GRS Best Practice Guide

Example 3—Using a JSON Object

6/5/2026

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