



# **Red Hat Decision Manager 7.2**

## **Interacting with Red Hat Decision Manager using KIE APIs**



# Red Hat Decision Manager 7.2 Interacting with Red Hat Decision Manager using KIE APIs

---

Red Hat Customer Content Services  
brms-docs@redhat.com

## Legal Notice

Copyright © 2019 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This document describes how to use KIE APIs to interact with Decision Servers, KIE containers, and business assets in Red Hat Decision Manager 7.2.

## Table of Contents

<b>PREFACE</b> .....	<b>3</b>
<b>CHAPTER 1. DECISION SERVER REST API FOR KIE CONTAINERS AND BUSINESS ASSETS</b> .....	<b>4</b>
1.1. SENDING REQUESTS WITH THE DECISION SERVER REST API USING A REST CLIENT OR CURL UTILITY	6
1.2. SENDING REQUESTS WITH THE DECISION SERVER REST API USING THE SWAGGER INTERFACE	10
1.3. SUPPORTED DECISION SERVER REST API ENDPOINTS	13
<b>CHAPTER 2. DECISION SERVER JAVA CLIENT API FOR KIE CONTAINERS AND BUSINESS ASSETS</b> ..	<b>14</b>
2.1. SENDING REQUESTS WITH THE DECISION SERVER JAVA CLIENT API	18
2.2. SUPPORTED DECISION SERVER JAVA CLIENTS	24
2.3. EXAMPLE REQUESTS WITH THE DECISION SERVER JAVA CLIENT API	25
<b>CHAPTER 3. DECISION SERVER AND KIE CONTAINER COMMANDS IN RED HAT DECISION MANAGER</b>	<b>29</b>
3.1. SAMPLE DECISION SERVER AND KIE CONTAINER COMMANDS	29
<b>CHAPTER 4. RUNTIME COMMANDS IN RED HAT DECISION MANAGER</b> .....	<b>44</b>
4.1. SAMPLE RUNTIME COMMANDS IN RED HAT DECISION MANAGER	44
<b>CHAPTER 5. DECISION MANAGER CONTROLLER REST API FOR DECISION SERVER TEMPLATES AND INSTANCES</b> .....	<b>60</b>
5.1. SENDING REQUESTS WITH THE DECISION MANAGER CONTROLLER REST API USING A REST CLIENT OR CURL UTILITY	62
5.2. SENDING REQUESTS WITH THE DECISION MANAGER CONTROLLER REST API USING THE SWAGGER INTERFACE	66
5.3. SUPPORTED DECISION MANAGER CONTROLLER REST API ENDPOINTS	69
<b>CHAPTER 6. DECISION MANAGER CONTROLLER JAVA CLIENT API FOR DECISION SERVER TEMPLATES AND INSTANCES</b> .....	<b>71</b>
6.1. SENDING REQUESTS WITH THE DECISION MANAGER CONTROLLER JAVA CLIENT API	75
6.2. SUPPORTED DECISION MANAGER CONTROLLER JAVA CLIENTS	78
6.3. EXAMPLE REQUESTS WITH THE DECISION MANAGER CONTROLLER JAVA CLIENT API	78
<b>CHAPTER 7. KNOWLEDGE STORE REST API FOR DECISION CENTRAL SPACES AND PROJECTS</b> ....	<b>84</b>
7.1. SENDING REQUESTS WITH THE KNOWLEDGE STORE REST API USING A REST CLIENT OR CURL UTILITY	85
7.2. SUPPORTED KNOWLEDGE STORE REST API ENDPOINTS	89
7.2.1. Spaces	89
7.2.2. Projects	94
7.2.3. Jobs (API requests)	100
<b>CHAPTER 8. ADDITIONAL RESOURCES</b> .....	<b>103</b>
<b>APPENDIX A. VERSIONING INFORMATION</b> .....	<b>104</b>



## PREFACE

As a business rules developer or systems administrator, you can use KIE APIs to interact with Decision Servers, KIE containers, and business assets in Red Hat Decision Manager. You can use the Decision Server REST API and Java client API to interact with KIE containers and business assets (such as business rules, processes, and solvers), the Decision Manager controller REST API and Java client API to interact with Decision Server templates and instances, and the Knowledge Store REST API to interact with spaces and projects in Decision Central.



### REST API ENDPOINTS FOR DECISION SERVER AND THE DECISION MANAGER CONTROLLER

The lists of REST API endpoints for Decision Server and the Decision Manager controller are published separately from this document and maintained dynamically to ensure that endpoint options and data are as current as possible. Use this document to understand what the Decision Server and Decision Manager controller REST APIs enable you to do and how to use them, and use the separately maintained lists of REST API endpoints for specific endpoint details.

For the full list of Decision Server REST API endpoints and descriptions, use one of the following resources:

- [Execution Server REST API](#) on the jBPM Documentation page (static)
- Swagger UI for the Decision Server REST API at <http://SERVER:PORT/kie-server/docs> (dynamic, requires running Decision Server)

For the full list of Decision Manager controller REST API endpoints and descriptions, use one of the following resources:

- [Controller REST API](#) on the jBPM Documentation page (static)
- Swagger UI for the Decision Manager controller REST API at <http://SERVER:PORT/CONTROLLER/docs> (dynamic, requires running Decision Manager controller)

### Prerequisites

- Red Hat Decision Manager is installed and running. For installation and startup options, see [Planning a Red Hat Decision Manager installation](#).
- You have access to Red Hat Decision Manager with the following user roles:
  - **kie-server**: For access to Decision Server API capabilities, and access to headless Decision Manager controller API capabilities without Decision Central (if applicable)
  - **rest-all**: For access to Decision Central API capabilities for the built-in Decision Manager controller and for the Decision Central Knowledge Store
  - **admin**: For full administrative access to Red Hat Decision Manager

Although these user roles are not all required for every KIE API, consider acquiring all of them to ensure that you can access any KIE API without disruption. For more information about user roles, see [Planning a Red Hat Decision Manager installation](#).

# CHAPTER 1. DECISION SERVER REST API FOR KIE CONTAINERS AND BUSINESS ASSETS

Red Hat Decision Manager provides a Decision Server REST API that you can use to interact with your KIE containers and business assets (such as business rules, processes, and solvers) in Red Hat Decision Manager without using the Decision Central user interface. This API support enables you to maintain your Red Hat Decision Manager resources more efficiently and optimize your integration and development with Red Hat Decision Manager.

With the Decision Server REST API, you can perform the following actions:

- Deploy or dispose KIE containers
- Retrieve and update KIE container information
- Return Decision Server status and basic information
- Retrieve and update business asset information
- Execute business assets (such as rules and processes)

Decision Server REST API requests require the following components:

## Authentication

The Decision Server REST API requires HTTP Basic authentication or token-based authentication for the user role **kie-server**. To view configured user roles for your Red Hat Decision Manager distribution, navigate to `~/$SERVER_HOME/standalone/configuration/application-roles.properties` and `~/application-users.properties`.

To add a user with the **kie-server** role, navigate to `~/$SERVER_HOME/bin` and run the following command:

```
$ ./add-user.sh -a --user <USERNAME> --password <PASSWORD> --role kie-server
```

For more information about user roles and Red Hat Decision Manager installation options, see [Planning a Red Hat Decision Manager installation](#).

## HTTP headers

The Decision Server REST API requires the following HTTP headers for API requests:

- **Accept**: Data format accepted by your requesting client:
  - **application/json** (JSON)
  - **application/xml** (XML, for JAXB or XSTREAM)
- **Content-Type**: Data format of your **POST** or **PUT** API request data:
  - **application/json** (JSON)
  - **application/xml** (XML, for JAXB or XSTREAM)
- **X-KIE-ContentType**: Required header for **application/xml** XSTREAM API requests and responses:



- o **XSTREAM**

## HTTP methods

The Decision Server REST API supports the following HTTP methods for API requests:

- **GET**: Retrieves specified information from a specified resource endpoint
- **POST**: Updates a resource or resource instance
- **PUT**: Updates or creates a resource or resource instance
- **DELETE**: Deletes a resource or resource instance

## Base URL

The base URL for Decision Server REST API requests is **http://SERVER:PORT/kie-server/services/rest/**, such as **http://localhost:8080/kie-server/services/rest/**.

## Endpoints

Decision Server REST API endpoints, such as **/server/containers/{containerId}** for a specified KIE container, are the URLs that you append to the Decision Server REST API base URL to access the corresponding resource or type of resource in Red Hat Decision Manager.

### Example request URL for /server/containers/{containerId} endpoint

**http://localhost:8080/kie-server/services/rest/server/containers/MyContainer**

## Request parameters and request data

Many Decision Server REST API requests require specific parameters in the request URL path to identify or filter specific resources and to perform specific actions. You can append URL parameters to the endpoint in the format **?<PARAM>=<VALUE>&<PARAM>=<VALUE>**.

### Example GET request URL with parameters

**http://localhost:8080/kie-server/services/rest/server/containers?groupId=com.redhat&artifactId=Project1&version=1.0&status=STARTED**

HTTP **POST** and **PUT** requests may additionally require a request body or file with data to accompany the request.

### Example POST request URL and JSON request body data

**http://localhost:8080/kie-server/services/rest/server/containers/MyContainer/release-id**

```
{
  "release-id": {
    "artifact-id": "Project1",
    "group-id": "com.redhat",
    "version": "1.1"
  }
}
```

## 1.1. SENDING REQUESTS WITH THE DECISION SERVER REST API USING A REST CLIENT OR CURL UTILITY

The Decision Server REST API enables you to interact with your KIE containers and business assets (such as business rules, processes, and solvers) in Red Hat Decision Manager without using the Decision Central user interface. You can send Decision Server REST API requests using any REST client or curl utility.

### Prerequisites

- Decision Server is installed and running.
- You have **kie-server** user role access to Decision Server.

### Procedure

1. Identify the relevant [API endpoint](#) to which you want to send a request, such as **[GET] /server/containers** to retrieve KIE containers from Decision Server.
2. In a REST client or curl utility, enter the following components for a **GET** request to **/server/containers**. Adjust any request details according to your use case.

For REST client:

- **Authentication:** Enter the user name and password of the Decision Server user with the **kie-server** role.
- **HTTP Headers:** Set the following header:
  - **Accept: application/json**
- **HTTP method:** Set to **GET**.
- **URL:** Enter the Decision Server REST API base URL and endpoint, such as **http://localhost:8080/kie-server/services/rest/server/containers**.

For curl utility:

- **-u:** Enter the user name and password of the Decision Server user with the **kie-server** role.
- **-H:** Set the following header:
  - **accept: application/json**
- **-X:** Set to **GET**.
- **URL:** Enter the Decision Server REST API base URL and endpoint, such as **http://localhost:8080/kie-server/services/rest/server/containers**.

```
curl -u 'baAdmin:password@1' -H "accept: application/json" -X GET
"http://localhost:8080/kie-server/services/rest/server/containers"
```

3. Execute the request and review the Decision Server response.  
Example server response (JSON):

■

```

{
  "type": "SUCCESS",
  "msg": "List of created containers",
  "result": {
    "kie-containers": {
      "kie-container": [
        {
          "container-id": "itorders_1.0.0-SNAPSHOT",
          "release-id": {
            "group-id": "itorders",
            "artifact-id": "itorders",
            "version": "1.0.0-SNAPSHOT"
          },
          "resolved-release-id": {
            "group-id": "itorders",
            "artifact-id": "itorders",
            "version": "1.0.0-SNAPSHOT"
          },
          "status": "STARTED",
          "scanner": {
            "status": "DISPOSED",
            "poll-interval": null
          },
          "config-items": [],
          "container-alias": "itorders"
        }
      ]
    }
  }
}

```

4. For this example, copy or note the project **group-id**, **artifact-id**, and **version** (GAV) data from one of the deployed KIE containers returned in the response.
5. In your REST client or curl utility, send another API request with the following components for a **PUT** request to **/server/containers/{containerId}** to deploy a new KIE container with the copied project GAV data. Adjust any request details according to your use case.  
For REST client:

- **Authentication:** Enter the user name and password of the Decision Server user with the **kie-server** role.
- **HTTP Headers:** Set the following headers:
  - **Accept:** **application/json**
  - **Content-Type:** **application/json**
- **HTTP method:** Set to **PUT**.
- **URL:** Enter the Decision Server REST API base URL and endpoint, such as **http://localhost:8080/kie-server/services/rest/server/containers/MyContainer**.
- **Request body:** Add a JSON request body with the configuration items for the new KIE container:

```

{
  "config-items": [
    {
      "itemName": "RuntimeStrategy",
      "itemValue": "SINGLETON",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "MergeMode",
      "itemValue": "MERGE_COLLECTIONS",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KBase",
      "itemValue": "",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KSession",
      "itemValue": "",
      "itemType": "java.lang.String"
    }
  ],
  "release-id": {
    "group-id": "itorders",
    "artifact-id": "itorders",
    "version": "1.0.0-SNAPSHOT"
  },
  "scanner": {
    "poll-interval": "5000",
    "status": "STARTED"
  }
}

```

For curl utility:

- **-u**: Enter the user name and password of the Decision Server user with the **kie-server** role.
- **-H**: Set the following headers:
  - **accept: application/json**
  - **content-type: application/json**
- **-X**: Set to **PUT**.
- **URL**: Enter the Decision Server REST API base URL and endpoint, such as **http://localhost:8080/kie-server/services/rest/server/containers/MyContainer**.
- **-d**: Add a JSON request body or file (**@file.json**) with the configuration items for the new KIE container:

```

curl -u 'baAdmin:password@1' -H "accept: application/json" -H
"content-type: application/json" -X PUT "http://localhost:8080/kie-

```

```
server/services/rest/server/containers/MyContainer" -d "{ \"config-
items\": [ { \"itemName\": \"RuntimeStrategy\", \"itemValue\":
\"SINGLETON\", \"itemType\": \"java.lang.String\" }, { \"itemName\":
\"MergeMode\", \"itemValue\": \"MERGE_COLLECTIONS\", \"itemType\":
\"java.lang.String\" }, { \"itemName\": \"KBase\", \"itemValue\":
\"\", \"itemType\": \"java.lang.String\" }, { \"itemName\":
\"KSession\", \"itemValue\": \"\", \"itemType\":
\"java.lang.String\" } ], \"release-id\": { \"group-id\":
\"itorders\", \"artifact-id\": \"itorders\", \"version\": \"1.0.0-
SNAPSHOT\" }, \"scanner\": { \"poll-interval\": \"5000\",
\"status\": \"STARTED\" } }"
```

```
curl -u 'baAdmin:password@1' -H "accept: application/json" -H
"content-type: application/json" -X PUT "http://localhost:8080/kie-
server/services/rest/server/containers/MyContainer" -d @my-
container-configs.json
```

- Execute the request and review the Decision Server response.

Example server response (JSON):

```
{
  "type": "SUCCESS",
  "msg": "Container MyContainer successfully deployed with module
itorders:itorders:1.0.0-SNAPSHOT.",
  "result": {
    "kie-container": {
      "container-id": "MyContainer",
      "release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "resolved-release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "status": "STARTED",
      "scanner": {
        "status": "STARTED",
        "poll-interval": 5000
      },
      "config-items": [],
      "messages": [
        {
          "severity": "INFO",
          "timestamp": {
            "java.util.Date": 1540584717937
          },
          "content": [
            "Container MyContainer successfully created with module
itorders:itorders:1.0.0-SNAPSHOT."
          ]
        }
      ]
    }
  },
}
```

```

    "container-alias": null
  }
}

```

If you encounter request errors, review the returned error code messages and adjust your request accordingly.

## 1.2. SENDING REQUESTS WITH THE DECISION SERVER REST API USING THE SWAGGER INTERFACE

The Decision Server REST API supports a Swagger web interface that you can use instead of a standalone REST client or curl utility to interact with your KIE containers and business assets (such as business rules, processes, and solvers) in Red Hat Decision Manager without using the Decision Central user interface.

### Prerequisites

- Decision Server is installed and running.
- You have **kie-server** user role access to Decision Server.

### Procedure

1. In a web browser, navigate to **http://SERVER:PORT/kie-server/docs**, such as **http://localhost:8080/kie-server/docs**, and log in with the user name and password of the Decision Server user with the **kie-server** role.
2. In the Swagger page, select the relevant API endpoint to which you want to send a request, such as **KIE Server :: Core → [GET] /server/containers** to retrieve KIE containers from Decision Server.
3. Click **Try it out** and provide any optional parameters by which you want to filter results, if needed.
4. In the **Response content type** drop-down menu, select the desired format of the server response, such as **application/json** for JSON format.
5. Click **Execute** and review the Decision Server response.

