# Genesys Rules Authoring Tool Help

## Beispiele für die Vorlagenentwicklung

5/3/2025

# Inhaltsverzeichnis

# Beispiele für die Vorlagenentwicklung

Dieser Abschnitt enthält einige Beispiele für Konfigurationen eines Regelentwicklers auf der Registerkarte 'Vorlagenentwicklung'. Ausführliche Informationen über die Konfiguration von Regelvorlagen für die Nutzung mit der Lösung 'Intelligent Workload Distribution' (iWD) von Genesys finden Sie im Handbuch *iWD und Genesys Rules System*.

# Beispiel 1: Bedingung und Aktion

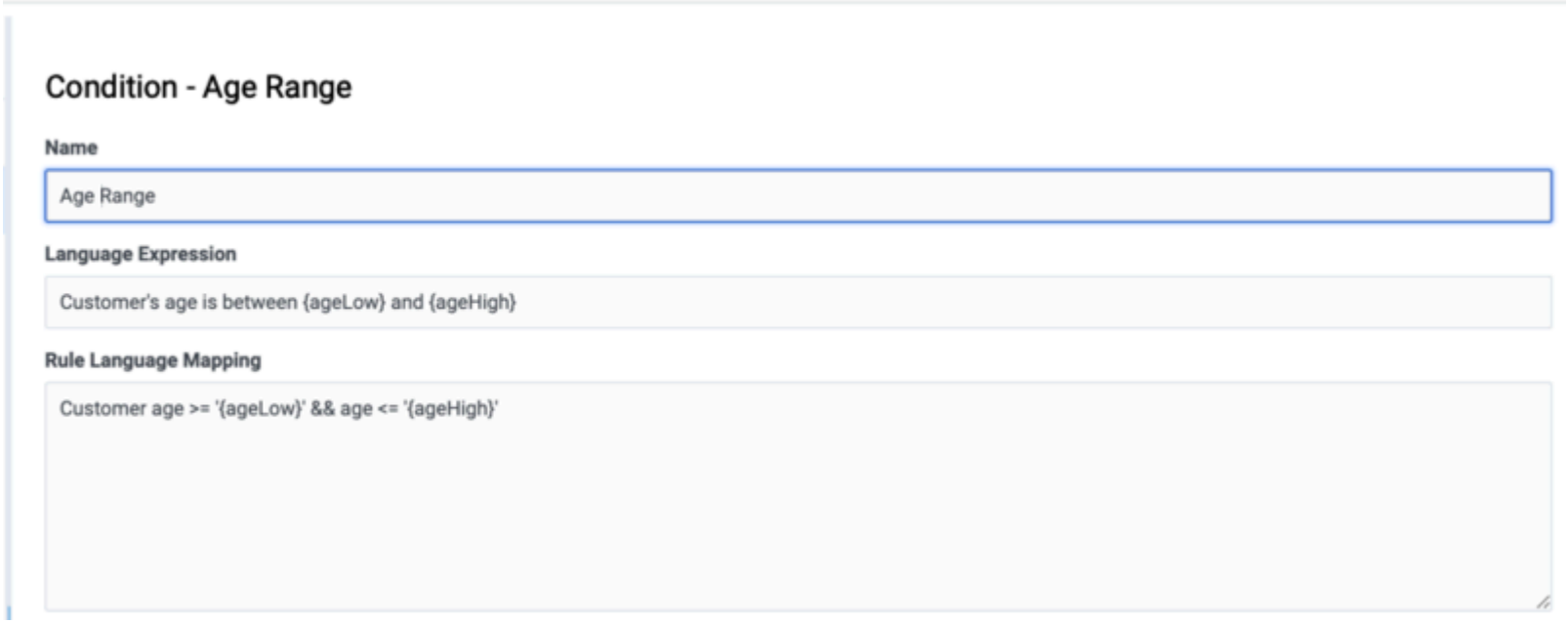
## Altersbereich-Bedingung

Wenn das Alter eines Kunden innerhalb eines bestimmten Bereichs liegt, wird eine bestimmte Agentengruppe als Ziel ausgewählt. In diesem Szenario lautet die Bedingung, ob das Alter des Kunden in den Bereich fällt. Im Genesys Rules Development Tool wurden die Bedingungen wie folgt konfiguriert:

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

Verwenden Sie in den Regelsprachausdrücken nicht das Wort 'end'. Dadurch werden Regelanalysefehler verursacht.

