Red Hat Process Automation Manager
7.8

Interacting with Red Hat Process Automation Manager using KIE APIs
Red Hat Process Automation Manager 7.8 Interacting with Red Hat Process Automation Manager using KIE APIs

Red Hat Customer Content Services
brms-docs@redhat.com
Abstract
This document describes how to use KIE APIs to interact with KIE Servers, KIE containers, and business assets in Red Hat Process Automation Manager 7.8.
Table of Contents

PREFACE ......................................................................................................................... 4

CHAPTER 1. KIE SERVER REST API FOR KIE CONTAINERS AND BUSINESS ASSETS .......... 5
  1.1. SENDING REQUESTS WITH THE KIE SERVER REST API USING A REST CLIENT OR CURL UTILITY 6
  1.2. SENDING REQUESTS WITH THE KIE SERVER REST API USING THE SWAGGER INTERFACE 11
  1.3. SUPPORTED KIE SERVER REST API ENDPOINTS .................................................... 15
    Endpoint requirements ............................................................................................... 16

CHAPTER 2. KIE SERVER JAVA CLIENT API FOR KIE CONTAINERS AND BUSINESS ASSETS ............. 17
  2.1. SENDING REQUESTS WITH THE KIE SERVER JAVA CLIENT API ........................... 21
  2.2. SUPPORTED KIE SERVER JAVA CLIENTS ............................................................... 26
  2.3. EXAMPLE REQUESTS WITH THE KIE SERVER JAVA CLIENT API .......................... 27

CHAPTER 3. KIE SERVER AND KIE CONTAINER COMMANDS IN RED HAT PROCESS AUTOMATION MANAGER .............................................................................................................. 33
  3.1. SAMPLE KIE SERVER AND KIE CONTAINER COMMANDS ....................................... 33

CHAPTER 4. RUNTIME COMMANDS IN RED HAT PROCESS AUTOMATION MANAGER ............... 48
  4.1. SAMPLE RUNTIME COMMANDS IN RED HAT PROCESS AUTOMATION MANAGER ........ 48

CHAPTER 5. PROCESS AUTOMATION MANAGER CONTROLLER REST API FOR KIE SERVER TEMPLATES AND INSTANCES ......................................................................................... 68
  5.1. SENDING REQUESTS WITH THE PROCESS AUTOMATION MANAGER CONTROLLER REST API USING A REST CLIENT OR CURL UTILITY ..................................................... 70
  5.2. SENDING REQUESTS WITH THE PROCESS AUTOMATION MANAGER CONTROLLER REST API USING THE SWAGGER INTERFACE ......................................................... 74
  5.3. SUPPORTED PROCESS AUTOMATION MANAGER CONTROLLER REST API ENDPOINTS .................................................................................................................. 77

CHAPTER 6. PROCESS AUTOMATION MANAGER CONTROLLER JAVA CLIENT API FOR KIE SERVER TEMPLATES AND INSTANCES ................................................................. 79
  6.1. SENDING REQUESTS WITH THE PROCESS AUTOMATION MANAGER CONTROLLER JAVA CLIENT API ..................................................................................................................... 82
  6.2. SUPPORTED PROCESS AUTOMATION MANAGER CONTROLLER JAVA CLIENTS .......... 85
  6.3. EXAMPLE REQUESTS WITH THE PROCESS AUTOMATION MANAGER CONTROLLER JAVA CLIENT API ................................................................................................................ 86

CHAPTER 7. KNOWLEDGE STORE REST API FOR BUSINESS CENTRAL SPACES AND PROJECTS ...... 90
  7.1. SENDING REQUESTS WITH THE KNOWLEDGE STORE REST API USING A REST CLIENT OR CURL UTILITY ........................................................................................................... 91
  7.2. SUPPORTED KNOWLEDGE STORE REST API ENDPOINTS .......................................... 95
    7.2.1. Spaces ............................................................................................................... 95
    7.2.2. Projects ............................................................................................................ 100
    7.2.3. Jobs (API requests) .......................................................................................... 106
    7.2.4. Branches .......................................................................................................... 108

CHAPTER 8. SECURITY MANAGEMENT REST API FOR BUSINESS CENTRAL GROUPS, ROLES, AND USERS ...................................................................................................................... 113
  8.1. SENDING REQUESTS WITH THE SECURITY MANAGEMENT REST API USING A REST CLIENT OR CURL UTILITY .................................................................................................. 114
  8.2. SUPPORTED SECURITY MANAGEMENT REST API ENDPOINTS ............................. 116
    8.2.1. Groups .............................................................................................................. 116
    8.2.2. Roles .................................................................................................................. 118
    8.2.3. Users ................................................................................................................. 118
    8.2.4. Permissions ....................................................................................................... 122
      8.2.4.1. Supported permissions in Business Central .................................................... 128
CHAPTER 9. EJB API FOR KIE SESSIONS AND TASK SERVICES ........................................... 130
9.1. SUPPORTED EJB SERVICES .................................................................................. 130
9.2. DEPLOYING AN EJB SERVICES WAR FILE ....................................................... 131

CHAPTER 10. ADDITIONAL RESOURCES ......................................................................... 133

APPENDIX A. VERSIONING INFORMATION ................................................................. 134
As a business rules developer or systems administrator, you can use KIE APIs to interact with KIE Servers, KIE containers, and business assets in Red Hat Process Automation Manager. You can use the KIE Server REST API and Java client API to interact with KIE containers and business assets (such as business rules, processes, and solvers), the Process Automation Manager controller REST API and Java client API to interact with KIE Server templates and instances, and the Knowledge Store REST API to interact with spaces and projects in Business Central.

REST API ENDPOINTS FOR KIE SERVER AND THE PROCESS AUTOMATION MANAGER CONTROLLER

The lists of REST API endpoints for KIE Server and the Process Automation 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 KIE Server and Process Automation 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 KIE 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 KIE Server REST API at http://SERVER:PORT/kie-server/docs (dynamic, requires running KIE Server)

For the full list of Process Automation 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 Process Automation Manager controller REST API at http://SERVER:PORT/CONTROLLER/docs (dynamic, requires running Process Automation Manager controller)

Prerequisites

- Red Hat Process Automation Manager is installed and running. For installation and startup options, see Planning a Red Hat Process Automation Manager installation.

- You have access to Red Hat Process Automation Manager with the following user roles:
  - **kie-server**: For access to KIE Server API capabilities, and access to headless Process Automation Manager controller API capabilities without Business Central (if applicable)
  - **rest-all**: For access to Business Central API capabilities for the built-in Process Automation Manager controller and for the Business Central Knowledge Store
  - **admin**: For full administrative access to Red Hat Process Automation 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 Process Automation Manager installation.
CHAPTER 1. KIE SERVER REST API FOR KIE CONTAINERS AND BUSINESS ASSETS

Red Hat Process Automation Manager provides a KIE 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 Process Automation Manager without using the Business Central user interface. This API support enables you to maintain your Red Hat Process Automation Manager resources more efficiently and optimize your integration and development with Red Hat Process Automation Manager.

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

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

KIE Server REST API requests require the following components:

Authentication

The KIE 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 Process Automation 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 Process Automation Manager installation options, see `Planning a Red Hat Process Automation Manager installation`.

HTTP headers

The KIE 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:
  - XSTREAM
HTTP methods

The KIE 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 KIE Server REST API requests is `http://SERVER:PORT/kie-server/services/rest/`, such as `http://localhost:8080/kie-server/services/rest/`.

Endpoints

KIE Server REST API endpoints, such as `/server/containers/{containerId}` for a specified KIE container, are the URIs that you append to the KIE Server REST API base URL to access the corresponding resource or type of resource in Red Hat Process Automation 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 KIE 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`

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

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

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

### Prerequisites

- KIE Server is installed and running.
- You have **kie-server** user role access to KIE 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 KIE 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 KIE Server user with the **kie-server** role.
   - **HTTP Headers**: Set the following header:
     - **Accept**: `application/json`
   - **HTTP method**: Set to **GET**.
   - **URL**: Enter the KIE 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 KIE Server user with the **kie-server** role.
   - `-H`: Set the following header:
     - **Accept**: `application/json`
   - `-X`: Set to **GET**.
   - **URL**: Enter the KIE Server REST API base URL and endpoint, such as `http://localhost:8080/kie-server/services/rest/server/containers`.

   ```bash
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 KIE Server response.

   Example server response (JSON):

   ```json
   {
     "type": "SUCCESS",
     "msg": "List of created containers",
     "result": {
       "kie-containers": {
         "kie-container": [ }
   ```
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 KIE Server user with the `kie-server` role.
   - **HTTP Headers**: Set the following headers:
     - **Accept**: application/json
     - **Content-Type**: application/json

   ```json
   "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"
   }
   }
   }
   }
   }
   ```

   **NOTE**

   When you add `fields=not_null` to **Content-Type**, the null fields are excluded from the REST API response.

   - **HTTP method**: Set to PUT.
   - **URL**: Enter the KIE 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:

   ```json
   { }
   ```
"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 KIE Server user with the **kie-server** role.

- **-H**: Set the following headers:
  - **Accept**: application/json
  - **Content-Type**: application/json

  **NOTE**
  When you add **fields=not_null** to **Content-Type**, the null fields are excluded from the REST API response.

- **-X**: Set to **PUT**.

- **URL**: Enter the KIE Server REST API base URL and endpoint, such as

- **-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" }}"

6. Execute the request and review the KIE Server response.
Example server response (JSON):

```
{
  "type": "SUCCESS",
  "msg": "Container MyContainer successfully deployed with module itorders:itorders:1.0.0-SNAPSHOT."
}
```

Red Hat Process Automation Manager 7.8 Interacting with Red Hat Process Automation Manager using KIE APIs
If you encounter request errors, review the returned error code messages and adjust your request accordingly.

**REST API REQUESTS FOR PROCESS INSTANCES**

For REST API requests that send complex data objects to the process instance endpoint `/server/containers/{containerId}/processes/{processId}/instances`, ensure that you include either the fully qualified class name (such as `com.myspace.Person`) or the simple class name (such as `Person`) in the request body. The class name is required for the request body to be mapped to the correct business object in Red Hat Process Automation Manager. If you exclude the class name from the request, KIE Server does not unmarshall the object to the expected type.

**Correct request body for process instance**

```json
{
    "id": 4,
    "lease": {
        "com.myspace.restcall.LeaseModel": {
            "annualRent": 109608,
            "isAutoApproved": false
        }
    }
}
```

**Incorrect request body for process instance**

```json
{
    "id": 4,
    "lease": {
        "annualRent": 109608,
        "isAutoApproved": false
    }
}
```

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

The KIE 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 Process Automation Manager without using the Business Central user interface.

**NOTE**

By default, the Swagger web interface for KIE Server is enabled by the `org.kie.swagger.server.ext.disabled=false` system property. To disable the Swagger web interface in KIE Server, set this system property to `true`. 
Prerequisites

- KIE Server is installed and running.
- You have kie-server user role access to KIE 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 KIE 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 and KIE containers → [GET] /server/containers to retrieve KIE containers from KIE 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 KIE Server response.

Example server response (JSON):

```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 **groupId**, **artifactId**, 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 and KIE containers** → [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:

     ```json
     {
       "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": {
         "groupId": "itorders",
         "artifactId": "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 KIE Server response.

