Red Hat Process Automation Manager 7.7

Custom tasks and work item handlers in Business Central

Last Updated: 2020-05-27

Red Hat Process Automation Manager 7.7 Custom tasks and work item handlers in Business Central

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

Legal Notice

Abstract

This document describes how to create, customize and manage custom tasks and work item handlers in Business Central in Red Hat Process Automation Manager 7.7.

Table of Contents

PREFACE 3

CHAPTER 1. MANAGING SERVICE TASKS IN BUSINESS CENTRAL 4

CHAPTER 2. CREATING WORK ITEM HANDLER PROJECTS 8

CHAPTER 3. WORK ITEM HANDLER PROJECT CUSTOMIZATION 11

CHAPTER 4. WORK ITEM DEFINITIONS 13
4.1. @WID ANNOTATION 13
4.2. TEXT FILE 16

CHAPTER 5. DEPLOYING CUSTOM TASKS 19
5.1. USING A BUSINESS CENTRAL SERVICE TASK REPOSITORY 19
5.2. UPLOADING JAR ARTIFACT TO BUSINESS CENTRAL 19
5.3. MANUALLY COPYING WORK ITEM DEFINITIONS TO BUSINESS CENTRAL MAVEN REPOSITORY 19

CHAPTER 6. REGISTERING CUSTOM TASKS 20
6.1. REGISTERING CUSTOM TASKS USING THE DEPLOYMENT DESCRIPTOR INSIDE BUSINESS CENTRAL 20
6.2. REGISTERING CUSTOM TASKS USING THE DEPLOYMENT DESCRIPTOR OUTSIDE BUSINESS CENTRAL 21

CHAPTER 7. PLACING CUSTOM TASKS 22

APPENDIX A. VERSIONING INFORMATION 23

PREFACE

As a business rules developer, you can create custom tasks and work item handlers in Business Central to execute custom code within your process flows and extend the operations available for use in Red Hat Process Automation Manager. You can use custom tasks to develop operations that Red Hat Process Automation Manager does not directly provide and include them in process diagrams.

In Business Central, each task in a process diagram has a **WorkItem** Java class with an associated **WorkItemHandler** Java class. The work item handler contains Java code registered with Business Central and implements **org.kie.api.runtime.process.WorkItemHandler**.

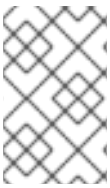
The Java code of the work item handler is executed when the task is triggered. You can customize and register a work item handler to execute your own Java code in custom tasks.

Prerequisites

- Business Central is deployed and is running on a web or application server.
- You are logged in to Business Central.
- Maven is installed.
- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.
- Your system has access to the Red Hat Maven repository either locally or online

CHAPTER 1. MANAGING SERVICE TASKS IN BUSINESS CENTRAL

Service tasks (work items) are tasks that you can customize and reuse across multiple business processes or across all projects in Business Central. Red Hat Process Automation Manager provides a set of service tasks within the service task repository in Business Central. You can enable or disable the default service tasks and upload custom service tasks into Business Central to implement the tasks in the relevant processes.



NOTE

Red Hat Process Automation Manager includes a limited set of supported custom tasks. Custom tasks that are not included in Red Hat Process Automation Manager are not supported.