Example server response (JSON):

```

{
  "type": "SUCCESS",
  "msg": "List of created containers",
  "result": {
    "kie-containers": {
      "kie-container": [
        {
          "container-id": "itorders_1.0.0-SNAPSHOT",
          "release-id": {
            "group-id": "itorders",
            "artifact-id": "itorders",
            "version": "1.0.0-SNAPSHOT"
          },
          "resolved-release-id": {

```

```

        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
    },
    "status": "STARTED",
    "scanner": {
        "status": "DISPOSED",
        "poll-interval": null
    },
    "config-items": [],
    "container-alias": "itorders"
}
]
}
}
}
}

```

6. For this example, copy or note the project **group-id**, **artifact-id**, and **version** (GAV) data from one of the deployed KIE containers returned in the response.
7. In the Swagger page, navigate to the **KIE Server :: Core** → **[PUT] /server/containers/{containerId}** endpoint to send another request to deploy a new KIE container with the copied project GAV data. Adjust any request details according to your use case.
8. Click **Try it out** and enter the following components for the request:
  - **containerId**: Enter the ID of the new KIE container, such as **MyContainer**.
  - **body**: Set the **Parameter content type** to the desired request body format, such as **application/json** for JSON format, and add a request body with the configuration items for the new KIE container:

```

{
  "config-items": [
    {
      "itemName": "RuntimeStrategy",
      "itemValue": "SINGLETON",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "MergeMode",
      "itemValue": "MERGE_COLLECTIONS",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KBase",
      "itemValue": "",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KSession",
      "itemValue": "",
      "itemType": "java.lang.String"
    }
  ],
}

```

```

    "release-id": {
      "group-id": "itorders",
      "artifact-id": "itorders",
      "version": "1.0.0-SNAPSHOT"
    },
    "scanner": {
      "poll-interval": "5000",
      "status": "STARTED"
    }
  }
}

```

9. In the **Response content type** drop-down menu, select the desired format of the server response, such as **application/json** for JSON format.
10. Click **Execute** and review the Decision Server response.  
Example server response (JSON):

```

{
  "type": "SUCCESS",
  "msg": "Container MyContainer successfully deployed with module itorders:itorders:1.0.0-SNAPSHOT.",
  "result": {
    "kie-container": {
      "container-id": "MyContainer",
      "release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "resolved-release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "status": "STARTED",
      "scanner": {
        "status": "STARTED",
        "poll-interval": 5000
      },
      "config-items": [],
      "messages": [
        {
          "severity": "INFO",
          "timestamp": {
            "java.util.Date": 1540584717937
          },
          "content": [
            "Container MyContainer successfully created with module itorders:itorders:1.0.0-SNAPSHOT."
          ]
        }
      ],
      "container-alias": null
    }
  }
}

```



-

If you encounter request errors, review the returned error code messages and adjust your request accordingly.

### 1.3. SUPPORTED DECISION SERVER REST API ENDPOINTS

The Decision Server REST API provides endpoints for the following types of resources in Red Hat Decision Manager:

- Decision Server and KIE containers
- KIE session assets (for runtime commands)
- DMN assets
- Planning solvers

The Decision Server REST API base URL is **`http://SERVER:PORT/kie-server/services/rest/`**. All requests require HTTP Basic authentication or token-based authentication for the **`kie-server`** user role.

For the full list of Decision Server REST API endpoints and descriptions, use one of the following resources:

- [Execution Server REST API](#) on the jBPM Documentation page (static)
- Swagger UI for the Decision Server REST API at **`http://SERVER:PORT/kie-server/docs`** (dynamic, requires running Decision Server)

## CHAPTER 2. DECISION SERVER JAVA CLIENT API FOR KIE CONTAINERS AND BUSINESS ASSETS

Red Hat Decision Manager provides a Decision Server Java client API that enables you to connect to Decision Server using REST protocol from your Java client application. You can use the Decision Server Java client API as an alternative to the Decision Server REST API to interact with your KIE containers and business assets (such as business rules, processes, and solvers) in Red Hat Decision Manager without using the Decision Central user interface. This API support enables you to maintain your Red Hat Decision Manager resources more efficiently and optimize your integration and development with Red Hat Decision Manager.

With the Decision Server Java client API, you can perform the following actions also supported by the Decision Server REST API:

- Deploy or dispose KIE containers
- Retrieve and update KIE container information
- Return Decision Server status and basic information
- Retrieve and update business asset information
- Execute business assets (such as rules and processes)

Decision Server Java client API requests require the following components:

### Authentication

The Decision Server Java client API requires HTTP Basic authentication for the user role **kie-server**. To view configured user roles for your Red Hat Decision Manager distribution, navigate to `~/$SERVER_HOME/standalone/configuration/application-roles.properties` and `~/application-users.properties`.

To add a user with the **kie-server** role, navigate to `~/$SERVER_HOME/bin` and run the following command:

```
$ ./add-user.sh -a --user <USERNAME> --password <PASSWORD> --role kie-server
```

For more information about user roles and Red Hat Decision Manager installation options, see [Planning a Red Hat Decision Manager installation](#).

### Project dependencies

The Decision Server Java client API requires the following dependencies on the relevant classpath of your Java project:

```
// For remote execution on Decision Server
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhdm.version}</version>
</dependency>

// For runtime commands
<dependency>
  <groupId>org.drools</groupId>
```

```

<artifactId>drools-compiler</artifactId>
<scope>runtime</scope>
<version>${rhdm.version}</version>
</dependency>

// For debug logging (optional):
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>

```

The **<version>** for Red Hat Decision Manager dependencies is the Maven artifact version for Red Hat Decision Manager currently used in your project (for example, 7.14.0.Final-redhat-00002).

## NOTE

Instead of specifying a Red Hat Decision Manager **<version>** for individual dependencies, consider adding the Red Hat Business Automation bill of materials (BOM) dependency to your project **pom.xml** file. The Red Hat Business Automation BOM applies to both Red Hat Decision Manager and Red Hat Process Automation Manager. When you add the BOM files, the correct versions of transitive dependencies from the provided Maven repositories are included in the project.

Example BOM dependency:

```

<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.2.0.GA-redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>

```

For more information about the Red Hat Business Automation BOM, see [What is the mapping between RHDM product and maven library version?](#).

## Client request configuration

All Java client requests with the Decision Server Java client API must define at least the following server communication components:

- Credentials of the **kie-server** user
- Decision Server location, such as **http://localhost:8080/kie-server/services/rest/server**
- Marshalling format for API requests and responses (JSON, JAXB, or XSTREAM)
- A **KieServicesConfiguration** object and a **KieServicesClient** object, which serve as the entry point for starting the server communication using the Java client API
- A **KieServicesFactory** object defining REST protocol and user access

- Any other client services used, such as **RuleServicesClient**, **ProcessServicesClient**, or **QueryServicesClient**

The following are examples of basic and advanced client configurations with these components:

### Basic client configuration example

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;

public class MyConfigurationObject {

    private static final String URL = "http://localhost:8080/kie-
server/services/rest/server";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static final MarshallingFormat FORMAT =
MarshallingFormat.JSON;

    private static KieServicesConfiguration conf;
    private static KieServicesClient kieServicesClient;

    public static void initialize() {
        conf = KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD);

        //If you use custom classes, such as Obj.class, add them to the
configuration.
        Set<Class<?>> extraClassList = new HashSet<Class<?>>();
        extraClassList.add(Obj.class);
        conf.addExtraClasses(extraClassList);

        conf.setMarshallingFormat(FORMAT);
        kieServicesClient = KieServicesFactory.newKieServicesClient(conf);
    }
}
```

### Advanced client configuration example with additional client services

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.CaseServicesClient;
import org.kie.server.client.DMNServicesClient;
import org.kie.server.client.DocumentServicesClient;
import org.kie.server.client.JobServicesClient;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.ProcessServicesClient;
import org.kie.server.client.QueryServicesClient;
import org.kie.server.client.RuleServicesClient;
import org.kie.server.client.SolverServicesClient;
import org.kie.server.client.UIServicesClient;
import org.kie.server.client.UserTaskServicesClient;
```

```

import org.kie.server.api.model.instance.ProcessInstance;
import org.kie.server.api.model.KieContainerResource;
import org.kie.server.api.model.ReleaseId;

public class MyAdvancedConfigurationObject {

    // REST API base URL, credentials, and marshalling format
    private static final String URL = "http://localhost:8080/kie-
server/services/rest/server";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static final MarshallingFormat FORMAT =
MarshallingFormat.JSON;

    private static KieServicesConfiguration conf;

    // KIE client for common operations
    private static KieServicesClient kieServicesClient;

    // Rules client
    private static RuleServicesClient ruleClient;

    // Process automation clients
    private static CaseServicesClient caseClient;
    private static DocumentServicesClient documentClient;
    private static JobServicesClient jobClient;
    private static ProcessServicesClient processClient;
    private static QueryServicesClient queryClient;
    private static UIServicesClient uiClient;
    private static UserTaskServicesClient userTaskClient;

    // DMN client
    private static DMNServicesClient dmnClient;

    // Planning client
    private static SolverServicesClient solverClient;

    public static void main(String[] args) {
        initializeKieServerClient();
        initializeDroolsServiceClients();
        initializeJbpmServiceClients();
        initializeSolverServiceClients();
    }

    public static void initializeKieServerClient() {
        conf = KieServicesFactory.newRestConfiguration(URL, USER,
PASSWORD);
        conf.setMarshallingFormat(FORMAT);
        kieServicesClient =
KieServicesFactory.newKieServicesClient(conf);
    }

    public static void initializeDroolsServiceClients() {
        ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);

```

```

        dmnClient =
kieServicesClient.getServicesClient(DMNServicesClient.class);
    }

    public static void initializeJbpmServiceClients() {
        caseClient =
kieServicesClient.getServicesClient(CaseServicesClient.class);
        documentClient =
kieServicesClient.getServicesClient(DocumentServicesClient.class);
        jobClient =
kieServicesClient.getServicesClient(JobServicesClient.class);
        processClient =
kieServicesClient.getServicesClient(ProcessServicesClient.class);
        queryClient =
kieServicesClient.getServicesClient(QueryServicesClient.class);
        uiClient =
kieServicesClient.getServicesClient(UIServicesClient.class);
        userTaskClient =
kieServicesClient.getServicesClient(UserTaskServicesClient.class);
    }

    public static void initializeSolverServiceClients() {
        solverClient =
kieServicesClient.getServicesClient(SolverServicesClient.class);
    }
}

```

## 2.1. SENDING REQUESTS WITH THE DECISION SERVER JAVA CLIENT API

The Decision Server Java client API enables you to connect to Decision Server using REST protocol from your Java client application. You can use the Decision Server Java client API as an alternative to the Decision Server REST API to interact with your KIE containers and business assets (such as business rules, processes, and solvers) in Red Hat Decision Manager without using the Decision Central user interface.

### Prerequisites

- Decision Server is installed and running.
- You have **kie-server** user role access to Decision Server.
- You have a Java project with Red Hat Decision Manager resources.

### Procedure

1. In your client application, ensure that the following dependencies have been added to the relevant classpath of your Java project:

```

// For remote execution on Decision Server
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhdm.version}</version>

```

```

</dependency>

// For runtime commands
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-compiler</artifactId>
  <scope>runtime</scope>
  <version>${rhdm.version}</version>
</dependency>

// For debug logging (optional):
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>

```

2. Download the **Red Hat Decision Manager 7.2.0 Source Distribution** from the [Red Hat Customer Portal](#) and navigate to `~/rhdm-7.2.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/server/client` to access the Decision Server Java clients.
3. In the `~/kie/server/client` folder, identify the relevant Java client for the request you want to send, such as **KieServicesClient** to access client services for KIE containers and other assets in Decision Server.
4. In your client application, create a **.java** class for the API request. The class must contain the necessary imports, Decision Server location and user credentials, a **KieServicesClient** object, and the client method to execute, such as **createContainer** and **disposeContainer** from the **KieServicesClient** client. Adjust any configuration details according to your use case.

### Creating and disposing a container

```

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.api.model.KieContainerResource;
import org.kie.server.api.model.ServiceResponse;

public class MyConfigurationObject {

    private static final String URL = "http://localhost:8080/kie-
server/services/rest/server";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static final MarshallingFormat FORMAT =
MarshallingFormat.JSON;

    private static KieServicesConfiguration conf;
    private static KieServicesClient kieServicesClient;

```

```

    public static void initialize() {
        conf = KieServicesFactory.newRestConfiguration(URL, USER,
            PASSWORD);

        public void disposeAndCreateContainer() {
            System.out.println("== Disposing and creating containers ==");

            // Retrieve list of KIE containers
            List<KieContainerResource> kieContainers =
            kieServicesClient.listContainers().getResult().getContainers();
            if (kieContainers.size() == 0) {
                System.out.println("No containers available...");
                return;
            }

            // Dispose KIE container
            KieContainerResource container = kieContainers.get(0);
            String containerId = container.getContainerId();
            ServiceResponse<Void> responseDispose =
            kieServicesClient.disposeContainer(containerId);
            if (responseDispose.getType() == ResponseType.FAILURE) {
                System.out.println("Error disposing " + containerId + ".
            Message: ");
                System.out.println(responseDispose.getMsg());
                return;
            }
            System.out.println("Success Disposing container " +
            containerId);
            System.out.println("Trying to recreate the container...");

            // Re-create KIE container
            ServiceResponse<KieContainerResource> createResponse =
            kieServicesClient.createContainer(containerId, container);
            if(createResponse.getType() == ResponseType.FAILURE) {
                System.out.println("Error creating " + containerId + ".
            Message: ");
                System.out.println(responseDispose.getMsg());
                return;
            }
            System.out.println("Container recreated with success!");
        }
    }
}

```

You define service responses using the `org.kie.server.api.model.ServiceResponse<T>` object, where **T** represents the type of returned response. The **ServiceResponse** object has the following attributes:

- **String message**: Returns the response message
- **ResponseType type**: Returns either **SUCCESS** or **FAILURE**
- **T result**: Returns the requested object



In this example, when you dispose a container, the **ServiceResponse** returns a **Void** response. When you create a container, the **ServiceResponse** returns a **KieContainerResource** object.



## NOTE

A conversation between a client and a specific Decision Server container in a clustered environment is secured by a unique **conversationID**. The **conversationID** is transferred using the **X-KIE-ConversationId** REST header. If you update the container, unset the previous **conversationID**. Use **KieServicesClient.completeConversation()** to unset the **conversationID** for Java API.

- Run the configured **.java** class from your project directory to execute the request, and review the Decision Server response.  
If you enabled debug logging, Decision Server responds with a detailed response according to your configured marshalling format, such as JSON.

Example server response for a new KIE container (log):

```
10:23:35.194 [main] INFO o.k.s.a.m.MarshallerFactory - Marshaller
extensions init
10:23:35.396 [main] DEBUG o.k.s.client.balancer.LoadBalancer - Load
balancer RoundRobinBalancerStrategy{availableEndpoints=
[http://localhost:8080/kie-server/services/rest/server]} selected
url 'http://localhost:8080/kie-server/services/rest/server'
10:23:35.398 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl -
About to send GET request to 'http://localhost:8080/kie-
server/services/rest/server'
10:23:35.440 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl -
About to deserialize content:
  {
    "type" : "SUCCESS",
    "msg" : "Kie Server info",
    "result" : {
      "kie-server-info" : {
        "id" : "default-kieserver",
        "version" : "7.11.0.Final-redhat-00003",
        "name" : "default-kieserver",
        "location" : "http://localhost:8080/kie-
server/services/rest/server",
        "capabilities" : [ "KieServer", "BRM", "BPM", "CaseMgmt",
"BPM-UI", "BRP", "DMN", "Swagger" ],
        "messages" : [ {
          "severity" : "INFO",
          "timestamp" : {
            "java.util.Date" : 1540814906533
          }
        }
      ],
      "content" : [ "Server KieServerInfo{serverId='default-
kieserver', version='7.11.0.Final-redhat-00003', name='default-
kieserver', location='http://localhost:8080/kie-
server/services/rest/server', capabilities=[KieServer, BRM, BPM,
CaseMgmt, BPM-UI, BRP, DMN, Swagger], messages=null}started
successfully at Mon Oct 29 08:08:26 EDT 2018" ]
    } ]
```

```

    }
  }
}'
  into type: 'class org.kie.server.api.model.ServiceResponse'
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
KieServicesClient connected to: default-kieserver version
7.11.0.Final-redhat-00003
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Supported capabilities by the server: [KieServer, BRM, BPM,
CaseMgmt, BPM-UI, BRP, DMN, Swagger]
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Building services client for server capability KieServer
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - No
builder found for 'KieServer' capability
10:23:35.654 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Building services client for server capability BRM
10:23:35.654 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Builder
'org.kie.server.client.helper.DroolsServicesClientBuilder@6b927fb'
for capability 'BRM'
10:23:35.655 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Capability implemented by {interface
org.kie.server.client.RuleServicesClient=org.kie.server.client.impl.
RuleServicesClientImpl@4a94ee4}
10:23:35.655 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Building services client for server capability BPM
10:23:35.656 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Builder
'org.kie.server.client.helper.JBPMServicesClientBuilder@4cc451f2'
for capability 'BPM'
10:23:35.672 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Capability implemented by {interface
org.kie.server.client.JobServicesClient=org.kie.server.client.impl.J
obServicesClientImpl@1189dd52, interface
org.kie.server.client.admin.ProcessAdminServicesClient=org.kie.serv
er.client.admin.impl.ProcessAdminServicesClientImpl@36bc55de,
interface
org.kie.server.client.DocumentServicesClient=org.kie.server.client.i
mpl.DocumentServicesClientImpl@564fab8c, interface
org.kie.server.client.admin.UserTaskAdminServicesClient=org.kie.serv
er.client.admin.impl.UserTaskAdminServicesClientImpl@16d04d3d,
interface
org.kie.server.client.QueryServicesClient=org.kie.server.client.impl
.QueryServicesClientImpl@49ec71f8, interface
org.kie.server.client.ProcessServicesClient=org.kie.server.client.im
pl.ProcessServicesClientImpl@1d2adfbc, interface
org.kie.server.client.UserTaskServicesClient=org.kie.server.client.i
mpl.UserTaskServicesClientImpl@36902638}
10:23:35.672 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Building services client for server capability CaseMgmt
10:23:35.672 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Builder
'org.kie.server.client.helper.CaseServicesClientBuilder@223d2c72'
for capability 'CaseMgmt'
10:23:35.676 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Capability implemented by {interface

