GRDT Help

Genesys Rules System 8.1.3

12/30/2021

Table of Contents

Genesys Rules Development Tool Help

The Genesys Rules Development Tool (GRDT) is an Eclipse plug-in that allows 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.

Working with Templates

Working with Editors

Rule Template Overview

Rule templates are created in GRDT, which is an Eclipse plug-in that can either be installed into a standalone Eclipse application or can be installed into Genesys Composer. Rule templates are used to define the building blocks that 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.

Rule templates are made up of several elements:

- Actions—Contain `then` expressions.
- Conditions—Contain `when` or `if` expressions.
- Enumerations—Define lists of possible choices that will be displayed to the business rule author.
- Fact models—All rule templates include a fact model with one or more facts.
- Events—From release 8.1.2, to support Complex Event Processing, template developers need to be able to designate certain facts as events.
- Functions—Sometimes used to support rule conditions and actions—for example, when parsing of timestamps is required.
- Parameters—Database, Web Service and Workforce Management parameters are used in the actions and conditions.

Releases up to and including 8.1.2

Rule templates are developed in the Genesys Rules Development Tool (GRDT). In releases up to and including 8.1.2, each time a rule template is published, a new version is created in the repository. The rule author will be able to select the latest version of the template when creating a rule package. Once a rule package is created, it will always use the same version of the rule template, even if newer versions are published. The rule author can choose to upgrade to a newer version of the rule template at any time, but this will not happen automatically.

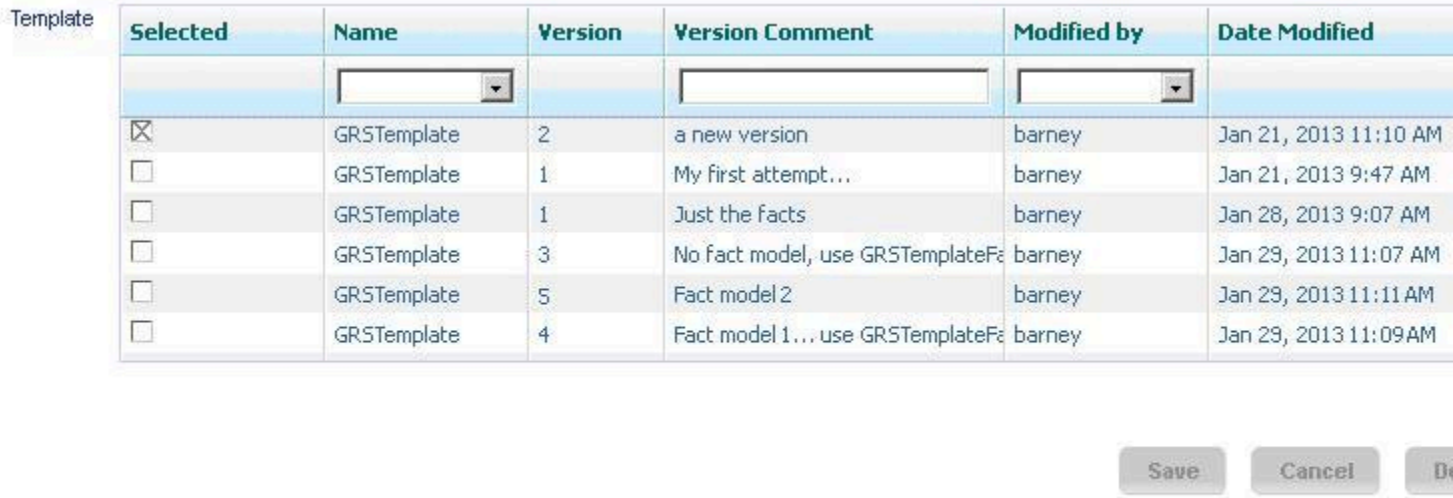
The rule developer should communicate to the rule author if a new version of the Rule Template is available and if they are advised to upgrade.

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.

Release 8.1.3

In release 8.1.3, multiple versions of templates can be created and stored for users to choose from in the Template Selection dialog. 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.

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.



Template Selection

Configuration Option

`display-n-template-versions`

Valid Values: Integer >=1

Default Value: 3

Description: Integer value specifying the maximum number of versions to display for any published template.

See also **Deploying GRAT in Genesys Administrator** for information about this configuration option.

