Genesys Rules Authoring Tool Help

Ejemplos de desarrollo de plantillas.

6/27/2025

Sumario

Ejemplos de desarrollo de plantillas.

Esta sección proporciona algunos ejemplos de lo que un desarrollador de reglas podría configurar en la ficha Desarrollo de plantillas. Para obtener información específica sobre cómo se configuran las plantillas de reglas para usarse con la solución de distribución inteligente de cargas de trabajo (iWD) de Genesys, consulte la guía *iWD y Genesys Rules System*.

Ejemplo 1: Condición y acción

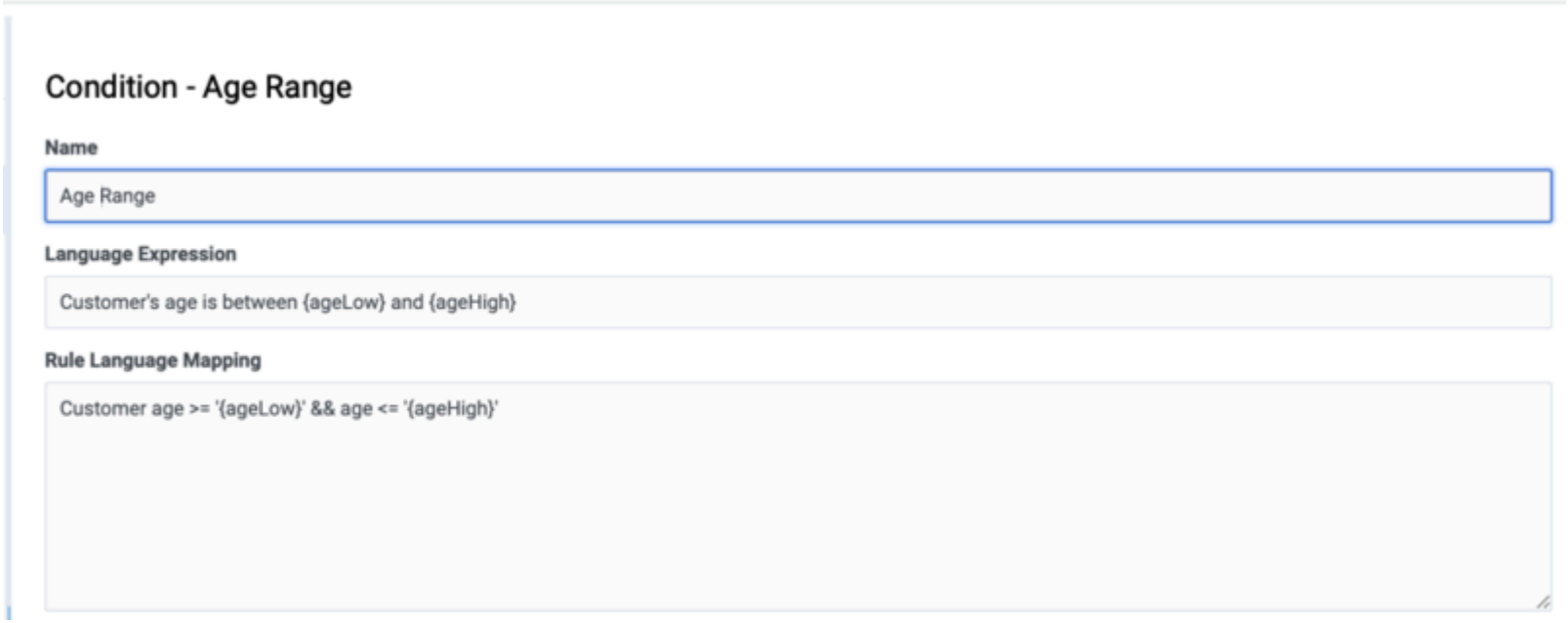
Condición de rango de edad

Si la edad de un cliente está dentro de un rango específico, se asignará a un grupo de agentes específico. En este escenario, la condición es si la edad del cliente se encuentra dentro del rango. En la herramienta de desarrollo de reglas de Genesys, las condiciones se configurarían de la siguiente manera:

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

No use la palabra 'fin' en expresiones de lenguaje de reglas. Esto provoca errores de análisis de reglas.

La siguiente figura muestra cómo aparecería esta condición en el desarrollo de plantillas de GRAT.