    Example server response (JSON):

    ```json
    {
      "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": {
        "groupId": "itorders",
        "artifactId": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "scanner": {
        "poll-interval": "5000",
        "status": "STARTED"
      }
    }
    ```
If you encounter request errors, review the returned error code messages and adjust your request accordingly.
REST API REQUESTS FOR PROCESS INSTANCES

For REST API requests that send complex data objects to the process instance endpoint `/server/containers/{containerId}/processes/{processId}/instances`, ensure that you include either the fully qualified class name (such as `com.myspace.Person`) or the simple class name (such as `Person`) in the request body. The class name is required for the request body to be mapped to the correct business object in Red Hat Process Automation Manager. If you exclude the class name from the request, KIE Server does not unmarshall the object to the expected type.

Correct request body for process instance

```json
{
  "id": 4,
  "lease": {
    "com.myspace.restcall.LeaseModel": {
      "annualRent": 109608,
      "isAutoApproved": false
    }
  }
}
```

Incorrect request body for process instance

```json
{
  "id": 4,
  "lease": {
    "annualRent": 109608,
    "isAutoApproved": false
  }
}
```

1.3. SUPPORTED KIE SERVER REST API ENDPOINTS

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

- KIE Server and KIE containers
- KIE session assets (for runtime commands)
- DMN assets
- Planning solvers
- Processes
- Process images
- Process and task forms
- Tasks
- Cases
The KIE 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 KIE 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 KIE Server REST API** at `http://SERVER:PORT/kie-server/docs` (dynamic, requires running KIE Server)

**NOTE**

By default, the Swagger web interface for KIE Server is enabled by the `org.kie.swagger.server.ext.disabled=false` system property. To disable the Swagger web interface in KIE Server, set this system property to `true`.

### Endpoint requirements

Note the following requirements for some of the KIE Server REST API endpoints:

- **Process images**: For API access to process images, the system property `<storesvgonsave enabled="true"/>` must be configured for your Red Hat Process Automation Manager project in `$SERVER_HOME/standalone/deployments/business-central.war/org.kie.workbench.KIEWebapp/profiles/jbpm.xml`. This property is set to `true` by default. If the API is not working with process images, set it to `true` in the file, restart your KIE Server, modify the relevant process and save it, and then build and deploy your project. This property enables SVG images to be stored so that they can be retrieved by the KIE Server REST API.

- **Custom queries**: Some custom query requests with the KIE Server REST API require a query mapper definition to map the query results to concrete objects. You can implement your own query result mappers or use the mappers provided with Red Hat Process Automation Manager. The query mappers in Red Hat Process Automation Manager are similar to other object-relational mapping (ORM) providers, such as Hibernate, which maps tables to entities. For example, you can use the `org.jbpm.kie.services.impl.query.mapper.ProcessInstanceQueryMapper`, also registered as `ProcessInstances`, in custom query endpoints for returning process instance data. Example POST endpoint with `ProcessInstances` mapper parameter:

```
http://localhost:8080/kie-server/services/rest/server/queries/definitions/jbpmProcessInstances?
mapper=ProcessInstances
```

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

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

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

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

KIE Server Java client API requests require the following components:

Authentication
The KIE Server Java client API requires HTTP Basic authentication for the user role kie-server. To view configured user roles for your Red Hat Process Automation Manager distribution, navigate to 

```
~/$SERVER_HOME/standalone/configuration/application-roles.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 Process Automation Manager installation options, see Planning a Red Hat Process Automation Manager installation.

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

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

<!-- For runtime commands -->
<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
</dependency>
```
The `<version>` for Red Hat Process Automation Manager dependencies is the Maven artifact version for Red Hat Process Automation Manager currently used in your project (for example, 7.39.0.Final-redhat-00005).

**NOTE**

Instead of specifying a Red Hat Process Automation 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:

```xml
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.8.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

For more information about the Red Hat Business Automation BOM, see What is the mapping between RHPAM product and maven library version?.

**Client request configuration**

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

- Credentials of the **kie-server** user
- KIE 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

```java
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

```java
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);
}
2.1. SENDING REQUESTS WITH THE KIE SERVER JAVA CLIENT API

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

Prerequisites

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

Procedure

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

```xml
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

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

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

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 KIE Server.

4. In your client application, create a .java class for the API request. The class must contain the necessary imports, KIE 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

```java
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);
        kieServicesClient = KieServicesConfiguration.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);

        // Trying to recreate the container...
    }
```

Red Hat Process Automation Manager 7.8 Interacting with Red Hat Process Automation Manager using KIE APIs
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 KIE 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.

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

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

```java
10:23:35.194 [main] INFO  o.k.s.a.m.MarshallerFactory - Marshaller extensions init
10:23:35.398 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to send GET request to 'http://localhost:8080/kgbranch-1'
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"]
}
]
}
}
type: 'class org.kie.server.api.model.ServiceResponse'
10:23:35.653 [main] DEBUG o.k.s.impl.KieServicesClientImpl - KieServicesClient
connected to: default-kieserver version 7.11.0.Final-redhat-00003
10:23:35.653 [main] DEBUG o.k.s.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.impl.KieServicesClientImpl - Building services client for
server capability KieServer
10:23:35.653 [main] DEBUG o.k.s.impl.KieServicesClientImpl - No builder found for
'KieServer' capability
10:23:35.654 [main] DEBUG o.k.s.impl.KieServicesClientImpl - Building services client for
server capability BRM
10:23:35.654 [main] DEBUG o.k.s.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.DroolsServicesClientBuilder@6b927fb' for capability 'BRM'
10:23:35.655 [main] DEBUG o.k.s.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.RuleServicesClient=org.kie.server.client.impl.RuleServicesClientImpl@4a9ee4}
10:23:35.655 [main] DEBUG o.k.s.impl.KieServicesClientImpl - Building services client for
server capability BPM
10:23:35.656 [main] DEBUG o.k.s.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.JBPMServicesClientBuilder@4c451f2' for capability 'BPM'
10:23:35.672 [main] DEBUG o.k.s.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.JobServicesClient=org.kie.server.client.impl.JobServicesClientImpl@1189d52, interface
org.kie.server.client.DocumentServicesClient=org.kie.server.client.impl.DocumentServicesClientImpl@564fabc8, interface
org.kie.server.client.admin.UserTaskAdminServicesClient=org.kie.server.client.impl.UserTaskAdminServicesClientImpl@16d04d3d, interface
org.kie.server.client.QueryServicesClient=org.kie.server.client.impl.QueryServicesClientImpl@49ec7f18, interface
org.kie.server.client.ProcessServicesClient=org.kie.server.client.impl.ProcessServicesClientImpl@1d2ad7be, interface
org.kie.server.client.UserTaskServicesClient=org.kie.server.client.impl.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 - 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.UIServicesClientImpl@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@49139829' 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.impl.SolverServicesClientImpl@77bd92c}
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.DMNServicesClientImpl@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
10:23:35.701 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to send PUT request to 'http://localhost:8080/kie-server/services/rest/server/containers/employeerostering3' with payload '{
  "container-id" : null,
  "release-id" : {
    "group-id" : "employeerostering",
    "artifact-id" : "employeerostering",
    "version" : "1.0.0-SNAPSHOT"
  },
  "resolved-release-id" : null,
  "status" : null,
  "scanner" : null,
  "config-items" : [],
  "messages" : []
}
If you encounter request errors, review the returned error code messages and adjust your Java configurations accordingly.

### 2.2. SUPPORTED KIE SERVER JAVA CLIENTS

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

- **KieServicesClient**: Used as the entry point for other KIE 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

- **ProcessServicesClient**: Used to start, signal, and abort processes or work items

- **QueryServicesClient**: Used to query processes, process nodes, and process variables

- **UserTaskServicesClient**: Used to perform all user-task operations, such as starting, claiming, or canceling a task, and to query tasks by a specified field, such as by user or by process instances ID

- **UIServicesClient**: Used to get String representation of forms (XML or JSON) and of a process image (SVG)

- **ProcessAdminServicesClient**: Provides an interface for operations with process instances (found in ~/org/kie/server/client/admin)

- **UserTaskAdminServicesClient**: Provides an interface for operations with user tasks (found in ~/org/kie/server/client/admin)

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

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


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


#### Listing KIE 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 KIE Server.

**Example request to return KIE Server capabilities**

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

    for (String capability : serverInfo.getCapabilities()) {
        System.out.print(" "+ capability);
    }
}
```
Listing KIE containers in KIE 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 KIE Server

```java
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

```java
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 KIE 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 KIE Server

Red Hat Process Automation Manager supports runtime commands that you can send to KIE 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 Process Automation 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;
A conversation between a client and a specific KIE 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.