Procedure

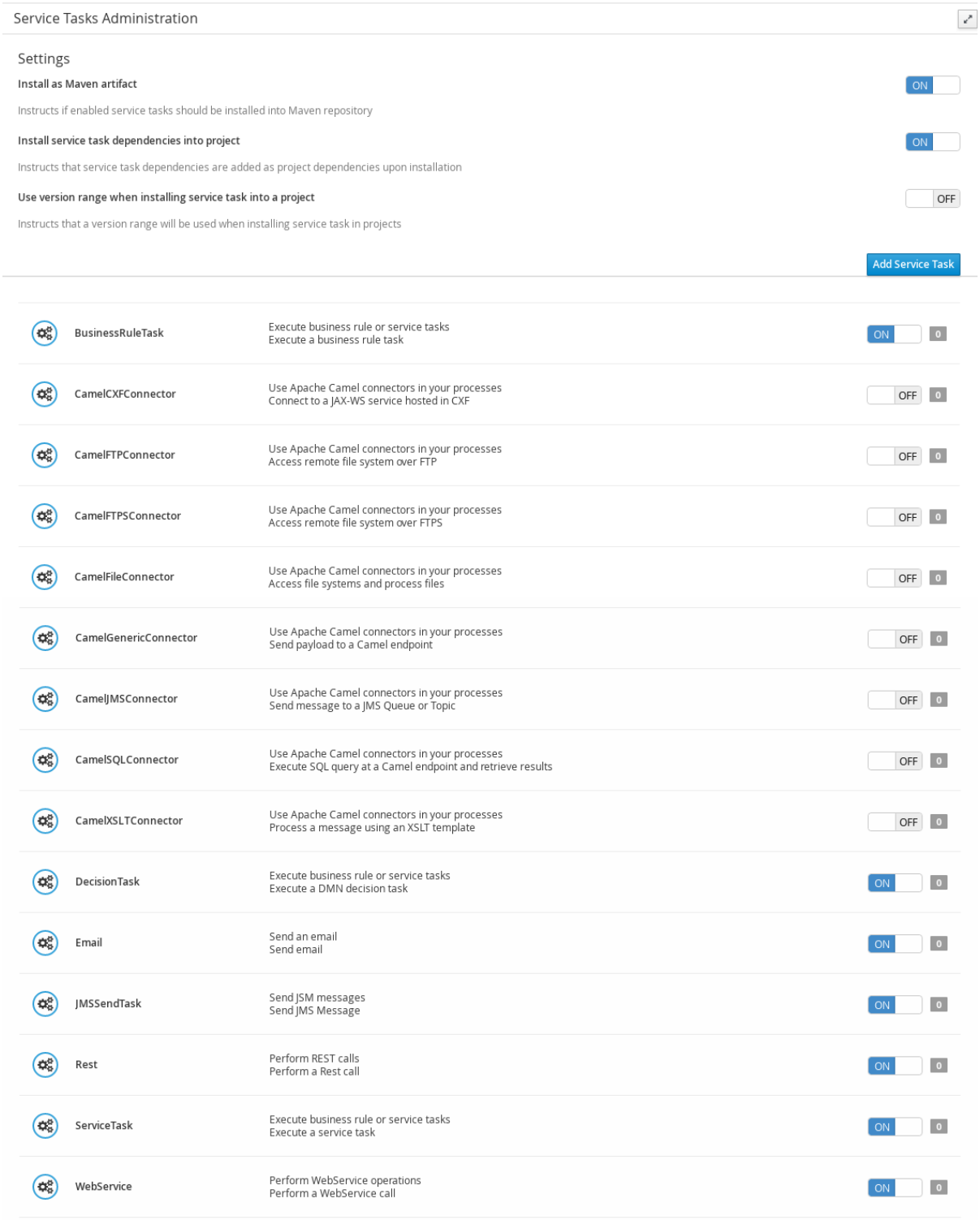
1. In Business Central, select the **Admin** icon in the top-right corner of the screen and select **Service Tasks Administration**.
   This page lists the service task installation settings and available service tasks for processes in projects throughout Business Central. The service tasks that you enable on this page become available in the project-level settings where you can then install each service task to be used in processes. The way in which the service tasks are installed in a project is determined by the global settings that you enable or disable under **Settings** on this **Service Tasks Administration** page.

2. Under **Settings**, enable or disable each setting to determine how the available service tasks will be implemented when a user installs them at the project level.
   The following service task settings are available:

   - **Install as Maven artifact** Uploads the service task JAR file to the Maven repository that is configured with Business Central, if the file is not already present.
   - **Install service task dependencies into project** Adds any service task dependencies to the **pom.xml** file of the project where the task is installed.
   - **Use version range when installing service task into project** Uses a version range instead of a fixed version of a service task that is added as a project dependency. Example: **[7.16,)** instead of **7.16.0.Final**

3. Enable or disable (set to **ON** or **OFF**) any available service tasks as needed. Service tasks that you enable will be displayed in project-level settings for all projects in Business Central.

Figure 1.1. Enable service tasks and service task settings



4. To add a custom service task, click **Add Service Task**, browse to the relevant JAR file, and click the **Upload** icon. The JAR file must contain work item handler implementations annotated with **@Wid**.