Die folgende Abbildung zeigt, wie diese Bedingung in der GRAT-Vorlagenentwicklung angezeigt wird.



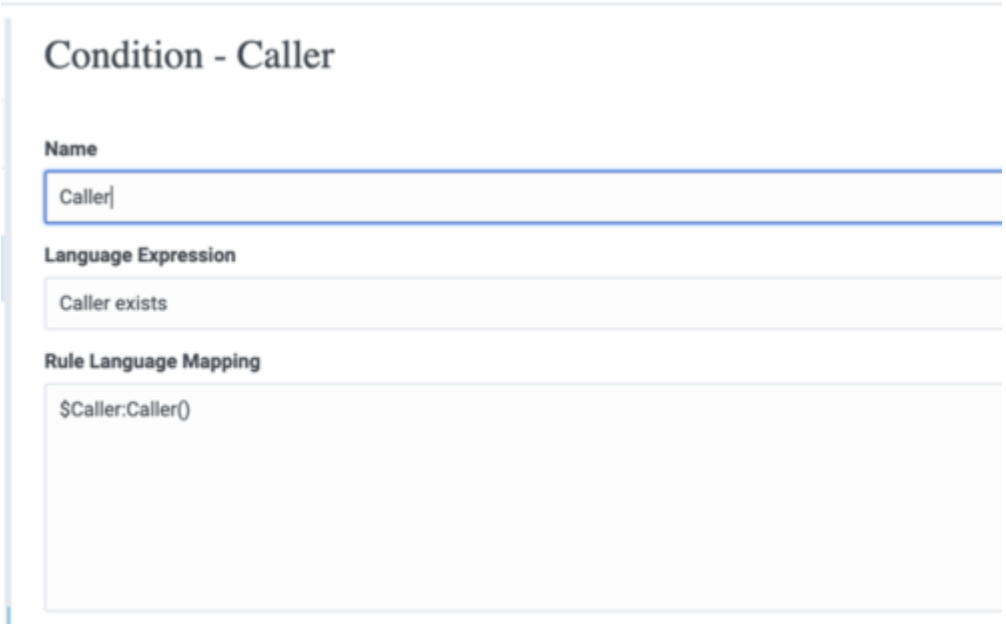

## Anruferbedingung

Zusätzlich zur Prüfung, ob der Anrufer (Caller) vorhanden ist, erstellt die nächste Bedingung auch die Variable `$Caller`, die von Aktionen verwendet wird, um die Caller-Fakt zu ändern. Der geänderte Caller wird in den Ergebnissen des Evaluierungsantrags zurückgegeben.

Sie können eine Variable nicht mehrfach innerhalb einer Regel erstellen, und Sie können keine Variablen in Aktionen verwenden, wenn die Variablen nicht in der Bedingung definiert wurden.

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

Die folgende Abbildung zeigt, wie diese Bedingung in der GRAT-Regelentwicklung angezeigt wird.




## Aktion der Ziel-Agentengruppe

Die Aktion würde wie folgt konfiguriert werden:

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

Die folgende Abbildung zeigt, wie diese Aktion in der GRAT-Regelentwicklung angezeigt wird.


## Parameter

Die Bedingung in diesem Beispiel hat zwei Parameter:

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

Die Aktion enthält den Parameter `{agentGroup}`. Der Parametereditor-Screenshot zeigt den Beispielparameter `{ageHigh}`.


