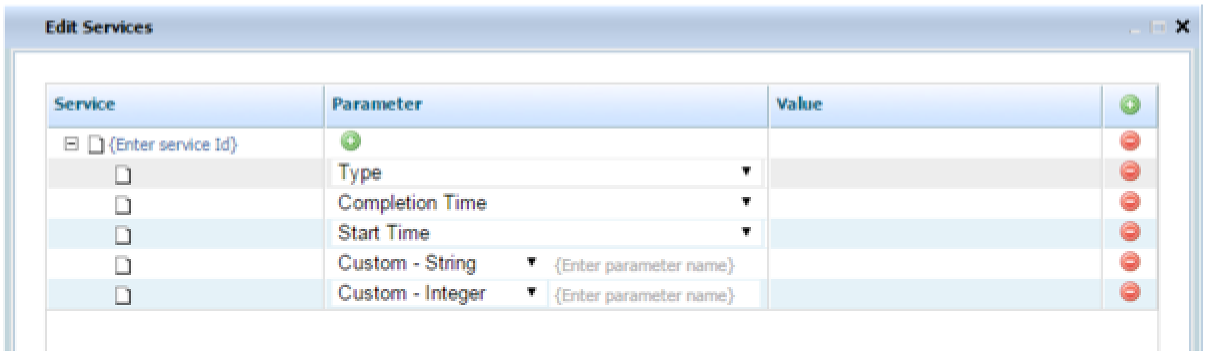
Edit Dialogs

To create entries for the **Givens** and Expectations of your Conversation Manager test scenario, select the relevant **@class** item and use the sample additional edit dialogs shown below.

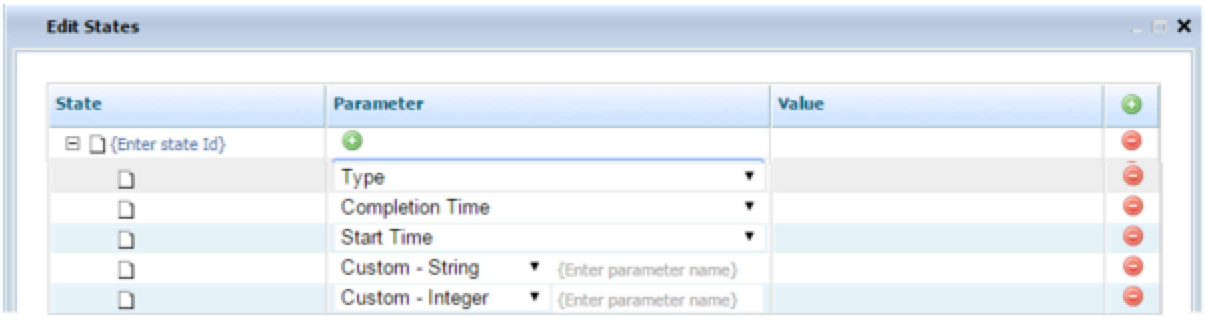
Givens


Edit Customer Profile

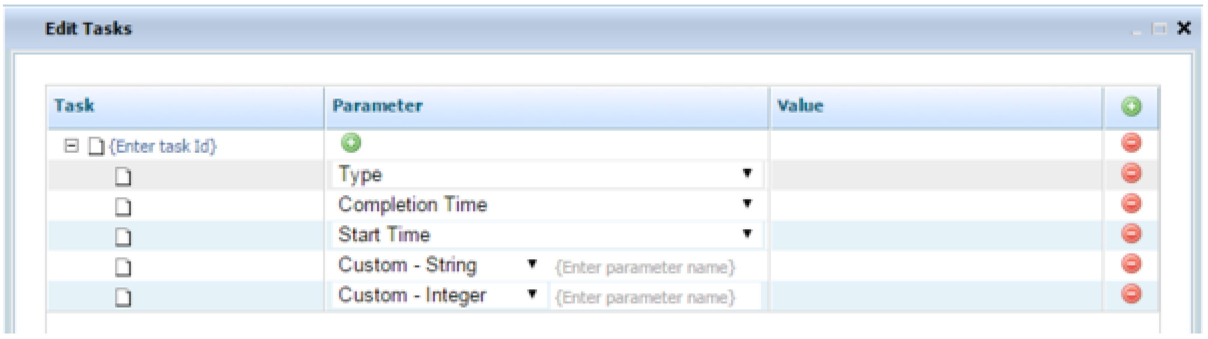
Customer	Parameter	Value	
Customer			
	Title		
	Phone Number		
	Last Name		


Edit Services

Service	Parameter	Value	
{Enter service Id}			
	Type		
	Completion Time		
	Start Time		
	Custom - String {Enter parameter name}		
	Custom - Integer {Enter parameter name}		

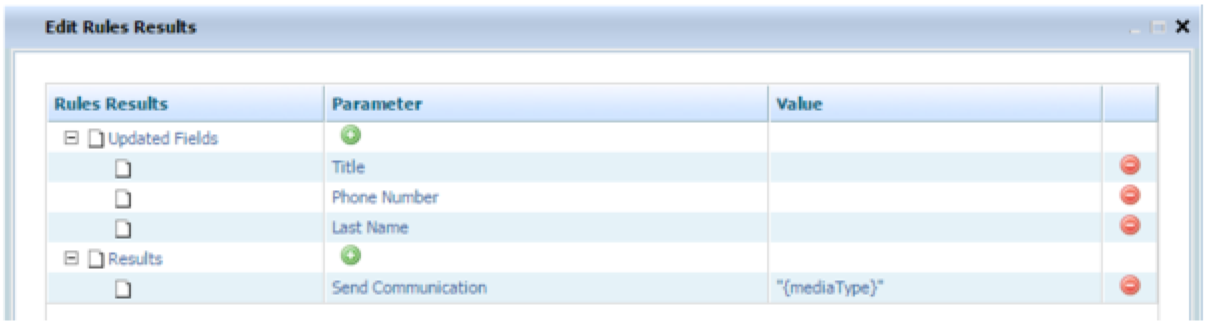

Edit States

State	Parameter	Value	
{Enter state Id}			
	Type		
	Completion Time		
	Start Time		
	Custom - String {Enter parameter name}		
	Custom - Integer {Enter parameter name}		


Edit Tasks

Task	Parameter	Value	
{Enter task Id}			
	Type		
	Completion Time		
	Start Time		
	Custom - String {Enter parameter name}		
	Custom - Integer {Enter parameter name}		

Expectations



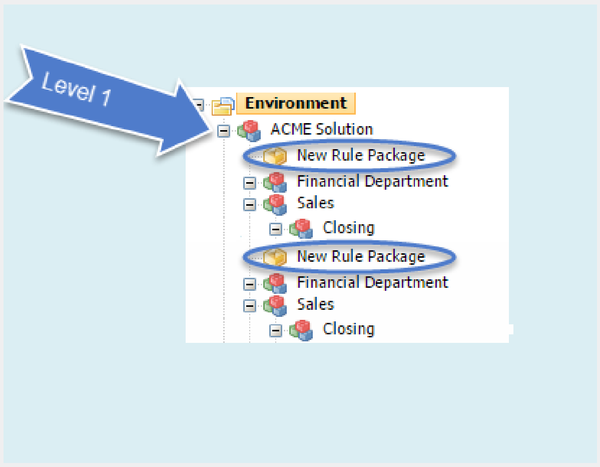
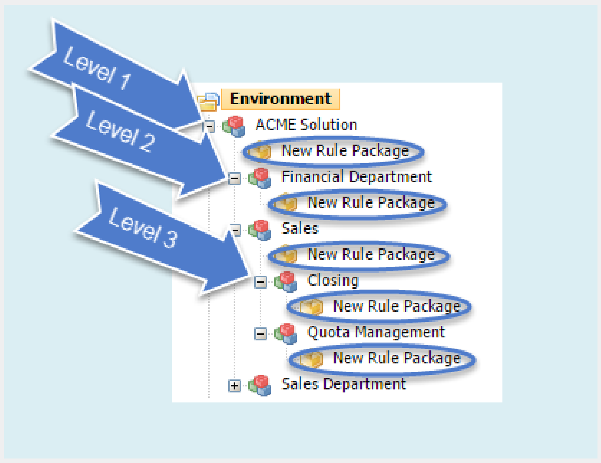
Edit Rules Results

Rules Results	Parameter	Value	
⊟ Updated Fields			
	Title		
	Phone Number		
	Last Name		
⊟ Results			
	Send Communication	"{mediaType}"	

Nested Solution Business Hierarchy

In release 8.5.1 of Genesys Rules Authoring tool, if you have permission to create a new rule (`Rule Package - Create`) you can now add a new Rule Package at any node in the business hierarchy (a *nested* solution), rather than just at the first level.

[+] FULL DESCRIPTION

Non-Nested Solution	Nested Solution
	

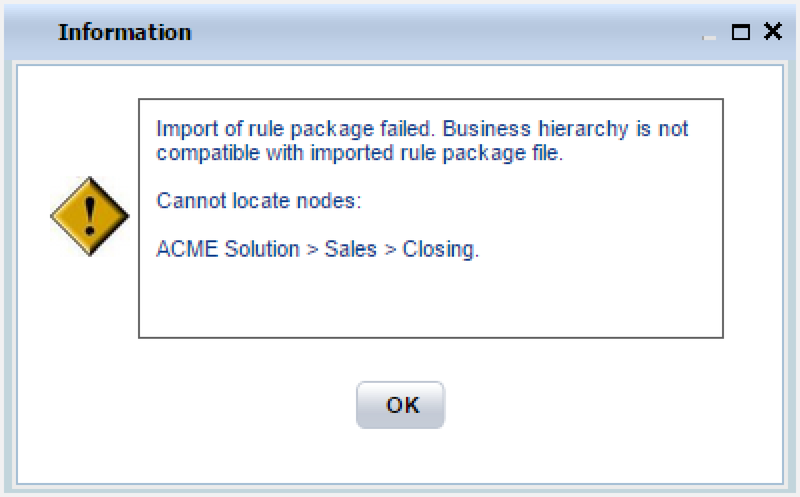
This means that:

- Your business hierarchy can be more easily organized.
- The need for lots of duplication and repetition at the Solution level in more complex business hierarchies is now removed.
- Individual users can be restricted using Role-Based Access Control to specific sub-nodes (for

example, Departments and Processes).