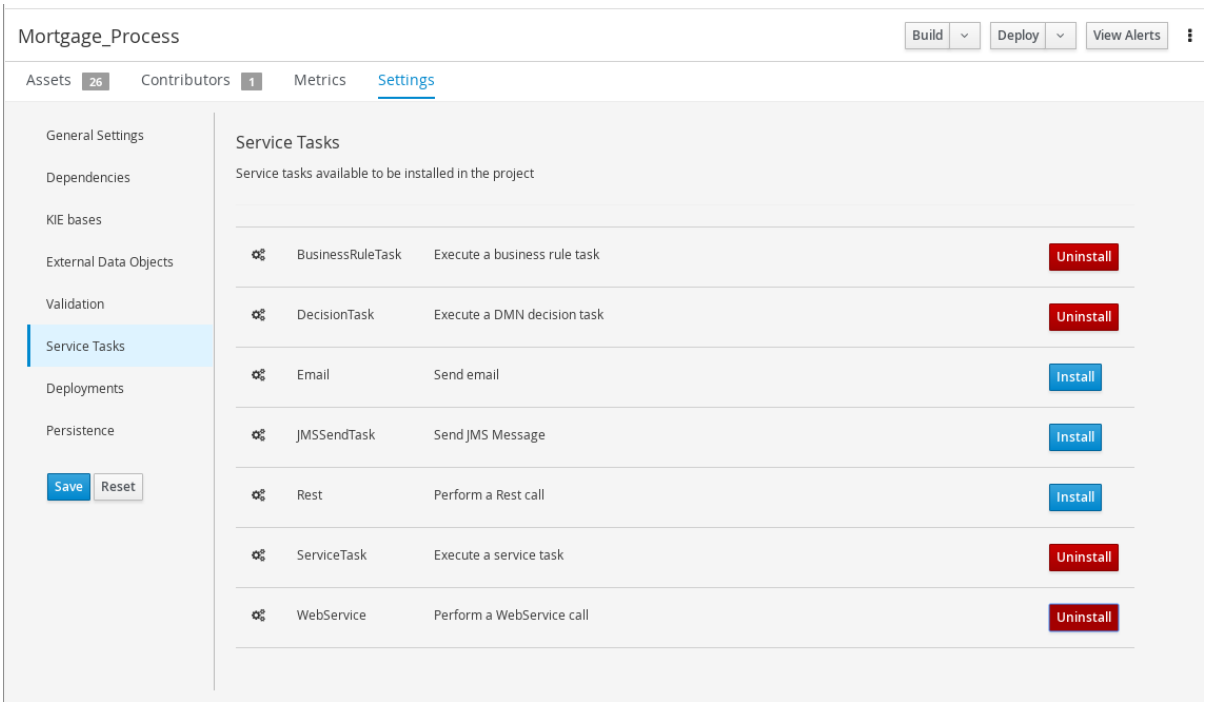
5. Optionally, to remove a service task, click **remove** on the row of the service task you want to remove and click **Ok** to confirm removal.

6. After you configure all required service tasks, navigate to a project in Business Central and go to the project **Settings** → **Service Tasks** page to view the available service tasks that you enabled.

7. For each service task, click **Install** to make the task available to the processes in that project or click **Uninstall** to exclude the task from the processes in the project.

8. If you are prompted for additional information when you install a service task, enter the required information and click **Install** again.
   The required parameters for the service task depend on the type of task. For example, rule and decision tasks require artifact GAV information (Group ID, Artifact ID, Version), email tasks require host and port access information, and REST tasks require API credentials. Other service tasks might not require any additional parameters.

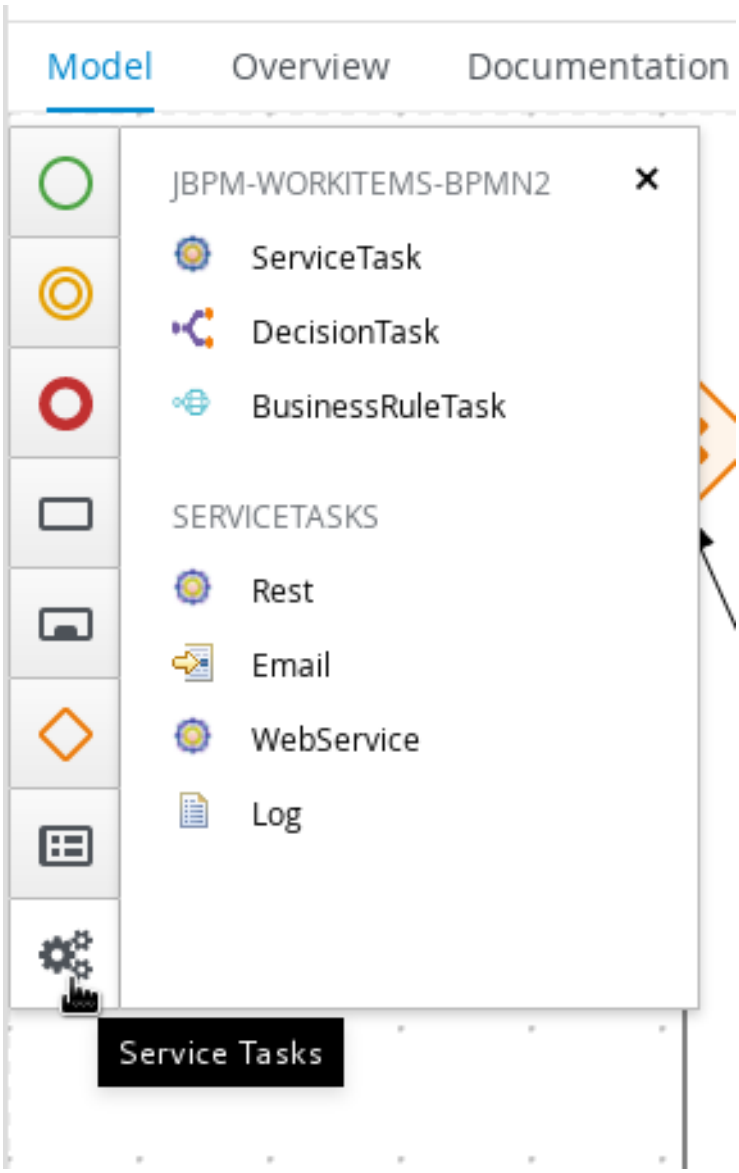
Figure 1.2. Install service tasks for use in processes