```

```

org.kie.server.client.admin.CaseAdminServicesClient=org.kie.server.c
lient.admin.impl.CaseAdminServicesClientImpl@2b662a77, interface
org.kie.server.client.CaseServicesClient=org.kie.server.client.impl.
CaseServicesClientImpl@7f0eb4b4}
10:23:35.676 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Building services client for server capability BPM-UI
10:23:35.676 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Builder
'org.kie.server.client.helper.JBPMUIServicesClientBuilder@5c33f1a9'
for capability 'BPM-UI'
10:23:35.677 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Capability implemented by {interface
org.kie.server.client.UIServicesClient=org.kie.server.client.impl.UI
ServicesClientImpl@223191a6}
10:23:35.678 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Building services client for server capability BRP
10:23:35.678 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Builder
'org.kie.server.client.helper.OptaplannerServicesClientBuilder@49139
829' for capability 'BRP'
10:23:35.679 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Capability implemented by {interface
org.kie.server.client.SolverServicesClient=org.kie.server.client.imp
l.SolverServicesClientImpl@77fbd92c}
10:23:35.679 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Building services client for server capability DMN
10:23:35.679 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Builder
'org.kie.server.client.helper.DMNServicesClientBuilder@67c27493' for
capability 'DMN'
10:23:35.680 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Capability implemented by {interface
org.kie.server.client.DMNServicesClient=org.kie.server.client.impl.D
MNServicesClientImpl@35e2d654}
10:23:35.680 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl -
Building services client for server capability Swagger
10:23:35.680 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - No
builder found for 'Swagger' capability
10:23:35.681 [main] DEBUG o.k.s.c.client.balancer.LoadBalancer - Load
balancer RoundRobinBalancerStrategy{availableEndpoints=
[http://localhost:8080/kie-server/services/rest/server]} selected
url 'http://localhost:8080/kie-server/services/rest/server'
10:23:35.701 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl -
About to send PUT request to 'http://localhost:8080/kie-
server/services/rest/server/containers/employee-rostering3' with
payload '{
  "container-id" : null,
  "release-id" : {
    "group-id" : "employee-rostering",
    "artifact-id" : "employee-rostering",
    "version" : "1.0.0-SNAPSHOT"
  },
  "resolved-release-id" : null,
  "status" : null,
  "scanner" : null,
  "config-items" : [ ],

```

```

    "messages" : [ ],
    "container-alias" : null
  }'
10:23:38.071 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl -
About to deserialize content:
  '{
    "type" : "SUCCESS",
    "msg" : "Container employee-rostering3 successfully deployed with
module employeerostering:employeerostering:1.0.0-SNAPSHOT.",
    "result" : {
      "kie-container" : {
        "container-id" : "employee-rostering3",
        "release-id" : {
          "group-id" : "employeerostering",
          "artifact-id" : "employeerostering",
          "version" : "1.0.0-SNAPSHOT"
        },
        "resolved-release-id" : {
          "group-id" : "employeerostering",
          "artifact-id" : "employeerostering",
          "version" : "1.0.0-SNAPSHOT"
        },
        "status" : "STARTED",
        "scanner" : {
          "status" : "DISPOSED",
          "poll-interval" : null
        },
        "config-items" : [ ],
        "messages" : [ {
          "severity" : "INFO",
          "timestamp" : {
            "java.util.Date" : 1540909418069
          }
        } ],
        "content" : [ "Container employee-rostering3 successfully
created with module employeerostering:employeerostering:1.0.0-
SNAPSHOT." ]
      } ],
      "container-alias" : null
    }
  }'
into type: 'class org.kie.server.api.model.ServiceResponse'

```

If you encounter request errors, review the returned error code messages and adjust your Java configurations accordingly.

## 2.2. SUPPORTED DECISION SERVER JAVA CLIENTS

The following are some of the Java client services available in the `org.kie.server.client` package of your Red Hat Decision Manager distribution. You can use these services to interact with related resources in Decision Server similarly to the Decision Server REST API.

- **KieServicesClient**: Used as the entry point for other Decision Server Java clients, and used to interact with KIE containers
- **JobServicesClient**: Used to schedule, cancel, re-queue, and get job requests

- **RuleServicesClient**: Used to send commands to the server to perform rule-related operations, such as executing rules or inserting objects into the KIE session
- **SolverServicesClient**: Used to perform all Red Hat Business Optimizer operations, such as getting the solver state and the best solution, or disposing a solver

The `getServicesClient` method provides access to any of these clients:

```
RuleServicesClient rulesClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
```

For the full list of available Decision Server Java clients, download the **Red Hat Decision Manager 7.2.0 Source Distribution** from the [Red Hat Customer Portal](#) and navigate to `~/rhd-7.2.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/server/client`.

## 2.3. EXAMPLE REQUESTS WITH THE DECISION SERVER JAVA CLIENT API

The following are examples of Decision Server Java client API requests for basic interactions with Decision Server. For the full list of available Decision Server Java clients, download the **Red Hat Decision Manager 7.2.0 Source Distribution** from the [Red Hat Customer Portal](#) and navigate to `~/rhd-7.2.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/server/client`.

### Listing Decision Server capabilities

You can use the `org.kie.server.api.model.KieServerInfo` object to identify server capabilities. The `KieServicesClient` client requires the server capability information to correctly produce service clients. You can specify the capabilities globally in `KieServicesConfiguration`; otherwise they are automatically retrieved from Decision Server.

### Example request to return Decision Server capabilities

```
public void listCapabilities() {
    KieServerInfo serverInfo =
kieServicesClient.getServerInfo().getResult();
    System.out.print("Server capabilities:");

    for (String capability : serverInfo.getCapabilities()) {
        System.out.print(" " + capability);
    }

    System.out.println();
}
```

### Listing KIE containers in Decision Server

KIE containers are represented by the `org.kie.server.api.model.KieContainerResource` object. The list of resources is represented by the `org.kie.server.api.model.KieContainerResourceList` object.

## Example request to return KIE containers from Decision Server

```
public void listContainers() {
    KieContainerResourceList containersList =
kieServicesClient.listContainers().getResult();
    List<KieContainerResource> kieContainers =
containersList.getContainers();
    System.out.println("Available containers: ");
    for (KieContainerResource container : kieContainers) {
        System.out.println("\t" + container.getContainerId() + " (" +
container.getReleaseId() + ")");
    }
}
```

You can optionally filter the KIE container results using an instance of the `org.kie.server.api.model.KieContainerResourceFilter` class, which is passed to the `org.kie.server.client.KieServicesClient.listContainers()` method.

## Example request to return KIE containers by release ID and status

```
public void listContainersWithFilter() {

    // Filter containers by releaseId
    "org.example:container:1.0.0.Final" and status FAILED
    KieContainerResourceFilter filter = new
KieContainerResourceFilter.Builder()
        .releaseId("org.example", "container", "1.0.0.Final")
        .status(KieContainerStatus.FAILED)
        .build();

    // Using previously created KieServicesClient
    KieContainerResourceList containersList =
kieServicesClient.listContainers(filter).getResult();
    List<KieContainerResource> kieContainers =
containersList.getContainers();

    System.out.println("Available containers: ");

    for (KieContainerResource container : kieContainers) {
        System.out.println("\t" + container.getContainerId() + " (" +
container.getReleaseId() + ")");
    }
}
```

## Creating and disposing KIE containers in Decision Server

You can use the `createContainer` and `disposeContainer` methods in the `KieServicesClient` client to dispose and create KIE containers. In this example, when you dispose a container, the `ServiceResponse` returns a `Void` response. When you create a container, the `ServiceResponse` returns a `KieContainerResource` object.

## Example request to dispose and re-create a KIE container

```
public void disposeAndCreateContainer() {
    System.out.println("== Disposing and creating containers ==");
```

```

    // Retrieve list of KIE containers
    List<KieContainerResource> kieContainers =
kieServicesClient.listContainers().getResult().getContainers();
    if (kieContainers.size() == 0) {
        System.out.println("No containers available...");
        return;
    }

    // Dispose KIE container
    KieContainerResource container = kieContainers.get(0);
    String containerId = container.getContainerId();
    ServiceResponse<Void> responseDispose =
kieServicesClient.disposeContainer(containerId);
    if (responseDispose.getType() == ResponseType.FAILURE) {
        System.out.println("Error disposing " + containerId + ".
Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Success Disposing container " + containerId);
    System.out.println("Trying to recreate the container...");

    // Re-create KIE container
    ServiceResponse<KieContainerResource> createResponse =
kieServicesClient.createContainer(containerId, container);
    if (createResponse.getType() == ResponseType.FAILURE) {
        System.out.println("Error creating " + containerId + ". Message:
");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Container recreated with success!");
}
}

```

### Executing runtime commands in Decision Server

Red Hat Decision Manager supports runtime commands that you can send to Decision Server for asset-related operations, such as inserting or retracting objects in a KIE session or firing all rules. The full list of supported runtime commands is located in the **org.drools.core.command.runtime** package in your Red Hat Decision Manager instance. You can use the **org.kie.api.command.KieCommands** class to insert commands, and use **org.kie.api.KieServices.get().getCommands()** to instantiate the **KieCommands** class. If you want to add multiple commands, use the **BatchExecutionCommand** wrapper.

### Example request to insert an object and fire all rules

```

import org.kie.api.command.Command;
import org.kie.api.command.KieCommands;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.RuleServicesClient;
import org.kie.server.client.KieServicesClient;
import org.kie.api.KieServices;

import java.util.Arrays;

```

```

...

public void executeCommands() {

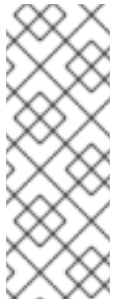
    String containerId = "hello";
    System.out.println("== Sending commands to the server ==");
    RuleServicesClient rulesClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
    KieCommands commandsFactory = KieServices.Factory.get().getCommands();

    Command<?> insert = commandsFactory.newInsert("Some String OBJ");
    Command<?> fireAllRules = commandsFactory.newFireAllRules();
    Command<?> batchCommand =
commandsFactory.newBatchExecution(Arrays.asList(insert, fireAllRules));

    ServiceResponse<String> executeResponse =
rulesClient.executeCommands(containerId, batchCommand);

    if(executeResponse.getType() == ResponseType.SUCCESS) {
        System.out.println("Commands executed with success! Response: ");
        System.out.println(executeResponse.getResult());
    } else {
        System.out.println("Error executing rules. Message: ");
        System.out.println(executeResponse.getMsg());
    }
}
}

```



#### NOTE

A conversation between a client and a specific Decision Server container in a clustered environment is secured by a unique **conversationID**. The **conversationID** is transferred using the **X-KIE-ConversationId** REST header. If you update the container, unset the previous **conversationID**. Use **KieServicesClient.completeConversation()** to unset the **conversationID** for Java API.



## CHAPTER 3. DECISION SERVER AND KIE CONTAINER COMMANDS IN RED HAT DECISION MANAGER

Red Hat Decision Manager supports server commands that you can send to Decision Server for server-related or container-related operations, such as retrieving server information or creating or deleting a container. The full list of supported Decision Server configuration commands is located in the `org.kie.server.api.commands` package in your Red Hat Decision Manager instance.

In the Decision Server REST API, you use the `org.kie.server.api.commands` commands as the request body for **POST** requests to `http://SERVER:PORT/kie-server/services/rest/server/config`. For more information about using the Decision Server REST API, see [Chapter 1, Decision Server REST API for KIE containers and business assets](#).

In the Decision Server Java client API, you use the corresponding method in the parent `KieServicesClient` Java client as an embedded API request in your Java application. All Decision Server commands are executed by methods provided in the Java client API, so you do not need to embed the actual Decision Server commands in your Java application. For more information about using the Decision Server Java client API, see [Chapter 2, Decision Server Java client API for KIE containers and business assets](#).

### 3.1. SAMPLE DECISION SERVER AND KIE CONTAINER COMMANDS

The following are sample Decision Server commands that you can use with the Decision Server REST API or Java client API for server-related or container-related operations in Decision Server:

- `GetServerInfoCommand`
- `GetServerStateCommand`
- `CreateContainerCommand`
- `GetContainerInfoCommand`
- `ListContainersCommand`
- `CallContainerCommand`
- `DisposeContainerCommand`
- `GetScannerInfoCommand`
- `UpdateScannerCommand`
- `UpdateReleaseIdCommand`

For the full list of supported Decision Server configuration and management commands, see the `org.kie.server.api.commands` package in your Red Hat Decision Manager instance.

You can run Decision Server commands individually or together as a batch REST API request or batch Java API request:

#### Batch REST API request to create, call, and dispose a KIE container (JSON)

```
{
```

```

"commands": [
  {
    "create-container": {
      "container": {
        "status": "STARTED",
        "container-id": "command-script-container",
        "release-id": {
          "version": "1.0",
          "group-id": "com.redhat",
          "artifact-id": "Project1"
        }
      }
    }
  },
  {
    "call-container": {
      "payload": "{\n  \"commands\" : [ {\n    \"fire-all-rules\" : {\n      \"max\" : -1,\n      \"out-identifier\" : null\n    } ]\n}",
      "container-id": "command-script-container"
    }
  },
  {
    "dispose-container": {
      "container-id": "command-script-container"
    }
  }
]
}

```

### Batch Java API request to retrieve, dispose, and re-create a KIE container

```

public void disposeAndCreateContainer() {
    System.out.println("== Disposing and creating containers ==");

    // Retrieve list of KIE containers
    List<KieContainerResource> kieContainers =
kieServicesClient.listContainers().getResult().getContainers();
    if (kieContainers.size() == 0) {
        System.out.println("No containers available...");
        return;
    }

    // Dispose KIE container
    KieContainerResource container = kieContainers.get(0);
    String containerId = container.getContainerId();
    ServiceResponse<Void> responseDispose =
kieServicesClient.disposeContainer(containerId);
    if (responseDispose.getType() == ResponseType.FAILURE) {
        System.out.println("Error disposing " + containerId + ". Message:
");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Success Disposing container " + containerId);
    System.out.println("Trying to recreate the container...");
}

```

```

    // Re-create KIE container
    ServiceResponse<KieContainerResource> createResponse =
kieServicesClient.createContainer(containerId, container);
    if(createResponse.getType() == ResponseType.FAILURE) {
        System.out.println("Error creating " + containerId + ". Message:
");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Container recreated with success!");
}

```

Each command in this section includes a REST request body example (JSON) for the Decision Server REST API and an embedded method example from the **KieServicesClient** Java client for the Decision Server Java client API.

### GetServerInfoCommand

Returns information about the Decision Server.