Importing Rule Packages

Because rules can be associated with sub-nodes in a nested hierarchy, when a rule package is imported, GRAT ensures that the business structure is compatible, and prevents an import if it is not. If GRAT finds an incompatibility, an error such as the following is displayed:



Import of rule package failed. Business hierarchy is not compatible with imported rule package file.

Cannot locate nodes:

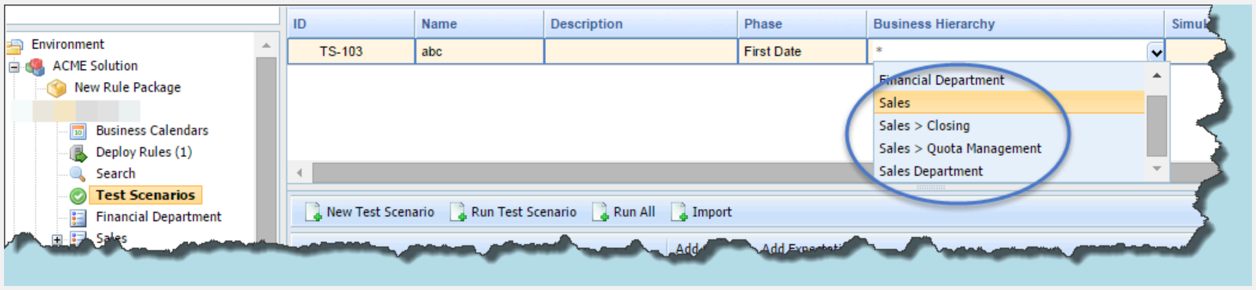
ACME Solution > Sales > Closing.

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.

Test Scenarios

For Test Scenarios, the Business Hierarchy drop-down displays the relative path underneath the selected rule package:



Deleting Business Nodes

Be careful never to delete any business structure nodes that contain active rule packages or rules, without first backing up the rule packages as XML files. While it is OK to add new nodes, or to "rename" nodes, proper permissions should be set up to prevent a GA/GAX user from accidentally deleting nodes which could cause rule packages / rules to become unreachable.

Enabling and Disabling the Feature

Because some customers might want to restrict their users to creating rule packages only under the **Solution** node, in release 8.5.100.21 a new configuration option—`enable-nested-solutions`—has been implemented to allow users to enable or disable this feature. Disabling this feature is recommended for iWD users.

- Option name—`enable-nested-solutions`
- Valid values—`true`/`false`
- Default value—`true`
- Description—Controls whether users can create new rule packages under any node in the hierarchy. For iWD, it is recommended to set this option to `false`.

New in 8.5.001.21

Business Calendar Enhancements

Business calendars have been enhanced in GRS release 8.5.2 to allow:

- Dynamic Timezone Support
- Differentiation between holidays and non-working days.

[+] FULL DESCRIPTION

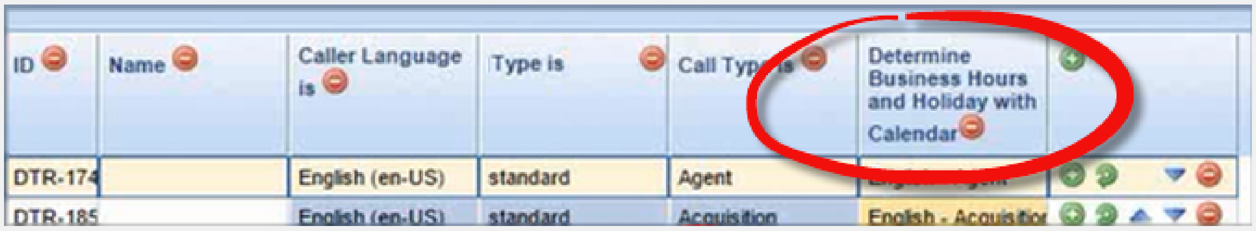
Dynamic Timezone Support

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). Business calendars have been enhanced to allow the timezone to be provided dynamically at rule-evaluation time. This feature enables you to configure a standard set of business calendars that can be re-used in any timezone.

In this release, the standard methods that can be accessed from within the rule template have been extended to allow the timezone ID to be passed in at rule evaluation time. 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.

Example

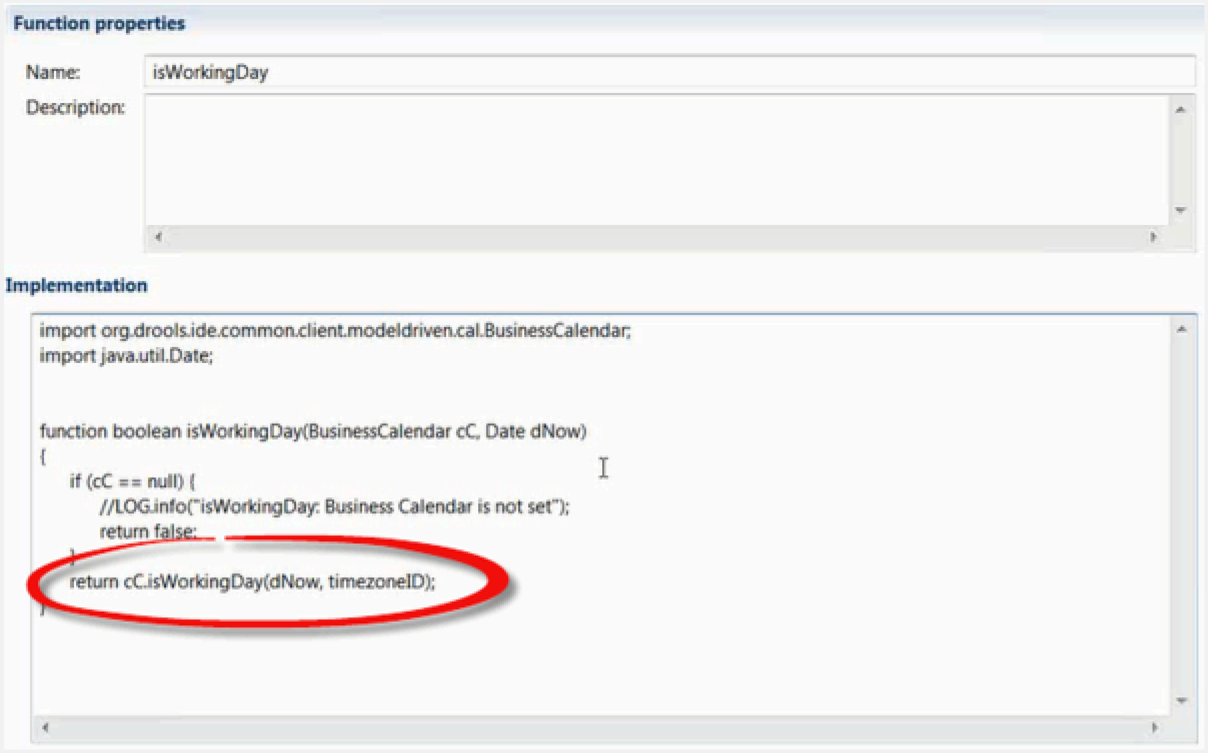
A rule, such as in the example below, asks which business calendar to use in evaluating the rule conditions;



ID	Name	Caller Language is	Type is	Call Type is	Determine Business Hours and Holiday with Calendar	
DTR-174		English (en-US)	standard	Agent	[illegible]	
DTR-185		English (en-US)	standard	Acquisition	English - Acquisitior	

and this can answer questions about whether today is a working day (by using the `isWorkingDay` method), or whether this time is in business hours, and so on. Now, you can configure the calling application (that is calling for the rule evaluation) to pass in a timezone ID as a parameter to the relevant business calendar function, and if present, this will override the configured timezone of the business calendar that the rule was created with.

For example, for the `isWorkingDay` method, the addition of the `timezoneID` parameter (highlighted below in the GRST template) enables this override to occur if the parameter is passed in:



isWorkingDay

```
import org.drools.ide.common.client.modeldriven.cal.BusinessCalendar;
import java.util.Date;


function boolean isWorkingDay(BusinessCalendar cC, Date dNow)
{
    if (cC == null) {
        //LOG.info("isWorkingDay: Business Calendar is not set");
        return false;
    }
    return cC.isWorkingDay(dNow, timezoneID);
}
```

The `isWorkingDay` condition is then evaluated based on the dynamic timezone passed in by the calling application.