9. Click **Save**.

10. Return to the project page, select or add a business process in the project, and in the process designer palette, select the **Service Tasks** option to view the available service tasks that you enabled and installed:

Figure 1.3. Access installed service tasks in process designer



CHAPTER 2. CREATING WORK ITEM HANDLER PROJECTS

Create the software project to contain all configurations, mappings, and executable code for the custom task.

You can create a work item handler from scratch or use a Maven archetype to create an example project. Red Hat Process Automation Manager provides the **jbpm-workitems-archetype** from the Red Hat Maven repository for this purpose.

Procedure

1. Open the command line and create a directory where you will build your work item handler such as **workitem-home**:

```
$ mkdir workitem-home
```

2. Check the Maven **settings.xml** file and ensure that the Red Hat Maven repository is included in the repository list.

NOTE

Setting up Maven is outside the scope of this guide.

For example, to add the online Red Hat Maven repository to your Maven **settings.xml** file:

```
<settings>
  <profiles>
    <profile>
      <id>my-profile</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
        <repository>
          <id>redhat-ga</id>
          <url>http://maven.repository.redhat.com/ga/</url>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
          <releases>
            <enabled>true</enabled>
          </releases>
        </repository>
        ...
      </repositories>
    </profile>
  </profiles>
  ...
</settings>
```

3. Find the Red Hat library version and perform one of the following tasks:

   - To find the library version online, see What is the mapping between Red Hat Process Automation Manager and the Maven library version?.

- To find the library version offline, check **Implementation-Version** in **business-central.war/META-INF/MANIFEST.MF** or **Implementation-Version** in **kie-server.war/META-INF/MANIFEST.MF**.

4. In the **workitem-home** directory, execute the following command:

```
$ mvn archetype:generate \
-DarchetypeGroupId=org.jbpm \
-DarchetypeArtifactId=jbpm-workitems-archetype \
-DarchetypeVersion=<redhat-library-version> \
-Dversion=1.0.0-SNAPSHOT \
-DgroupId=com.redhat \
-DartifactId=myworkitem \
-DclassPrefix=MyWorkItem
```

Table 2.1. Parameter descriptions

Parameter	Description
-DarchetypeGroupId	Specific to the archetype and must remain unchanged.
-DarchetypeArtifactId	Specific to the archetype and must remain unchanged.
-DarchetypeVersion	Red Hat library version that is searched for when Maven attempts to download the **jbpm-workitems-archetype** artifact.
-Dversion	Version of your specific project. For example, **1.0.0-SNAPSHOT**.
-DgroupId	Maven group of your specific project. For example, **com.redhat**.
-DartifactId	Maven ID of your specific project. For example, **myworkitem**.
-DclassPrefix	String added to the beginning of Java classes when Maven generates the classes for easier identification. For example, **MyWorkItem**.

A **myworkitem** folder is created in the **workitem-home** directory. For example:

```
assembly/
  assembly.xml
src/
  main/
    java/
      com/
        redhat/
          MyWorkItemWorkItemHandler.java
    repository/
    resources/
```

```
  test/
    java/
      com/
        redhat/
          MyWorkItemWorkItemHandlerTest.java
          MyWorkItemWorkItemIntegrationTest.java
    resources/
      com/
        redhat/
pom.xml
```

5. Add any Maven dependencies required by the work item handler class to the **pom.xml** file.
6. To create a deployable JAR for this project, in the parent project folder where the pom.xml file is located, execute the following command:

```
$ mvn clean package
```

Several files are created in the **target**/ directory which include the following two main files:

Table 2.2. File descriptions

Parameter	Description
myworkitems-<version>.jar	Used for direct deployment to Red Hat Process Automation Manager.
myworkitems-<version>.zip	Used for deployment using a service repository.