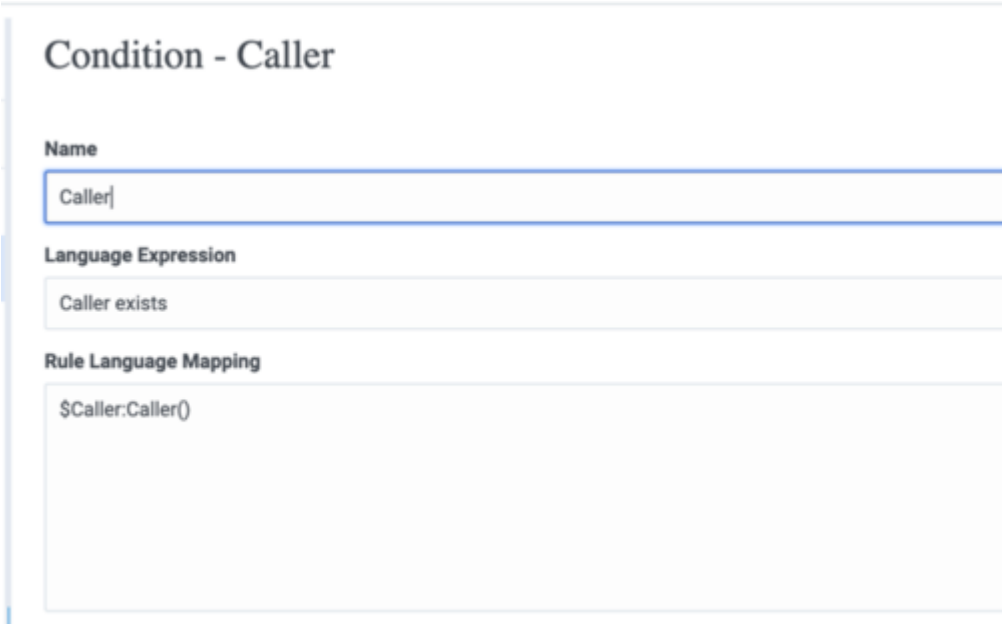
Condición de la persona que llama

Además de probar que la persona que llama sí existe, la siguiente condición también crea la variable `$Caller` que usan las acciones para modificar el dato de la persona que llama. La persona que llama modificada, se devolverá en los resultados de la solicitud de evaluación.

No puede crear una variable más de una vez dentro de una regla, y no puede usar variables en acciones si las variables no se han definido en la condición.

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

La figura a continuación muestra cómo aparecería esta condición en el desarrollo de reglas GRAT.



Acciones del grupo de agente de destino

La acción se configuraría del siguiente modo:

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

La figura a continuación muestra cómo aparecería esta acción en el desarrollo de reglas GRAT.

Parámetros

La condición en este ejemplo tiene dos parámetros:

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

La acción tiene el parámetro `{agentGroup}`. La captura de pantalla del editor de parámetros muestra un parámetro `{ageHigh}` como ejemplo.

Cómo funciona

La forma en que funcionaría el ejemplo anterior es la siguiente:

1. El desarrollador de reglas crea un modelo de datos (o el modelo de datos podría incluirse como parte de una plantilla de reglas listo para usarse con una solución Genesys particular). El modelo de datos describe las propiedades del `dato del Cliente` y el dato de `la persona que llama`. En este caso, podemos ver que el dato del `Cliente` tiene una propiedad llamada edad (probablemente un número entero) y el dato `persona que llama` tiene una propiedad llamada `targetAgentGroup` (muy probablemente una cadena).