## Funktionsweise

Die Funktionsweise des vorherigen Beispiels lautet wie folgt:

1. Der Regelentwickler erstellt ein Faktenmodell (oder das Faktenmodell kann als Teil einer Regelvorlage enthalten sein, die standardmäßig in einer bestimmten Genesys-Lösung zur Verfügung steht). Das Faktenmodell beschreibt die Eigenschaften des Fakts `Customer fact` und den `Caller`. In diesem Fall sehen wir, dass der Kunde eine Eigenschaft namens age (wahrscheinlich eine Ganzzahl) aufweist und der `Caller`-Fakt eine Eigenschaft namens `{targetAgentGroup}` (höchstwahrscheinlich eine Zeichenfolge) hat.

2. Der Regelentwickler erstellt die Parameter `ageLow` und `ageHigh`, die bearbeitbare Felder werden können. Diese kann der Business-Benutzer ausfüllen wird, wenn er eine Geschäftsregel erstellt, die diese Regelvorlage verwendet. Diese Parameter hätten den Typ `Eingabewert`, wobei der `Werttyp` wahrscheinlich eine Ganzzahl wäre. Der Regelentwickler kann optional die möglichen Werte einschränken, die der Business-Benutzer eingeben kann, indem er ein Minimum und/oder Maximum eingibt.
3. Der Regelentwickler erstellt außerdem den Parameter `agentGroup`, der wahrscheinlich eine auswählbare Liste ist, in der dem Business-Benutzer eine Dropdown-Liste mit Werten angezeigt wird, die aus dem Genesys Configuration Server oder aus einer externen Datenquelle abgerufen werden. Das Verhalten dieses Parameters hängt vom Parametertyp ab, der vom Regelentwickler ausgewählt wurde.
4. Der Regelentwickler erstellt wie zuvor beschrieben eine Regelaktion und eine Regelbedingung. Aktion und Bedingung enthalten Regelsprachenzuordnungen, die die Regelengine anweisen, welche Fakten basierend auf Informationen, die (als Teil der Regelauswertungsanforderung von einer Client-Anwendung wie einer SCXML-Anwendung) an die Regelengine weitergeleitet werden, verwendet oder aktualisiert werden sollen.
5. Der Regelentwickler veröffentlicht die Regelvorlage im Regel-Repository.
6. Der Regelautor verwendet diese Regelvorlage, um eine oder mehrere Geschäftsregeln zu erstellen, die die Bedingungen und Aktionen in den Kombinationen verwenden, die erforderlich sind, um die Geschäftslogik zu beschreiben, die der Regelautor durchsetzen möchte. In diesem Fall werden die oben beschriebenen Bedingungen und Aktionen wahrscheinlich zusammen in einer einzelnen Regel verwendet, aber die Bedingungen und Aktionen können auch mit anderen verfügbaren Bedingungen und Aktionen kombiniert werden, um unterschiedliche Geschäftsrichtlinien zu erstellen.
7. Der Regelautor stellt das Regelpaket auf dem Regelengine-Anwendungsserver bereit.
8. Eine Client-Anwendung, z. B. eine VXML- oder SCXML-Anwendung, ruft die Regelengine auf und legt das zu evaluierende Regelpaket fest. Die Anforderung an die Regelengine enthält die Eingabe- und Ausgabeparameter für das Faktenmodell. In diesem Beispiel müsste die Eigenschaft 'Alter' des Fakts 'Kunde' enthalten sein. Dieses Alter wurde möglicherweise über GVP erfasst oder vor dem Aufruf der Regelengine aus einer Kundendatenbank extrahiert. Basierend auf dem Wert der Fakteneigenschaft `Customer.age`, die als Teil der Regelauswertungsanforderung an die Regelengine übergeben wird, wertet die Regelengine eine bestimmte Gruppe der bereitgestellten Regeln aus. In diesem Beispiel wird ausgewertet, ob `Customer.age` zwischen der unteren und oberen Grenze liegt, die der Regelautor in der Regel angegeben hat.
9. Wenn die Regel von der Regelengine als wahr ausgewertet wird, wird die Eigenschaft `targetAgentGroup` des Fakts `Caller` mit dem Namen der Agentengruppe aktualisiert, die vom Geschäftsregelautor beim Schreiben der Regel ausgewählt wurde. Der Wert der Eigenschaft `Caller.targetAgentGroup` wird zur weiteren Verarbeitung an die Client-Anwendung zurückgegeben. In diesem Beispiel wird möglicherweise der Wert von `Caller.targetAgentGroup` einer Composer-Anwendungsvariablen zugeordnet, die dann an den Zielblock weitergeleitet wird, um den Genesys Universal Routing Server aufzufordern, diese Agentengruppe zu wählen.