CHAPTER 3. WORK ITEM HANDLER PROJECT CUSTOMIZATION

You can customize the code of a work item handler project. There are two Java methods required by a work item handler, **executeWorkItem** and **abortWorkItem**.

Table 3.1. Java method descriptions

Java Method	Description
executeWorkItem(WorkItem workItem, WorkItemManager manager)	Executed by default when the work item handler is run.
abortWorkItem(WorkItem workItem, WorkItemManager manager)	Executed when the work item is aborted.

In both methods, the **WorkItem** parameter contains any of the parameters entered into the custom task through a GUI or API call, and the **WorkItemManager** parameter is responsible for tracking the state of the custom task.

Example code structure

```
public class MyWorkItemWorkItemHandler extends AbstractLogOrThrowWorkItemHandler {

  public void executeWorkItem(WorkItem workItem, WorkItemManager manager) {
    try {
      RequiredParameterValidator.validate(this.getClass(), workItem);

      // sample parameters
      String sampleParam = (String) workItem.getParameter("SampleParam");
      String sampleParamTwo = (String) workItem.getParameter("SampleParamTwo");

      // complete workitem impl...

      // return results
      String sampleResult = "sample result";
      Map<String, Object> results = new HashMap<String, Object>();
      results.put("SampleResult", sampleResult);
      manager.completeWorkItem(workItem.getId(), results);
    } catch(Throwable cause) {
      handleException(cause);
    }
  }

  @Override
  public void abortWorkItem(WorkItem workItem, WorkItemManager manager) {
    // similar
  }
}
```

Table 3.2. Parameter descriptions

Parameter	Description
RequiredParameterValidator.validate(this.getClass(), workItem);	Checks that all parameters marked “required” are present. If they are not, an **IllegalArgumentException** is thrown.
String sampleParam = (String) workItem.getParameter("SampleParam");	Example of getting a parameter from the **WorkItem** class. The name is always a string. For example, **WorkItem**. In the example, **SampleParam** is always a string but the object associated with it can be many things and require a cast in order to avoid errors.
// **complete workitem impl…**	Executes the custom Java code when a parameter is received.
results.put("SampleResult", sampleResult);	Passes results to the custom task. The results are placed in the data output areas of the custom task.
manager.completeWorkItem(workItem.getId(), results);	Marks the work item handler as complete. The **WorkItemManager** controls the state of the work item and is responsible for getting the **WorkItem** ID and associate the results with the correct custom task.
abortWorkItem()	Aborts the custom Java code. May be left blank if the work item is not designed to be aborted

NOTE

Red Hat Process Automation Manager includes a limited set of supported custom tasks. Custom tasks that are not included in Red Hat Process Automation Manager are not supported.

CHAPTER 4. WORK ITEM DEFINITIONS

Red Hat Process Automation Manager requires a work item definition (WID) file to identify the data fields to show in Business Central and accept API calls. The WID file is a mapping between user interactions with Red Hat Process Automation Manager and the data that is passed to the work item handler. The WID file also handles the UI details such as the name of the custom task, the category it is displayed as on the palette in Business Central, the icon used to designate the custom task, and the work item handler the custom task will map to.

In Red Hat Process Automation Manager you can create a WID file in two ways:

- Use a **@Wid** annotation when coding the work item handler.
- Create a **.wid** text file. For example, **definitions-example.wid**.

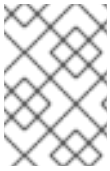
4.1. @WID ANNOTATION

The **@Wid** annotation is automatically created when you generate a work item handler project using the Maven archetype. You can also add the annotation manually.

@Wid Example

```
@Wid(widfile="MyWorkItemDefinitions.wid",
    name="MyWorkItemDefinitions",
    displayName="MyWorkItemDefinitions", icon="",
    defaultHandler="mvel: new
    com.redhat.MyWorkItemWorkItemHandler()",
    documentation = "myworkitem/index.html",
    parameters={
      @WidParameter(name="SampleParam", required = true),
      @WidParameter(name="SampleParamTwo", required = true)
    }, results={
      @WidResult(name="SampleResult")
    },
    mavenDepends={
      @WidMavenDepends(group="com.redhat",
      artifact="myworkitem",
      version="7.26.0.Final-example-00004")
    },
    serviceInfo={
        @WidService(
                category = "myworkitem",
                description = "${description}",
                keywords = "",
                action = @WidAction(title = "Sample Title"),
                authinfo = @WidAuth(required = true, params = {"SampleParam", "SampleParamTwo"},
                        paramsdescription = {"SampleParam", "SampleParamTwo"},
                        referencesite = "referenceSiteURL")
        )
    }
)
```