2. El desarrollador de la regla crea los parámetros `ageLow` y `ageHigh`, que se convertirán en campos editables que el usuario de negocio llenará cuando esté creando una regla comercial que use esta plantilla de regla. Estos parámetros serían del tipo `Valor de entrada` donde el `Tipo de valor` probablemente sea un entero. El desarrollador de la regla puede, de manera opcional, restringir los posibles valores que el usuario de negocio podrá ingresar colocando un mínimo y/o un máximo.
3. El desarrollador de la regla también crea el parámetro `agentGroup`, que probablemente será una lista seleccionable mediante la cual se presentará al usuario de negocio una lista desplegable de valores que se extraen del servidor Genesys Configuration Server o de un origen de datos externo. El comportamiento de este parámetro depende del tipo de parámetro seleccionado por el desarrollador de la regla.
4. El desarrollador de la regla crea una acción de regla y una condición de regla, como se describió anteriormente. La acción y la condición incluyen asignaciones de lenguaje de reglas que indican al motor de reglas qué datos usar o actualizar según la información que se pasa al motor de reglas como parte (de la solicitud de evaluación de reglas que proviene de una aplicación cliente como una aplicación SCXML).
5. El desarrollador de reglas publica la plantilla de reglas en el repositorio de reglas.
6. El autor de reglas usa esta plantilla de reglas para crear una o más reglas de negocios que utilizan las condiciones y acciones combinadas según se requiera para describir la lógica de negocios que el autor de reglas quiere aplicar. En este caso, las condiciones y acciones descritas anteriormente probablemente se usarían juntas en una sola regla, pero las condiciones y acciones también podrían combinarse con otras condiciones y acciones disponibles para crear diferentes políticas de negocio.
7. El autor de reglas implementa el paquete de reglas en el servidor de aplicaciones del motor de reglas.
8. Una aplicación cliente, como una aplicación VXML o SCXML, invoca al motor de reglas y especifica el paquete de reglas que se evaluará. La solicitud al Motor de reglas incluirá los parámetros de entrada y salida para el modelo de datos. En este ejemplo, tendría que incluir la propiedad de edad del dato del cliente. Esta edad podría haberse recopilado a través de GVP o extraída de una base de datos de clientes antes de que se llamara al motor de reglas. Según el valor de la propiedad de dato `Customer.age` que se pasa al motor de reglas como parte de la solicitud de evaluación de reglas, el motor de reglas evaluará un conjunto particular de las reglas que se han implementado. En este ejemplo, evaluará si `Customer.age` se encuentra entre los límites inferior y superior que el autor de las reglas especificó en la regla.
9. Si el motor de reglas evalúa la regla como verdadera, la propiedad `targetAgentGroup` del dato `dela persona que llamar` se actualizará con el nombre del grupo de agentes seleccionado por el autor de las reglas de negocio cuando escribió la regla. El valor de la propiedad `Caller.targetAgentGroup` se devolverá a la aplicación cliente para su posterior procesamiento. En este ejemplo, tal vez el valor de `Caller.targetAgentGroup` se asignará a una variable de aplicación Composer que luego pasará al bloque destino para pedirle al servidor Genesys Universal Routing Server que apunte a ese grupo de agentes.

Ejemplo 2: Función

Las funciones se utilizan para elementos más complejos y están escritas en Java. En este ejemplo, la función se usa para comparar fechas. Se puede

configurar del siguiente modo:

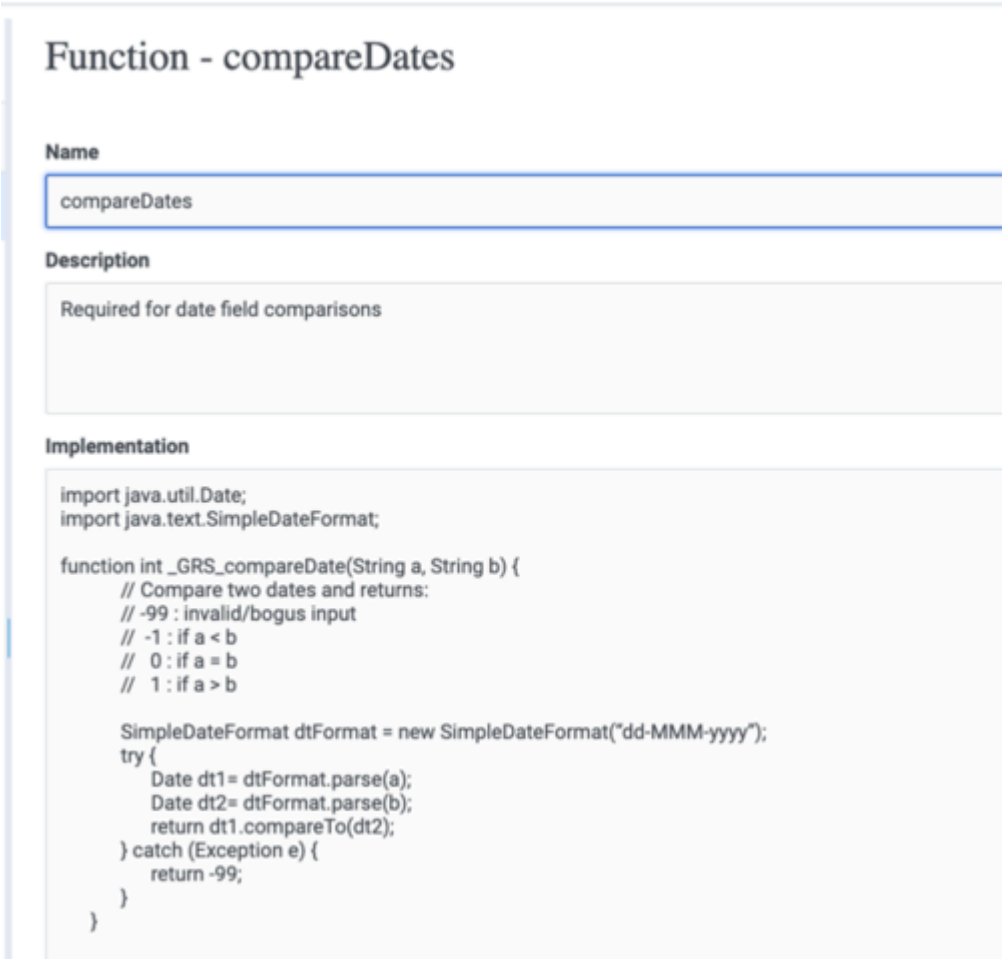
```
Name: compareDates
Description: This function is required to compare dates.
Implementation:
import java.util.Date;
import java.text.SimpleDateFormat;

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;
            }
      }
```