New Method Signatures to the BusinessCalendar Object

The following new method signatures have been added to the `BusinessCalendar` object, and can be invoked from within a rule function:

- `public boolean isWorkingDay(Date theDate, String timeZoneID);`
- `public boolean isHoliday(Date theDate, String timeZoneID);`
- `public boolean isException(Date theDate, String timeZoneID);`
- `public boolean isWorkingTime(Date theTime, String timeZoneID);`
- `public int diffWorkingDays(Date date1, Date date2, String timeZoneID);`
- `public int diffWorkingHours(Date date1, Date date2,String timeZoneID);`
- `public int diffWorkingMinutes(Date date1, Date date2, String timeZoneID);`
- `public long diffWorkingSeconds(Date time1, Date time2, String timeZoneID);`

- `public Date beginningOfWorkingDay (Date time, String timeZoneID);`
- `public Date endOfWorkingDay (Date time, String timeZoneID);`

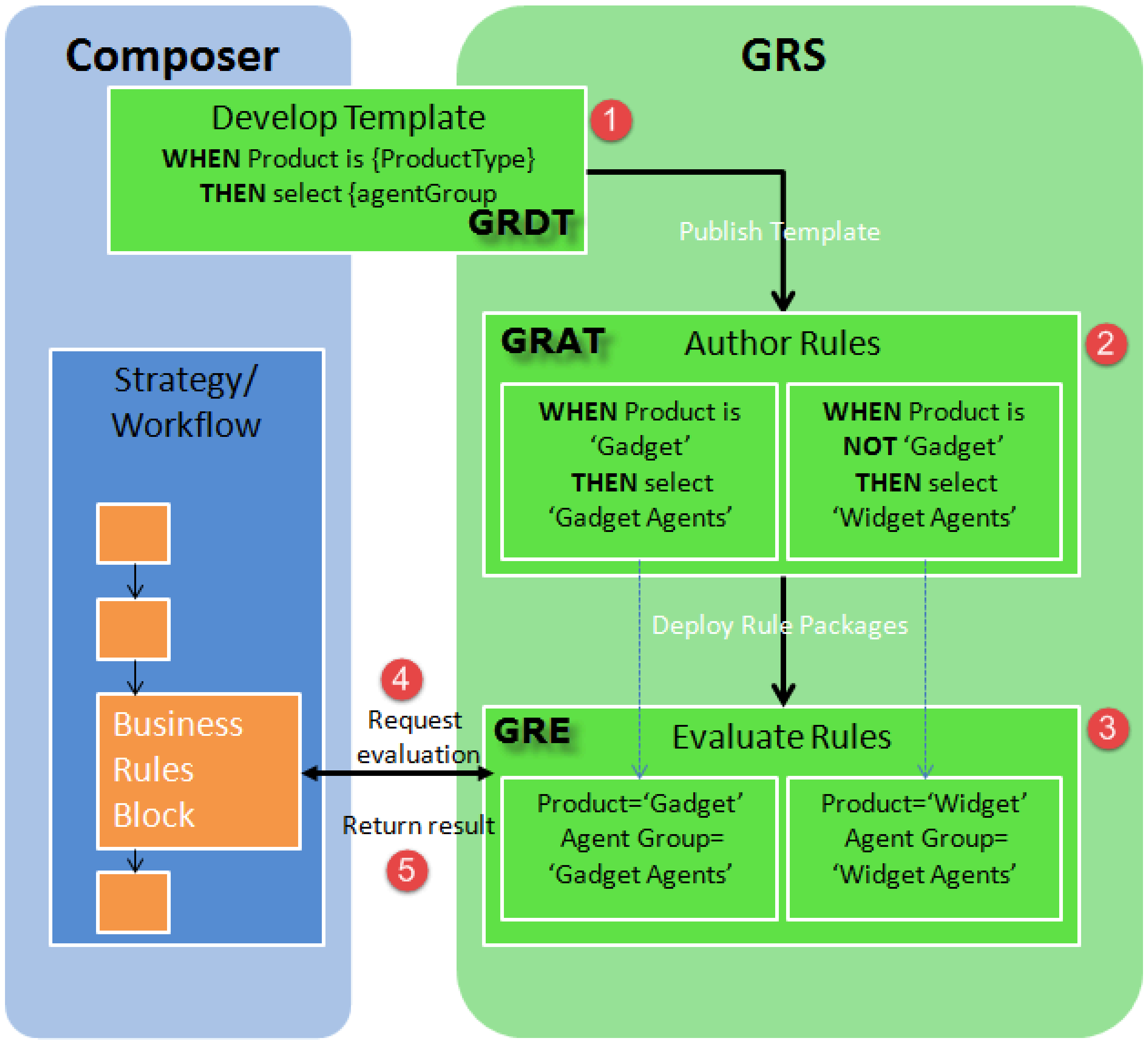
Differentiation between Holidays and Non-Working Days

Business calendars have been enhanced to distinguish between holidays and non-working days. Four new methods have been added to the business calendar object:

- `isHoliday()`—Returns whether this calendar day is a holiday. Holidays are always non-working days.
- `isHoliday(date)`—As for isHoliday() but allows you to specify the date to check.
- `isException()`—Returns whether the day is an exception to the standard work schedule. An exception includes either a time change or a holiday.
- `isException(date)`—As for isException() but for a specified day.

GRS/Composer Process Flow

Basic Relationships and Flow


Composer
GRS
Develop Template
WHEN Product is {ProductType}
THEN select {agentGroup
GRDT
1
Publish Template
GRAT
Author Rules
2
WHEN Product is 'Gadget' THEN select 'Gadget Agents'
WHEN Product is NOT 'Gadget' THEN select 'Widget Agents'
Deploy Rule Packages
Strategy/ Workflow
Business Rules Block
4
Request evaluation
Return result
5
GRE
Evaluate Rules
3
Product='Gadget' Agent Group= 'Gadget Agents'
Product='Widget' Agent Group= 'Widget Agents'

1 Create a template using the GRDT.

2 Create a rule package in GRAT and deploy it to the GRE.

3 The deployed rule package arrives to GRE and awaits evaluation requests from Composer's Business Rules block.

4 Composer's Business Rules block sends evaluation requests to GRE.

5 GRE returns the results in a variable to Composer's Business Rule block.

Variations

It is possible to:

- Create a template from scratch in GRDT and publish to GRAT.
- Import a template project into GRDT, optionally make some modifications, publish to GRAT (for example, the sample from Genesys Proactive Engagement).
- Import a template XML file into GRAT (eg, sample from Conversation Rules or iWD) and use it directly. If you want to modify the template, you can import into GRDT using the **GRS Server Explorer** window, modify, the publish it to GRAT.

  + DETAIL Importing/Exporting Templates

Working with Rule Templates

The topics in this section describes in detail how to work with GRS rule templates.

Rule Template Components

There are a number of components that can be created in a rule template.

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.

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.

Fact Models

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.

Events

In release 8.1.2, 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 was enhanced to 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. 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. Refer to the Genesys Rules Development Tool Help for detailed information about creating parameters in the Genesys Rules Development Tool, including examples of parameters.

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.

Database Parameters

Database 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 the Genesys Rules Development Tool, 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.

Property	Mandatory/optional	Description
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`.

In the Genesys Rules Development Tool, 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.

In the Genesys Rules Development Tool, 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 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” under the Genesys Rules Development Tool, 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.

Rule Language Mapping

When rule developers create the conditions or actions in a rule template, they enter the rule language mapping. Up to and including Genesys Rules System 8.1.2, the 5.1 Drools Rule Language is used. Details of this can be found here:

```
 http://downloads.jboss.com/drools/docs/5.1.1.34858.FINAL/drools-expert/html/ch04.html
```

However, for use in JBOSS environments, you should reference the 5.2 version here:

```
 http://downloads.jboss.com/drools/docs/5.2.FINAL/drools-expert/html/ch05.html
```

For GRS 8.1.3 and higher, use the 5.5 versions, found here:

```
 http://docs.jboss.org/drools/release/5.5.0.Final/drools-expert-docs/html_single/#d0e4033
```

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 building a rule template in GRDT, 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.