Listing available business processes in a KIE container

You can use the QueryServicesClient client to list available process definitions. The QueryServicesClient methods use pagination, so in addition to the query you make, you must provide the current page and the number of results per page. In this example, the query starts on page 0 and lists the first 1000 results.

Example request to list business processes in KIE Server

```java
public void listProcesses() {
    System.out.println("== Listing Business Processes ==");
    QueryServicesClient queryClient = kieServicesClient.getServicesClient(QueryServicesClient.class);
    List<ProcessDefinition> findProcessesByContainerId = queryClient.findProcessesByContainerId("rewards", 0, 1000);
    for (ProcessDefinition def : findProcessesByContainerId) {
        System.out.println(def.getName() + " - " + def.getId() + " v" + def.getVersion());
    }
}
```
Starting a business process in a KIE container

You can use the `ProcessServicesClient` client to start a business process. Ensure that any custom classes that you require for your process are added into the `KieServicesConfiguration` object, using the `addExtraClasses()` method.

Example request to start a business process

```java
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
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.client.ProcessServicesClient;
...

public static void startProcess() {
    // Client configuration setup
    KieServicesConfiguration config = KieServicesFactory.newRestConfiguration(SERVER_URL, LOGIN, PASSWORD);

    // Add custom classes, such as Obj.class, to the configuration
    Set<Class<?>> extraClassList = new HashSet<Class<?>>();
    extraClassList.add(Obj.class);
    config.addExtraClasses(extraClassList);
    config.setMarshallingFormat(MarshallingFormat.JSON);

    // ProcessServicesClient setup
    KieServicesClient client = KieServicesFactory.newKieServicesClient(config);
    ProcessServicesClient processServicesClient = client.getServicesClient(ProcessServicesClient.class);

    // Create an instance of the custom class
    Obj obj = new Obj();
    obj.setOk("ok");

    Map<String, Object> variables = new HashMap<String, Object>();
    variables.put("test", obj);

    // Start the process with custom class
    processServicesClient.startProcess(CONTAINER, processId, variables);
}
```

Running a custom query

You can use the `QueryDefinition` object of the `QueryServicesClient` client to register and execute custom queries in KIE Server.
Example request to register and execute a custom query in KIE Server

```java
// Client setup
KieServicesConfiguration conf = KieServicesFactory.newRestConfiguration(SERVER_URL, LOGIN, PASSWORD);
KieServicesClient client = KieServicesFactory.newKieServicesClient(conf);

// Get the QueryServicesClient
QueryServicesClient queryClient = client.getServicesClient(QueryServicesClient.class);

// Build the query
QueryDefinition queryDefinition = QueryDefinition.builder().name(QUERY_NAME)
    .expression("select * from Task t")
    .source("java:jboss/datasources/ExampleDS")
    .target("TASK").build();

// Specify that two queries cannot have the same name
queryClient.unregisterQuery(QUERY_NAME);

// Register the query
queryClient.registerQuery(queryDefinition);

// Execute the query with parameters: query name, mapping type (to map the fields to an object),
page number, page size, and return type
List<TaskInstance> query = queryClient.query(QUERY_NAME,
QueryServicesClient.QUERY_MAP_TASK, 0, 100, TaskInstance.class);

// Read the result
for (TaskInstance taskInstance : query) {
   System.out.println(taskInstance);
}
```

In this example, the `target` instructs the query service to apply default filters. Alternatively, you can set filter parameters manually. The `Target` class supports the following values:

```java
public enum Target {
   PROCESS,
   TASK,
   BA_TASK,
   PO_TASK,
   JOBS,
   CUSTOM;
}
```
CHAPTER 3. KIE SERVER AND KIE CONTAINER COMMANDS IN RED HAT PROCESS AUTOMATION MANAGER

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

In the KIE 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 KIE Server REST API, see Chapter 1, KIE Server REST API for KIE containers and business assets.

In the KIE 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 KIE Server commands are executed by methods provided in the Java client API, so you do not need to embed the actual KIE Server commands in your Java application. For more information about using the KIE Server Java client API, see Chapter 2, KIE Server Java client API for KIE containers and business assets.

3.1. SAMPLE KIE SERVER AND KIE CONTAINER COMMANDS

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

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

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

You can run KIE 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)

```json
{
    "commands": [
        
```
Batch Java API request to retrieve, dispose, and re-create a KIE container

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

GetServerInfoCommand

Returns information about the KIE Server.

Example REST request body (JSON)

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

Example Java client method

```java
KieServerInfo serverInfo = kieServicesClient.getServerInfo();
```

Example server response (JSON)

```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"
          ],
          "messages": [
            {
              "severity": "INFO",
              "timestamp": {
                "java.util.Date": 1538502533321
              }
            }
          ]
        }
      }
    }
  ]
}
```

System.out.println("Error creating " + containerId + ". Message: ");
System.out.println(responseDispose.getMsg());
return;
}
System.out.println("Container recreated with success!");
}
GetServerStateCommand

Returns information about the current state and configurations of the KIE Server.

**Example REST request body (JSON)**

```json
{
    "commands": [ {
        "get-server-state" : { }
    } ]
}
```

**Example Java client method**

```java
KieServerStateInfo serverStateInfo = kieServicesClient.getServerState();
```

**Example server response (JSON)**

```json
{
    "response": [ {
        "type": "SUCCESS",
        "msg": "Successfully loaded server state for server id default-kieserver",
        "result": { 
            "kie-server-state-info": { 
                "controller": [ "http://localhost:8080/business-central/rest/controller"
                ],
                "config": { 
                    "config-items": [ 
                        { "itemName": "org.kie.server.location",
                        "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"
                        }
                        ]
                }
            }
        }
    }
}
```
CHAPTER 3. KIE SERVER AND KIE CONTAINER COMMANDS IN RED HAT PROCESS AUTOMATION MANAGER

CreateContainerCommand

Creates a KIE container in the KIE Server.

```json
{
    "itemName": "org.kie.server.controller",
    "itemValue": "http://localhost:8080/business-central/rest/controller",
    "itemType": "java.lang.String"
}
}
}
"containers": [
{
    "container-id": "employee-rostering",
    "release-id": {
        "group-id": "employee-rostering",
        "artifact-id": "employee-rostering",
        "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": "employee-rostering"
}
}
}

CreateContainerCommand

Creates a KIE container in the KIE Server.
Table 3.1. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>container</td>
<td>Map containing the <code>container-id, release-id</code> data (group ID, artifact ID, version), <code>status</code>, and any other components of the new KIE container</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example REST request body (JSON)

```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

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

Example server response (JSON)

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

Returns information about a specified KIE container in KIE Server.

Table 3.2. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>container-id</td>
<td>ID of the KIE container</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example REST request body (JSON)

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

Example Java client method

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

Example server response (JSON)

```json
{
    "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
}
```
ListContainersCommand

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

Table 3.3. Command attributes

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

Example REST request body (JSON)

```json
{
  "commands" : [ {
    "list-containers" : {
      "kie-container-filter" : {
        "release-id-filter" : { },
        "container-status-filter" : {
        }
      }
    }
  }
}
```
Example Java client method

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

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

Example server response (JSON)

```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": {
                                "group-id": "com.redhat",
                                "artifact-id": "Project1",
                                "version": "1.0"
                            },
                            "status": "STARTED",
                            "scanner": {
                                "status": "STARTED",
                                "poll-interval": 5000
                            },
                            "config-items": [
                                {
                                    "itemName": "RuntimeStrategy",
                                    "itemValue": "SINGLETON",
                                    "itemType": "java.lang.String"
                                },
                                {
                                    "itemName": "MergeMode",
                                    "itemValue": "MERGE_COLLECTIONS",
                                    "itemType": "java.lang.String"
                                },
                                {
                                    "itemName": "KBase",
                                    "itemValue": "com.redhat",
                                    "itemType": "java.lang.String"
                                }
                            ]
                        }
                    ]
                }
            }
        }
    ]
}
```
CallContainerCommand

Calls a KIE container and executes one or more runtime commands. For information about Red Hat Process Automation Manager runtime commands, see Chapter 4, Runtime commands in Red Hat Process Automation Manager.

Table 3.4. Command attributes

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

Example REST request body (JSON)

```json
{
  "commands" : [
    {
      "call-container" : {
        "payload" : {
          "lookup" : "defaultKieSession",
          "commands" : [{
            "fire-all-rules" : {
              "max" : -1,
              "out-identifier" : null
            }
          }]
        }
      }
    }
  ]
}
```
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 KIE Server.

Table 3.5. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>container-id</td>
<td>ID of the KIE container to be disposed</td>
<td>Required</td>
</tr>
</tbody>
</table>

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)
GetScannerInfoCommand

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

Table 3.6. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>container-id</td>
<td>ID of the KIE container where the KIE scanner is used</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example REST request body (JSON)

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

Example Java client method

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

Example server response (JSON)

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

**Example REST request body (JSON)**
```json
{
    "commands": [
        {
            "update-scanner": {
                "scanner": {
                    "status": "STARTED",
                    "poll-interval": 10000
                },
                "container-id": "command-script-container"
            }
        }
    ]
}
```

**Example Java client method**
```java
KieScannerResource scannerResource = new KieScannerResource();
scannerResource.setPollInterval(10000);
scannerResource.setStatus(KieScannerStatus.STARTED);

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

**Example server response (JSON)**
```json
{
    "response": [
        
```
UpdateReleaseIdCommand

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

Table 3.8. Command attributes

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

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

```java
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 PROCESS AUTOMATION MANAGER

Red Hat Process Automation Manager supports runtime commands that you can send to KIE 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 Process Automation Manager instance.

In the KIE 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 KIE Server REST API, see Chapter 1, KIE Server REST API for KIE containers and business assets.

In the KIE 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 KIE Server Java client API, see Chapter 2, KIE Server Java client API for KIE containers and business assets.