#### Example REST request body (JSON)

```

{
  "commands" : [ {
    "get-server-info" : { }
  } ]
}

```

#### Example Java client method

```

KieServerInfo serverInfo = kieServicesClient.getServerInfo();

```

#### Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Kie Server info",
      "result": {
        "kie-server-info": {
          "id": "default-kieserver",
          "version": "7.11.0.Final-redhat-00001",
          "name": "default-kieserver",
          "location": "http://localhost:8080/kie-
server/services/rest/server",
          "capabilities": [
            "KieServer",
            "BRM",
            "BPM",
            "CaseMgmt",
            "BPM-UI",
            "BRP",
            "DMN",
            "Swagger"
          ]
        }
      }
    }
  ]
}

```



```

        "itemValue": "http://localhost:8080/kie-
server/services/rest/server",
        "itemType": "java.lang.String"
    },
    {
        "itemName": "org.kie.server.controller.user",
        "itemValue": "controllerUser",
        "itemType": "java.lang.String"
    },
    {
        "itemName": "org.kie.server.controller",
        "itemValue": "http://localhost:8080/decision-
central/rest/controller",
        "itemType": "java.lang.String"
    }
]
},
"containers": [
    {
        "container-id": "employee-rostering",
        "release-id": {
            "group-id": "employeerostering",
            "artifact-id": "employeerostering",
            "version": "1.0.0-SNAPSHOT"
        },
        "resolved-release-id": null,
        "status": "STARTED",
        "scanner": {
            "status": "STOPPED",
            "poll-interval": null
        },
        "config-items": [
            {
                "itemName": "KBase",
                "itemValue": "",
                "itemType": "BPM"
            },
            {
                "itemName": "KSession",
                "itemValue": "",
                "itemType": "BPM"
            },
            {
                "itemName": "MergeMode",
                "itemValue": "MERGE_COLLECTIONS",
                "itemType": "BPM"
            },
            {
                "itemName": "RuntimeStrategy",
                "itemValue": "SINGLETON",
                "itemType": "BPM"
            }
        ],
        "messages": [],
        "container-alias": "employeerostering"
    }
}

```

```

    ]
  }
}
]
}

```

## CreateContainerCommand

Creates a KIE container in the Decision Server.

**Table 3.1. Command attributes**

Name	Description	Requirement
<b>container</b>	Map containing the <b>container-id</b> , <b>release-id</b> data (group ID, artifact ID, version), <b>status</b> , and any other components of the new KIE container	Required

## Example REST request body (JSON)

```

{
  "commands" : [ {
    "create-container" : {
      "container" : {
        "status" : null,
        "messages" : [ ],
        "container-id" : "command-script-container",
        "release-id" : {
          "version" : "1.0",
          "group-id" : "com.redhat",
          "artifact-id" : "Project1"
        },
        "config-items" : [ ]
      }
    }
  } ]
}

```

## Example Java client method

```

ServiceResponse<KieContainerResource> response =
kieServicesClient.createContainer("command-script-container", resource);

```

## Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully deployed
with module com.redhat:Project1:1.0.",

```

```

"result": {
  "kie-container": {
    "container-id": "command-script-container",
    "release-id": {
      "version" : "1.0",
      "group-id" : "com.redhat",
      "artifact-id" : "Project1"
    },
    "resolved-release-id": {
      "version" : "1.0",
      "group-id" : "com.redhat",
      "artifact-id" : "Project1"
    },
    "status": "STARTED",
    "scanner": {
      "status": "DISPOSED",
      "poll-interval": null
    },
    "config-items": [],
    "messages": [
      {
        "severity": "INFO",
        "timestamp": {
          "java.util.Date": 1538762455510
        },
        "content": [
          "Container command-script-container successfully
created with module com.redhat:Project1:1.0."
        ]
      }
    ],
    "container-alias": null
  }
}
]
}

```

### GetContainerInfoCommand

Returns information about a specified KIE container in Decision Server.

**Table 3.2. Command attributes**

Name	Description	Requirement
<code>container-id</code>	ID of the KIE container	Required

### Example REST request body (JSON)

```

{
  "commands" : [ {
    "get-container-info" : {
      "container-id" : "command-script-container"
    }
  }
]
}

```

```

    }
  } ]
}

```

### Example Java client method

```

ServiceResponse<KieContainerResource> response =
kieServicesClient.getContainerInfo("command-script-container");

```

### Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Info for container command-script-container",
      "result": {
        "kie-container": {
          "container-id": "command-script-container",
          "release-id": {
            "group-id": "com.redhat",
            "artifact-id": "Project1",
            "version": "1.0"
          },
          "resolved-release-id": {
            "group-id": "com.redhat",
            "artifact-id": "Project1",
            "version": "1.0"
          },
          "status": "STARTED",
          "scanner": {
            "status": "DISPOSED",
            "poll-interval": null
          },
          "config-items": [
            ],
          "container-alias": null
        }
      }
    }
  ]
}

```

### ListContainersCommand

Returns a list of KIE containers that have been created in the Decision Server.

**Table 3.3. Command attributes**

Name	Description	Requirement
------	-------------	-------------



Name	Description	Requirement
<b>kie-container-filter</b>	Optional map containing <b>release-id-filter</b> , <b>container-status-filter</b> , and any other KIE container properties by which you want to filter results	Optional

### Example REST request body (JSON)

```
{
  "commands" : [ {
    "list-containers" : {
      "kie-container-filter" : {
        "release-id-filter" : { },
        "container-status-filter" : {
          "accepted-status" : ["FAILED"]
        }
      }
    }
  } ]
}
```

### Example Java client method

```
KieContainerResourceFilter filter = new
KieContainerResourceFilter.Builder()
    .status(KieContainerStatus.FAILED)
    .build();

KieContainerResourceList containersList =
kieServicesClient.listContainers(filter);
```

### Example server response (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "List of created containers",
      "result": {
        "kie-containers": {
          "kie-container": [
            {
              "container-id": "command-script-container",
              "release-id": {
                "group-id": "com.redhat",
                "artifact-id": "Project1",
                "version": "1.0"
              },
              "resolved-release-id": {
```



Calls a KIE container and executes one or more runtime commands. For information about Red Hat Decision Manager runtime commands, see [Chapter 4, Runtime commands in Red Hat Decision Manager](#).

**Table 3.4. Command attributes**

Name	Description	Requirement
<b>container-id</b>	ID of the KIE container to be called	Required
<b>payload</b>	One or more commands in a <b>BatchExecutionCommand</b> wrapper to be executed on the KIE container	Required

### Example REST request body (JSON)

```
{
  "commands" : [ {
    "call-container" : {
      "payload" : "{\n  \"lookup\" : \"defaultKieSession\", \n
\"commands\" : [ {\n    \"fire-all-rules\" : {\n      \"max\" : -1, \n
\"out-identifier\" : null\n    } \n  } ] \n}",
      "container-id" : "command-script-container"
    }
  } ]
}
```

### Example Java client method

```
List<Command<?>> commands = new ArrayList<Command<?>>();
BatchExecutionCommand batchExecution1 =
commandsFactory.newBatchExecution(commands, "defaultKieSession");
commands.add(commandsFactory.newFireAllRules());

ServiceResponse<ExecutionResults> response1 =
ruleClient.executeCommandsWithResults("command-script-container",
batchExecution1);
```

### Example server response (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": "{\n  \"results\" : [ ], \n  \"facts\" : [ ] \n}"
    }
  ]
}
```

### DisposeContainerCommand

Disposes a specified KIE container in the Decision Server.

**Table 3.5. Command attributes**

Name	Description	Requirement
<code>container-id</code>	ID of the KIE container to be disposed	Required

**Example REST request body (JSON)**

```
{
  "commands" : [ {
    "dispose-container" : {
      "container-id" : "command-script-container"
    }
  } ]
}
```

**Example Java client method**

```
ServiceResponse<Void> response =
kieServicesClient.disposeContainer("command-script-container");
```

**Example server response (JSON)**

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully
disposed.",
      "result": null
    }
  ]
}
```

**GetScannerInfoCommand**

Returns information about the KIE scanner used for automatic updates in a specified KIE container, if applicable.

**Table 3.6. Command attributes**

Name	Description	Requirement
<code>container-id</code>	ID of the KIE container where the KIE scanner is used	Required

**Example REST request body (JSON)**

```
{
  "commands" : [ {
    "get-scanner-info" : {
```

```

        "container-id" : "command-script-container"
      }
    } ]
  }

```

### Example Java client method

```

ServiceResponse<KieScannerResource> response =
kieServicesClient.getScannerInfo("command-script-container");

```

### Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Scanner info successfully retrieved",
      "result": {
        "kie-scanner": {
          "status": "DISPOSED",
          "poll-interval": null
        }
      }
    }
  ]
}

```

### UpdateScannerCommand

Starts or stops a KIE scanner that controls polling for updated KIE container deployments.



#### NOTE

Avoid using a KIE scanner with business processes. Using a KIE scanner with processes can lead to unforeseen updates that can then cause errors in long-running processes when changes are not compatible with running process instances.

**Table 3.7. Command attributes**

Name	Description	Requirement
<b>container-id</b>	ID of the KIE container where the KIE scanner is used	Required
<b>status</b>	Status to be set on the KIE scanner ( <b>STARTED</b> , <b>STOPPED</b> )	Required
<b>poll-interval</b>	Permitted polling duration in milliseconds	Required only when starting scanner

### Example REST request body (JSON)

```

{
  "commands" : [ {
    "update-scanner" : {
      "scanner" : {
        "status" : "STARTED",
        "poll-interval" : 10000
      },
      "container-id" : "command-script-container"
    }
  } ]
}

```

### Example Java client method

```

KieScannerResource scannerResource = new KieScannerResource();
scannerResource.setPollInterval(10000);
scannerResource.setStatus(KieScannerStatus. STARTED);

ServiceResponse<KieScannerResource> response =
kieServicesClient.updateScanner("command-script-container",
scannerResource);

```

### Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Kie scanner successfully created.",
      "result": {
        "kie-scanner": {
          "status": "STARTED",
          "poll-interval": 10000
        }
      }
    }
  ]
}

```

### UpdateReleaseIdCommand

Updates the release ID data (group ID, artifact ID, version) for a specified KIE container.

**Table 3.8. Command attributes**

Name	Description	Requirement
<b>container-id</b>	ID of the KIE container to be updated	Required
<b>releaseId</b>	Updated GAV (group ID, artifact ID, version) data to be applied to the KIE container	Required

### Example REST request body (JSON)

```

{
  "commands" : [ {
    "update-release-id" : {
      "releaseId" : {
        "version" : "1.1",
        "group-id" : "com.redhat",
        "artifact-id" : "Project1"
      },
      "container-id" : "command-script-container"
    }
  } ]
}

```

### Example Java client method

```

ServiceResponse<ReleaseId> response =
kieServicesClient.updateReleaseId("command-script-container",
"com.redhat:Project1:1.1");

```

### Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Release id successfully updated",
      "result": {
        "release-id": {
          "group-id": "com.redhat",
          "artifact-id": "Project1",
          "version": "1.1"
        }
      }
    }
  ]
}

```

## CHAPTER 4. RUNTIME COMMANDS IN RED HAT DECISION MANAGER

Red Hat Decision Manager supports runtime commands that you can send to Decision Server for asset-related operations, such as executing all rules or inserting or retracting objects in a KIE session. The full list of supported runtime commands is located in the `org.drools.core.command.runtime` package in your Red Hat Decision Manager instance.

In the Decision Server REST API, you use the global `org.drools.core.command.runtime` commands or the rule-specific `org.drools.core.command.runtime.rule` commands as the request body for **POST** requests to `http://SERVER:PORT/kie-server/services/rest/server/containers/instances/{containerId}`. For more information about using the Decision Server REST API, see [Chapter 1, Decision Server REST API for KIE containers and business assets](#).

In the Decision Server Java client API, you can embed these commands in your Java application along with the relevant Java client. For example, for rule-related commands, you use the `RuleServicesClient` Java client with the embedded commands. For more information about using the Decision Server Java client API, see [Chapter 2, Decision Server Java client API for KIE containers and business assets](#).

### 4.1. SAMPLE RUNTIME COMMANDS IN RED HAT DECISION MANAGER

The following are sample runtime commands that you can use with the Decision Server REST API or Java client API for asset-related operations in Decision Server:

- `BatchExecutionCommand`
- `InsertObjectCommand`
- `RetractCommand`
- `ModifyCommand`
- `GetObjectCommand`
- `GetObjectsCommand`
- `InsertElementsCommand`
- `FireAllRulesCommand`
- `QueryCommand`
- `SetGlobalCommand`
- `GetGlobalCommand`

For the full list of supported runtime commands, see the `org.drools.core.command.runtime` package in your Red Hat Decision Manager instance.

Each command in this section includes a REST request body example (JSON) for the Decision Server REST API and an embedded Java command example for the Decision Server Java client API. The Java examples use an object `org.drools.compiler.test.Person` with the fields `name` (String) and `age`



(Integer).

### BatchExecutionCommand

Contains multiple commands to be executed together.

**Table 4.1. Command attributes**

Name	Description	Requirement
<b>commands</b>	List of commands to be executed.	Required
<b>lookup</b>	Sets the KIE session ID on which the commands will be executed. For stateless KIE sessions, this attribute is required. For stateful KIE sessions, this attribute is optional and if not specified, the default KIE session is used.	Required for stateless KIE session, optional for stateful KIE session

### Example JSON request body

```
{
  "lookup": "ksession1",
  "commands": [ {
    "insert": {
      "object": {
        "org.drools.compiler.test.Person": {
          "name": "john",
          "age": 25
        }
      }
    }
  },
  {
    "fire-all-rules": {
      "max": 10,
      "out-identifier": "firedActivations"
    }
  }
]
}
```

### Example Java command

```
BatchExecutionCommand command = new BatchExecutionCommand();
command.setLookup("ksession1");

InsertObjectCommand insertObjectCommand = new InsertObjectCommand(new
Person("john", 25));
FireAllRulesCommand fireAllRulesCommand = new FireAllRulesCommand();

command.getCommands().add(insertObjectCommand);
command.getCommands().add(fireAllRulesCommand);

ksession.execute(command);
```

## Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": 0,
              "key": "firedActivations"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}

```

### InsertObjectCommand

Inserts an object into the KIE session.

**Table 4.2. Command attributes**

Name	Description	Requirement
<b>object</b>	The object to be inserted	Required
<b>out-identifier</b>	ID of the <b>FactHandle</b> created from the object insertion and added to the execution results	Optional
<b>return-object</b>	Boolean to determine whether the object must be returned in the execution results (default: <b>true</b> )	Optional
<b>entry-point</b>	Entry point for the insertion	Optional

### Example JSON request body

```

{
  "commands": [ {
    "insert": {
      "entry-point": "my stream",
      "object": {
        "org.drools.compiler.test.Person": {
          "age": 25,
          "name": "john"
        }
      }
    },
    "out-identifier": "john",
  }
]
}

```

```

        "return-object": false
      }
    ]
  }

```

### Example Java command

```

Command insertObjectCommand =
  CommandFactory.newInsert(new Person("john", 25), "john", false,
    null);

ksession.execute(insertObjectCommand);

```

### Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": [
            {
              "value": {
                "org.drools.core.common.DefaultFactHandle": {
                  "external-form": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
                }
              },
              "key": "john"
            }
          ]
        }
      }
    }
  ]
}

```

### RetractCommand

Retracts an object from the KIE session.

**Table 4.3. Command attributes**

Name	Description	Requirement
<b>fact-handle</b>	The <b>FactHandle</b> associated with the object to be retracted	Required

### Example JSON request body

-

```
{
  "commands": [ {
    "retract": {
      "fact-handle": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
    }
  }
]
}
```

### Example Java command: Use FactHandleFromString

```
RetractCommand retractCommand = new RetractCommand();
retractCommand.setFactHandleFromString("123:234:345:456:567");
```

### Example Java command: Use FactHandle from inserted object

```
RetractCommand retractCommand = new RetractCommand(factHandle);
```

### Example server response (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}
```

## ModifyCommand

Modifies a previously inserted object in the KIE session.