DROOLS5 Keywords

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()
```

Creating a Template in GRDT

Rule Templates are created as Projects in GRDT. The **New Project Wizard** is used to create new templates.

New Project Wizard

The **New Project Wizard** guides you through the steps to create a new Rule Template Project. This wizard can be accessed in the following ways:

- Select **File > New > Rule Template Project** from the menu bar.
- Right-click within the Project Explorer and select **New > Project** from the context-sensitive menu.

The wizard leads you through the following steps:

1. The first screen prompts you to select which wizard you need that is, which type of project you wish to create. If it is not already selected, navigate to **Genesys Rules System > Rule Template Project** and click **Next**.
2. Enter a name for the new template project. Template names must be unique within a tenant. Either use the default location, or clear the check-box and select a new location. Click **Next**.
3. Verify the type of template you are creating in the drop-down list. The default template type is `Stateless`. To create an iWD template, select `iWD`. To create a template for Genesys Web Engagement, select type **CEP**.

   Important
   CEP support requires the presence of GWE to function.

   Important
   The iCFD template type has been removed. The iWD template type is the only reserved template type. Any other new template types required can be created by using the **Configure Types** link (see step 4).

4. Click the **Configure Types** link to create new template types or maintain existing ones.
5. Use the **Enable event support** check box to indicate whether the template will support events. In templates that support events, the **Fact Model** editor can be used to create both facts and events.
6. Select the appropriate Tenant, and enter a description of the template.
7. Click **Finish**. The new template project will now appear in the Project Explorer.

Editing and Configuring Rule Templates

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

Renaming Rule Templates

Renaming rule templates does not change the name within the repository. When you publish the renamed rule template, it will be added to the repository as a new template, and the template with the old name will still exist.

You can rename your template by right-clicking the template in the Project Explorer and selecting **Rename** from the context-sensitive menu, by selecting the template and navigating to **File > Rename**, or by selecting the **F2** key on your keyboard. Many features can be accessed in similar ways.

Important

In release 8.1.2, 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**.**

Copying Rule Templates

Existing rule templates can be copied to be used as a basis for a new template. Right-click the template in the Project Explorer, and select **Copy**. As with renaming, there are multiple ways to access the Copy functionality, such as the **Ctrl + C** keyboard shortcut, **Edit > Copy**, and so on.

Deleting Rule Templates

Rule templates can be deleted using the GRS Server Explorer, provided that:

- The user has rule template delete permissions, and;
- The rule is not used in any rule package.

Publish Template

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.

Since release 8.1.3, 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.

To publish the template to the repository:

1. Select the template in the Project Explorer and right-click.
2. Add a version comment.
3. Select **Publish**.
4. Click **Next** to enter a publish comment.
5. Click **Finish** to publish the template.

Imported Templates

If you plan to import a Genesys-supplied template (such as for IWD or GPE or Conversation Rules) directly into GRAT, no Publish step from GRDT is required.

Templates imported into GRDT must still be published to GRAT before they can be used to create rules.

Video Series - Building a Rule Template in GRDT

Name	Description	Link	Duration
Creating a template project	Setting up the Eclipse/ Composer environment, initial configuration and creating a template project	▶	7.59
Building the Fact Model	Building the Facts (data) that will be passed in by the application for evaluation by the rules engine.	▶	5.32
Creating Enumerations	Creating Enumerations (static lists) that the rule author will select from in GRAT to build a rule.	▶	2:13
Creating Parameters	Creating the Parameters that will be used in Conditions and Actions	▶	7:11
Creating Conditions	Creating the Conditions that the rule author will select from in GRAT to build a rule.	▶	5:23
Creating Actions	Creating the Actions that the rule author will select from in GRAT to build a rule.	▶	6:18
Publishing	Publishing the template from GRDT to GRAT to enable the rule author to use it.	▶	2:11

Working with Rules

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 Explorer Tree, four tabs are displayed in the Details Panel:

- 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.

As well as creating a rule package, the GRAT enables you to import and export existing rule packages. This ability enables you, for example, to import a rules package from a test environment to a production environment, or to export a rules package for backup prior to upgrading.

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 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.

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.

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 Explorer 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

Before release 8.5.0, rules in Decision Tables were executed from the bottom up. From release 8.5.0, 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.

Locking of Rules

When you make any modifications to the body of a rule, you "lock" the rule, which prevents others from being able to make changes to the same rule at the same time. The unsaved changes icon will appear on the **Rule Summary** to alert you that you need to save your changes. For any other user, the **Lock** icon appears on the rule summary and the **Save** and **Cancel** buttons are disabled. In addition, other users are unable to make changes to the rule because it is marked "read only".

You can modify multiple rules at a time, without explicitly saving your changes as you move from one rule to the next. The **Rule Summary** will indicate whether there are any unsaved changes that need to be saved. Once the rule is saved, it is "unlocked" and other users will be able to modify it. You can also **Cancel** any unsaved changes, reverting the rule back to the last saved state.

If you log out of your session, you will be prompted if you have unsaved changes. You may then either go back and save your changes, or continue with the logout. In the latter case, the changes you made will be lost and not committed, and the rules will be unlocked.

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.

Quickest Way to Create New Rule

The quickest way to create a new rule is to copy an existing one that you know works correctly.

For Linear Rules, click here.

For Decision Tables, click here.

Creating a Linear Rule