4.1. SAMPLE RUNTIME COMMANDS IN RED HAT PROCESS AUTOMATION MANAGER

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

- `BatchExecutionCommand`
- `InsertObjectCommand`
- `RetractCommand`
- `ModifyCommand`
- `GetObjectCommand`
- `GetObjectsCommand`
- `InsertElementsCommand`
- `FireAllRulesCommand`
- `StartProcessCommand`
- `SignalEventCommand`
- `CompleteWorkItemCommand`
- `AbortWorkItemCommand`
- `QueryCommand`
- `SetGlobalCommand`
- `GetGlobalCommand`
For the full list of supported runtime commands, see the `org.drools.core.command.runtime` package in your Red Hat Process Automation Manager instance.

Each command in this section includes a REST request body example (JSON) for the KIE Server REST API and an embedded Java command example for the KIE 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>commands</td>
<td>List of commands to be executed.</td>
<td>Required</td>
</tr>
<tr>
<td>lookup</td>
<td>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.</td>
<td>Required for stateless KIE session, optional for stateful KIE session</td>
</tr>
</tbody>
</table>

**NOTE**

KIE session IDs are in the `kmodule.xml` file of your Red Hat Process Automation Manager project. To view or add a KIE session ID in Business Central to use with the `lookup` command attribute, navigate to the relevant project in Business Central and go to project `Settings → KIE bases → KIE sessions`. If no KIE bases exist, click `Add KIE base → KIE sessions` to define the new KIE base and KIE sessions.

**Example JSON request body**

```json
{
"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

```
InsertObjectCommand insertCommand = new InsertObjectCommand(new Person("john", 25));
FireAllRulesCommand fireCommand = new FireAllRulesCommand();

BatchExecutionCommand batch = new BatchExecutionCommandImpl(Arrays.asList(insertCommand, fireCommand), "ksession1");
```

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**

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

**Example JSON request body**

```
{
  "commands": [
```

Red Hat Process Automation Manager 7.8 Interacting with Red Hat Process Automation Manager using KIE APIs

50
"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
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact-handle</td>
<td>The FactHandle associated with the object to be retracted</td>
<td>Required</td>
</tr>
</tbody>
</table>

### Example JSON request body

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

#### Example Java command: Use FactHandleFromString

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

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

```java
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**
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact-handle</td>
<td>The FactHandle associated with the object to be modified</td>
<td>Required</td>
</tr>
<tr>
<td>setters</td>
<td>List of setters for object modifications</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example JSON request body

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

Example Java command

```java
ModifyCommand modifyCommand = new ModifyCommand(factHandle);
List<Setter> setters = new ArrayList<Setter>();
setters.add(new SetterImpl("age", "25");
modifyCommand.setSetters(setters);
```

Example server response (JSON)

```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

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

Example JSON request body

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

Example Java command

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

Example server response (JSON)

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

GetObjectsCommand
Retrieves all objects from the KIE session as a collection.

Table 4.6. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>object-filter</td>
<td>Filter for the objects returned from the KIE session</td>
<td>Optional</td>
</tr>
<tr>
<td>out-identifier</td>
<td>Identifier to be used in the execution results</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Example JSON request body

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

Example Java command

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

Example server response (JSON)

```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"?>
<object xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="person">
<age>25</age><name>john</name>
 </object>",
                },
                {
                  "org.drools.compiler.test.Person": {
                    "name": "john",
                    "age": 25
                  }
                }
              ],
              "key": "objects"
            }
          ]
        }
      }
    }
  ]
}
```
InsertElementsCommand

Inserts a list of objects into the KIE session.

**Table 4.7. Command attributes**

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

**Example JSON request body**

```json
{
  "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

```java
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)

```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

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

**Example JSON request body**

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

**Example Java command**

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

**Example server response (JSON)**

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

**StartProcessCommand**
Starts a process using the process ID. You can also pass parameters and initial data to be inserted.

Table 4.9. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>processId</td>
<td>ID of the process to be started</td>
<td>Required</td>
</tr>
<tr>
<td>parameters</td>
<td>A Map &lt;String, Object&gt; argument to pass parameters in the process startup</td>
<td>Optional</td>
</tr>
<tr>
<td>data</td>
<td>List of objects to be inserted into the KIE session before the process startup</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Example JSON request body

```json
{
  "commands": [
    {
      "start-process": {
        "processId": "myProject.myProcess",
        "data": null,
        "parameter": [],
        "out-identifier": null
      }
    }
  ]
}
```

Example Java command

```java
StartProcessCommand startProcessCommand = new StartProcessCommand();
startProcessCommand.setProcessId("org.drools.task.processOne");
```

Example server response (JSON)

```json
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [],
      "facts": []
    }
  }
}
```

SignalEventCommand

Sends a signal event to the KIE session.

Table 4.10. Command attributes
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>event-type</td>
<td>Type of the incoming event</td>
<td>Required</td>
</tr>
<tr>
<td>process-instance-id</td>
<td>ID of the process instance to be signalled</td>
<td>Optional</td>
</tr>
<tr>
<td>event</td>
<td>Data of the incoming event</td>
<td>Optional</td>
</tr>
</tbody>
</table>

**Example JSON request body**

```json
{
  "commands": [
    {
      "signal-event": {
        "process-instance-id": 1001,
        "correlation-key": null,
        "event-type": "start",
        "event": {
          "org.kie.server.testing.Person": {
            "fullname": "john",
            "age": 25
          }
        }
      }
    }
  ]
}
```

**Example Java command**

```java
SignalEventCommand signalEventCommand = new SignalEventCommand();
signalEventCommand.setProcessInstanceId(1001);
signalEventCommand.setEventType("start");
signalEventCommand.setEvent(new Person("john", 25));
```

**Example server response (JSON)**

```json
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [],
      "facts": []
    }
  }
}
```

**CompleteWorkItemCommand**

Completes a work item in the KIE session.

Table 4.11 Command attributes
Table 4.11. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>workItemId</td>
<td>ID of the work item to be completed</td>
<td>Required</td>
</tr>
<tr>
<td>results</td>
<td>Result of the work item</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Example JSON request body

```json
{
  "commands": [
    {
      "complete-work-item": {
        "id": 1001
      }
    }
  ]
}
```

Example Java command

```java
CompleteWorkItemCommand completeWorkItemCommand = new CompleteWorkItemCommand();
completeWorkItemCommand.setWorkItemId(1001);
```

Example server response (JSON)

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

AbortWorkItemCommand

Aborts a work item in the KIE session in the same way as 
`ksession.getWorkItemManager().abortWorkItem(workItemId)`. 

Table 4.12. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>workItemId</td>
<td>ID of the work item to be aborted</td>
<td>Required</td>
</tr>
</tbody>
</table>
Example JSON request body

```json
{
    "commands": [
        {
            "abort-work-item": {
                "id": 1001
            }
        }
    ]
}
```

Example Java command

```java
AbortWorkItemCommand abortWorkItemCommand = new AbortWorkItemCommand();
abortWorkItemCommand.setWorkItemId(1001);
```

Example server response (JSON)

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

QueryCommand

Executes a query defined in the KIE base.

Table 4.13. Command attributes

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

Example JSON request body

```json
{
    "commands": [
        {
            "commands": [
```

62
Example Java command

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

Example server response (JSON)

```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",
                                                    }},
                                                    "key": "$person"
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```
SetGlobalCommand

Sets an object to a global state.

Table 4.14. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>ID of the global variable defined in the KIE base</td>
<td>Required</td>
</tr>
<tr>
<td>object</td>
<td>Object to be set into the global variable</td>
<td>Optional</td>
</tr>
</tbody>
</table>
**Example JSON request body**

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

**Example Java command**

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

**Example server response (JSON)**

```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"
        }
      ]
    }
  }
}
```
GetGlobalCommand

Receives a previously defined global object.

Table 4.15. Command attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>ID of the global variable defined in the KIE base</td>
<td>Required</td>
</tr>
<tr>
<td>out-identifier</td>
<td>ID to be used in the execution results</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Example JSON request body

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

Example Java command

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

Example server response (JSON)

```json
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called."
    },
    {
      "execution-results": {
        "results": [
          {
            "value": null,
            "key": "helperOutput"
          }
        ],
        "facts": []
      }
    }
  ]
}
```
CHAPTER 5. PROCESS AUTOMATION MANAGER
CONTROLLER REST API FOR KIE SERVER TEMPLATES AND
INSTANCES

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

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

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

Requests to the Process Automation Manager controller REST API require the following components:

Authentication

The Process Automation 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 Business Central and you want to use the built-in Process Automation Manager controller
- **kie-server** user role if you installed the headless Process Automation Manager controller separately from Business Central

To view configured user roles for your Red Hat Process Automation 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 Process Automation 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/business-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 Process Automation Manager installation options, see Planning a Red Hat Process Automation Manager installation.

HTTP headers

The Process Automation 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 Process Automation 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 Process Automation Manager controller REST API requests is http://SERVER:PORT/CONTROLLER/rest/, such as http://localhost:8080/business-central/rest/ if you are using the Process Automation Manager controller built in to Business Central.

Endpoints

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

Example request URL for /controller/management/servers/{serverTemplateld} endpoint


Request parameters and request data

Some Process Automation 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
location=http://localhost:8080/kie-server/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


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

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

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

Prerequisites

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

Procedure

1. Identify the relevant API endpoint to which you want to send a request, such as [GET] /controller/management/servers to retrieve KIE Server templates from the Process Automation 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 Process Automation Manager controller user with the rest-all role or the headless Process Automation 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 Process Automation Manager controller REST API base URL and endpoint, such as http://localhost:8080/business-central/rest/controller/management/servers.

For curl utility:
- **-u**: Enter the user name and password of the Process Automation Manager controller user with the rest-all role or the headless Process Automation Manager controller user with the kie-server role.
- **-H**: Set the following header:
  - Accept: application/json
- **-X**: Set to GET.

URL: Enter the Process Automation Manager controller REST API base URL and endpoint, such as http://localhost:8080/business-central/rest/controller/management/servers.