Table 4.1. @Wid descriptions

	Description
@Wid	Top-level annotation to auto-generate WID files.
widfile	Name of the file that is automatically created for the custom task when it is deployed in Red Hat Process Automation Manager.
name	Name of the custom task, used internally. This name must be unique to custom tasks deployed in Red Hat Process Automation Manager.
displayName	Displayed name of the custom task. This name is displayed in the palette in Business Central.
icon	Path from **src/main/resources/** to an icon located in the current project. The icon is displayed in the palette in Business Central. The icon, if specified, must be a PNG or GIF file and 16x16 pixels. This value can be left blank to use a default "Service Task" icon.
description	Description of the custom task.
defaultHandler	The work item handler Java class that is linked to the custom task. This entry is in the format **<language> : <class>**. Red Hat Process Automation Manager recommends using **mvel** as the language value for this attribute but **java** can also be used. For more information about mvel, see MVEL Documentation.
documentation	Path to an HTML file in the current project that contains a description of the custom task.
@WidParameter	Child annotation of **@Wid**. Specifies values that will be populated in the Business Central GUI or expected by API calls as data inputs for the custom task. More than one parameter can be specified: **name** - A name for the parameter.  NOTE Due to the possibility of this name being used in API calls over transfer methods such as REST or SOAP, this name should not contain spaces or special characters. **required** - Boolean value indicating whether the parameter is required for the custom task to execute.

	Description
@WidResult	Child annotation of **@Wid**. Specifies values that will be populated in the Business Central GUI or expected by API calls as data outputs for the custom task. You can specify more than one result: **name** - A name for the result.  NOTE Due to the possibility of this name being used in API calls over transfer methods such as REST or SOAP, this name should not contain spaces or special characters. **@WidMavenDepends** - Child annotation of**@Wid**. Specifies Maven dependencies that will be required for the correct functioning of the work item handler. You can specify more than one dependency: **group** - Maven group ID of the dependency. **artifact** - Maven artifact ID of the dependency. **version** - Maven version number of the dependency.
@WidService	Child annotation of **@Wid**. Specifies values that will be populated in the service repository. **category** - The UI palette category that the handler will be placed. This value should match the **category** field of the **@Wid** annotation. **description** - Description of the handler that will be displayed in the service repository. **keywords** - Comma-separated list of keywords that apply to the handler. Note: Currently not used by the Business Central service repository. **action** - The **@WidAction** object that describes the handler purpose. Contains the fields **title** and **description**. **authinfo** - The **@WidAuth** object that defines the authentication requirement. Optional. Contains the fields **required**, **params**, **paramsdescription**, **referencesite**.
@WidAction	Object of **@WidService** that describes the handler purpose. **title** - The title for the handler action. **description** - The description for the handler action.

	Description
@WidAuth	Object of **@WidService** that defines the authentication required by the handler. **required** - The boolean value that determines whether authentication is required. **params** - The array containing the authentication parameters required. **paramsdescription** - The array containing the descriptions for each authentication parameter. **referencesite** - The URL to where the handler documentation can be found. Note: Currently not used by the Business Central service repository.

4.2. TEXT FILE

A global **WorkDefinitions** WID text file is automatically generated by new projects when a business process is added. The WID text file is similar to the JSON format but is not a completely valid JSON file. You can open this file in Business Central. You can create additional WID files by selecting **Add Asset** > **Work item definitions** from an existing project.

Text file example

```
[
  [
    "name" : "MyWorkItemDefinitions",
    "displayName" : "MyWorkItemDefinitions",
    "category" : "",
    "description" : "",
    "defaultHandler" : "mvel: new com.redhat.MyWorkItemWorkItemHandler()",
    "documentation" : "myworkitem/index.html",
    "parameters" : [
      "SampleParam" : new StringDataType(),
      "SampleParamTwo" : new StringDataType()
    ],
    "results" : [
      "SampleResult" : new StringDataType()
    ],
    "mavenDependencies" : [
      "com.redhat:myworkitem:7.26.0.Final-example-00004"
    ],
    "icon" : ""
  ]
]
```

The file is structured as a plain-text file using a JSON-like structure. The filename extension is **.wid**.