Para las clases proporcionadas por el usuario, el archivo .jar debe estar en CLASSPATH tanto para GRAT como para GRE.

La figura siguiente muestra cómo aparecería esta función en el desarrollo de reglas GRAT.



Function - compareDates

Name

compareDates

Description

Required for date field comparisons

Implementation

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

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

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

Ejemplo 3: Cómo usar un objeto JSON

Los desarrolladores de plantillas pueden crear plantillas que permitan a las aplicaciones cliente pasar datos a GRE como objetos JSON sin tener que asignar explícitamente cada campo al modelo de dato.

Importante

Las reglas basadas en plantillas que usan esta funcionalidad, por el momento no admiten la creación de escenarios de prueba.

Este ejemplo muestra cómo crear una plantilla que contenga una clase (llamada `MyJson`) para pasar un objeto JSON.

Inicio

1. Cree la siguiente clase e impórtela en una plantilla de regla:

```
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. Cree un objeto de dato ficticio con el mismo nombre (`MyJson`) en la plantilla.
3. Agregue `MyJson.class` a la ruta de clase de GRAT y GRE.
4. Cree la siguiente condición y acción:

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

5. Use esta condición y acción en una regla dentro del paquete `json.test`. Se generará lo siguiente:

```
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. Implemente el paquete `json.test` en GRE.
7. Ejecute la siguiente solicitud de ejecución desde RESTClient:

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

8. Se genera la siguiente respuesta:

```
{"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"]}}}}
```


Fin

Ejemplo 4: Desarrollo de plantillas para habilitar escenarios de prueba

> ### Importante
> La creación y edición de eventos no se admite en la versión inicial 9.0.0 de GRAT, por lo que no se pueden desarrollar plantillas que admitan escenarios de prueba. Sin embargo, las plantillas que admiten eventos/escenarios de prueba creados en la herramienta Genesys Rules Development Tool en 8.5 aún pueden desarrollarse en GRDT, importarse a GRAT 9.0 y usarse para crear paquetes de reglas que admitan eventos/escenarios de prueba.

Para obtener más información sobre este tema, consulte *Desarrollo de plantillas para habilitar escenarios de prueba* en la Ayuda de GRDT 8.1.3.

Asignación de varias instancias de un parámetro de regla a una sola definición de parámetro

Al momento de crear parámetros, en lugar de crear los parámetros `ageLow` y `ageHigh`, el desarrollador de plantillas de reglas puede crear un único parámetro `{age}` y usar la notación de subrayado que se muestra en el siguiente ejemplo para crear índices para escenarios en los que se requieren múltiples instancias de parámetros con el mismo tipo `(edad)` (más comúnmente utilizado con rangos). Por ejemplo: `{age_1}, {age_2}....{age_n}` Estos se convertirán en campos editables. Esta característica se usa generalmente para definir rangos de manera más eficiente.

Dato/condición

Se puede hacer referencia a los datos en condiciones y acciones prefijando el nombre del dato con un signo `$`. Por ejemplo, se puede hacer referencia al dato `Persona que llama` con el nombre `$Caller`. GRS generará implícitamente una condición que asocia la variable `$Caller` al dato `Persona que llama` (es decir, `$Caller:Persona que llama()`).

La condición `$Caller:Persona que llama()` requiere un objeto `Persona que llama` como entrada para la ejecución de reglas para que esta condición se evalúe como verdadera.

Plantilla:BestPractices