```bash
```

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

```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",
"GUID",
"BPMN",
"TLD",
"REST",
"INTEGRATION"
]
4. In your REST client or curl utility, send another API request with the following components for a **PUT** request to `/controller/management/servers/{serverTemplateId}` to create a new KIE Server template. Adjust any request details according to your use case.

For REST client:

- **Authentication**: Enter the user name and password of the Process Automation Manager controller user with the **rest-all** role or the headless Process Automation 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 Process Automation Manager controller REST API base URL and endpoint, such as `http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver`.

- **Request body**: Add a JSON request body with the configurations for the new KIE Server template:

  ```json
  {
    "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 Process Automation Manager controller user with the **rest-all** role or the headless Process Automation 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 Process Automation Manager controller REST API base URL and endpoint, such as http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver.

-d: Add a JSON request body or file (@file.json) with the configurations for the new KIE Server template:

```
```

```
```

5. Execute the request and confirm the successful Process Automation 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 PROCESS AUTOMATION MANAGER CONTROLLER REST API USING THE SWAGGER INTERFACE

The Process Automation 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 KIE Server templates, instances, and associated KIE containers in Red Hat Process Automation Manager without using the Business Central user interface.

**NOTE**
By default, the Swagger web interface for the Process Automation Manager controller is enabled by the org.kie.workbench.swagger.disabled=false system property. To disable the Swagger web interface for the Process Automation Manager controller, set this system property to true.

Prerequisites

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

Procedure

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

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

2. In the Swagger page, select the relevant API endpoint to which you want to send a request, such as Controller :: KIE Server templates and KIE containers+ [GET] /controller/management/servers to retrieve KIE Server templates from the Process Automation 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 KIE Server response. Example server response (JSON):

```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"
              }
            }
          }
        }
      ]
    }
  ]
}
```
6. In the Swagger page, navigate to the **Controller :: KIE Server templates and KIE containers** → **[GET] /controller/management/servers/{serverTemplateId}** endpoint to send another
request to create a new KIE Server template. Adjust any request details according to your use case.

7. Click **Try it out** and enter the following components for the request:

   - **serverTemplateId**: Enter the ID of the new KIE 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 KIE Server template:

   ```json
   {    
      "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 Process Automation Manager controller response. If you encounter request errors, review the returned error code messages and adjust your request accordingly.

### 5.3. SUPPORTED PROCESS AUTOMATION MANAGER CONTROLLER REST API ENDPOINTS

The Process Automation Manager controller REST API provides endpoints for interacting with KIE Server templates (configurations), KIE Server instances (remote servers), and associated KIE containers (deployment units). The Process Automation 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 Business Central and you want to use the built-in Process Automation Manager controller, or the `kie-server` user role if you installed the headless Process Automation Manager controller separately from Business Central.

For the full list of Process Automation 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 Process Automation Manager controller REST API at `http://SERVER:PORT/CONTROLLER/docs` (dynamic, requires running Process Automation Manager controller)
NOTE

By default, the Swagger web interface for the Process Automation Manager controller is enabled by the `org.kie.workbench.swagger.disabled=false` system property. To disable the Swagger web interface for the Process Automation Manager controller, set this system property to `true`.

If you are using the Process Automation Manager controller built in to Business Central, the Swagger page associated with the Process Automation Manager controller is identified as the "Business Central API" for Business Central REST services. If you are using the headless Process Automation Manager controller without Business Central, the Swagger page associated with the headless Process Automation Manager controller is identified as the "Controller API". In both cases, the Process Automation Manager controller REST API endpoints are the same.
CHAPTER 6. PROCESS AUTOMATION MANAGER CONTROLLER JAVA CLIENT API FOR KIE SERVER TEMPLATES AND INSTANCES

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

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

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

Process Automation Manager controller Java client API requests require the following components:

### Authentication

The Process Automation 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 Business Central and you want to use the built-in Process Automation Manager controller
- **kie-server** user role if you installed the headless Process Automation Manager controller separately from Business Central

To view configured user roles for your Red Hat Process Automation 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 Process Automation 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):

```xml
<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/business-
```
For more information about user roles and Red Hat Process Automation Manager installation options, see Planning a Red Hat Process Automation Manager installation.

Project dependencies

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

```xml
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-controller-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

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

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

<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

The `<version>` for Red Hat Process Automation Manager dependencies is the Maven artifact version for Red Hat Process Automation Manager currently used in your project (for example, 7.39.0.Final-redhat-00005).
NOTE
Instead of specifying a Red Hat Process Automation 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:

```xml
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.8.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

For more information about the Red Hat Business Automation BOM, see What is the mapping between RHPAM product and maven library version?.

Client request configuration
All Java client requests with the Process Automation Manager controller Java client API must define at least the following controller communication components:

- Credentials of the `rest-all` user if you installed Business Central, or the `kie-server` user if you installed the headless Process Automation Manager controller separately from Business Central

- Process Automation Manager controller location for REST or WebSocket protocol:
  - Example REST URL: `http://localhost:8080/business-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 Process Automation 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

```java
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/business-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 PROCESS AUTOMATION MANAGER CONTROLLER JAVA CLIENT API
The Process Automation Manager controller Java client API enables you to connect to the Process Automation Manager controller using REST or WebSocket protocols from your Java client application. You can use the Process Automation Manager controller Java client API as an alternative to the Process Automation Manager controller REST API to interact with your KIE Server templates (configurations), KIE Server instances (remote servers), and associated KIE containers (deployment units) in Red Hat Process Automation Manager without using the Business Central user interface.

Prerequisites

- KIE Server is installed and running.
- The Process Automation Manager controller or headless Process Automation Manager controller is installed and running.
- You have **rest-all** user role access to the Process Automation Manager controller if you installed Business Central, or **kie-server** user role access to the headless Process Automation Manager controller installed separately from Business Central.
- You have a Java project with Red Hat Process Automation 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>${rhpam.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>
   ```

client/src/main/java/org/kie/server/controller/client to access the Process Automation 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 KIE 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 Process Automation 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 KIE Server template and KIE containers

```java
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/business-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()),
```
```
Run the configured .java class from your project directory to execute the request, and review the Process Automation Manager controller response.

If you enabled debug logging, KIE 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.

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

If you enabled debug logging, KIE 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 PROCESS AUTOMATION 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 Process Automation Manager distribution. You can use these services to interact with related resources in the Process Automation Manager controller similarly to the Process Automation Manager controller REST API.

- **KieServerControllerClient**: Used as the entry point for communicating with the Process Automation Manager controller

- **RestKieServerControllerClient**: Implementation used to interact with KIE Server templates and KIE containers in REST protocol (found in ~/org/kie/server/controller/client/rest)

- **WebSocketKieServerControllerClient**: Implementation used to interact with KIE Server templates and KIE containers in WebSocket protocol (found in ~/org/kie/server/controller/client/websocket)

For the full list of available Process Automation Manager controller Java clients, download the Red Hat
6.3. EXAMPLE REQUESTS WITH THE PROCESS AUTOMATION MANAGER CONTROLLER JAVA CLIENT API

The following are examples of Process Automation Manager controller Java client API requests for basic interactions with the Process Automation Manager controller. For the full list of available Process Automation Manager controller Java clients, download the Red Hat Process Automation Manager 7.8.0 Source Distribution from the Red Hat Customer Portal and navigate to

```
```

Creating and interacting with KIE Server templates and KIE containers

You can use the `ServerTemplate` and `ContainerSpec` services in the REST or WebSocket Process Automation Manager controller clients to create, dispose, and update KIE 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 KIE Server template and KIE containers

```java
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/business-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());
    }
}
```
Listing KIE Server templates and specifying connection timeout (REST)

When you use REST protocol for Process Automation 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

```java
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/business-central/rest/controller";

    // 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(500l, KieScannerStatus.SCANNING);
        containerConfigMap.put(Capability.RULE, ruleConfig);
        ReleaseId releaseld = new ReleaseId("org.kie.server.testing", "stateless-session-kjar", "1.0.0-SNAPSHOT");
        ContainerSpec containerSpec = new ContainerSpec("example-container-id", "example-client-name", serverTemplate, releaseld, KieContainerStatus.STOPPED, containerConfigMap);
        client.saveContainerSpec(serverTemplate.getId(), containerSpec);
        return containerSpec;
    }
```

Example REST request to return server templates and specify connection timeout

```java
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/business-central/rest/controller";
```
Listing KIE Server templates and specifying event notifications (WebSocket)

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

Example WebSocket request to return server templates and specify event notifications

```java
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,
                null,
                null);
    }
}
```

Listing KIE Server templates and specifying event notifications (WebSocket)

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

Example WebSocket request to return server templates and specify event notifications

```java
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));
}
```

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,
                null,
                null);
    }
}
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 BUSINESS CENTRAL SPACES AND PROJECTS

Red Hat Process Automation Manager provides a Knowledge Store REST API that you can use to interact with your projects and spaces in Red Hat Process Automation Manager without using the Business Central user interface. The Knowledge Store is the artifact repository for assets in Red Hat Process Automation Manager. This API support enables you to facilitate and automate maintenance of Business 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 Process Automation 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 Process Automation Manager installation options, see Planning a Red Hat Process Automation 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
- **PUT**: Updates a resource
- **DELETE**: Deletes a resource
The base URL for Knowledge Store REST API requests is `http://SERVER:PORT/business-central/rest/`, such as `http://localhost:8080/business-central/rest/`.

**NOTE**

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

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

`http://localhost:8080/business-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**


```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."
}
```

## 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 Process Automation Manager without using the Business Central user interface. You can send Knowledge Store REST API requests using any REST client or curl utility.

**Prerequisites**

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

**Procedure**

1. Identify the relevant **API endpoint** to which you want to send a request, such as `[GET] /spaces` to retrieve spaces in Business 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 Business 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/business-central/rest/spaces`.

For curl utility:

- `-u`: Enter the user name and password of the Business 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/business-central/rest/spaces`.