Table 4.2. Text file descriptions

	Description
name	Name of the custom task, used internally. This name must be unique to custom tasks deployed in Red Hat Process Automation Manager.
displayName	Displayed name of the custom task. This name is displayed in the palette in Business Central.
icon	Path from **src/main/resources/** to an icon located in the current project. The icon is displayed in the palette in Business Central. The icon, if specified, must be a PNG or GIF file and 16x16 pixels. This value can be left blank to use a default "Service Task" icon.
category	Name of a category within the Business Central palette under which this custom task is displayed.
description	Description of the custom task.
defaultHandler	The work item handler Java class that is linked to the custom task. This entry is in the format **<language> : <class>**. Red Hat Process Automation Manager recommends using **mvel** as the language value for this attribute but **java** can also be used. For more information about mvel, see MVEL Documentation.
documentation	Path to an HTML file in the current project that contains a description of the custom task.
parameters	Specifies the values to be populated in the Business Central GUI or expected by API calls as data inputs for the custom task. Parameters use the **<key> : <DataType>** format. Accepted data types are **StringDataType()**, **IntegerDataType()**, and **ObjectDataType()**. More than one parameter can be specified.
results	Specifies the values to be populated in the Business Central GUI or expected by API calls as data outputs for the custom task. Results use the **<key> : <DataType>** format. Accepted data types are **StringDataType()**, **IntegerDataType()**, and **ObjectDataType()**. More than one result can be specified.
mavenDependencies	Optional: Specifies Maven dependencies required for the correct functioning of the work item handler. Dependencies can also be specified in the work item handler **pom.xml** file. Dependencies are in the format **<group>:<artifact>:<version>**. More than one dependency may be specified

Red Hat Process Automation Manager tries to locate a ***.wid** file in two locations by default:

- Within Business Central in the project's top-level **global/** directory. This is the location of the default **WorkDefinitions.wid** file that is created automatically when a project first adds a business process asset.

- Within Business Central in the project's **src/main/resources/** directory. This is where WID files created within a project in Business Central will be placed. A WID file may be created at any level of a Java package, so a WID file created at a package location of **<default>** will be created directly inside **src/main/resources/** while a WID file created at a package location of **com.redhat** will be created at **src/main/resources/com/redhat/**

WARNING

Red Hat Process Automation Manager does not validate that the value for the **defaultHandler** tag is executable or is a valid Java class. Specifying incorrect or invalid classes for this tag will return errors.

CHAPTER 5. DEPLOYING CUSTOM TASKS

Work item handlers, as custom code, are created outside of Red Hat Process Automation Manager. To use the code in your custom task, the code must be deployed to the server. Work item handler projects must be Java JAR files that can be placed into a Maven repository.

In Red Hat Process Automation Manager you can deploy custom tasks using three methods:

- Within a Business Central service task repository. For more information, see Chapter 1, *Managing service tasks in Business Central* .
- Within Business Central where you can use both the legacy and current editors to upload the work item handler JAR to the Business Central Maven repository as an artifact.
- Outside of Business Central, you can manually copy the JAR files into the Maven repository.

5.1. USING A BUSINESS CENTRAL SERVICE TASK REPOSITORY

You can enable, disable, and deploy service tasks within a Business Central service task repository. For more information, see Chapter 1, *Managing service tasks in Business Central* .

5.2. UPLOADING JAR ARTIFACT TO BUSINESS CENTRAL

You can upload the work item handler JAR to the Business Central Maven repository as an artifact by using the legacy and current editors.

Procedure

1. In Business Central, select the **Admin** icon in the top-right corner of the screen and select **Artifacts**.
2. Click **Upload**.
3. In the **Artifact Upload** window, click the **Choose File** icon.
4. Navigate to the location of the work item handler JAR, select the file and click **Open**.
5. In the pop-up dialog, click the **Upload** icon.
   The artifact is uploaded and can now be viewed on the **Artifacts** page and referenced.

5.3. MANUALLY COPYING WORK ITEM DEFINITIONS TO BUSINESS CENTRAL MAVEN REPOSITORY

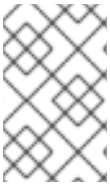
Business Central automatically creates or reuses the Maven repository folder. By default the location is based on the location of the user launched Red Hat JBoss EAP. For example, the full default path would be **<startup location>/repositories/kie/global**. It is possible to replicate a standard Maven repository folder layout of **<groupId>/<artifactId>/<versionId>/** in this folder and copy work item handler JAR files to this location. For example:

```
<startup location>/repositories/kie/global/com/redhat/myworkitem/1.0.0-SNAPSHOT/myworkitems-1.0.0-SNAPSHOT.jar
```

Any artifacts copied in this fashion are available to Red Hat Process Automation Manager without a server restart. Viewing the artifact in the Business Central **Artifacts** page requires clicking **Refresh**.