Version Comment

In order to provide details about the differences between template versions, rules template developers in GRDT can now 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

To import a template project:

1. Right-click anywhere in the Project Explorer and select **Import**, or select **New > Import** from the Eclipse/Composer menu.
2. On the **Select** screen of the Import wizard, in the **General** folder, select **Existing Projects Into Workspace**.
3. Click **Next**.
4. On the **Import Projects** screen, select the **Select** root directory , click **Browse** to browse to your file system, and select the template project to import.
5. Click **Finish**.

To export a template project:

1. Right-click anywhere in the Project Explorer and select **Export**, or select **New > Export** from the Eclipse/Composer menu.
2. On the **Select** screen of the **Export** wizard, in the **General** folder, select **File System**.
3. Click **Next**.
4. On the **File System** screen, select one or more template projects to export, or one or more components with the projects you wish to export.
5. Browse to the location on the file system to which you want to export.
6. Click **Finish**.

> ### 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 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.

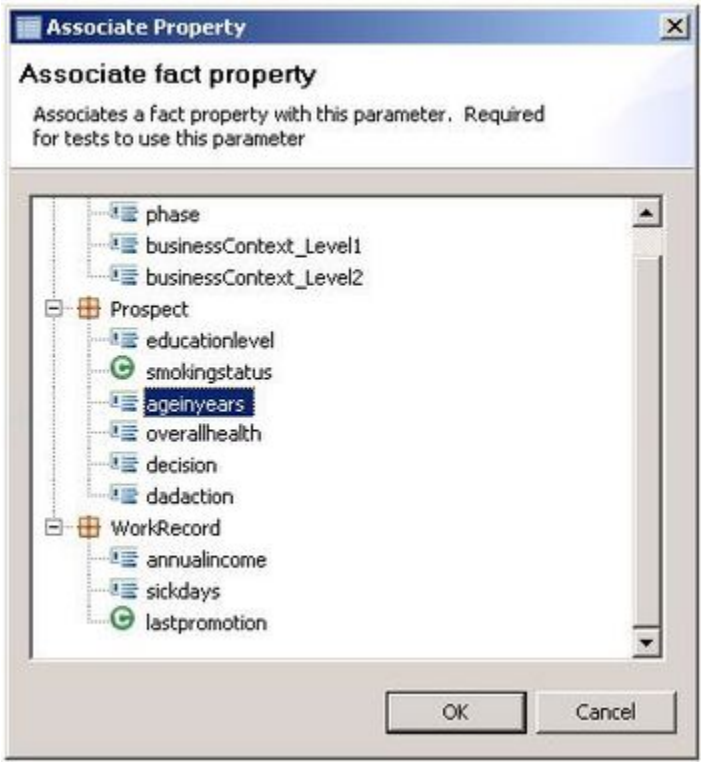
Developing Templates to Enable Test Scenarios

Mapping Parameters to a Fact Model

Rules authors build rule-test scenarios using parameters, in the same way as they define rules. However, when the tests execute, GRAT maps the parameter values to the underlying fact model developed by the template developer, who understands the relationship between the parameters and the fact model. So, for rule testing to operate correctly, template developers must map parameters back to the underlying fact model.

For example, the {age} parameter may be related to the `ageinyears` field of the `Prospect` fact. So, if age is set to 25, then, when executing the test, GRAT needs to allocate a `Prospect` fact and set the `ageinyears` field to 25. The same is true for the expected results.

The **Associate Property** dialog enables this mapping.



In general, there should be a one-to-one mapping between a parameter and a fact. However, this may be too restrictive for all implementations. GRDT lets you map a single parameter to multiple fact values. For example, the {age} parameter could be defined once, but reused to represent both a

customer’s age and the age of an order.

So, {age} could map to both:

- Customer.ageinyears
- Order.ageoforder

Where this occurs, GRAT displays the parameter in the **Add Given** or **Add Expect** drop-down list in parentheses, so that the GRAT user can select the correct mapping

Example

In the following example, only {age} has this special designation because of the ambiguity in the definition.

Add Given…

- {age} (Customer.ageinyears)
- {age} (Order.ageoforder)
- {gender}
- {education}

To hide this ambiguity from the rule author, you should declare a different parameter for each usage: for example, {customerAge} and {orderAge}.