```
```

3. Execute the request and review the KIE Server response.

Example server response (JSON):

```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"
          }
        ]
      }
    ]
  }
]```
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:

```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"
        }
    ],
    "owner": "admin",
    "defaultGroupId": "com.myspace"
}
```

```json
{
    "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"
        }
    ]
}
```
Authentication: Enter the user name and password of the Business Central user with the rest-all role.

HTTP Headers: Set the following header:
- Accept: application/json
- Accept-Language: en-US
- 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/business-central/rest/spaces/MySpace/projects.

Request body: Add a JSON request body with the identification data for the new project:

```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."
}
```

For curl utility:

- `-u`: Enter the user name and password of the Business Central user with the rest-all role.
- `-H`: Set the following headers:
  - Accept: application/json
  - Accept-Language: en-US (If not defined, the default locale from the JVM is reflected)
  - Content-Type: application/json
- `-X`: Set to POST.
- URL: Enter the Knowledge Store REST API base URL and endpoint, such as http://localhost:8080/business-central/rest/spaces/MySpace/projects.
- `-d`: Add a JSON request body or file (`@file.json`) with the identification data for the new project:

```
```

```
```
5. Execute the request and review the KIE Server response.

Example server response (JSON):

```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 Process Automation 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 Business Central. The Knowledge Store REST API base URL is `http://SERVER:PORT/business-central/rest/`. All requests require HTTP Basic authentication or token-based authentication for the `rest-all` user role.

**[GET] /spaces**

Returns all spaces in Business Central.

**Example server response (JSON)**

```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"
            }
        ]
    }
]```
[GET] /spaces/{spaceName}

Returns information about a specified space.
Table 7.1. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space to be retrieved</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

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": null,
            "publicURIs": [
                {
                    "protocol": "git",
                    "uri": "git://localhost:9418/MySpace/example-Evaluation_Process"
                },
                {
                    "protocol": "ssh",
                    "uri": "ssh://localhost:8001/MySpace/example-Evaluation_Process"
                }
            ]
        }
    ]
}
```
POST /spaces

Creates a space in Business Central.

Table 7.2. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>The <strong>name</strong>, <strong>description</strong>, <strong>owner</strong>, <strong>defaultGroupId</strong>, and any other components of the new space</td>
<td>Request body</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example request body (JSON)

```json
{
  "name": "NewSpace",
  "description": "My new space.",
  "owner": "admin",
  "defaultGroupId": "com.newspace"
}
```
Example server response (JSON)

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

[PUT] /spaces

Updates description, owner, and defaultGroupId of a space in Business Central.

Example request body (JSON)

```json
{
  "name": "MySpace",
  "description": "This is updated description",
  "owner": "admin",
  "defaultGroupId": "com.updatedGroupId"
}
```

Example server response (JSON)

```json
{
  "jobId": "1592214574454-1",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "owner": "admin",
  "defaultGroupId": "com.updatedGroupId",
  "description": "This is updated description"
}
```

[DELETE] /spaces/{spaceName}

Deletes a specified space from Business Central.

Table 7.3. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space to be deleted</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```json
{
  "jobId": "1541127032997-8",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "owner": "admin",
```
7.2.2. Projects

The Knowledge Store REST API supports the following endpoints for managing, building, and deploying projects in Business Central. The Knowledge Store REST API base URL is http://SERVER:PORT/business-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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space for which you are retrieving projects</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```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": "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"
      }
    ]
  }
]
```
[GET] /spaces/{spaceName}/projects/{projectName}

Returns information about a specified project in a specified space.

Table 7.5. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>
Example server response (JSON)

```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."
}
```

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

Creates a project in a specified space.

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

Example request body (JSON)

```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)
[DELETE] /spaces/{spaceName}/projects/{projectName}

Deletes a specified project from a specified space.

Table 7.7. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project to be deleted</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

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."
}
```

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

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

Table 7.8. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space to which you are cloning a project</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>body</td>
<td>The name, description, and Git repository username, password, and gitURL for the project to be cloned</td>
<td>Request body</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example request body (JSON)

```
{
    "jobId": "1541128617727-10",
    "status": "APPROVED",
    "projectName": "Employee_Rostering",
    "spaceName": "MySpace"
}
```
Example server response (JSON)

```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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project to be compiled</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```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
### Example server response (JSON)

```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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project to be tested</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

### Example server response (JSON)

```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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Type</td>
<td>Requirement</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project to be deployed</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

**Example server response (JSON)**

```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)**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPTED</td>
<td>The request was accepted and is being processed.</td>
</tr>
<tr>
<td>BAD_REQUEST</td>
<td>The request contained incorrect content and was not accepted.</td>
</tr>
<tr>
<td>RESOURCE_NOT_EXIST</td>
<td>The requested resource (path) does not exist.</td>
</tr>
<tr>
<td>DUPLICATERESOURCE</td>
<td>The resource already exists.</td>
</tr>
<tr>
<td>SERVER_ERROR</td>
<td>An error occurred in KIE Server.</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>The request finished successfully.</td>
</tr>
<tr>
<td>FAIL</td>
<td>The request failed.</td>
</tr>
<tr>
<td>APPROVED</td>
<td>The request was approved.</td>
</tr>
<tr>
<td>DENIED</td>
<td>The request was denied.</td>
</tr>
</tbody>
</table>
The job ID for the request could not be found due to one of the following reasons:

- The request was explicitly removed.
- 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.
- The request never existed.

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/business-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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>ID of the job to be retrieved (example: 1541010216919-1)</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

**Example server response (JSON)**

```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.
    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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>ID of the job to be deleted (example: 1541010216919-1)</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>
7.2.4. Branches

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

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

Returns all branches in a specified project and space.

Table 7.16. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space for which you are retrieving projects</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project for which you are retrieving branches</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```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. Build: SUCCESSFUL"
  ]
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches

Adds a specified branch in a specified project and space.

Table 7.17. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>
Table 7.18. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project where the branch is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>branchName</td>
<td>Name of the branch to be deleted</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```json
{
    "jobId": "1576175811421-5",
    "status": "APPROVED",
    "spaceName": "Space123",
    "projectName": "ProjABC",
    "newBranchName": "b1",
    "baseBranchName": "master",
    "userIdentifier": "bc"
}
```
Compiles a specified branch in a specified project and space. If `branchName` is not specified, then request applies to the master branch.

Table 7.19. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spaceName</code></td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td><code>projectName</code></td>
<td>Name of the project where the branch is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td><code>branchName</code></td>
<td>Name of the branch to be compiled</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```json
{
    "jobId": "1576175811233-4",
    "status": "APPROVED",
    "spaceName": "Space123",
    "projectName": "ProjABC",
    "branchName": "b1",
}
```

Installs a specified branch in a specified project and space. If `branchName` is not specified, then request applies to the master branch.

Table 7.20. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spaceName</code></td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td><code>projectName</code></td>
<td>Name of the project where the branch is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td><code>branchName</code></td>
<td>Name of the branch to be installed</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>
Example server response (JSON)

```json
{
    "jobId": "1576175811233-4",
    "status": "APPROVED",
    "spaceName": "Space123",
    "projectName": "ProjABC",
    "branchName": "b1",
}
```

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

Tests a specified branch in a specified project and space. If `branchName` is not specified, then request applies to the master branch.

Table 7.21. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space where the project located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project where the branch is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>branchName</td>
<td>Name of the branch to be tested</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```json
{
    "jobId": "1576175811233-4",
    "status": "APPROVED",
    "spaceName": "Space123",
    "projectName": "ProjABC",
    "branchName": "b1",
}
```

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

Deploys a specified branch in a specified project and space. If `branchName` is not specified, then request applies to the master branch.

Table 7.22. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceName</td>
<td>Name of the space where the project is located</td>
<td>String</td>
<td>Required</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project where the branch is located</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>
**Example server response (JSON)**

```json
{
    "jobId": "1576175811233-4",
    "status": "APPROVED",
    "spaceName": "Space123",
    "projectName": "ProjABC",
    "branchName": "b1",
}
```
Red Hat Process Automation Manager provides a Security Management REST API that you can use to manage the groups, roles, and users in Red Hat Process Automation Manager without using the Business Central user interface. This API support enables you to facilitate and automate management of Business Central groups, roles, users, and granted permissions.

With the Security Management REST API, you can perform the following actions:

- Retrieve information about all groups, roles, users, and their granted permissions
- Create, update, or delete groups and users
- Update granted permissions for groups, roles, and users
- Retrieve information about groups and roles assigned to the users

Security Management REST API requests require the following components:

**Authentication**

The Security Management REST API requires HTTP Basic authentication or token-based authentication for the user role admin. To view configured user roles for your Red Hat Process Automation Manager distribution, navigate to `~/$SERVER_HOME/standalone/configuration/application-roles.properties` and `~/application-users.properties`.

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

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

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

**HTTP headers**

The Security Management 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 Security Management 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
- **PUT**: Updates a resource
- **DELETE**: Deletes a resource
Base URL


NOTE

The REST API base URL for the Security Management, Knowledge Store, and Process Automation Manager controller built into Business Central are the same because all are considered part of Business Central REST services.

Endpoints

Security Management REST API endpoints, such as `/users/{userName}` for a specified user, are the URIs that you append to the Security Management REST API base URL to access the corresponding resource or type of resource in Red Hat Process Automation Manager.

Example request URL for `/users/{userName}` endpoint


Request data

HTTP POST requests in the Security Management 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/business-central/rest/users/newUser/groups`

[ "newGroup"
]

8.1. SENDING REQUESTS WITH THE SECURITY MANAGEMENT REST API USING A REST CLIENT OR CURL UTILITY

The Security Management REST API enables you to manage the groups, roles, and users in Red Hat Process Automation Manager without using the Business Central user interface. You can send Security Management REST API requests using any REST client or curl utility.

Prerequisites

- Business Central is installed and running.
- You have `admin` user role access to Business Central.

Procedure

1. Identify the relevant API endpoint to which you want to send a request, such as `[GET] /groups` to retrieve groups in Business Central.

2. In a REST client or curl utility, enter the following components for a GET request to `/groups`. Adjust any request details according to your use case.
   For REST client:
• **Authentication**: Enter the user name and password of the Business Central user with the `admin` role.

• **HTTP Headers**: Set the following header:
  - **Accept**: `application/json`

• **HTTP method**: Set to `GET`.

• **URL**: Enter the Security Management REST API base URL and endpoint, such as `http://localhost:8080/business-central/rest/groups`.

For curl utility:

• `-u`: Enter the user name and password of the Business Central user with the `admin` role.

• `-H`: Set the following header:
  - **Accept**: `application/json`

• `-X`: Set to `GET`.

• **URL**: Enter the Security Management REST API base URL and endpoint, such as `http://localhost:8080/business-central/rest/groups`.