**Table 4.4. Command attributes**

Name	Description	Requirement
<b>fact-handle</b>	The <b>FactHandle</b> associated with the object to be modified	Required
<b>setters</b>	List of setters for object modifications	Required

### Example JSON request body

```
{
```

```

    "commands": [ {
      "modify": {
        "fact-handle": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap",
        "setters": {
          "accessor": "age",
          "value": 25
        }
      }
    }
  ]
}

```

### Example Java command

```

ModifyCommand modifyCommand = new ModifyCommand(factHandle);

List<Setter> setters = new ArrayList<Setter>();
setters.add(new SetterImpl("age", "25"));

modifyCommand.setSetters(setters);

```

### Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}

```

### GetObjectCommand

Retrieves an object from a KIE session.

**Table 4.5. Command attributes**

Name	Description	Requirement
<b>fact-handle</b>	The <b>FactHandle</b> associated with the object to be retrieved	Required
<b>out-identifier</b>	ID of the <b>FactHandle</b> created from the object insertion and added to the execution results	Optional

## Example JSON request body

```
{
  "commands": [ {
    "get-object": {
      "fact-handle": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap",
      "out-identifier": "john"
    }
  }
]
```

## Example Java command

```
GetObjectCommand getObjectCommand = new GetObjectCommand();
getObjectCommand.setFactHandleFromString("123:234:345:456:567");
getObjectCommand.setOutIdentifier("john");
```

## Example server response (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": null,
              "key": "john"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

## GetObjectsCommand

Retrieves all objects from the KIE session as a collection.

**Table 4.6. Command attributes**

Name	Description	Requirement
<b>object-filter</b>	Filter for the objects returned from the KIE session	Optional
<b>out-identifier</b>	Identifier to be used in the execution results	Optional

**Example JSON request body**

```
{
  "commands": [ {
    "get-objects": {
      "out-identifier": "objects"
    }
  }
]
```

**Example Java command**

```
GetObjectsCommand getObjectsCommand = new GetObjectsCommand();
getObjectsCommand.setOutIdentifier("objects");
```

**Example server response (JSON)**

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": [
                {
                  "org.apache.xerces.dom.ElementNSImpl": "<?xml
version=\"1.0\" encoding=\"UTF-16\"?>\n<object
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:type=\"person\"><age>25</age><name>john</name>\n </object>"
                },
                {
                  "org.drools.compiler.test.Person": {
                    "name": "john",
                    "age": 25
                  }
                }
              ],
              "key": "objects"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

**InsertElementsCommand**

Inserts a list of objects into the KIE session.

Table 4.7. Command attributes

Name	Description	Requirement
<b>objects</b>	The list of objects to be inserted into the KIE session	Required
<b>out-identifier</b>	ID of the <b>FactHandle</b> created from the object insertion and added to the execution results	Optional
<b>return-object</b>	Boolean to determine whether the object must be returned in the execution results. Default value: <b>true</b> .	Optional
<b>entry-point</b>	Entry point for the insertion	Optional

### Example JSON request body

```
{
  "commands": [ {
    "insert-elements": {
      "objects": [
        {
          "containedObject": {
            "@class": "org.drools.compiler.test.Person",
            "age": 25,
            "name": "john"
          }
        },
        {
          "containedObject": {
            "@class": "Person",
            "age": 35,
            "name": "sarah"
          }
        }
      ]
    }
  }
]
```

### Example Java command

```
List<Object> objects = new ArrayList<Object>();
objects.add(new Person("john", 25));
objects.add(new Person("sarah", 35));
```

```
Command insertElementsCommand =
CommandFactory.newInsertElements(objects);
```

### Example server response (JSON)



```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": [
            {
              "value": {
                "org.drools.core.common.DefaultFactHandle": {
                  "external-form": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
                }
              },
              "key": "john"
            },
            {
              "value": {
                "org.drools.core.common.DefaultFactHandle": {
                  "external-form": "0:4:436792766:-
2127720266:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
                }
              },
              "key": "sarah"
            }
          ]
        }
      }
    }
  ]
}

```

### FireAllRulesCommand

Executes all rules in the KIE session.

**Table 4.8. Command attributes**

Name	Description	Requirement
<b>max</b>	Maximum number of rules to be executed. The default is <b>-1</b> and does not put any restriction on execution.	Optional
<b>out-identifier</b>	ID to be used for retrieving the number of fired rules in execution results.	Optional
<b>agenda-filter</b>	Agenda Filter to be used for rule execution.	Optional

### Example JSON request body

```
{
```

```

    "commands" : [ {
      "fire-all-rules": {
        "max": 10,
        "out-identifier": "firedActivations"
      }
    } ]
  }

```

### Example Java command

```

FireAllRulesCommand fireAllRulesCommand = new FireAllRulesCommand();
fireAllRulesCommand.setMax(10);
fireAllRulesCommand.setOutIdentifier("firedActivations");

```

### Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": 0,
              "key": "firedActivations"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}

```

### QueryCommand

Executes a query defined in the KIE base.

**Table 4.9. Command attributes**

Name	Description	Requirement
<b>name</b>	Query name.	Required
<b>out-identifier</b>	ID of the query results. The query results are added in the execution results with this identifier.	Optional
<b>arguments</b>	List of objects to be passed as a query parameter.	Optional

### Example JSON request body

■

```

{
  "commands": [
    {
      "query": {
        "name": "persons",
        "arguments": [],
        "out-identifier": "persons"
      }
    }
  ]
}

```

### Example Java command

```

QueryCommand queryCommand = new QueryCommand();
queryCommand.setName("persons");
queryCommand.setOutIdentifier("persons");

```

### Example server response (JSON)

```

{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [
        {
          "value": {
            "org.drools.core.runtime.rule.impl.FlatQueryResults": {
              "idFactHandleMaps": {
                "type": "LIST",
                "componentType": null,
                "element": [
                  {
                    "type": "MAP",
                    "componentType": null,
                    "element": [
                      {
                        "value": {
                          "org.drools.core.common.DisconnectedFactHandle": {
                            "id": 1,
                            "identityHashCode": 1809949690,
                            "objectHashCode": 1809949690,
                            "recency": 1,
                            "object": {
                              "org.kie.server.testing.Person": {
                                "fullname": "John Doe",
                                "age": 47
                              }
                            }
                          }
                        },
                        "entryPointId": "DEFAULT",
                        "traitType": "NON_TRAIT",
                        "external-form":

```

```

"0:1:1809949690:1809949690:1:DEFAULT:NON_TRAIT:org.kie.server.testing.Pe

```

```

rson"
    }
    },
    "key": "$person"
  }
]
},
"idResultMaps": {
  "type": "LIST",
  "componentType": null,
  "element": [
    {
      "type": "MAP",
      "componentType": null,
      "element": [
        {
          "value": {
            "org.kie.server.testing.Person": {
              "fullname": "John Doe",
              "age": 47
            }
          },
          "key": "$person"
        }
      ]
    }
  ]
},
"identifiers": {
  "type": "SET",
  "componentType": null,
  "element": [
    "$person"
  ]
},
"key": "persons"
},
"facts": []
}
}
}

```

### SetGlobalCommand

Sets an object to a global state.

**Table 4.10. Command attributes**

Name	Description	Requirement
<b>identifier</b>	ID of the global variable defined in the KIE base	Required
<b>object</b>	Object to be set into the global variable	Optional
<b>out</b>	Boolean to exclude the global variable you set from the execution results	Optional
<b>out-identifier</b>	ID of the global execution result	Optional

### Example JSON request body

```
{
  "commands": [
    {
      "set-global": {
        "identifier": "helper",
        "object": {
          "org.kie.server.testing.Person": {
            "fullname": "kyle",
            "age": 30
          }
        }
      },
      "out-identifier": "output"
    }
  ]
}
```

### Example Java command

```
SetGlobalCommand setGlobalCommand = new SetGlobalCommand();
setGlobalCommand.setIdentifier("helper");
setGlobalCommand.setObject(new Person("kyle", 30));
setGlobalCommand.setOut(true);
setGlobalCommand.setOutIdentifier("output");
```

### Example server response (JSON)

```
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [
        {
          "value": {
            "org.kie.server.testing.Person": {
              "fullname": "kyle",
              "age": 30
            }
          }
        }
      ]
    }
  }
}
```

```

    }
    },
    "key": "output"
  }
],
"facts": []
}
}
}

```

### GetGlobalCommand

Retrieves a previously defined global object.

**Table 4.11. Command attributes**

Name	Description	Requirement
<b>identifier</b>	ID of the global variable defined in the KIE base	Required
<b>out-identifier</b>	ID to be used in the execution results	Optional

### Example JSON request body

```

{
  "commands": [ {
    "get-global": {
      "identifier": "helper",
      "out-identifier": "helperOutput"
    }
  }
]
}

```

### Example Java command

```

GetGlobalCommand getGlobalCommand = new GetGlobalCommand();
getGlobalCommand.setIdentifier("helper");
getGlobalCommand.setOutIdentifier("helperOutput");

```

### Example server response (JSON)

```

{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": null,
              "key": "helperOutput"
            }
          ]
        }
      }
    }
  ]
}

```

```
}  
],  
"facts": []  
}  
}  
]  
}
```

## CHAPTER 5. DECISION MANAGER CONTROLLER REST API FOR DECISION SERVER TEMPLATES AND INSTANCES

Red Hat Decision Manager provides a Decision Manager controller REST API that you can use to interact with your Decision Server templates (configurations), Decision Server instances (remote servers), and associated KIE containers (deployment units) in Red Hat Decision Manager without using the Decision Central user interface. This API support enables you to maintain your Red Hat Decision Manager servers and resources more efficiently and optimize your integration and development with Red Hat Decision Manager.

With the Decision Manager controller REST API, you can perform the following actions:

- Retrieve information about Decision Server templates, instances, and associated KIE containers
- Update, start, or stop KIE containers associated with Decision Server templates and instances
- Create, update, or delete Decision Server templates
- Create, update, or delete Decision Server instances

Requests to the Decision Manager controller REST API require the following components:

### Authentication

The Decision Manager controller REST API requires HTTP Basic authentication or token-based authentication for the following user roles, depending on controller type:

- **rest-all** user role if you installed Decision Central and you want to use the built-in Decision Manager controller
- **kie-server** user role if you installed the headless Decision Manager controller separately from Decision Central

To view configured user roles for your Red Hat Decision Manager distribution, navigate to `~/$SERVER_HOME/standalone/configuration/application-roles.properties` and `~/application-users.properties`.

To add a user with the **kie-server** role or the **rest-all** role or both, navigate to `~/$SERVER_HOME/bin` and run the following command with the role or roles specified:

```
$ ./add-user.sh -a --user <USERNAME> --password <PASSWORD> --role kie-server,rest-all
```

To configure the **kie-server** or **rest-all** user with Decision Manager controller access, navigate to `~/$SERVER_HOME/standalone/configuration/standalone-full.xml`, uncomment the **org.kie.server** properties (if applicable), and add the controller user login credentials and controller location (if needed):

```
<property name="org.kie.server.location"
value="http://localhost:8080/kie-server/services/rest/server"/>
<property name="org.kie.server.controller"
value="http://localhost:8080/decision-central/rest/controller"/>
<property name="org.kie.server.controller.user" value="baAdmin"/>
<property name="org.kie.server.controller.pwd" value="password@1"/>
<property name="org.kie.server.id" value="default-kieserver"/>
```



For more information about user roles and Red Hat Decision Manager installation options, see [Planning a Red Hat Decision Manager installation](#).

## HTTP headers

The Decision Manager controller REST API requires the following HTTP headers for API requests:

- **Accept**: Data format accepted by your requesting client:
  - `application/json` (JSON)
  - `application/xml` (XML, for JAXB)
- **Content-Type**: Data format of your **POST** or **PUT** API request data:
  - `application/json` (JSON)
  - `application/xml` (XML, for JAXB)

## HTTP methods

The Decision Manager controller REST API supports the following HTTP methods for API requests:

- **GET**: Retrieves specified information from a specified resource endpoint
- **POST**: Updates a resource or resource instance
- **PUT**: Creates a resource or resource instance
- **DELETE**: Deletes a resource or resource instance

## Base URL

The base URL for Decision Manager controller REST API requests is

`http://SERVER:PORT/CONTROLLER/rest/`, such as `http://localhost:8080/decision-central/rest/` if you are using the Decision Manager controller built in to Decision Central.

## Endpoints

Decision Manager controller REST API endpoints, such as `/controller/management/servers/{serverId}` for a specified Decision Server template, are the URIs that you append to the Decision Manager controller REST API base URL to access the corresponding server resource or type of server resource in Red Hat Decision Manager.

### Example request URL for `/controller/management/servers/{serverId}` endpoint

`http://localhost:8080/decision-central/rest/controller/management/servers/default-kieserver`

## Request parameters and request data

Some Decision Manager controller REST API requests require specific parameters in the request URL path to identify or filter specific resources and to perform specific actions. You can append URL parameters to the endpoint in the format `?<PARAM>=<VALUE>&<PARAM>=<VALUE>`.

### Example DELETE request URL with parameters

`http://localhost:8080/decision-central/rest/controller/server/new-kieserver-instance?location=http://localhost:8080/kieserver/services/rest/server`

HTTP **POST** and **PUT** requests may additionally require a request body or file with data to accompany the request.

### Example PUT request URL and JSON request body data

`http://localhost:8080/decision-central/rest/controller/management/servers/new-kieserver`

```
{
  "server-id": "new-kieserver",
  "server-name": "new-kieserver",
  "container-specs": [],
  "server-config": {},
  "capabilities": [
    "RULE",
    "PROCESS",
    "PLANNING"
  ]
}
```

## 5.1. SENDING REQUESTS WITH THE DECISION MANAGER CONTROLLER REST API USING A REST CLIENT OR CURL UTILITY

The Decision Manager controller REST API enables you to interact with your Decision Server templates (configurations), Decision Server instances (remote servers), and associated KIE containers (deployment units) in Red Hat Decision Manager without using the Decision Central user interface. You can send Decision Manager controller REST API requests using any REST client or curl utility.

### Prerequisites

- Decision Server is installed and running.
- The Decision Manager controller or headless Decision Manager controller is installed and running.
- You have **rest-all** user role access to the Decision Manager controller if you installed Decision Central, or **kie-server** user role access to the headless Decision Manager controller installed separately from Decision Central.

### Procedure

1. Identify the relevant [API endpoint](#) to which you want to send a request, such as **[GET] /controller/management/servers** to retrieve Decision Server templates from the Decision Manager controller.
2. In a REST client or curl utility, enter the following components for a **GET** request to **controller/management/servers**. Adjust any request details according to your use case.  
For REST client:
  - **Authentication:** Enter the user name and password of the Decision Manager controller user with the **rest-all** role or the headless Decision Manager controller user with the **kie-server** role.

- **HTTP Headers:** Set the following header:
  - **Accept:** `application/json`
- **HTTP method:** Set to **GET**.
- **URL:** Enter the Decision Manager controller REST API base URL and endpoint, such as `http://localhost:8080/decision-central/rest/controller/management/servers`.

For curl utility:

- **-u:** Enter the user name and password of the Decision Manager controller user with the **rest-all** role or the headless Decision Manager controller user with the **kie-server** role.
- **-H:** Set the following header:
  - **accept:** `application/json`
- **-X:** Set to **GET**.
- **URL:** Enter the Decision Manager controller REST API base URL and endpoint, such as `http://localhost:8080/decision-central/rest/controller/management/servers`.

```
curl -u 'baAdmin:password@1' -H "accept: application/json" -X GET
"http://localhost:8080/decision-
central/rest/controller/management/servers"
```

3. Execute the request and review the Decision Manager controller response.  
Example server response (JSON):

```
{
  "server-template": [
    {
      "server-id": "default-kieserver",
      "server-name": "default-kieserver",
      "container-specs": [
        {
          "container-id": "employeerostering_1.0.0-SNAPSHOT",
          "container-name": "employeerostering",
          "server-template-key": {
            "server-id": "default-kieserver",
            "server-name": "default-kieserver"
          },
          "release-id": {
            "group-id": "employeerostering",
            "artifact-id": "employeerostering",
            "version": "1.0.0-SNAPSHOT"
          },
          "configuration": {
            "RULE": {
              "org.kie.server.controller.api.model.spec.RuleConfig": {
                "pollInterval": null,
                "scannerStatus": "STOPPED"
              }
            }
          }
        }
      ]
    }
  ]
}
```

```

        }
      },
      "PROCESS": {
"org.kie.server.controller.api.model.spec.ProcessConfig": {
    "runtimeStrategy": "SINGLETON",
    "kbase": "",
    "ksession": "",
    "mergeMode": "MERGE_COLLECTIONS"
  }
},
"status": "STARTED"
},
{
  "container-id": "mortgage-process_1.0.0-SNAPSHOT",
  "container-name": "mortgage-process",
  "server-template-key": {
    "server-id": "default-kieserver",
    "server-name": "default-kieserver"
  },
  "release-id": {
    "group-id": "mortgage-process",
    "artifact-id": "mortgage-process",
    "version": "1.0.0-SNAPSHOT"
  },
  "configuration": {
    "RULE": {
"org.kie.server.controller.api.model.spec.RuleConfig": {
    "pollInterval": null,
    "scannerStatus": "STOPPED"
  }
},
      "PROCESS": {
"org.kie.server.controller.api.model.spec.ProcessConfig": {
    "runtimeStrategy": "PER_PROCESS_INSTANCE",
    "kbase": "",
    "ksession": "",
    "mergeMode": "MERGE_COLLECTIONS"
  }
},
"status": "STARTED"
}
},
"server-config": {},
"server-instances": [
  {
    "server-instance-id": "default-kieserver-
instance@localhost:8080",
    "server-name": "default-kieserver-
instance@localhost:8080",
    "server-template-id": "default-kieserver",
    "server-url": "http://localhost:8080/kie-

```

```

server/services/rest/server"
    }
  ],
  "capabilities": [
    "RULE",
    "PROCESS",
    "PLANNING"
  ]
}
]
}

```

- In your REST client or curl utility, send another API request with the following components for a **PUT** request to `/controller/management/servers/{serverId}` to create a new Decision Server template. Adjust any request details according to your use case.