Using ESP-type Templates

There are some special considerations in developing templates for ESP-type templates, for products such as iWD.

With ESP templates, instead of building a set of facts and passing them to be executed, you create a KeyValueCollection (KVC) and populate it with key/value pairs of test data. In order to enable this mapping between a parameters and the correct key to use in the KVC collection, you need to create a dummy fact model in GRDT to represent the keys that will be inserted into the KVC.

For example, with iWD, this means modifying the iWD Standard Rules Table to insert a fact and a field for each key that is used in the template.

Fact Name

You will need to define a reserved fact name (for example, _GRS_ESP_Fact) that is processed differently. For this fact, the fields are mapped to a KVC instead of the traditional Fact model. The fact name must be used because there is no type associated with a fact. Types are only associated with individual fields.

Field Name

You must develop a convention (such as prefixing all field names with `grs_`) . This is because fact fields must start with a lower-case letter (a DROOLS restriction) and GRDT enforces this convention, but iWD key names all begin with IWD. Since the existing iWD key names are like “IWD_businessValue”, you will need to adopt some naming convention. If the `grs_` prefix is present, GRAT will remove it and insert the remaining value into the KVC as the key (for example, `grs_IWD_channel` is inserted into KVC as `IWD_channel`)

The rule template developer must then map each parameter name (used in conditions/actions) to the appropriate field within `_GRS_ESP_Fact`.

The rule author can then use GRAT to create a rule and a test scenario for that rule.

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.

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.

Reserving Templates

Reserving rule templates ensures that no other template developers can modify the template you are working on.

To reserve the template:

1. Right-click the template in the Project Explorer
2. Select **Reservations**. If you have the template reserved already, you will be prompted to release it. If you do not have the template reserved, you will be prompted to reserve it. Once the reservation is made, it will be maintained even when working offline.

Important

You cannot reserve templates that do not yet exist on the server. Reservation is used to ensure that other users do not update the template while you are working on it, so there is no need to reserve a template before you have published it.

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. This is what appears in the Project Explorer to identify 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.

> **Important**
> Always enclose language expressions in double quotes.

- `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.

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. This is what will appear in the Project Explorer to identify the condition.
- `Language Expression`—The plain language expression of the condition that the rule author sees when constructing a business rule in the Rules Authoring Tool.

> **Important**
> Always enclose language expressions in double quotes.

- `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.

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 in GRAT, 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 and Events 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 and event.

Facts

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

- `Name`—A name for the fact.
- `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.

Events

> **Important**
>
> Event support is implemented in anticipation of the Genesys Web Engagement (GWE) product. Event support functionality requires the presence of GWE in order to function.

Event support is available for templates of type CEP where event support is declared when the template is created. Event support is not available for iWD-type templates.

To create a new event, click in the `Facts/Events` pane of the Fact Model Editor.

Each event that is created in the editor contains the following fields:

- Name—A name for the event.
- Description—A brief description of the event
- Sensitive—Determines whether the event is logged when the Rules Engine evaluates events. If this field is selected, the event is not logged. If this field is cleared, the event 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.
  - Metadata—Determines the expiry date of the event

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 the GRAT and the GRE.

Parameters Editor

The Parameters Editor allows you to create rule parameters, which are optionally used in rule conditions and actions. You can also map a parameter to a fact model by using the button. See Mapping Parameters to a Fact Model.

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

`Name`: A name for the parameter.

`Description`: A brief description of the parameter.

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:

- Configuration Server
- Input Value
- Database
- Operational

- Select Enumeration
- Web Service
- Workforce Management
- Calendar

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—Business Attribute, Business Context, List Object, or List— as described below:

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.

Selecting `List Object` prompts you to select the List and Section values for the List Object from a user-defined set of List objects.

Selecting `List` prompts you to select from a predefined set of lists as shown below:

- Agent Groups
- Agent Skills
- Agents
- Extensions
- External Route Points
- Interaction Queue
- Media Types
- Place Groups
- Places
- Route Points
- Switch
- T-Server
- Virtual Route Points

Input Value

Input Value parameters are simply parameters for which the rules author can provide a value based on the defined parameter type (Boolean, Integer, Numeric, String, Date, or Time). 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.

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

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`.

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 GRDT, 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.

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 GRDT 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 GRDT, 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” 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.

Calendar

Calendar parameters indicate to the 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.

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://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 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.