```bash
curl -u 'baAdmin:password@1' -H "Accept: application/json" -X GET "http://localhost:8080/business-central/rest/groups"
```

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

```
[
  {
    "group1"
  },
  {
    "group2"
  }
]
```

4. In your REST client or curl utility, send another API request with the following components for a **POST** request to `/users/{userName}/groups` to update the groups assigned to a user. Adjust any request details according to your use case.
For REST client:

• **Authentication**: Enter the user name and password of the Business Central user with the `admin` role.

• **HTTP Headers**: Set the following header:
  - **Accept**: `application/json`

  - **Content-Type**: `application/json`

• **HTTP method**: Set to **POST**.
8.2. SUPPORTED SECURITY MANAGEMENT REST API ENDPOINTS

The Security Management REST API provides endpoints for managing groups, roles, users, and permissions in Business Central. It includes the security and permission management tasks that an administrator can also perform using the Security Management page in Business Central.

8.2.1. Groups

- **URL**: Enter the Security Management REST API base URL and endpoint, such as http://localhost:8080/business-central/rest/users/newUser/groups.

- **Request body**: Add a JSON request body with the identification data for the new group:

```json
[
    "newGroup"
]
```

For curl utility:

- **-u**: Enter the user name and password of the Business Central user with the admin role.

- **-H**: Set the following headers:
  - `Accept: application/json`
  - `Content-Type: application/json`

- **-X**: Set to POST.

- **URL**: Enter the Security Management REST API base URL and endpoint, such as http://localhost:8080/business-central/rest/users/newUser/groups.

- **-d**: Add a JSON request body or file (`@file.json`) with the identification data for the new group:

```bash
```

```bash
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X POST "http://localhost:8080/business-central/rest/users/newUser/groups" -d @user-groups.json
```

5. Execute the request and review the KIE Server response.

Example server response (JSON):

```json
{
    "status": "OK",
    "message": "Groups [newGroup] are assigned successfully to user wbadmin"
}
```

If you encounter request errors, review the returned error code messages and adjust your request accordingly.
The Security Management REST API supports the following endpoints for managing groups in Business Central. The Security Management REST API base URL is http://SERVER:PORT/business-central/rest/. All requests require HTTP Basic authentication or token-based authentication for the admin user role.

**[GET] /groups**
Returns all groups in Business Central.

**Example server response (JSON)**
```
[
  {
    "group1"
  },
  {
    "group2"
  }
]
```

**[POST] /groups**
Creates a group in Business Central. A group must have at least one user assigned.

**Table 8.1. Request parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>Name of the group and users assigned to the new group</td>
<td>Request body</td>
<td>Required</td>
</tr>
</tbody>
</table>

**Example request body (JSON)**
```
{
  "name": "groupName",
  "users": [
    "userNames"
  ]
}
```

**Example server response (JSON)**
```
{
  "status": "OK",
  "message": "Group newGroup is created successfully."
}
```

**[DELETE] /groups/{groupName}**
Deletes a specified group from Business Central.

**Table 8.2. Request parameters**
### Example server response (JSON)

```json
{
  "status": "OK",
  "message": "Group newGroup is deleted successfully."
}
```

### 8.2.2. Roles

The Security Management REST API supports the following endpoints for managing roles in Business Central. The Security Management REST API base URL is `http://SERVER:PORT/business-central/rest/`. All requests require HTTP Basic authentication or token-based authentication for the `admin` user role.

**[GET] /roles**

Returns all roles in Business Central.

**Example server response (JSON)**

```json
[
  {
    "name": "process-admin"
  },
  {
    "name": "manager"
  },
  {
    "name": "admin"
  }
]
```

### 8.2.3. Users

The Security Management REST API supports the following endpoints for managing users in Business Central. The Security Management REST API base URL is `http://SERVER:PORT/business-central/rest/`. All requests require HTTP Basic authentication or token-based authentication for the `admin` user role.

**[GET] /users**

Returns all users in Business Central.

**Example server response (JSON)**

```json
[
]
```
"newUser",
"user1",
"user2",
]

[GET] /users/{userName}/groups
Returns all groups assigned to a specified user.

Table 8.3. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>Name of the user for whom you are retrieving assigned groups</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```json
[
  {
    "group1"
  },
  {
    "group2"
  }
]
```

[GET] /users/{userName}/roles
Returns all roles assigned to a specified user.

Table 8.4. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>Name of the user for whom you are retrieving assigned roles</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```json
[
  {
    "name": "process-admin"
  },
  {
    "name": "manager"
  },
  {
    "name": "admin"
  }
]
```
[POST] /users

Creates a specified user with specified roles and groups.

Example request body (JSON)

```
{
   "name": "newUser",
   "roles": [
      "admin",
      "developer"
   ],
   "groups": [
      "group1",
      "group2"
   ]
}
```

Example server response (JSON)

```
{
   "status": "OK",
   "message": "User newUser is created successfully."
}
```

[Post] /users/{userName}/changePassword

Changes the password of a specified user.

Table 8.5. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>Name of the user for whom you are changing the password</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example request command

```
```

Example server response (JSON)

```
{
   "status": "OK",
   "message": "Password for newUser has been updated successfully."
}
```

[DELETE] /users/{userName}

Deletes a specified user from Business Central.

Table 8.6. Request parameters
**Table 8.7. Request parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>Name of the user to be deleted</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

**Example server response (JSON)**

```
{
    "status": "OK",
    "message": "User newUser is deleted successfully."
}
```

**[POST] /users/{userName}/groups**

Overwrites the existing groups assigned to a specified user with new groups.

**Table 8.7. Request parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>Name of the user for whom you are updating groups</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

**Example request body (JSON)**

```
[
    "newGroup"
]
```

**Example server response (JSON)**

```
{
    "status": "OK",
    "message": "Groups [newGroup] are assigned successfully to user wbadmin"
}
```

**[POST] /users/{userName}/roles**

Overwrites the existing roles assigned to a specified user with new roles.

**Table 8.8. Request parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>Name of the user for whom you are updating roles</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

**Example request body (JSON)**
8.2.4. Permissions

The Security Management REST API supports the following endpoints for managing permissions granted to the groups, roles, and users in Business Central. The Security Management REST API base URL is http://SERVER:PORT/business-central/rest/. All requests require HTTP Basic authentication or token-based authentication for the **admin** user role.

**[GET] /groups/{groupName}/permissions**

Returns all permissions granted to a specified group.

**Table 8.9. Request parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>groupName</strong></td>
<td>Name of the group for whom you are retrieving permissions</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

**Example server response (JSON)**

```json
[
  "admin"
]
```

---

```json
{
  "status": "OK",
  "message": "Roles [admin] are assigned successfully to user wbadmin"
}
```
[GET] /roles/{roleName}/permissions

Returns all permissions granted to a specified role.

Table 8.10. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>roleName</td>
<td>Name of the role for whom you are retrieving permissions</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```json
{
  "homePage": "HomePerspective",
  "priority": -10,
  "project": {
    "read": {
      "access": false,
      "exceptions": []
    },
    "create": null,
    "update": null,
    "delete": null,
    "build": null
  },
  "pages": {
    "read": {
      "access": true,
      "exceptions": []
    },
    "build": null
  },
  "workbench": {
    "editDataObject": false,
    "plannerAvailable": false,
    "editGlobalPreferences": false,
    "editProfilePreferences": false,
    "accessDataTransfer": false,
    "jarDownload": true,
    "editGuidedDecisionTableColumns": true
  }
}
```
[GET] /users/{userName}/permissions

Returns all permissions granted to a specified user.

Table 8.11. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>Name of the user for whom you are retrieving permissions</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example server response (JSON)

```json
{
  "homePage": null,
  "priority": null,
  "project": {
    "read": {
      "access": false,
      "exceptions": []
    },
    "build": null
  },
  "workbench": {
    "editDataObject": false,
    "plannerAvailable": false,
    "editGlobalPreferences": false,
    "editProfilePreferences": false,
    "accessDataTransfer": false,
    "jarDownload": true,
    "editGuidedDecisionTableColumns": true
  }
}
```
[Post] /groups/{groupName}/permissions
Updates the permissions of a specified group.

Table 8.12. Request parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName</td>
<td>Name of the group for whom you are updating permissions</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example request body (JSON)

```json
{
   "spaces": {
      "read": {
         "access": true,
         "exceptions": [
            "MySpace"
         ]
      },
      "create": null,
      "update": null,
      "delete": null,
      "build": null
   },
   "editor": {
      "read": {
         "access": false,
         "exceptions": [
            "GuidedDecisionTreeEditorPresenter"
         ]
      },
      "create": null,
      "update": null,
      "delete": null,
      "build": null
   },
   "pages": {
      "read": {
         "access": true,
         "exceptions": []
      },
      "build": null
   },
   "workbench": {
      "editDataObject": false,
      "plannerAvailable": false,
      "editGlobalPreferences": false,
      "editProfilePreferences": false,
      "accessDataTransfer": false,
      "jarDownload": true,
      "editGuidedDecisionTableColumns": true
   }
}
```
Example server response (JSON)

```
{
  "status": "OK",
  "message": "Group newGroup permissions are updated successfully."
}
```

[Post] /roles/{roleName}/permissions
Updates the permissions of a specified role.