Follow these steps to create a linear rule:

1. Navigate to the rule package to which the new rule will belong in the **Explorer Tree** (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 Explorer Tree and click on 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, the rule is marked with a flag (🚩) to 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.
    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 in GRDT, there may be constraints on the values that can be entered.
11. Click **Validate** to validate the syntax of the linear rule. The **Validate** option appears in the drop-down located in the lower left side of panel.
12. Click **Save** to save your changes.

> **Important**
>
> When you make any modifications to the body of a rule, you "lock" the rule, which prevents others from being able to make changes to the same rule at the same time. The unsaved icon will appear on the rule summary to alert you that you need to save your changes. For any other user, the locked icon appears on the rule

> summary and the Save and Cancel buttons are disabled. In addition, other users are unable to make changes to the rule because it is marked "read only".

When editing rules, be careful not to clear your browsing history or cookie data, as this might cause the rule to be stuck in a locked state. Unsaved changes could be lost.

Linear Rules—Examples

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.

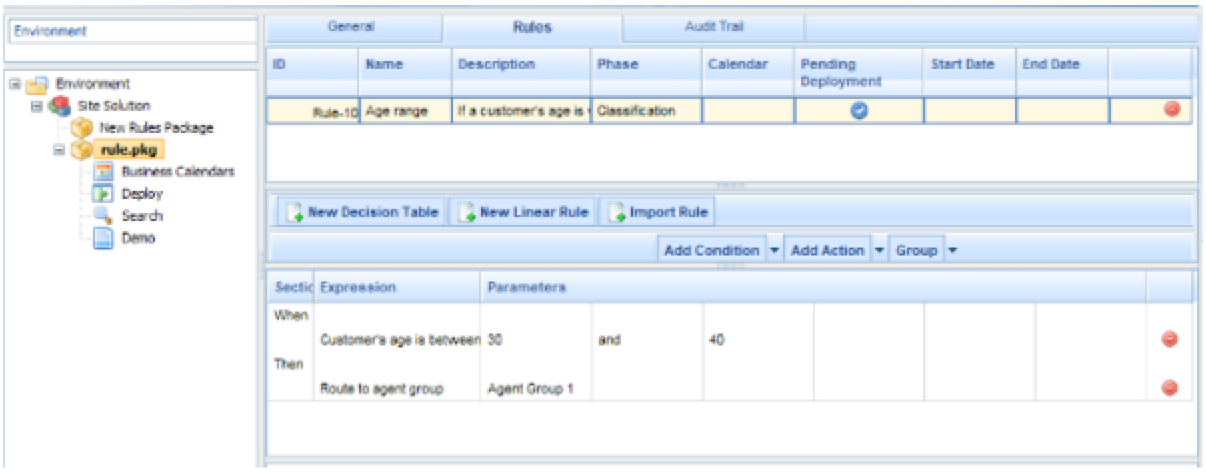
The parameters that are contained in each condition and action are represented by the names that are entered for them. The business rule author must replace this name either by entering a value (such as for an age range) or by selecting an option from the drop-down list (such as for an Agent Group).

Example 1—Route Age Range to Agent Group

WHEN a customer's age is within the range of 30-40 years, THEN the customer's interaction will be routed to Agent Group 1. In GRAT, 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 GRDT. 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.

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. The figure below shows the linear rule in the Genesys Rules Authoring Tool.

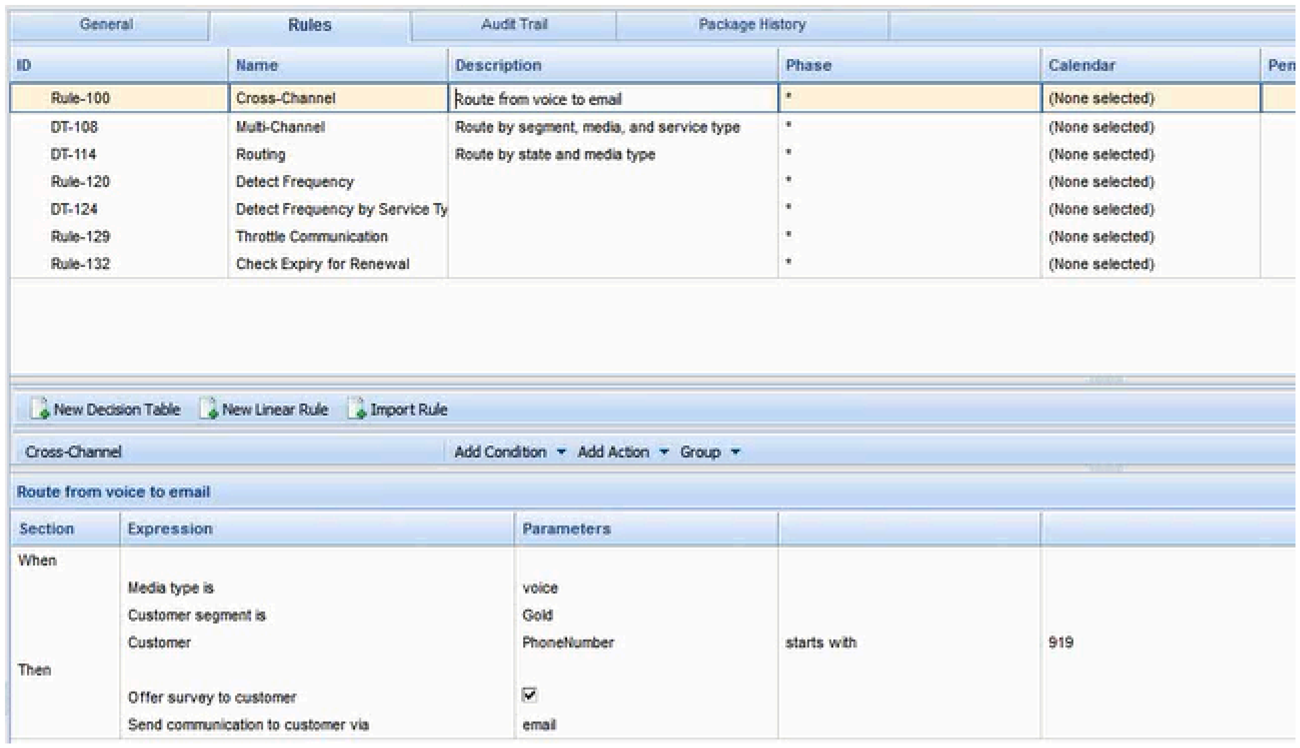


ID	Name	Description	Phase	Calendar	Pending Deployment	Start Date	End Date	
Rule-10	Age range	If a customer's age is	Classification		✓			●

Sectic	Expression	Parameters						
When								
	Customer's age is between	30	and	40				●
Then								
	Route to agent group	Agent Group 1						●

Example 2—Route from Voice to E-mail

This example is typical of a Conversation Manager scenario. WHEN a customer calls in (media type=`voice`) and they are a `Gold` segment customer whose phone number starts with 919, THEN offer a survey to the customer and send it by e-mail.



ID	Name	Description	Phase	Calendar	Pen
Rule-100	Cross-Channel	Route from voice to email	*	(None selected)	
DT-108	Multi-Channel	Route by segment, media, and service type	*	(None selected)	
DT-114	Routing	Route by state and media type	*	(None selected)	
Rule-120	Detect Frequency		*	(None selected)	
DT-124	Detect Frequency by Service Ty		*	(None selected)	
Rule-129	Throttle Communication		*	(None selected)	
Rule-132	Check Expiry for Renewal		*	(None selected)	

Cross-Channel

Route from voice to email

Section	Expression	Parameters		
When				
	Media type is	voice		
	Customer segment is	Gold		
	Customer	PhoneNumber	starts with	919
Then				
	Offer survey to customer	✓		
	Send communication to customer via	email		

Creating a Decision Table

> **Important**
>
> Decision tables can have a maximum of 30 columns.

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 Explorer Tree (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 in the Explorer Tree and click on 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, depending upon 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, the rule is marked with a flag (🚩) 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 in release 8.5.0 you can re-order the rows when creating and editing a decision table.

   > **Important**
   >
   > By default, up to release 8.5.0, rules were executed from the bottom up. In release 8.5.0, your 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
>
> In release 8.5.001, you can now 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 in GRDT, 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.

11. Click **Validate** to validate the syntax of the linear rule.
12. Click **Save** to save your changes.

> ### Important
>
> When you make any modifications to the body of a rule, you "lock" the rule, which prevents others from being able to make changes to the same rule at the same time.
>
> The unsaved icon will appear on the rule summary to alert you that you need to save your changes. For any other user, the locked icon appears on the rule summary and the **Save** and **Cancel** buttons are disabled. In addition, other users are unable to make changes to the rule because it is marked "read only".

> ### Important
>
> When editing rules, be careful not to clear your browsing history or cookie data, as this might cause the rule to be stuck in a locked state. 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.

Decision Tables—Examples

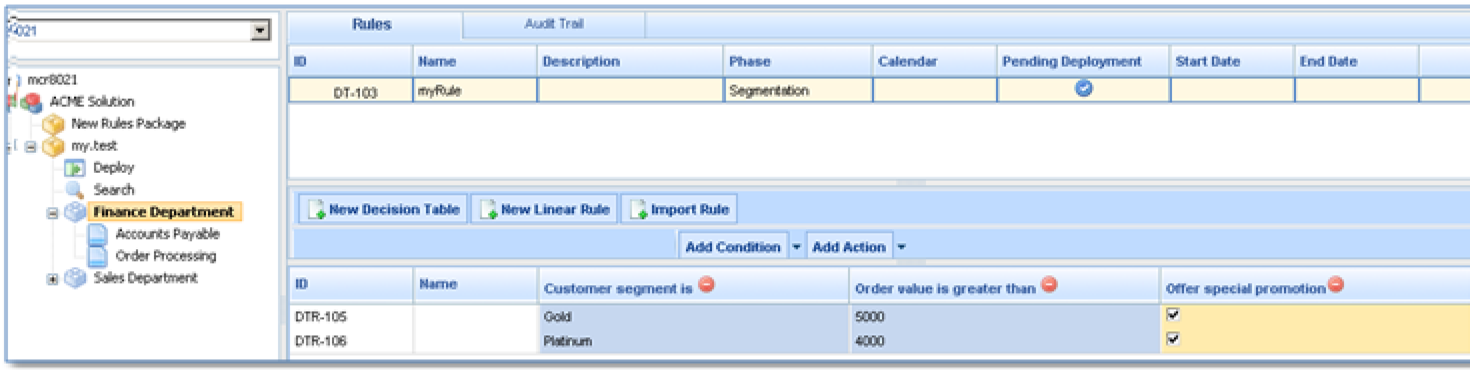
Important

Decision Tables can have a maximum of 30 columns.

Example 1—Pre–8.5.0

Decision tables allow you to create a number of rules that have the same set of conditions (WHEN) and actions (THEN) that are to be used for a complex (structured) business case. Use decision tables to avoid dozens of linear rules that have an identical structure in the system.

Choices in decision tables must 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 may be executed in no guaranteed order. The last row that is executed will determine the final result.



ID	Name	Description	Phase	Calendar	Pending Deployment	Start Date	End Date
DT-103	myRule		Segmentation		✓		

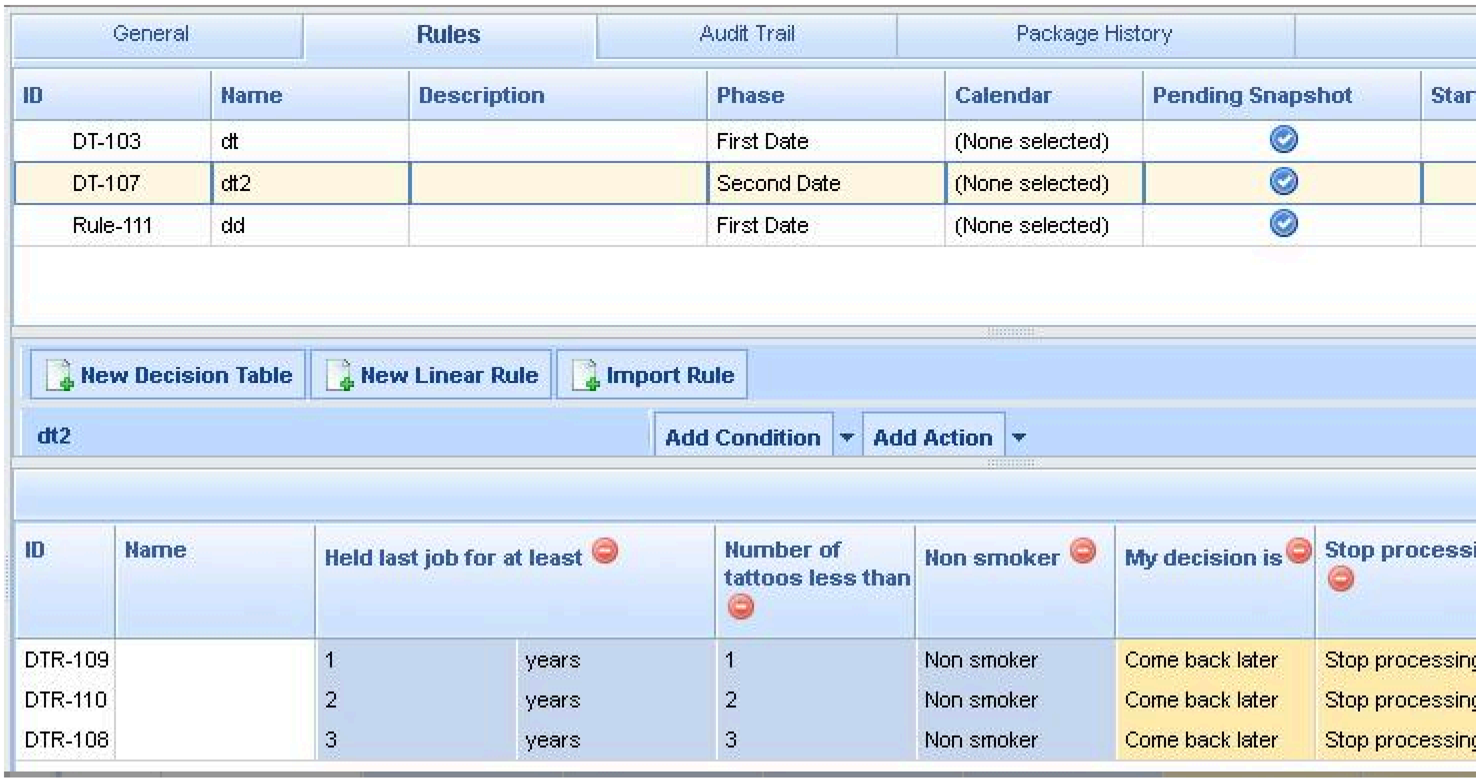
ID	Name	Customer segment is	Order value is greater than	Offer special promotion
DTR-105		Gold	5000	✓
DTR-106		Platinum	4000	✓

Sample Decision

When you are editing rules, be careful not to clear your cookie data, as this might cause the rule to become stuck in a locked state until the session times out (the default is 30 minutes). Consult the documentation for the browser that you are using for more information about how to prevent a user from clearing cookie data.

Example 2—Post–8.5.0 Rule Evaluation Order

By default, up to release 8.5.0, rules were executed from the bottom up (a DROOLS constraint). In release 8.5.0, 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. A new configuration option (evaluate-decision-table-rows-top-down) allows the administrator to set the order of evaluation (top to bottom or bottom to top).



ID	Name	Description	Phase	Calendar	Pending Snapshot	Star
DT-103	dt		First Date	(None selected)	✓	
DT-107	dt2		Second Date	(None selected)	✓	
Rule-111	dd		First Date	(None selected)	✓	

ID	Name	Held last job for at least		Number of tattoos less than	Non smoker	My decision is	Stop processi
DTR-109		1	years	1	Non smoker	Come back later	Stop processing