## Beispiel 2: Funktion

Funktionen werden für komplexere Elemente verwendet und sind in Java geschrieben. In diesem Beispiel wird die Funktion zum Vergleichen von

Datumswerten verwendet. Sie wird wie folgt konfiguriert:

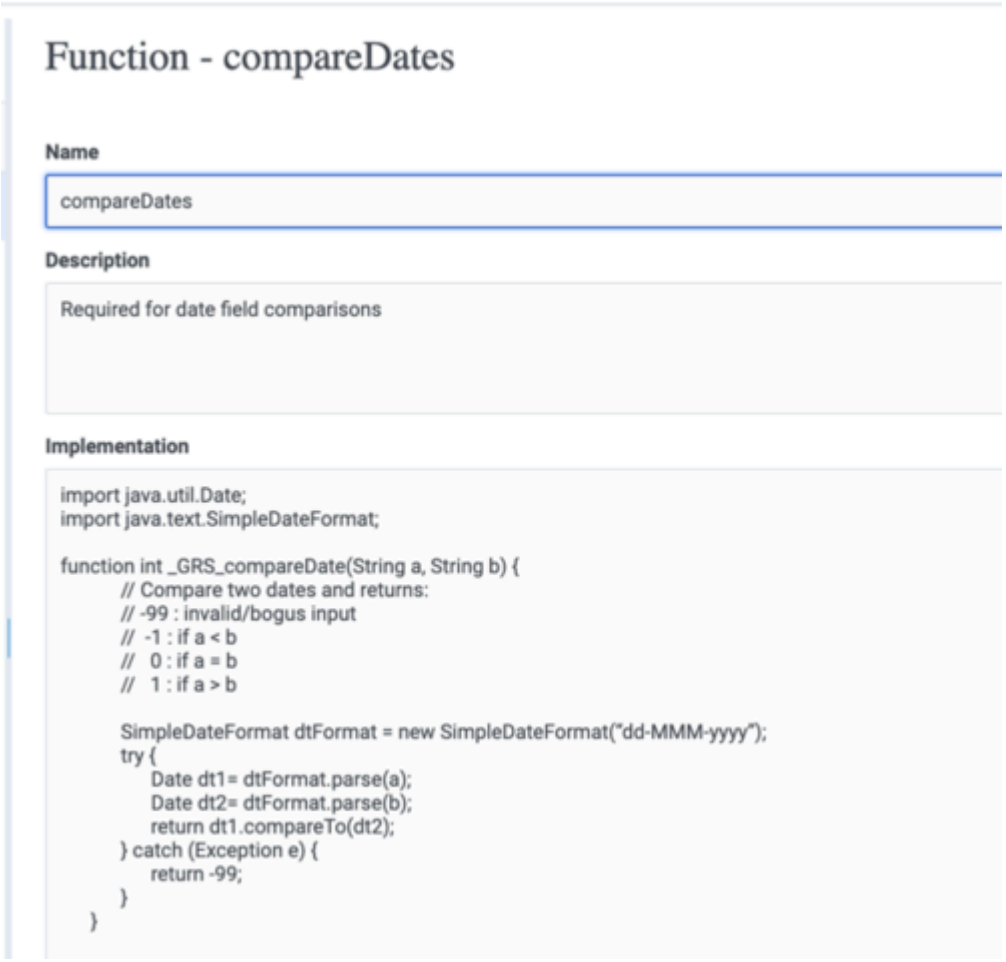
```
Name: compareDates
Description: This function is required to compare dates.
Implementation:
import java.util.Date;
import java.text.SimpleDateFormat;

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;
            }
      }
```

Für vom Benutzer angegebene Klassen muss sich die JAR-Datei sowohl für GRAT als auch für GRE im CLASSPATH befinden.