For REST client:

- **Authentication:** Enter the user name and password of the Decision Manager controller user with the **rest-all** role or the headless Decision Manager controller user with the **kie-server** role.
- **HTTP Headers:** Set the following headers:
  - **Accept:** `application/json`
  - **Content-Type:** `application/json`
- **HTTP method:** Set to **PUT**.
- **URL:** Enter the Decision Manager controller REST API base URL and endpoint, such as `http://localhost:8080/decision-central/rest/controller/management/servers/new-kieserver`.
- **Request body:** Add a JSON request body with the configurations for the new Decision Server template:

```

{
  "server-id": "new-kieserver",
  "server-name": "new-kieserver",
  "container-specs": [],
  "server-config": {},
  "capabilities": [
    "RULE",
    "PROCESS",
    "PLANNING"
  ]
}

```

For curl utility:

- **-u:** Enter the user name and password of the Decision Manager controller user with the **rest-all** role or the headless Decision Manager controller user with the **kie-server** role.
- **-H:** Set the following headers:
  - **accept:** `application/json`

- **content-type: application/json**
- **-X:** Set to **PUT**.
- **URL:** Enter the Decision Manager controller REST API base URL and endpoint, such as **http://localhost:8080/decision-central/rest/controller/management/servers/new-kieserver**.
- **-d:** Add a JSON request body or file (**@file.json**) with the configurations for the new Decision Server template:

```
curl -u 'baAdmin:password@1' -H "accept: application/json" -H
"content-type: application/json" -X PUT
"http://localhost:8080/decision-
central/rest/controller/management/servers/new-kieserver" -d "{
  \"server-id\": \"new-kieserver\", \"server-name\": \"new-
kieserver\", \"container-specs\": [], \"server-config\": {},
  \"capabilities\": [ \"RULE\", \"PROCESS\", \"PLANNING\" ]}"
```

```
curl -u 'baAdmin:password@1' -H "accept: application/json" -H
"content-type: application/json" -X PUT
"http://localhost:8080/decision-
central/rest/controller/management/servers/new-kieserver" -d @my-
server-template-configs.json
```

5. Execute the request and confirm the successful Decision Manager controller response. If you encounter request errors, review the returned error code messages and adjust your request accordingly.

## 5.2. SENDING REQUESTS WITH THE DECISION MANAGER CONTROLLER REST API USING THE SWAGGER INTERFACE

The Decision Manager controller REST API supports a Swagger web interface that you can use instead of a standalone REST client or curl utility to interact with your Decision Server templates, instances, and associated KIE containers in Red Hat Decision Manager without using the Decision Central user interface.

### Prerequisites

- The Decision Manager controller is installed and running.
- You have **rest-all** user role access to the Decision Manager controller if you installed Decision Central, or **kie-server** user role access to the headless Decision Manager controller installed separately from Decision Central.

### Procedure

1. In a web browser, navigate to **http://SERVER:PORT/CONTROLLER/docs**, such as **http://localhost:8080/decision-central/docs**, and log in with the user name and password of the Decision Manager controller user with the **rest-all** role or the headless Decision Manager controller user with the **kie-server** role.



## NOTE

If you are using the Decision Manager controller built in to Decision Central, the Swagger page associated with the Decision Manager controller is identified as the "Decision Central API" for Decision Central REST services. If you are using the headless Decision Manager controller without Decision Central, the Swagger page associated with the headless Decision Manager controller is identified as the "Controller API". The Decision Manager controller REST API endpoints in both cases are the same.

2. In the Swagger page, select the relevant API endpoint to which you want to send a request, such as **Controller :: Management** → **[GET] /controller/management/servers** to retrieve Decision Server templates from the Decision Manager controller.
3. Click **Try it out** and provide any optional parameters by which you want to filter results, if applicable.
4. In the **Response content type** drop-down menu, select the desired format of the server response, such as **application/json** for JSON format.
5. Click **Execute** and review the Decision Server response.

Example server response (JSON):

```
{
  "server-template": [
    {
      "server-id": "default-kieserver",
      "server-name": "default-kieserver",
      "container-specs": [
        {
          "container-id": "employeerostering_1.0.0-SNAPSHOT",
          "container-name": "employeerostering",
          "server-template-key": {
            "server-id": "default-kieserver",
            "server-name": "default-kieserver"
          },
          "release-id": {
            "group-id": "employeerostering",
            "artifact-id": "employeerostering",
            "version": "1.0.0-SNAPSHOT"
          },
          "configuration": {
            "RULE": {
              "org.kie.server.controller.api.model.spec.RuleConfig": {
                "pollInterval": null,
                "scannerStatus": "STOPPED"
              }
            },
            "PROCESS": {
              "org.kie.server.controller.api.model.spec.ProcessConfig": {
                "runtimeStrategy": "SINGLETON",
                "kbase": "",
                "ksession": "",
                "mergeMode": "MERGE_COLLECTIONS"
              }
            }
          }
        }
      ]
    }
  ]
}
```

```

        }
      }
    },
    "status": "STARTED"
  },
  {
    "container-id": "mortgage-process_1.0.0-SNAPSHOT",
    "container-name": "mortgage-process",
    "server-template-key": {
      "server-id": "default-kieserver",
      "server-name": "default-kieserver"
    },
    "release-id": {
      "group-id": "mortgage-process",
      "artifact-id": "mortgage-process",
      "version": "1.0.0-SNAPSHOT"
    },
    "configuration": {
      "RULE": {
"org.kie.server.controller.api.model.spec.RuleConfig": {
      "pollInterval": null,
      "scannerStatus": "STOPPED"
    }
      },
      "PROCESS": {
"org.kie.server.controller.api.model.spec.ProcessConfig": {
      "runtimeStrategy": "PER_PROCESS_INSTANCE",
      "kbase": "",
      "ksession": "",
      "mergeMode": "MERGE_COLLECTIONS"
    }
      }
    },
    "status": "STARTED"
  }
],
"server-config": {},
"server-instances": [
  {
    "server-instance-id": "default-kieserver-
instance@localhost:8080",
    "server-name": "default-kieserver-
instance@localhost:8080",
    "server-template-id": "default-kieserver",
    "server-url": "http://localhost:8080/kie-
server/services/rest/server"
  }
],
"capabilities": [
  "RULE",
  "PROCESS",
  "PLANNING"
]

```



```

    ]
  }
}

```

6. In the Swagger page, navigate to the **Controller :: Management** → **[GET] /controller/management/servers/{serverId}** endpoint to send another request to create a new Decision Server template. Adjust any request details according to your use case.
7. Click **Try it out** and enter the following components for the request:
  - **serverId**: Enter the ID of the new Decision Server template, such as **new-kieserver**.
  - **body**: Set the **Parameter content type** to the desired request body format, such as **application/json** for JSON format, and add a request body with the configurations for the new Decision Server template:

```

{
  "server-id": "new-kieserver",
  "server-name": "new-kieserver",
  "container-specs": [],
  "server-config": {},
  "capabilities": [
    "RULE",
    "PROCESS",
    "PLANNING"
  ]
}

```

8. In the **Response content type** drop-down menu, select the desired format of the server response, such as **application/json** for JSON format.
9. Click **Execute** and confirm the successful Decision Manager controller response. If you encounter request errors, review the returned error code messages and adjust your request accordingly.

### 5.3. SUPPORTED DECISION MANAGER CONTROLLER REST API ENDPOINTS

The Decision Manager controller REST API provides endpoints for interacting with Decision Server templates (configurations), Decision Server instances (remote servers), and associated KIE containers (deployment units). The Decision Manager controller REST API base URL is **http://SERVER:PORT/CONTROLLER/rest/**. All requests require HTTP Basic authentication or token-based authentication for the **rest-all** user role if you installed Decision Central and you want to use the built-in Decision Manager controller, or the **kie-server** user role if you installed the headless Decision Manager controller separately from Decision Central.

For the full list of Decision Manager controller REST API endpoints and descriptions, use one of the following resources:

- [Controller REST API](#) on the jBPM Documentation page (static)
- Swagger UI for the Decision Manager controller REST API at **http://SERVER:PORT/CONTROLLER/docs** (dynamic, requires running Decision Manager controller)

**NOTE**

If you are using the Decision Manager controller built in to Decision Central, the Swagger page associated with the Decision Manager controller is identified as the "Decision Central API" for Decision Central REST services. If you are using the headless Decision Manager controller without Decision Central, the Swagger page associated with the headless Decision Manager controller is identified as the "Controller API". The Decision Manager controller REST API endpoints in both cases are the same.

## CHAPTER 6. DECISION MANAGER CONTROLLER JAVA CLIENT API FOR DECISION SERVER TEMPLATES AND INSTANCES

Red Hat Decision Manager provides a Decision Manager controller Java client API that enables you to connect to the Decision Manager controller using REST or WebSocket protocol from your Java client application. You can use the Decision Manager controller Java client API as an alternative to the Decision Manager controller REST API to interact with your Decision Server templates (configurations), Decision Server instances (remote servers), and associated KIE containers (deployment units) in Red Hat Decision Manager without using the Decision Central user interface. This API support enables you to maintain your Red Hat Decision Manager servers and resources more efficiently and optimize your integration and development with Red Hat Decision Manager.

With the Decision Manager controller Java client API, you can perform the following actions also supported by the Decision Manager controller REST API:

- Retrieve information about Decision Server templates, instances, and associated KIE containers
- Update, start, or stop KIE containers associated with Decision Server templates and instances
- Create, update, or delete Decision Server templates
- Create, update, or delete Decision Server instances

Decision Manager controller Java client API requests require the following components:

### Authentication

The Decision Manager controller Java client API requires HTTP Basic authentication for the following user roles, depending on controller type:

- **rest-all** user role if you installed Decision Central and you want to use the built-in Decision Manager controller
- **kie-server** user role if you installed the headless Decision Manager controller separately from Decision Central

To view configured user roles for your Red Hat Decision Manager distribution, navigate to `~/$SERVER_HOME/standalone/configuration/application-roles.properties` and `~/application-users.properties`.

To add a user with the **kie-server** role or the **rest-all** role or both, navigate to `~/$SERVER_HOME/bin` and run the following command with the role or roles specified:

```
$ ./add-user.sh -a --user <USERNAME> --password <PASSWORD> --role kie-server,rest-all
```

To configure the **kie-server** or **rest-all** user with Decision Manager controller access, navigate to `~/$SERVER_HOME/standalone/configuration/standalone-full.xml`, uncomment the **org.kie.server** properties (if applicable), and add the controller user login credentials and controller location (if needed):

```
<property name="org.kie.server.location"
value="http://localhost:8080/kie-server/services/rest/server"/>
<property name="org.kie.server.controller"
```

```
value="http://localhost:8080/decision-central/rest/controller"/>
<property name="org.kie.server.controller.user" value="baAdmin"/>
<property name="org.kie.server.controller.pwd" value="password@1"/>
<property name="org.kie.server.id" value="default-kieserver"/>
```

For more information about user roles and Red Hat Decision Manager installation options, see [Planning a Red Hat Decision Manager installation](#).

## Project dependencies

The Decision Manager controller Java client API requires the following dependencies on the relevant classpath of your Java project:

```
// For remote execution on controller
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-controller-client</artifactId>
  <version>${rhdm.version}</version>
</dependency>

// For REST client
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-client</artifactId>
  <version>${resteasy.version}</version>
</dependency>

// For WebSocket client
<dependency>
  <groupId>io.undertow</groupId>
  <artifactId>undertow-websockets-jsr</artifactId>
  <version>${undertow.version}</version>
</dependency>

// For debug logging (optional):
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

The **<version>** for Red Hat Decision Manager dependencies is the Maven artifact version for Red Hat Decision Manager currently used in your project (for example, 7.14.0.Final-redhat-00002).



## NOTE

Instead of specifying a Red Hat Decision Manager **<version>** for individual dependencies, consider adding the Red Hat Business Automation bill of materials (BOM) dependency to your project **pom.xml** file. The Red Hat Business Automation BOM applies to both Red Hat Decision Manager and Red Hat Process Automation Manager. When you add the BOM files, the correct versions of transitive dependencies from the provided Maven repositories are included in the project.

Example BOM dependency:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.2.0.GA-redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

For more information about the Red Hat Business Automation BOM, see [What is the mapping between RHDM product and maven library version?](#)

## Client request configuration

All Java client requests with the Decision Manager controller Java client API must define at least the following controller communication components:

- Credentials of the **rest-all** user if you installed Decision Central, or the **kie-server** user if you installed the headless Decision Manager controller separately from Decision Central
- Decision Manager controller location for REST or WebSocket protocol:
  - Example REST URL: **http://localhost:8080/decision-central/rest/controller**
  - Example WebSocket URL: **ws://localhost:8080/headless-controller/websocket/controller**
- Marshalling format for API requests and responses (JSON or JAXB)
- A **KieServerControllerClient** object, which serves as the entry point for starting the server communication using the Java client API
- A **KieServerControllerClientFactory** defining REST or WebSocket protocol and user access
- The Decision Manager controller client service or services used, such as **listServerTemplates**, **getServerTemplate**, or **getServerInstances**

The following are examples of REST and WebSocket client configurations with these components:

### Client configuration example with REST

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
```

```
import
org.kie.server.controller.client.KieServerControllerClientFactory;

public class ListServerTemplatesExample {

    private static final String URL = "http://localhost:8080/decision-
central/rest/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static final MarshallingFormat FORMAT =
MarshallingFormat.JSON;

    public static void main(String[] args) {
        KieServerControllerClient client =
KieServerControllerClientFactory.newRestClient(URL,
USER,
PASSWORD);

        final ServerTemplateList serverTemplateList =
client.listServerTemplates();
        System.out.println(String.format("Found %s server template(s)
at controller url: %s",
serverTemplateList.getServerTemplates().length,
URL));
    }
}
```

### Client configuration example with WebSocket

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import
org.kie.server.controller.client.KieServerControllerClientFactory;

public class ListServerTemplatesExample {

    private static final String URL = "ws://localhost:8080/my-
controller/websocket/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static final MarshallingFormat FORMAT =
MarshallingFormat.JSON;

    public static void main(String[] args) {
        KieServerControllerClient client =
KieServerControllerClientFactory.newWebSocketClient(URL,
USER,
PASSWORD);
```

```

        final ServerTemplateList serverTemplateList =
client.listServerTemplates();
        System.out.println(String.format("Found %s server template(s)
at controller url: %s",

serverTemplateList.getServerTemplates().length,
                                URL));
    }
}

```

## 6.1. SENDING REQUESTS WITH THE DECISION MANAGER CONTROLLER JAVA CLIENT API

The Decision Manager controller Java client API enables you to connect to the Decision Manager controller using REST or WebSocket protocols from your Java client application. You can use the Decision Manager controller Java client API as an alternative to the Decision Manager controller REST API to interact with your Decision Server templates (configurations), Decision Server instances (remote servers), and associated KIE containers (deployment units) in Red Hat Decision Manager without using the Decision Central user interface.

### Prerequisites

- Decision Server is installed and running.
- The Decision Manager controller or headless Decision Manager controller is installed and running.
- You have **rest-all** user role access to the Decision Manager controller if you installed Decision Central, or **kie-server** user role access to the headless Decision Manager controller installed separately from Decision Central.
- You have a Java project with Red Hat Decision Manager resources.

### Procedure

1. In your client application, ensure that the following dependencies have been added to the relevant classpath of your Java project:

```

// For remote execution on controller
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-controller-client</artifactId>
  <version>${rhdm.version}</version>
</dependency>

// For REST client
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-client</artifactId>
  <version>${resteasy.version}</version>
</dependency>

// For WebSocket client

```

```

<dependency>
  <groupId>io.undertow</groupId>
  <artifactId>undertow-websockets-jsr</artifactId>
  <version>${undertow.version}</version>
</dependency>

// For debug logging (optional):
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>

```

2. Download the **Red Hat Decision Manager 7.2.0 Source Distribution** from the [Red Hat Customer Portal](#) and navigate to `~/rhdm-7.2.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-controller/kie-server-controller-client/src/main/java/org/kie/server/controller/client` to access the Decision Manager controller Java clients.
3. In the `~/kie/server/controller/client` folder, identify the relevant Java client implementation for the request you want to send, such as the **RestKieServerControllerClient** implementation to access client services for Decision Server templates and KIE containers in REST protocol.
4. In your client application, create a `.java` class for the API request. The class must contain the necessary imports, the Decision Manager controller location and user credentials, a **KieServerControllerClient** object, and the client method to execute, such as **createServerTemplate** and **createContainer** from the **RestKieServerControllerClient** implementation. Adjust any configuration details according to your use case.

### Creating and interacting with a Decision Server template and KIE containers

```

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.KieContainerStatus;
import org.kie.server.api.model.KieScannerStatus;
import org.kie.server.api.model.ReleaseId;
import org.kie.server.controller.api.model.spec.*;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;

public class RestTemplateContainerExample {

    private static final String URL = "http://localhost:8080/decision-
central/rest/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static KieServerControllerClient client;

```



```

    public static void main(String[] args) {
        KieServerControllerClient client =
KieServerControllerClientFactory.newRestClient(URL,

USER,

PASSWORD,

MarshallingFormat.JSON);
        // Create server template and KIE container, start and stop
KIE container, and delete server template
        ServerTemplate serverTemplate = createServerTemplate();
        ContainerSpec container = createContainer(serverTemplate);
        client.startContainer(container);
        client.stopContainer(container);
        client.deleteServerTemplate(serverTemplate.getId());
    }