DTR-110		2	years	2	Non smoker	Come back later	Stop processing
DTR-108		3	years	3	Non smoker	Come back later	Stop processing

Decision Table with Rule Evaluation Order

Configuration Info

The `evaluate-decision-table-rows-top-down` configuration option controls this behavior. This determines the order that the Decision Table rows are written out to the DRL. If you changes this default option, you will see a change in behavior immediately when using GRAT's Test Scenario feature, but will need to re-deploy the rule package in order for the change to be observed in GRE.

- Section: `settings`
- Option Name: `evaluate-decision-table-rows-top-down`
- Values: `true, false`
- Default: `false` (to maintain backwards compatibility)

Example 3—Wildcards

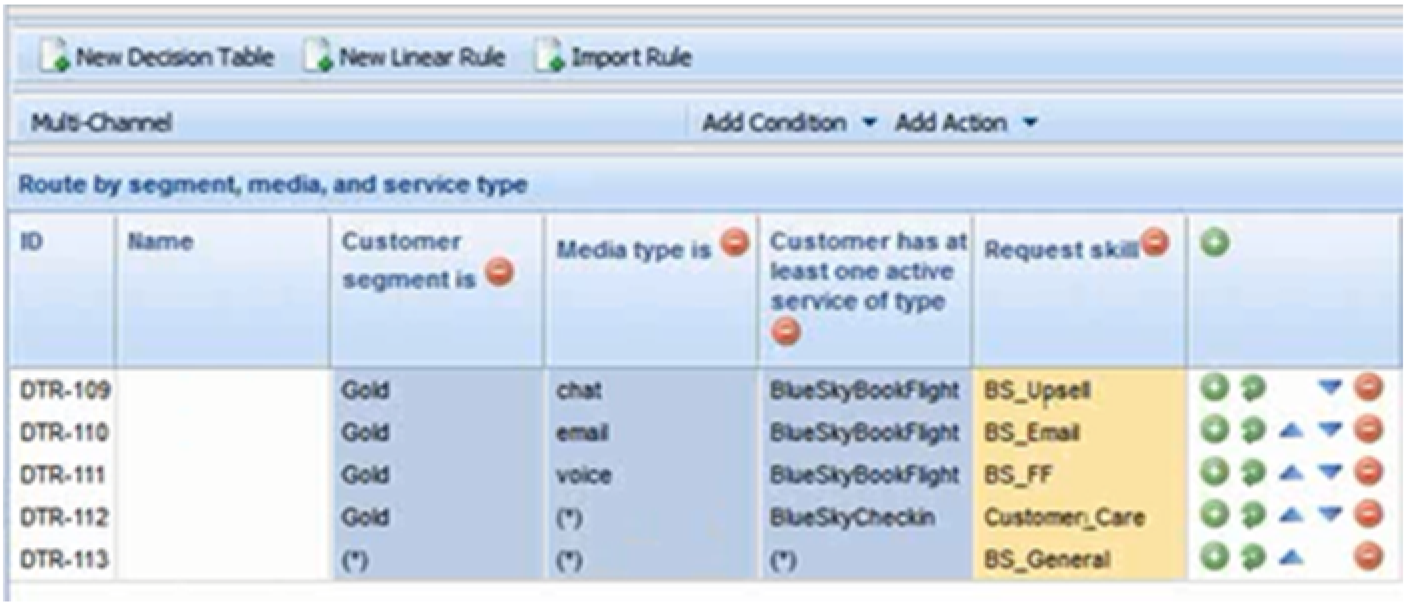
In release 8.5.001.00, you can now use a wildcard symbol (*) in row data in a decision table (if the feature is configured by administrators). The wild card indicates that, for this row, the value for the parameter where it is used is unimportant and not to be evaluated. A wildcard selection now appears 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.



ID	Name	Customer segment is	Media type is	Customer has at least one active service of type	Request skill
DTR-109		Gold	chat	BlueSkyBookFlight	BS_Upsell
DTR-110		Gold	email	BlueSkyBookFlight	BS_Email
DTR-111		Gold	voice	BlueSkyBookFlight	BS_FF
DTR-112		Gold	(*)	BlueSkyCheckin	Customer_Care
DTR-113		(*)	(*)	(*)	BS_General

Decision Table with Wildcards

Deploying a Rules Package

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 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.

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 Explorer Tree, 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 application clusters configured in Genesys Administrator, 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.

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 succeeded or failed. This gives useful information on how a deployment has progressed: you can 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.

Important

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.

To show the report:

1. Click on the **Failed/Successful** 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

Important

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

Video Series - Building a Rule Package and Rule in GRAT

Name	Description	Link	Duration
Getting Started	Creating a rule package.	▶	5.32
Creating a Linear Rule	How to create a simple linear rule.	▶	6.43
Creating a Decision Tale Rule	How to create a Decision Table rule.	▶	4.35
Creating Test Scenarios	Creating scenarios for testing rules.	▶	8.31
Deploying the Rule Package	How to deploy the rule package to a rules engine that will execute the rule(s).	▶	3.10

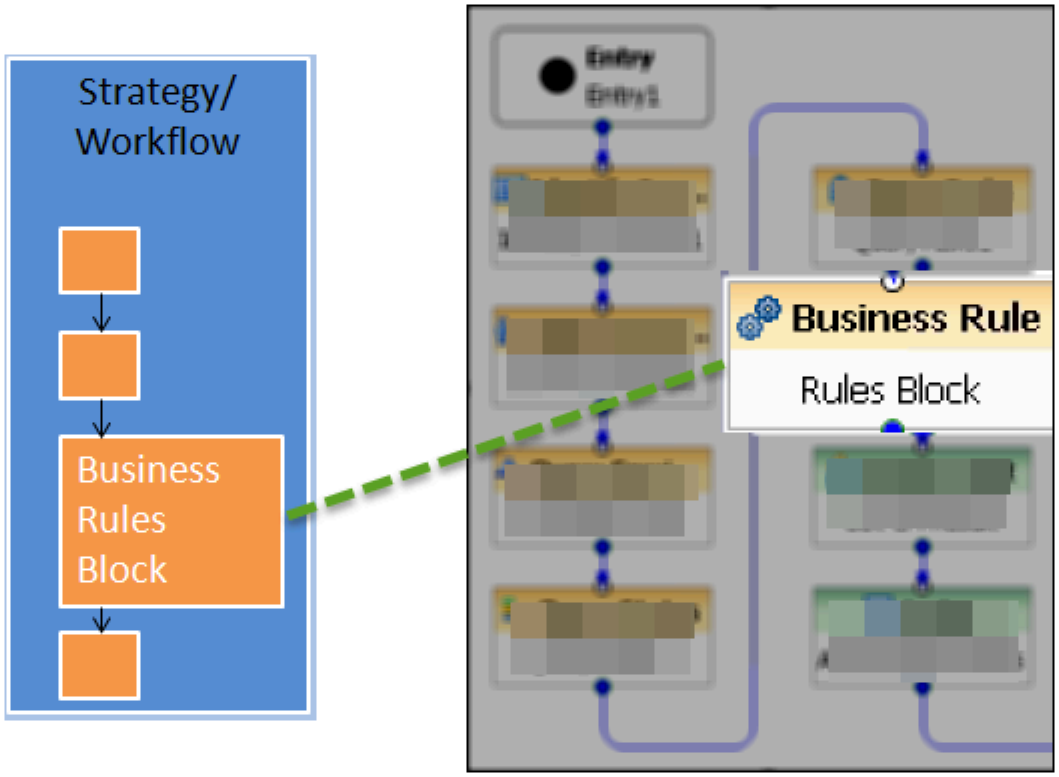
Working with Composer's Business Rules Block

Working with Composer Workflows

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

Important

In the process flow diagram at the beginning of this document, we used a schematic to indicate how the Composer Business Rule block fits into a workflow. In the graphic below you can see that we are not concerned with the details of the workflow application, only in how a rule that a rule author has created is consumed by the Composer/ORS application that the workflow represents.



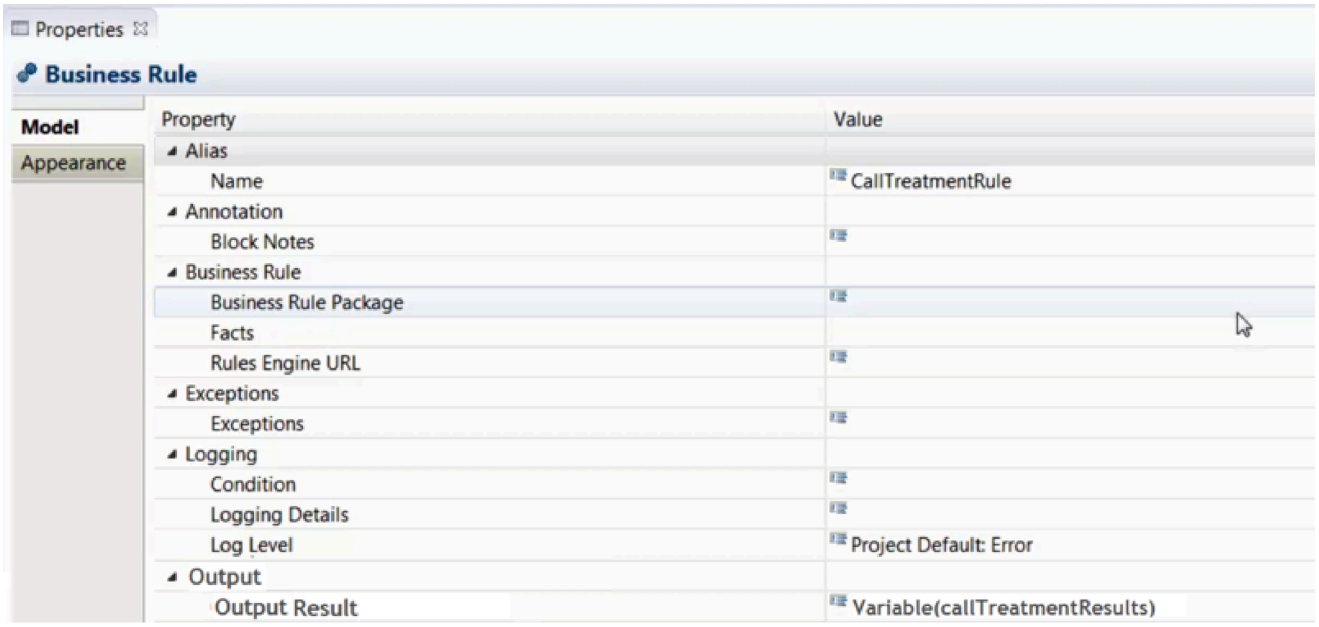
To edit the Business Rule block in Composer, double-click it.

Important

Because you will specify a rules package and a rule engine in this block, you must have already deployed a rules package to the relevant rules engine and configured a connection to the rule engine before the Business Rule block can be completed.

Business Rule Block Parameters and Properties

When you open the Business Rule block, it looks something like this:



Property	Value
Alias	
Name	CallTreatmentRule
Annotation	
Block Notes	
Business Rule	
Business Rule Package	
Facts	
Rules Engine URL	
Exceptions	
Exceptions	
Logging	
Condition	
Logging Details	
Log Level	Project Default: Error
Output	
Output Result	Variable(callTreatmentResults)

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

Important

This last step (evaluation) happens as part of a strategy or workflow application that Composer developer creates.

The Business Rule block has, among others, the following properties:

Business Rule

Business Rule Package Property

Use to select the Rule Package (collection of related rules) you would like to execute. The rule package must have already been deployed to the GRE. You will only be able to select deployed rule packages.

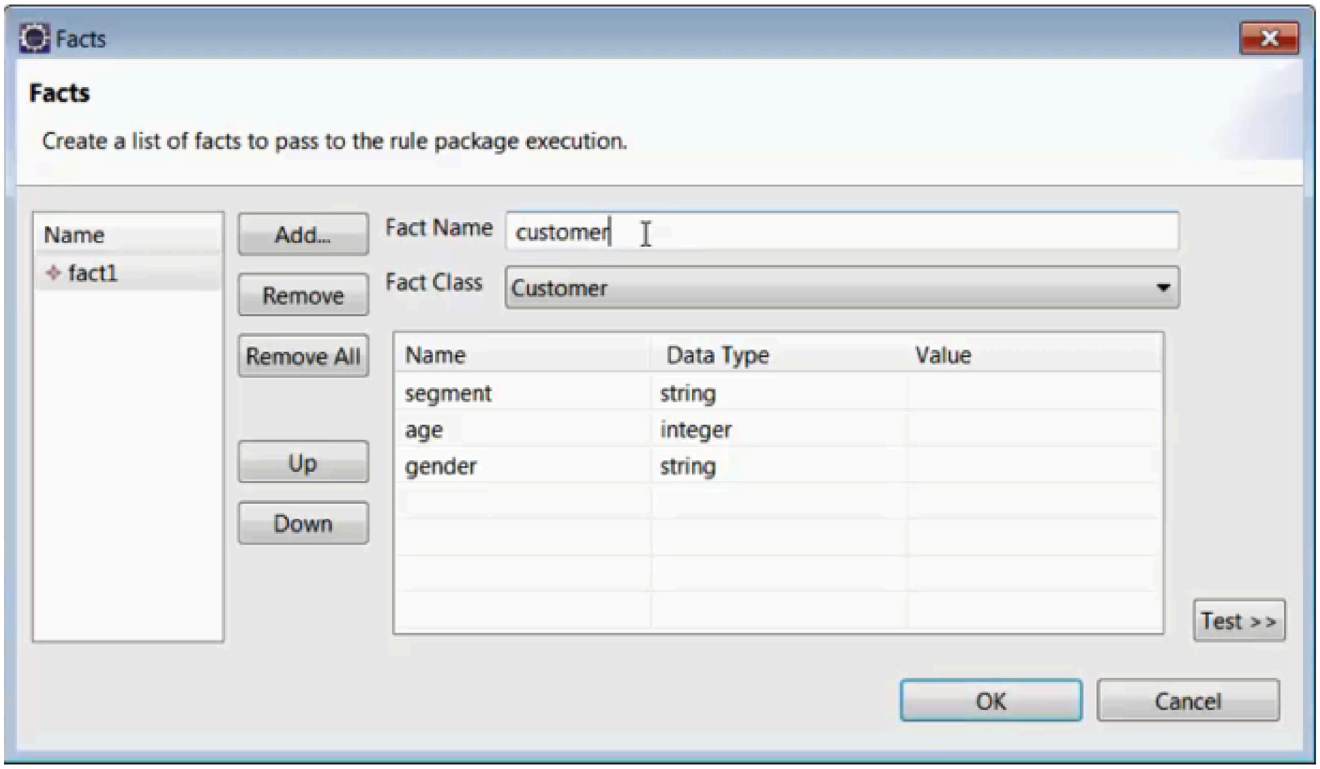
Packaging rules together allows the business analyst to define which rules will support a particular application. Before using this property, you must request Composer to connect to the Genesys Rules Authoring Tool Server using the information specified in Business Rule Preferences. After a successful connection, the Business Rule Package dialog appears.

Select a Rule Package and click **OK**. The dialog closes and the name of the Rule Package appears under Value.

Facts Property

Once you have selected the rule package, the Fact model associated with the template that the rule package is based on will automatically populate as selections in a drop-down menu in the Fact Class field.

Example:



Create a list of facts to pass to the rule package execution.

Name
fact1

Fact Name: customer
Fact Class: Customer

Name	Data Type	Value
segment	string	
age	integer	
gender	string	

Select a Fact class and create a new Fact based on it. This Fact will display all the fields that were defined in the template. At this point you can assign values to those Fact fields.

These values can be literals or variables that the workflow application is using. The available values are displayed in the Values drop-down menu.

You must also define here the output Fact (Output Result Property below)—the Fact that will contain the decision made by the rules engine and is passed back to the rule block in the workflow application to be processed by the workflow logic.

Rules Engine URL

Select the variable containing the Genesys Rules Engine URL.

Output Result Property

Use this property to save the results of the business rule execution to a variable.

The format of returned data is JSON. Any post-processing work to be done on returned results can be done in the existing Assign Block which provides access to ECMAScript functions. It supports writing simple or complex expressions to extract values out of JSON strings and arrays. In a workflow, the Output Result can be attached to User Data.

Important

The **Output Result** property takes effect only during application runtime. Its purpose is to take the output of the rule execution (at runtime) and store returned results back in the specified application variable so other parts of the application can access the data.

Working With Returned Data

Below is an example (using output variable `callTreatmentResults`) on how to work with data returned by the Business Rules block. A sample of the output can look like the snippet below, which will be stored in `callTreatmentResults`.