CHAPTER 6. REGISTERING CUSTOM TASKS

Red Hat Process Automation Manager must know how to associate a custom task work item with the code executed by the work item handler. The work item definition file links the custom task with the work item handler by name and Java class. The work item handler's Java class has to be registered as usable in Red Hat Process Automation Manager.



NOTE

Service repositories contain domain-specific services that provide integration of your processes with different types of systems. Registering a custom task is not necessary when using a service repository because the import process registers the custom task.

Red Hat Process Automation Manager creates a WID file by default for projects that contain at least one business process. You can create a WID file when registering a work item handler or edit the default WID file. For more information about WID file locations or formatting, see Chapter 4, *Work item definitions*.

For non-service repository deployments, work item handlers can be registered in two ways:

- Registering using the deployment descriptor.
- Registering using the spring component registration.

6.1. REGISTERING CUSTOM TASKS USING THE DEPLOYMENT DESCRIPTOR INSIDE BUSINESS CENTRAL

You can register a custom task work item with the work item handler using the deployment descriptor in Business Central.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and select the project name.
2. In the project pane, select **Settings → Deployments → Work Item Handlers**.
3. Click **Add Work Item Handler**.
4. In the **Name** field, enter the display name for the custom task.
5. From the **Resolver** list, select **MVEL**, **Reflection** or **Spring**.
6. In the **Value** field, enter the value based on the resolver type:
   - For MVEL, use the format **new <full Java package>.<Java work item handler class name>()**
     Example: **new com.redhat.MyWorkItemWorkItemHandler()**
   - For Reflection, use the format **<full Java package>.<Java work item handler class name>**
     Example: **com.redhat.MyWorkItemWorkItemHandler**
   - For Spring, use the format **<Spring bean identifier>**
     Example: **workItemSpringBean**

7. Click **Save** to save your changes

6.2. REGISTERING CUSTOM TASKS USING THE DEPLOYMENT DESCRIPTOR OUTSIDE BUSINESS CENTRAL

You can register a custom task work item with the work item handler using the deployment descriptor outside Business Central.

Procedure

1. Open the file **src/main/resources/META-INF/kie-deployment-descriptor.xml**.
2. Add the following content based on the resolver type under **<work-item-handlers>**:
   - For MVEL, add the following:

```
<work-item-handler>
  <resolver>mvel</resolver>
  <identifier>new com.redhat.MyWorkItemWorkItemHandler()</identifier>
  <parameters/>
  <name>MyWorkItem</name>
</work-item-handler>
```

   - For Reflections, add the following:

```
<work-item-handler>
  <resolver>reflection</resolver>
  <identifier>com.redhat.MyWorkItemWorkItemHandler</identifier>
  <parameters/>
  <name>MyWorkItem</name>
</work-item-handler>
```

   - For Spring, add the following and ensure the identifier is the identifier of a Spring bean:

```
<work-item-handler>
  <resolver>spring</resolver>
  <identifier>beanIdentifier</identifier>
  <parameters/>
  <name>MyWorkItem</name>
</work-item-handler>
```

NOTE

If you are using Spring to discover and configure Spring beans, it is possible to use an annotation of the **org.springframework.stereotype.Component** class to automatically register work item handlers.

Within a work item handler, add the annotation **@Component("<Name>")** before the declaration of the work item handler class. For example:
@Component("MyWorkItem") public class MyWorkItemWorkItemHandler extends AbstractLogOrThrowWorkItemHandler {

CHAPTER 7. PLACING CUSTOM TASKS

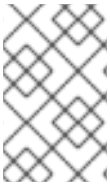
When a custom task is registered in Red Hat Process Automation Manager it appears in the process designer palette. The custom task is named and categorized according to the entries in its corresponding WID file.

Prerequisites

- A custom task is registered in Red Hat Process Automation Manager. For information about registering custom tasks, see Chapter 6, *Registering custom tasks*.
- The custom task is named and categorized according to the corresponding WID file. For more information about WID file locations or formatting, see Chapter 4, *Work item definitions*.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click a project.
2. Select the business process that you want to add a custom task to.
3. Select the custom task from the palette and drag it to the BPMN2 diagram.
4. Optional: Change the custom task attributes. For example, change the data output and input from the corresponding WID file.



NOTE

If the WID file is not visible in your project and no **Work Item Definition** object is visible in the **Others** category of your project, you must register the custom task. For more information about registering a custom task, see Chapter 6, *Registering custom tasks*.

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Monday, May 25, 2020.