    // Re-create and configure server template
    protected static ServerTemplate createServerTemplate() {
        ServerTemplate serverTemplate = new ServerTemplate();
        serverTemplate.setId("example-client-id");
        serverTemplate.setName("example-client-name");

serverTemplate.setCapabilities(Arrays.asList(Capability.PROCESS.name
()),

Capability.RULE.name(),

Capability.PLANNING.name()));

        client.saveServerTemplate(serverTemplate);

        return serverTemplate;
    }

    // Re-create and configure KIE containers
    protected static ContainerSpec createContainer(ServerTemplate
serverTemplate){
        Map<Capability, ContainerConfig> containerConfigMap = new
HashMap();

        ProcessConfig processConfig = new
ProcessConfig("PER_PROCESS_INSTANCE", "kieBase", "kieSession",
"MERGE_COLLECTION");
        containerConfigMap.put(Capability.PROCESS, processConfig);

        RuleConfig ruleConfig = new RuleConfig(5001,
KieScannerStatus.SCANNING);
        containerConfigMap.put(Capability.RULE, ruleConfig);

        ReleaseId releaseId = new
ReleaseId("org.kie.server.testing", "stateless-session-kjar",
"1.0.0-SNAPSHOT");

        ContainerSpec containerSpec = new ContainerSpec("example-

```

```

        container-id", "example-client-name", serverTemplate, releaseId,
        KieContainerStatus.STOPPED, containerConfigMap);
            client.saveContainerSpec(serverTemplate.getId(),
            containerSpec);

        return containerSpec;
    }
}

```

5. Run the configured `.java` class from your project directory to execute the request, and review the Decision Manager controller response.

If you enabled debug logging, Decision Server responds with a detailed response according to your configured marshalling format, such as JSON. If you encounter request errors, review the returned error code messages and adjust your Java configurations accordingly.

## 6.2. SUPPORTED DECISION MANAGER CONTROLLER JAVA CLIENTS

The following are some of the Java client services available in the `org.kie.server.controller.client` package of your Red Hat Decision Manager distribution. You can use these services to interact with related resources in the Decision Manager controller similarly to the Decision Manager controller REST API.

- **KieServerControllerClient**: Used as the entry point for communicating with the Decision Manager controller
- **RestKieServerControllerClient**: Implementation used to interact with Decision Server templates and KIE containers in REST protocol (found in `~/org/kie/server/controller/client/rest`)
- **WebSocketKieServerControllerClient**: Implementation used to interact with Decision Server templates and KIE containers in WebSocket protocol (found in `~/org/kie/server/controller/client/websocket`)

For the full list of available Decision Manager controller Java clients, download the **Red Hat Decision Manager 7.2.0 Source Distribution** from the [Red Hat Customer Portal](#) and navigate to `~/rhdm-7.2.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-controller/kie-server-controller-client/src/main/java/org/kie/server/controller/client`.

## 6.3. EXAMPLE REQUESTS WITH THE DECISION MANAGER CONTROLLER JAVA CLIENT API

The following are examples of Decision Manager controller Java client API requests for basic interactions with the Decision Manager controller. For the full list of available Decision Manager controller Java clients, download the **Red Hat Decision Manager 7.2.0 Source Distribution** from the [Red Hat Customer Portal](#) and navigate to `~/rhdm-7.2.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-controller/kie-server-controller-client/src/main/java/org/kie/server/controller/client`.

### Creating and interacting with Decision Server templates and KIE containers

You can use the `ServerTemplate` and `ContainerSpec` services in the REST or WebSocket Decision Manager controller clients to create, dispose, and update Decision Server templates and KIE containers, and to start and stop KIE containers, as illustrated in this example.

**Example request to create and interact with a Decision Server template and KIE containers**

```

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.KieContainerStatus;
import org.kie.server.api.model.KieScannerStatus;
import org.kie.server.api.model.ReleaseId;
import org.kie.server.controller.api.model.spec.*;
import org.kie.server.controller.client.KieServerControllerClient;
import
org.kie.server.controller.client.KieServerControllerClientFactory;

public class RestTemplateContainerExample {

    private static final String URL = "http://localhost:8080/decision-
central/rest/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static KieServerControllerClient client;

    public static void main(String[] args) {
        KieServerControllerClient client =
KieServerControllerClientFactory.newRestClient(URL,
USER,
PASSWORD,
MarshallingFormat.JSON);
        // Create server template and KIE container, start and stop KIE
container, and delete server template
        ServerTemplate serverTemplate = createServerTemplate();
        ContainerSpec container = createContainer(serverTemplate);
        client.startContainer(container);
        client.stopContainer(container);
        client.deleteServerTemplate(serverTemplate.getId());
    }

    // Re-create and configure server template
    protected static ServerTemplate createServerTemplate() {
        ServerTemplate serverTemplate = new ServerTemplate();
        serverTemplate.setId("example-client-id");
        serverTemplate.setName("example-client-name");

serverTemplate.setCapabilities(Arrays.asList(Capability.PROCESS.name(),
Capability.RULE.name(),
Capability.PLANNING.name()));

        client.saveServerTemplate(serverTemplate);
    }
}

```

```

        return serverTemplate;
    }

    // Re-create and configure KIE containers
    protected static ContainerSpec createContainer(ServerTemplate
serverTemplate){
        Map<Capability, ContainerConfig> containerConfigMap = new
HashMap();

        ProcessConfig processConfig = new
ProcessConfig("PER_PROCESS_INSTANCE", "kieBase", "kieSession",
"MERGE_COLLECTION");
        containerConfigMap.put(Capability.PROCESS, processConfig);

        RuleConfig ruleConfig = new RuleConfig(5001,
KieScannerStatus.SCANNING);
        containerConfigMap.put(Capability.RULE, ruleConfig);

        ReleaseId releaseId = new ReleaseId("org.kie.server.testing",
"stateless-session-kjar", "1.0.0-SNAPSHOT");

        ContainerSpec containerSpec = new ContainerSpec("example-
container-id", "example-client-name", serverTemplate, releaseId,
KieContainerStatus.STOPPED, containerConfigMap);
        client.saveContainerSpec(serverTemplate.getId(),
containerSpec);

        return containerSpec;
    }
}

```

### Listing Decision Server templates and specifying connection timeout (REST)

When you use REST protocol for Decision Manager controller Java client API requests, you can provide your own `javax.ws.rs.core.Configuration` specification to modify the underlying REST client API, such as connection timeout.

### Example REST request to return server templates and specify connection timeout

```

import java.util.concurrent.TimeUnit;
import javax.ws.rs.core.Configuration;
import org.jboss.resteasy.client.jaxrs.ResteasyClientBuilder;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import
org.kie.server.controller.client.KieServerControllerClientFactory;

public class RESTTimeoutExample {

    private static final String URL = "http://localhost:8080/decision-
central/rest/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

```

```

public static void main(String[] args) {

    // Specify connection timeout
    final Configuration configuration =
        new ResteasyClientBuilder()
            .establishConnectionTimeout(10,
                                       TimeUnit.SECONDS)
            .socketTimeout(60,
                           TimeUnit.SECONDS)
            .getConfiguration();
    KieServerControllerClient client =
KieServerControllerClientFactory.newRestClient(URL,
USER,
PASSWORD,
MarshallingFormat.JSON,
configuration);

    // Retrieve list of server templates
    final ServerTemplateList serverTemplateList =
client.listServerTemplates();
    System.out.println(String.format("Found %s server template(s)
at controller url: %s",
serverTemplateList.getServerTemplates().length,
                                URL));
}
}

```

### Listing Decision Server templates and specifying event notifications (WebSocket)

When you use WebSocket protocol for Decision Manager controller Java client API requests, you can enable event notifications based on changes that happen in the particular Decision Manager controller to which the client API is connected. For example, you can receive notifications when Decision Server templates or instances are connected to or updated in the Decision Manager controller.

#### Example WebSocket request to return server templates and specify event notifications

```

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.events.*;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import
org.kie.server.controller.client.KieServerControllerClientFactory;
import org.kie.server.controller.client.event.EventHandler;

public class WebSocketEventsExample {

    private static final String URL = "ws://localhost:8080/my-
controller/websocket/controller";

```

```

private static final String USER = "baAdmin";
private static final String PASSWORD = "password@1";

public static void main(String[] args) {
    KieServerControllerClient client =
KieServerControllerClientFactory.newWebSocketClient(URL,
USER,
PASSWORD,
MarshallingFormat.JSON,
new TestEventHandler());

    // Retrieve list of server templates
    final ServerTemplateList serverTemplateList =
client.listServerTemplates();
    System.out.println(String.format("Found %s server template(s)
at controller url: %s",
serverTemplateList.getServerTemplates().length,
URL));
    try {
        Thread.sleep(60 * 1000);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// Set up event notifications
static class TestEventHandler implements EventHandler {

    @Override
    public void onServerInstanceConnected(ServerInstanceConnected
serverInstanceConnected) {
        System.out.println("serverInstanceConnected = " +
serverInstanceConnected);
    }

    @Override
    public void onServerInstanceDeleted(ServerInstanceDeleted
serverInstanceDeleted) {
        System.out.println("serverInstanceDeleted = " +
serverInstanceDeleted);
    }

    @Override
    public void
onServerInstanceDisconnected(ServerInstanceDisconnected
serverInstanceDisconnected) {
        System.out.println("serverInstanceDisconnected = " +
serverInstanceDisconnected);
    }

    @Override

```

```
        public void onServerTemplateDeleted(ServerTemplateDeleted
serverTemplateDeleted) {
            System.out.println("serverTemplateDeleted = " +
serverTemplateDeleted);
        }

        @Override
        public void onServerTemplateUpdated(ServerTemplateUpdated
serverTemplateUpdated) {
            System.out.println("serverTemplateUpdated = " +
serverTemplateUpdated);
        }

        @Override
        public void onServerInstanceUpdated(ServerInstanceUpdated
serverInstanceUpdated) {
            System.out.println("serverInstanceUpdated = " +
serverInstanceUpdated);
        }

        @Override
        public void onContainerSpecUpdated(ContainerSpecUpdated
containerSpecUpdated) {
            System.out.println("onContainerSpecUpdated = " +
containerSpecUpdated);
        }
    }
}
```

## CHAPTER 7. KNOWLEDGE STORE REST API FOR DECISION CENTRAL SPACES AND PROJECTS

Red Hat Decision Manager provides a Knowledge Store REST API that you can use to interact with your projects and spaces in Red Hat Decision Manager without using the Decision Central user interface. The Knowledge Store is the artifact repository for assets in Red Hat Decision Manager. This API support enables you to facilitate and automate maintenance of Decision Central projects and spaces.

With the Knowledge Store REST API, you can perform the following actions:

- Retrieve information about all projects and spaces
- Create, update, or delete projects and spaces
- Build, deploy, and test projects
- Retrieve information about previous Knowledge Store REST API requests, or *jobs*

Knowledge Store REST API requests require the following components:

### Authentication

The Knowledge Store REST API requires HTTP Basic authentication or token-based authentication for the user role **rest-all**. To view configured user roles for your Red Hat Decision Manager distribution, navigate to `~/$SERVER_HOME/standalone/configuration/application-roles.properties` and `~/application-users.properties`.

To add a user with the **rest-all** role, navigate to `~/$SERVER_HOME/bin` and run the following command:

```
$ ./add-user.sh -a --user <USERNAME> --password <PASSWORD> --role rest-all
```

For more information about user roles and Red Hat Decision Manager installation options, see [Planning a Red Hat Decision Manager installation](#).

### HTTP headers

The Knowledge Store REST API requires the following HTTP headers for API requests:

- **Accept**: Data format accepted by your requesting client:
  - **application/json** (JSON)
- **Content-Type**: Data format of your **POST** or **PUT** API request data:
  - **application/json** (JSON)

### HTTP methods

The Knowledge Store REST API supports the following HTTP methods for API requests:

- **GET**: Retrieves specified information from a specified resource endpoint
- **POST**: Creates or updates a resource
- **DELETE**: Deletes a resource



## Base URL

The base URL for Knowledge Store REST API requests is `http://SERVER:PORT/decision-central/rest/`, such as `http://localhost:8080/decision-central/rest/`.



### NOTE

The REST API base URL for the Knowledge Store and for the Decision Manager controller built in to Decision Central are the same because both are considered part of Decision Central REST services.

## Endpoints

Knowledge Store REST API endpoints, such as `/spaces/{spaceName}` for a specified space, are the URIs that you append to the Knowledge Store REST API base URL to access the corresponding resource or type of resource in Red Hat Decision Manager.

### Example request URL for `/spaces/{spaceName}` endpoint

`http://localhost:8080/decision-central/rest/spaces/MySpace`

## Request data

HTTP **POST** requests in the Knowledge Store REST API may require a JSON request body with data to accompany the request.

### Example POST request URL and JSON request body data

`http://localhost:8080/decision-central/rest/spaces/MySpace/projects`

```

{
  "name": "Employee_Rostering",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner.
Assigns employees to shifts based on their skill."
}

```

## 7.1. SENDING REQUESTS WITH THE KNOWLEDGE STORE REST API USING A REST CLIENT OR CURL UTILITY

The Knowledge Store REST API enables you to interact with your projects and spaces in Red Hat Decision Manager without using the Decision Central user interface. You can send Knowledge Store REST API requests using any REST client or curl utility.

### Prerequisites

- Decision Central is installed and running.
- You have `rest-all` user role access to Decision Central.

### Procedure

1. Identify the relevant [API endpoint](#) to which you want to send a request, such as **[GET] /spaces** to retrieve spaces in Decision Central.
2. In a REST client or curl utility, enter the following components for a **GET** request to **/spaces**. Adjust any request details according to your use case.

For REST client:

- **Authentication:** Enter the user name and password of the Decision Central user with the **rest-all** role.
- **HTTP Headers:** Set the following header:
  - **Accept:** **application/json**
- **HTTP method:** Set to **GET**.
- **URL:** Enter the Knowledge Store REST API base URL and endpoint, such as **http://localhost:8080/decision-central/rest/spaces**.

For curl utility:

- **-u:** Enter the user name and password of the Decision Central user with the **rest-all** role.
- **-H:** Set the following header:
  - **accept:** **application/json**
- **-X:** Set to **GET**.
- **URL:** Enter the Knowledge Store REST API base URL and endpoint, such as **http://localhost:8080/decision-central/rest/spaces**.

```
curl -u 'baAdmin:password@1' -H "accept: application/json" -X GET
"http://localhost:8080/decision-central/rest/spaces"
```

3. Execute the request and review the Decision Server response.  
Example server response (JSON):

```
[
  {
    "name": "MySpace",
    "description": null,
    "projects": [
      {
        "name": "Employee_Rostering",
        "spaceName": "MySpace",
        "groupId": "employeerostering",
        "version": "1.0.0-SNAPSHOT",
        "description": "Employee rostering problem optimisation
using Planner. Assigns employees to shifts based on their skill.",
        "publicURIs": [
          {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-
Employee_Rostering"
          }
        ]
      }
    ]
  }
]
```

```

        {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-
Employee_Rostering"
        }
    ]
},
{
    "name": "Mortgage_Process",
    "spaceName": "MySpace",
    "groupId": "mortgage-process",
    "version": "1.0.0-SNAPSHOT",
    "description": "Getting started loan approval process in
BPMN2, decision table, business rules, and forms.",
    "publicURIs": [
        {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-
Mortgage_Process"
        },
        {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-
Mortgage_Process"
        }
    ]
}
],
"owner": "admin",
"defaultGroupId": "com.myspace"
},
{
    "name": "MySpace2",
    "description": null,
    "projects": [
        {
            "name": "IT_Orders",
            "spaceName": "MySpace",
            "groupId": "itorders",
            "version": "1.0.0-SNAPSHOT",
            "description": "Case Management IT Orders project",
            "publicURIs": [
                {
                    "protocol": "git",
                    "uri": "git://localhost:9418/MySpace/example-IT_Orders-
1"
                },
                {
                    "protocol": "ssh",
                    "uri": "ssh://localhost:8001/MySpace/example-IT_Orders-
1"
                }
            ]
        }
    ]
},
"owner": "admin",

```

```

    "defaultGroupId": "com.myspace"
  }
]

```

4. In your REST client or curl utility, send another API request with the following components for a **POST** request to `/spaces/{spaceName}/projects` to create a project within a space. Adjust any request details according to your use case.

For REST client:

- **Authentication:** Enter the user name and password of the Decision Central user with the `rest-all` role.
- **HTTP Headers:** Set the following header:
  - **Accept:** `application/json`
  - **Content-Type:** `application/json`
- **HTTP method:** Set to **POST**.
- **URL:** Enter the Knowledge Store REST API base URL and endpoint, such as `http://localhost:8080/decision-central/rest/spaces/MySpace/projects`.
- **Request body:** Add a JSON request body with the identification data for the new project:

```

{
  "name": "Employee_Rostering",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using
Planner. Assigns employees to shifts based on their skill."
}

```

For curl utility:

- **-u:** Enter the user name and password of the Decision Central user with the `rest-all` role.
- **-H:** Set the following headers:
  - **accept:** `application/json`
  - **content-type:** `application/json`
- **-X:** Set to **POST**.
- **URL:** Enter the Knowledge Store REST API base URL and endpoint, such as `http://localhost:8080/decision-central/rest/spaces/MySpace/projects`.
- **-d:** Add a JSON request body or file (`@file.json`) with the identification data for the new project:

```

curl -u 'baAdmin:password@1' -H "accept: application/json" -H
"content-type: application/json" -X POST
"http://localhost:8080/decision-
central/rest/spaces/MySpace/projects" -d "{ \"name\":
\"Employee_Rostering\", \"groupId\": \"employeerostering\",

```

```
\ "version\": \"1.0.0-SNAPSHOT\", \"description\": \"Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.\"}"
```

```
curl -u 'baAdmin:password@1' -H "accept: application/json" -H
"content-type: application/json" -X POST
"http://localhost:8080/decision-
central/rest/spaces/MySpace/projects" -d @my-project.json
```

- Execute the request and review the Decision Server response.

Example server response (JSON):

```
{
  "jobId": "1541017411591-6",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "projectName": "Employee_Rostering",
  "projectGroupId": "employeerostering",
  "projectVersion": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using
Planner. Assigns employees to shifts based on their skill."
}
```

If you encounter request errors, review the returned error code messages and adjust your request accordingly.

## 7.2. SUPPORTED KNOWLEDGE STORE REST API ENDPOINTS

The Knowledge Store REST API provides endpoints for managing spaces and projects in Red Hat Decision Manager and for retrieving information about previous Knowledge Store REST API requests, or *jobs*.

### 7.2.1. Spaces

The Knowledge Store REST API supports the following endpoints for managing spaces in Decision Central. The Knowledge Store REST API base URL is **http://SERVER:PORT/decision-central/rest/**. All requests require HTTP Basic authentication or token-based authentication for the **rest-all** user role.

#### [GET] /spaces

Returns all spaces in Decision Central.

#### Example server response (JSON)

```
[
  {
    "name": "MySpace",
    "description": null,
    "projects": [
      {
        "name": "Employee_Rostering",
        "spaceName": "MySpace",
        "groupId": "employeerostering",
```

```

        "version": "1.0.0-SNAPSHOT",
        "description": "Employee rostering problem optimisation using
Planner. Assigns employees to shifts based on their skill.",
        "publicURIs": [
            {
                "protocol": "git",
                "uri": "git://localhost:9418/MySpace/example-
Employee_Rostering"
            },
            {
                "protocol": "ssh",
                "uri": "ssh://localhost:8001/MySpace/example-
Employee_Rostering"
            }
        ]
    },
    {
        "name": "Mortgage_Process",
        "spaceName": "MySpace",
        "groupId": "mortgage-process",
        "version": "1.0.0-SNAPSHOT",
        "description": "Getting started loan approval process in BPMN2,
decision table, business rules, and forms.",
        "publicURIs": [
            {
                "protocol": "git",
                "uri": "git://localhost:9418/MySpace/example-
Mortgage_Process"
            },
            {
                "protocol": "ssh",
                "uri": "ssh://localhost:8001/MySpace/example-
Mortgage_Process"
            }
        ]
    }
],
"owner": "admin",
"defaultGroupId": "com.myspace"
},
{
    "name": "MySpace2",
    "description": null,
    "projects": [
        {
            "name": "IT_Orders",
            "spaceName": "MySpace",
            "groupId": "itorders",
            "version": "1.0.0-SNAPSHOT",
            "description": "Case Management IT Orders project",
            "publicURIs": [
                {
                    "protocol": "git",
                    "uri": "git://localhost:9418/MySpace/example-IT_Orders-1"
                }
            ]
        }
    ]
}

```

```

        "protocol": "ssh",
        "uri": "ssh://localhost:8001/MySpace/example-IT_Orders-1"
    }
  ]
}
],
"owner": "admin",
"defaultGroupId": "com.myspace"
}
]

```

**[GET] /spaces/{spaceName}**

Returns information about a specified space.

**Table 7.1. Request parameters**

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space to be retrieved	String	Required

**Example server response (JSON)**

```

{
  "name": "MySpace",
  "description": null,
  "projects": [
    {
      "name": "Mortgage_Process",
      "spaceName": "MySpace",
      "groupId": "mortgage-process",
      "version": "1.0.0-SNAPSHOT",
      "description": "Getting started loan approval process in BPMN2,
decision table, business rules, and forms.",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-
Mortgage_Process"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-
Mortgage_Process"
        }
      ]
    },
    {
      "name": "Employee_Rostering",
      "spaceName": "MySpace",
      "groupId": "employeerostering",
      "version": "1.0.0-SNAPSHOT",
      "description": "Employee rostering problem optimisation using
Planner. Assigns employees to shifts based on their skill.",

```

```

    "publicURIs": [
      {
        "protocol": "git",
        "uri": "git://localhost:9418/MySpace/example-
Employee_Rostering"
      },
      {
        "protocol": "ssh",
        "uri": "ssh://localhost:8001/MySpace/example-
Employee_Rostering"
      }
    ]
  },
  {
    "name": "Evaluation_Process",
    "spaceName": "MySpace",
    "groupId": "evaluation",
    "version": "1.0.0-SNAPSHOT",
    "description": "Getting started Business Process for evaluating
employees",
    "publicURIs": [
      {
        "protocol": "git",
        "uri": "git://localhost:9418/MySpace/example-
Evaluation_Process"
      },
      {
        "protocol": "ssh",
        "uri": "ssh://localhost:8001/MySpace/example-
Evaluation_Process"
      }
    ]
  },
  {
    "name": "IT_Orders",
    "spaceName": "MySpace",
    "groupId": "itorders",
    "version": "1.0.0-SNAPSHOT",
    "description": "Case Management IT Orders project",
    "publicURIs": [
      {
        "protocol": "git",
        "uri": "git://localhost:9418/MySpace/example-IT_Orders"
      },
      {
        "protocol": "ssh",
        "uri": "ssh://localhost:8001/MySpace/example-IT_Orders"
      }
    ]
  }
],
"owner": "admin",
"defaultGroupId": "com.myspace"
}

```

**[POST] /spaces**



Creates a space in Decision Central.

**Table 7.2. Request parameters**

Name	Description	Type	Requirement
<b>body</b>	The <b>name</b> , <b>description</b> , <b>owner</b> , <b>defaultGroupId</b> , and any other components of the new space	Request body	Required

### Example request body (JSON)

```
{
  "name": "NewSpace",
  "description": "My new space.",
  "owner": "admin",
  "defaultGroupId": "com.newspace"
}
```

### Example server response (JSON)

```
{
  "jobId": "1541016978154-3",
  "status": "APPROVED",
  "spaceName": "NewSpace",
  "owner": "admin",
  "defaultGroupId": "com.newspace",
  "description": "My new space."
}
```

### [DELETE] /spaces/{spaceName}

Deletes a specified space from Decision Central.

**Table 7.3. Request parameters**

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space to be deleted	String	Required

### Example server response (JSON)

```
{
  "jobId": "1541127032997-8",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "owner": "admin",
  "description": "My deleted space.",
  "repositories": null
}
```

## 7.2.2. Projects

The Knowledge Store REST API supports the following endpoints for managing, building, and deploying projects in Decision Central. The Knowledge Store REST API base URL is

**http://SERVER:PORT/decision-central/rest/**. All requests require HTTP Basic authentication or token-based authentication for the **rest-all** user role.

### [GET] /spaces/{spaceName}/projects

Returns projects in a specified space.

**Table 7.4. Request parameters**

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space for which you are retrieving projects	String	Required

### Example server response (JSON)

```
[
  {
    "name": "Mortgage_Process",
    "spaceName": "MySpace",
    "groupId": "mortgage-process",
    "version": "1.0.0-SNAPSHOT",
    "description": "Getting started loan approval process in BPMN2,
decision table, business rules, and forms.",
    "publicURIs": [
      {
        "protocol": "git",
        "uri": "git://localhost:9418/MySpace/example-Mortgage_Process"
      },
      {
        "protocol": "ssh",
        "uri": "ssh://localhost:8001/MySpace/example-Mortgage_Process"
      }
    ]
  },
  {
    "name": "Employee_Rostering",
    "spaceName": "MySpace",
    "groupId": "employeeerostering",
    "version": "1.0.0-SNAPSHOT",
    "description": "Employee rostering problem optimisation using
Planner. Assigns employees to shifts based on their skill.",
    "publicURIs": [
      {
        "protocol": "git",
        "uri": "git://localhost:9418/MySpace/example-
Employee_Rostering"
      },
      {
        "protocol": "ssh",
```

```

        "uri": "ssh://localhost:8001/MySpace/example-
Employee_Rostering"
    }
  ]
},
{
  "name": "Evaluation_Process",
  "spaceName": "MySpace",
  "groupId": "evaluation",
  "version": "1.0.0-SNAPSHOT",
  "description": "Getting started Business Process for evaluating
employees",
  "publicURIs": [
    {
      "protocol": "git",
      "uri": "git://localhost:9418/MySpace/example-
Evaluation_Process"
    },
    {
      "protocol": "ssh",
      "uri": "ssh://localhost:8001/MySpace/example-
Evaluation_Process"
    }
  ]
},
{
  "name": "IT_Orders",
  "spaceName": "MySpace",
  "groupId": "itorders",
  "version": "1.0.0-SNAPSHOT",
  "description": "Case Management IT Orders project",
  "publicURIs": [
    {
      "protocol": "git",
      "uri": "git://localhost:9418/MySpace/example-IT_Orders"
    },
    {
      "protocol": "ssh",
      "uri": "ssh://localhost:8001/MySpace/example-IT_Orders"
    }
  ]
}
]

```

### [GET] /spaces/{spaceName}/projects/{projectName}

Returns information about a specified project in a specified space.

**Table 7.5. Request parameters**

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space where the project is located	String	Required

Name	Description	Type	Requirement
<b>projectName</b>	Name of the project to be retrieved	String	Required

### Example server response (JSON)

```
{
  "name": "Employee_Rostering",
  "spaceName": "MySpace",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner.
  Assigns employees to shifts based on their skill.",
  "publicURIs": [
    {
      "protocol": "git",
      "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
    },
    {
      "protocol": "ssh",
      "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
    }
  ]
}
```

### [POST] /spaces/{spaceName}/projects

Creates a project in a specified space.

Table 7.6. Request parameters

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space in which the new project will be created	String	Required
<b>body</b>	The <b>name</b> , <b>groupId</b> , <b>version</b> , <b>description</b> , and any other components of the new project	Request body	Required

### Example request body (JSON)

```
{
  "name": "Employee_Rostering",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner.
  Assigns employees to shifts based on their skill."
}
```

**Example server response (JSON)**

```
{
  "jobId": "1541017411591-6",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "projectName": "Employee_Rostering",
  "projectGroupId": "employee rostering",
  "projectVersion": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner.
  Assigns employees to shifts based on their skill."
}
```

**[DELETE] /spaces/{spaceName}/projects/{projectName}**

Deletes a specified project from a specified space.

**Table 7.7. Request parameters**

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space where the project is located	String	Required
<b>projectName</b>	Name of the project to be deleted	String	Required

**Example server response (JSON)**

```
{
  "jobId": "1541128617727-10",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

**[POST] /spaces/{spaceName}/git/clone**

Clones a project into a specified space from a specified Git address.

**Table 7.8. Request parameters**

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space to which you are cloning a project	String	Required
<b>body</b>	The <b>name</b> , <b>description</b> , and Git repository <b>userName</b> , <b>password</b> , and <b>gitURL</b> for the project to be cloned	Request body	Required

**Example request body (JSON)**

```
{
  "name": "Employee_Rostering",
  "description": "Employee rostering problem optimisation using Planner.
  Assigns employees to shifts based on their skill.",
  "userName": "baAdmin",
  "password": "password@1",
  "gitURL": "git://localhost:9418/MySpace/example-Employee_Rostering"
}
```

**Example server response (JSON)**

```
{
  "jobId": "1541129488547-13",
  "status": "APPROVED",
  "cloneProjectRequest": {
    "name": "Employee_Rostering",
    "description": "Employee rostering problem optimisation using
    Planner. Assigns employees to shifts based on their skill.",
    "userName": "baAdmin",
    "password": "password@1",
    "gitURL": "git://localhost:9418/MySpace/example-Employee_Rostering"
  },
  "spaceName": "MySpace2"
}
```

**[POST] /spaces/{spaceName}/projects/{projectName}/maven/compile**

Compiles a specified project in a specified space (equivalent to `mvn compile`).

**Table 7.9. Request parameters**

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space where the project is located	String	Required
<b>projectName</b>	Name of the project to be compiled	String	Required

**Example server response (JSON)**

```
{
  "jobId": "1541128617727-10",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

**[POST] /spaces/{spaceName}/projects/{projectName}/maven/test**

Tests a specified project in a specified space (equivalent to `mvn test`).

Table 7.10. Request parameters

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space where the project is located	String	Required
<b>projectName</b>	Name of the project to be tested	String	Required

### Example server response (JSON)

```
{
  "jobId": "1541132591595-19",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

### [POST] /spaces/{spaceName}/projects/{projectName}/maven/install

Installs a specified project in a specified space (equivalent to `mvn install`).

Table 7.11. Request parameters

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space where the project is located	String	Required
<b>projectName</b>	Name of the project to be installed	String	Required

### Example server response (JSON)

```
{
  "jobId": "1541132668987-20",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

### [POST] /spaces/{spaceName}/projects/{projectName}/maven/deploy

Deploys a specified project in a specified space (equivalent to `mvn deploy`).

Table 7.12. Request parameters

Name	Description	Type	Requirement
<b>spaceName</b>	Name of the space where the project is located	String	Required
<b>projectName</b>	Name of the project to be deployed	String	Required

### Example server response (JSON)

```
{
  "jobId": "1541132816435-21",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

### 7.2.3. Jobs (API requests)

All **POST** and **DELETE** requests in the Knowledge Store REST API return a job ID associated with each request, in addition to the returned request details. You can use a job ID to view the request status or delete a sent request.

Knowledge Store REST API requests, or *jobs*, can have the following statuses:

**Table 7.13. Job statuses (API request statuses)**

Status	Description
<b>ACCEPTED</b>	The request was accepted and is being processed.
<b>BAD_REQUEST</b>	The request contained incorrect content and was not accepted.
<b>RESOURCE_NOT_EXIST</b>	The requested resource (path) does not exist.
<b>DUPLICATE_RESOURCE</b>	The resource already exists.
<b>SERVER_ERROR</b>	An error occurred in Decision Server.
<b>SUCCESS</b>	The request finished successfully.
<b>FAIL</b>	The request failed.
<b>APPROVED</b>	The request was approved.



Status	Description
<b>DENIED</b>	The request was denied.
<b>GONE</b>	The job ID for the request could not be found due to one of the following reasons: <ul style="list-style-type: none"> <li>• The request was explicitly removed.</li> <li>• The request finished and has been deleted from a status cache. A request is removed from a status cache after the cache has reached its maximum capacity.</li> <li>• The request never existed.</li> </ul>

The Knowledge Store REST API supports the following endpoints for retrieving or deleting sent API requests. The Knowledge Store REST API base URL is **http://SERVER:PORT/decision-central/rest/**. All requests require HTTP Basic authentication or token-based authentication for the **rest-all** user role.

#### [GET] /jobs/{jobId}

Returns the status of a specified job (a previously sent API request).

**Table 7.14. Request parameters**

Name	Description	Type	Requirement
<b>jobId</b>	ID of the job to be retrieved (example: <b>1541010216919-1</b> )	String	Required

#### Example server response (JSON)

```
{
  "status": "SUCCESS",
  "jobId": "1541010216919-1",
  "result": null,
  "lastModified": 1541010218352,
  "detailedResult": [
    "level:INFO, path:null, text:Build of module 'Mortgage_Process'
    (requested by system) completed.\n Build: SUCCESSFUL"
  ]
}
```

#### [DELETE] /jobs/{jobId}

Deletes a specified job (a previously sent API request). If the job is not being processed yet, this request removes the job from the job queue. This request does not cancel or stop an ongoing job.

**Table 7.15. Request parameters**

Name	Description	Type	Requirement
<b>jobId</b>	ID of the job to be deleted (example: <b>1541010216919-1</b> )	String	Required

### Example server response (JSON)

```
{
  "status": "GONE",
  "jobId": "1541010216919-1",
  "result": null,
  "lastModified": 1541132054916,
  "detailedResult": [
    "level:INFO, path:null, text:Build of module 'Mortgage_Process'
    (requested by system) completed.\n Build: SUCCESSFUL"
  ]
}
```

## CHAPTER 8. ADDITIONAL RESOURCES

- *Managing and monitoring Decision Server*
- *Packaging and deploying a Red Hat Decision Manager project*

## **APPENDIX A. VERSIONING INFORMATION**

Documentation last updated on Wednesday, February 13, 2019.