Die folgende Abbildung zeigt, wie diese Funktion in der GRAT-Regelentwicklung Development angezeigt wird.



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;
        }
    }
```

## Beispiel 3: Verwenden eines JSON-Objekts

Vorlagenentwickler können Vorlagen erstellen, die es Client-Anwendungen ermöglichen, Fakten als JSON-Objekte an GRE weiterzuleiten, ohne jedes Feld explizit dem Faktenmodell zuordnen zu müssen.

> ### Wichtig
> Regeln, die auf Vorlagen basieren, die diese Funktion verwenden, unterstützen derzeit nicht die Erstellung von Testszenarien.

Dieses Beispiel zeigt, wie Sie eine Vorlage erstellen, die eine Klasse (namens `MyJson`) für die Übergabe eines JSON-Objekts enthält.

**Start**

1. Erstellen Sie die folgende Klasse, und importieren Sie sie in eine Regelvorlage:

```
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. Erstellen Sie ein Dummy-Faktenobjekt mit demselben Namen (`MyJson`) in der Vorlage.
3. Fügen Sie `MyJson.class` zum Klassenpfad von GRAT und GRE hinzu.
4. Erstellen Sie die folgende Bedingung und Aktion:

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

5. Verwenden Sie diese Bedingung und Aktion in einer Regel im Paket `json.test`. Folgendes wird generiert:

```
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. Stellen Sie das Paket `json.test` in GRE bereit.
7. Führen Sie folgende Ausführungsanforderung aus dem REST-Client aus:

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

8. Die folgende Antwort wird generiert:

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

**Ende**

## Beispiel 4: Entwickeln von Vorlagen zum Aktivieren von Testszenarien

### Wichtig

Das Erstellen und Bearbeiten von Ereignissen wird in der GRAT-Anfangsversion 9.0.0 nicht unterstützt, so dass Vorlagen, die Testszenarien unterstützen, nicht bereitgestellt werden können. Vorlagen, die Ereignisse/Testszenarien unterstützen, die im Genesys Rules Development Tool 8.5 erstellt wurden, können jedoch weiterhin in GRDT entwickelt, in GRAT 9.0 importiert und zur Erstellung von Regelpaketen verwendet werden, die Ereignisse/Testszenarien unterstützen.

Weitere Informationen zu diesem Thema finden Sie unter *Entwickeln von Vorlagen zur Aktivierung von Testszenarien* in der Hilfe von GRDT 8.1.3.

### Zuordnen mehrerer Instanzen eines Regelparameters zu einer einzelnen Parameterdefinition

Zum Zeitpunkt der Parametererstellung kann der Entwickler der Regelvorlage anstelle der Parameter `ageLow` und `ageHigh` einen einzelnen Parameter `{age}` erstellen und die Unterstrichnotation wie im unteren Beispiel nutzen, um Indizes für Szenarien zu erstellen, bei denen mehrere Instanzen von Parametern des gleichen Typs (`Alter`) erforderlich sind (am häufigsten für Bereiche verwendet). Beispiel: `{age_1}, {age_2}....{age_n}` Diese Felder werden zu bearbeitbaren Feldern. Diese Funktion wird in der Regel zur effizienteren Bereichsdefinition verwendet.

### Fakt/Bedingung

Fakten können in Bedingungen und Aktionen referenziert werden, indem der Name des Fakts durch ein `$`-Zeichen vorangestellt wird. Beispielsweise kann auf den Fakt `Caller` durch den Namen `$Caller` verwiesen werden. GRS generiert implizit eine Bedingung, durch die die Variable `$Caller` dem Fakt `Caller` zugeordnet wird (d. h. `$Caller:Caller()`).

Die Bedingung `$Caller:Caller()` erfordert ein `Caller`-Objekt als Eingabe für die Ausführung der Regeln, damit diese Bedingung als wahr ausgewertet wird.

Vorlage:BestPractices