```
callTreatmentResults='({
        'knowledgebase-response':{
                "InOutFacts":{'named-fact':[
                   {
                                "id":"customer",
                                "fact":{
                                   "@class':"call.treatment.Customer",
                                   "segment":"gold"
                                }
                   },
                   {
                                "id":"callinfo",
                                "fact":{
                                   "@class':"call.treatment.CallInfo",
                                   "intention":"Address Change"
                                }
                   },
                   {
                                "id":"callTreatment",
                                "fact":{
                                   "routeToAgentGroup":"Accounts"
                                   "@class':"call.treatment.CallTreatment",
                                   "maxWaitTime":"10"
                                   "offerCallBack":"false"
                                }
                    }
                 ]}
                }}
)
```

This data structure contains the evaluation of the rule(s) as determined by the rules engine. These results are passed back to the Composer application for decision-making and onward processing.

Video Series - Using Composer to execute a rule block

Name	Description	Link	Duration
Getting Started	First steps in creating a Composer workflow that will use the rule.	▶	3.26
Creating a New Composer Project	Setting up the Composer workflow project.	▶	10.16
Testing the Rule Block	Testing the new rule block	▶	2:21

Genesys Rules System 101—Video Topics

The following video topics show how to create a simple template from scratch in GRDT, publish it to GRAT, create a rule package and finally connect with the Composer Business Rules block to use the evaluation decision in a workflow. The use case is a simple call treatment flow which uses the customer's segment and call intention to make business decisions about how to route the contact.

You can follow the videos in sequence as they appear on these tabs.

<tabber>

Part 1 - Building a template in GRDT=

Building a template in GRDT

Name	Description	Link	Duration
Creating a template project	Setting up the Eclipse/ Composer environment, initial configuration and creating a template project	▶	7.59
Building the Fact Model	Building the Facts (data) that will be passed in by the application for evaluation by the rules engine.	▶	5.32
Creating Enumerations	Creating Enumerations (static lists) that the rule author will select from in GRAT to build a rule.	▶	2:13
Creating Parameters	Creating the Parameters that will be used in Conditions and Actions	▶	7:11
Creating Conditions	Creating the Conditions that the rule author will select from in GRAT to build a rule.	▶	5:23

Name	Description	Link	Duration
Creating Actions	Creating the Actions that the rule author will select from in GRAT to build a rule.	▶	6:18
Publishing	Publishing the template from GRDT to GRAT to enable the rule author to use it.	▶	2:11

|-| Part 2 - Building a Rule Package and Rule in GRAT =

Building a Rule Package and Rule

Name	Description	Link	Duration
Getting Started	Creating a rule package.	▶	5.32
Creating a Linear Rule	How to create a simple linear rule.	▶	6.43
Creating a Decision Table Rule	How to create a Decision Table rule.	▶	4.35
Creating Test Scenarios	Creating scenarios for testing rules.	▶	8.31
Deploying the Rule Package	How to deploy the rule package to a rules engine that will execute the rule(s).	▶	3.10

|-| Part 3 - Using Composer to execute a rule block=

Using Composer to execute a rule block

Name	Description	Link	Duration
Getting Started	First steps in creating a Composer workflow that will use the rule.	▶	3.26
Creating a New Composer Project	Setting up the Composer workflow project.	▶	10.16
Testing the Rule Block	Testing the new rule block	▶	2:21

Examples of Rule Template Development

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

Example 1—Condition & Action

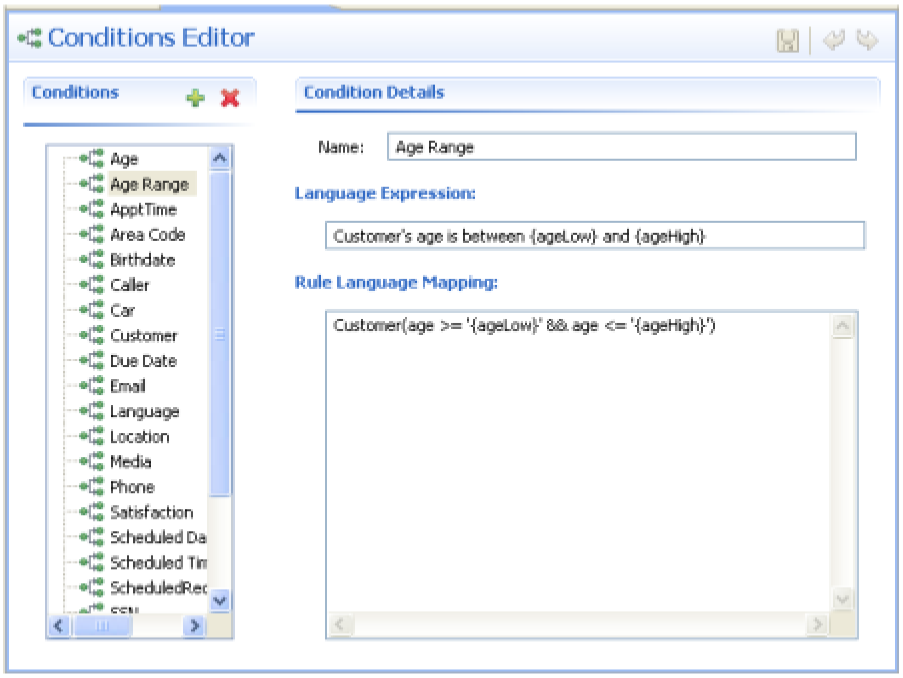
Age Range Condition

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 the Genesys Rules Development Tool.



Name: Age Range

Customer's age is between {ageLow} and {ageHigh}

Customer(age >= '{ageLow}' && age <= '{ageHigh}')

Age Range Condition

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 could instead 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.

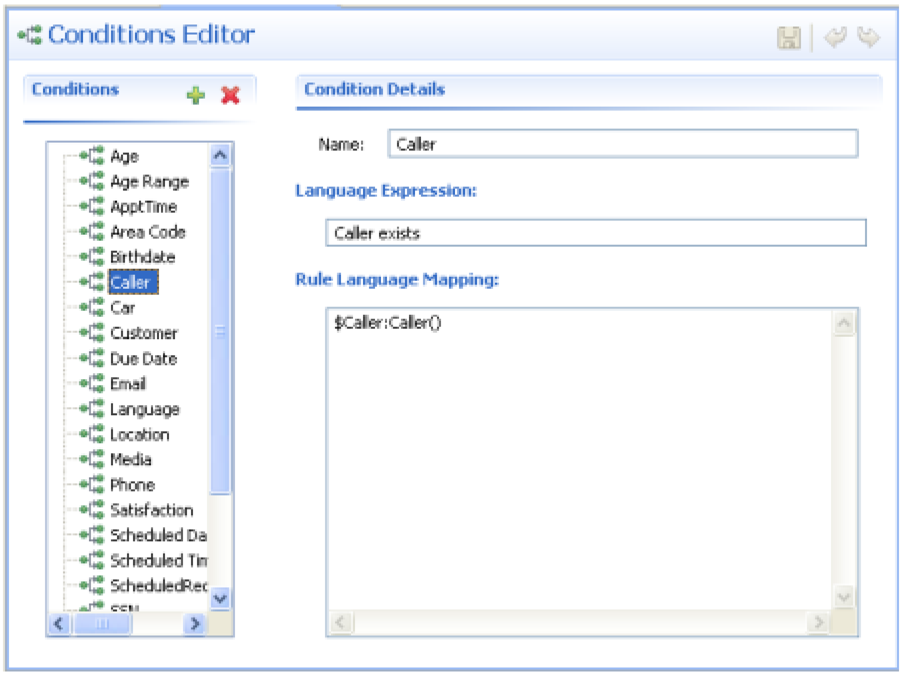
Caller Condition

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 the Genesys Rules Development Tool.



Name: Caller

Language Expression:

Caller exists

Rule Language Mapping:

```
$Caller:Caller()
```

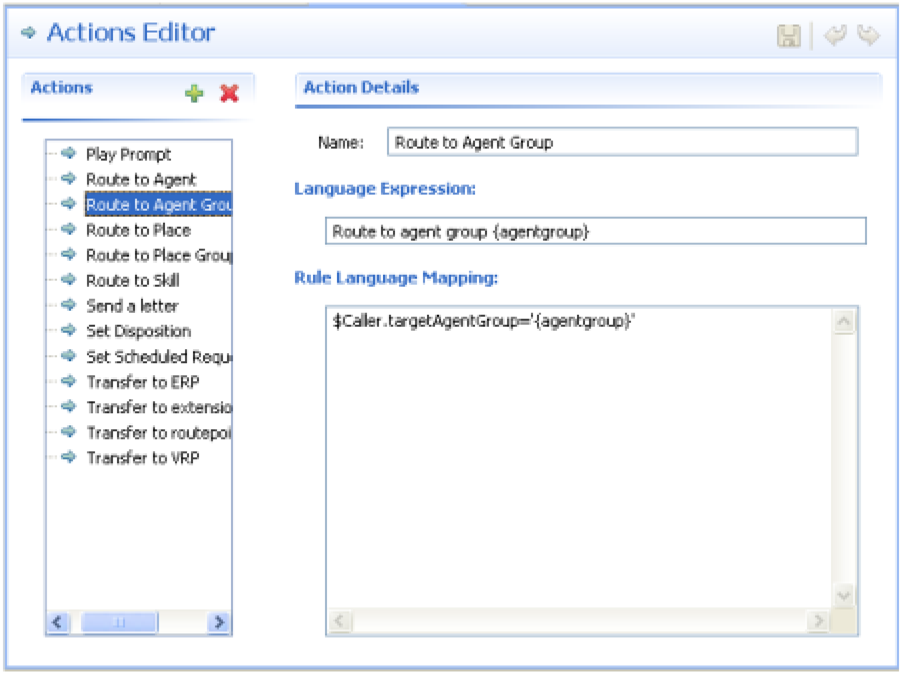
Caller Condition

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 the Genesys Rules Development Tool.



Name: Route to Agent Group

Language Expression: Route to agent group {agentgroup}

Rule Language Mapping:

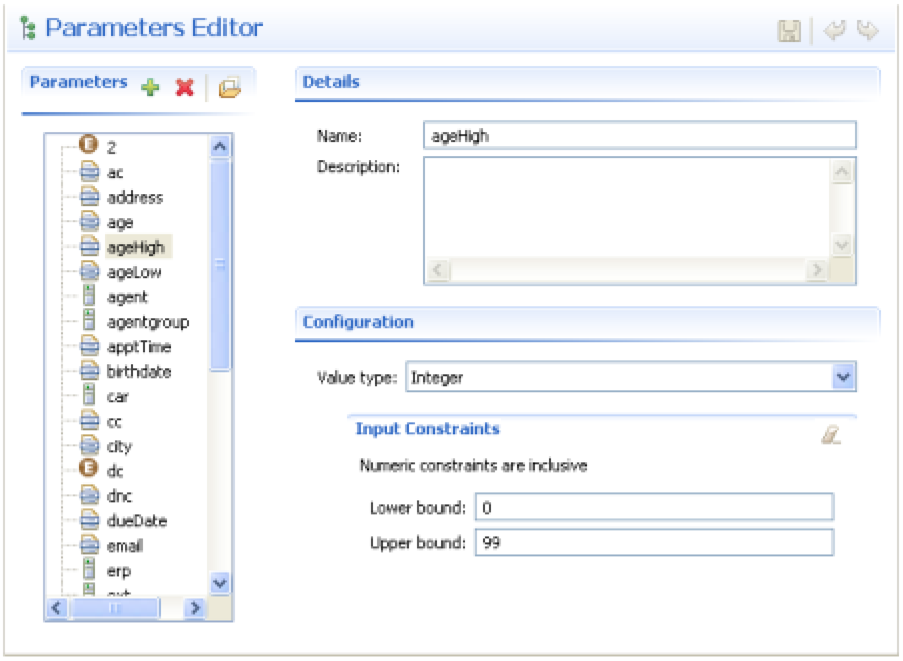
```
$Caller.targetAgentGroup='{agentgroup}'
```

Target Agent Group

The condition in this example has two parameters:

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

The action has the `"{agentGroup}"` parameter. Parameters are also configured in the Genesys Rules Development Tool. The Parameters Editor screenshot shows a sample `"{ageHigh}"` parameter. Refer to the Genesys Rules Development Tool Help for more details about how to configure parameters.



Name: ageHigh
Description:
Value type: Integer
Numeric constraints are inclusive
Lower bound: 0
Upper bound: 99

Parameters Editor Screen

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 Lower Bound and/or an Upper Bound.
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 or cluster.

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:

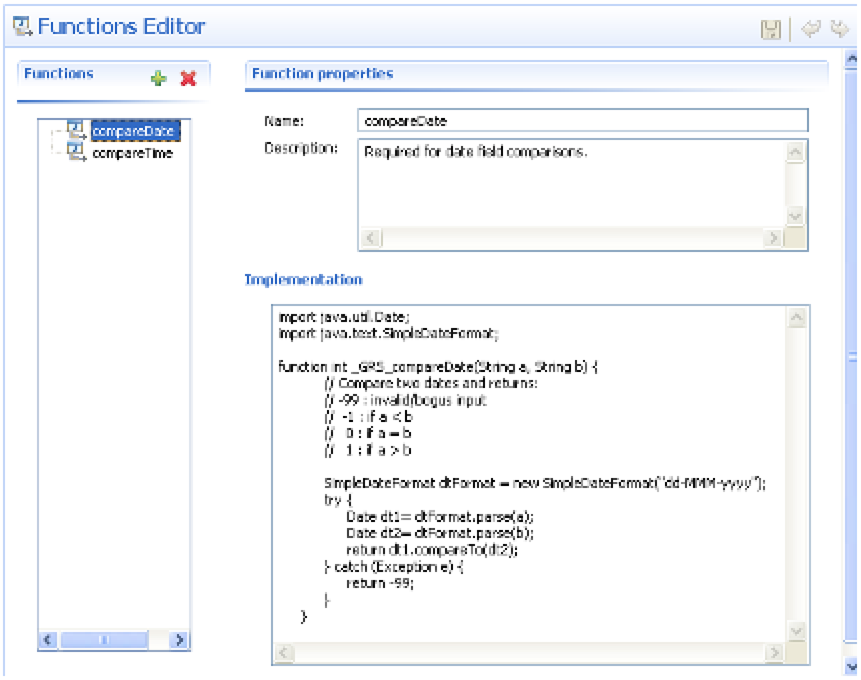
```
Name: compareDates
Description: This function is required to compare dates.
Implementation:
import java.util.Date;
import java.text.SimpleDateFormat;

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 the Genesys Rules Development Tool.



Name: compareDate

Description: Required for date field comparisons.

```
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;
        }
    }
```

compareDate Function

Example 3—Using a JSON Object

Since release 8.1.3, 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

Solution-Specific Templates

Click on these links for information about the standard templates supplied out-of-box with the relevant products:

- **Conversation Rules Template Guide**
- **Genesys Proactive Engagement**
- **intelligent Workload Distribution**