Table 8.13. Request parameters
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>roleName</td>
<td>Name of the role for whom you are updating permissions</td>
<td>String</td>
<td>Required</td>
</tr>
</tbody>
</table>

Example request body (JSON)

```json
{
  "homepage": "HomePerspective",
  "priority": 10,
  "pages": {
    "create": true,
    "read": false,
    "delete": false,
    "update": false,
    "exceptions": [{
      "name": "HomePerspective",
      "permissions": {
        "read": true
      }
    }]
  },
  "project": {
    "create": true,
    "read": true,
    "delete": false,
    "update": false,
    "Build": false
  },
  "spaces": {
    "create": true,
    "read": true,
    "delete": false,
    "update": false
  },
  "editor": {
    "read": true
  },
  "workbench": {
    "editDataObject": true,
    "plannerAvailable": true,
    "editGlobalPreferences": true,
    "editProfilePreferences": true,
    "accessDataTransfer": true,
    "jarDownload": true,
    "editGuidedDecisionTableColumns": true
  }
}
```

Example server response (JSON)

```json
{
  "status": "OK",
}
<table>
<thead>
<tr>
<th>Integer value</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than -5</td>
<td>VERY LOW</td>
</tr>
<tr>
<td>Between -5 and 0</td>
<td>LOW</td>
</tr>
<tr>
<td>Equal to 0</td>
<td>NORMAL</td>
</tr>
<tr>
<td>Between 0 and 5</td>
<td>HIGH</td>
</tr>
<tr>
<td>Greater than 5</td>
<td>VERY HIGH</td>
</tr>
</tbody>
</table>

Home Page

Home Page indicates the default landing page for users.

Workbench

Workbench consists of the following defined permissions:

```json
{
    "editDataObject": true,
    "plannerAvailable": true,
    "editGlobalPreferences": true,
    "editProfilePreferences": true,
    "accessDataTransfer": true,
    "jarDownload": true,
    "editGuidedDecisionTableColumns": true
}
```

Pages, Editor, Spaces, and Projects

The following are possible values for the permissions based on the resource type:

- **PAGES**: read, create, update, delete
- **EDITOR**: read
• **SPACES**: read, create, update, delete

• **PROJECT**: read, create, update, delete, build

You can use the following code to add exceptions to **Pages**, **Editor**, **Spaces**, and **Projects** permissions:

```json
{
    "pages": {
        "read": false,
        "exceptions": [
            {
                "resourceName": "ProcessInstances",
                "permissions": {
                    "read": false
                }
            },
            {
                "resourceName": "ProcessDefinitions",
                "permissions": {
                    "read": false
                }
            }
        ]
    }
}
```

The `name` attribute is an identifier of a resource that you add as an exception. Use the following REST API endpoints to get the list of possible identifiers. The REST API base URL is `http://SERVER:PORT/business-central/rest/`.

- **[GET] /perspectives**: Returns perspective names of all pages in Business Central
- **[GET] /editors**: Returns all editors in Business Central
- **[GET] /spaces**: Returns all spaces in Business Central
- **[GET] /spaces/{spaceName}/projects**: Returns projects in a specified space

**Example server response for pages (JSON)**

```json
"pages": {
    "create": true,
    "read": false,
    "exceptions": [
        {
            "name": "HomePerspective",
            "permissions": {
                "read": true
            }
        }
    ]
}
```
CHAPTER 9. EJB API FOR KIE SESSIONS AND TASK SERVICES

Red Hat Process Automation Manager provides an Enterprise JavaBeans (EJB) API that you can use for embedded use cases to access KieSession and TaskService objects remotely from an application. The EJB API enables close transaction integration between the process engine in Red Hat Process Automation Manager and remote customer applications.

Although KIE Server does not support EJB, you can use EJB as a remote protocol for the process engine similar to remote REST or JMS operations with KIE Server.

The implementation of the EJB interface is a single framework-independent and container-agnostic API that you can use with framework-specific code. The EJB services are exposed through the org.jbpm.services.api and org.jbpm.services.ejb packages in Red Hat Process Automation Manager. The implementation does not support the RuleService class, but the ProcessService class exposes an execute method that enables you to use various rule-related commands, such as InsertCommand and FireAllRulesCommand.

NOTE

Contexts and Dependency Injection (CDI) is also supported through the org.jbpm.services.cdi package in Red Hat Process Automation Manager. However, to avoid conflicts in your EJB integration, do not use EJB and CDI together.

9.1. SUPPORTED EJB SERVICES

For the full list of available Enterprise JavaBeans (EJB) services in Red Hat Process Automation Manager, download the Red Hat Process Automation Manager 7.8.0 Maven Repository from the Red Hat Customer Portal and navigate to ~/jboss-rhba-7.8.0.GA-maven-repository/maven-repository/org/jbpm/jbpm-services-ejb-*. The artifacts that provide the EJB interface to the jBPM services are in the following packages:

- org.jbpm.services.ejb.api: Contains extensions of the jBPM services API for the EJB interface
- org.jbpm.services.ejb.impl: Contains EJB wrappers on top of the core service implementation
- org.jbpm.services.ejb.client: Contains the EJB remote client implementation, supported on Red Hat JBoss EAP only

The artifacts that provide the EJB interface to the jBPM services are in the following packages:

- org.jbpm.services.ejb.client: Contains the EJB remote client implementation, supported on Red Hat JBoss EAP only

The org.jbpm.services.ejb.api package contains the following service interfaces that you can use with remote EJB clients:

- **DefinitionServiceEJBRemote**: Use this interface to gather information about processes (ID, name, and version), process variables (name and type), defined reusable subprocesses, domain-specific services, user tasks, and user task inputs and outputs.

- **DeploymentServiceEJBRemote**: Use this interface to initiate deployments and undeployments. The interface includes the methods deploy, undeploy, getRuntimeManager, getDeployedUnits, isDeployed, activate, deactivate, and getDeployedUnit. Calling the deploy method with an instance of DeploymentUnit deploys the unit into the runtime engine by building a RuntimeManager instance. After a successful deployment, an instance of DeployedUnit is created and cached for further use. (To use these methods, you must install the artifacts of the project in a Maven repository.)

- **ProcessServiceEJBRemote**: Use this interface to control the life cycle of one or more processes and work items.
- **RuntimeDataServiceEJBRemote**: Use this interface to retrieve data related to the run time, such as process instances, process definitions, node instance information, and variable information. The interface includes several convenience methods for gathering task information based on owner, status, and time.

- **UserTaskServiceEJBRemote**: Use this interface to control the life cycle of a user task. The interface includes several convenience methods for interacting with user tasks, such as `activate`, `start`, `stop`, and `execute`.

- **QueryServiceEJBRemote**: Use this interface for advanced queries.

- **ProcessInstanceMigrationServiceEJBRemote**: Use this interface to migrate process instances when a new version of a process definition is deployed.

If you run EJB applications and Business Central on the same KIE Server instance, you can synchronize the information between EJB and Business Central at a specified interval by setting the `org.jbpm.deploy.sync.int` system property. After the service finishes the synchronization, you can access the updated information using REST operations.

**NOTE**

EJB services in Red Hat Process Automation Manager are intended for embedded use cases. If you run EJB applications and Business Central on the same KIE Server instance, you must also add the `kie-services` package on the class path of your EJB application.

### 9.2. DEPLOYING AN EJB SERVICES WAR FILE

You can use the Enterprise JavaBeans (EJB) interface to create and deploy an EJB services WAR file that you want to use as part of your Red Hat Process Automation Manager distribution.

**Procedure**

1. Register a human task callback using a startup Java class, such as the following example:

   ```java
   @Singleton
   @Startup
   public class StartupBean {

   @PostConstruct
   public void init() {
     System.setProperty("org.jbpm.ht.callback", "jaas");
   }

   }
   ```

2. Build your EJB project to generate the WAR file according to your project configuration.

3. Deploy the generated file on the Red Hat JBoss EAP instance where Red Hat Process Automation Manager is running.

   Avoid using the `Singleton` strategy for your runtime sessions. The `Singleton` strategy can cause applications to load the same `ksession` instance multiple times from the underlying file system and cause optimistic lock exceptions.

   If you want to deploy the EJB WAR file on a Red Hat JBoss EAP instance separate from the one where Red Hat Process Automation Manager is running, configure your application or the application server to invoke a remote EJB and to propagate the security context.
If you are using Hibernate to create a database schema for Red Hat Process Automation Manager, update the `persistence.xml` file in Business Central and set the value of the `hibernate.hbm2ddl.auto` property to `update` instead of `create`.

4. Test the deployment locally by creating a basic web application and injecting the EJB services, as shown in the following example:

```java
@EJB(lookup = "ejb:/sample-war-ejb-app/ProcessServiceEJBImpl!org.jbpm.services.ejb.api.ProcessServiceEJBRemote")
private ProcessServiceEJBRemote processService;

@EJB(lookup = "ejb:/sample-war-ejb-app/UserTaskServiceEJBImpl!org.jbpm.services.ejb.api.UserTaskServiceEJBRemote")
private UserTaskServiceEJBRemote userTaskService;

@EJB(lookup = "ejb:/sample-war-ejb-app/RuntimeDataServiceEJBImpl!org.jbpm.services.ejb.api.RuntimeDataServiceEJBRemote")
private RuntimeDataServiceEJBRemote runtimeDataService;
```

For more information about developing and deploying EJB applications with Red Hat JBoss EAP, see *Developing EJB Applications*. 
CHAPTER 10. ADDITIONAL RESOURCES

- Managing and monitoring KIE Server
- Packaging and deploying a Red Hat Process Automation Manager project
APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Monday, November 15, 2021.