Workflow Guide

/ ForgeRock Identity Management 7.1

Latest update: 7.1.6

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to enabling and using ForgeRock® Identity Management workflows.

Table of Contents

Overview

IDM provides an embedded workflow and business process engine based on Flowable and the Business Process Model and Notation (BPMN) 2.0 standard. This guide describes how to configure the workflow engine, and how to manage workflow tasks and processes using the REST interface and the Admin UI.

Quick Start

 About Workflow Learn about BPMN 2.0 and the workflow tools.	 Enable Workflows Configure the workflow engine and datasource to enable workflows.
 Invoke Workflows Learn where and how you can trigger workflows.	 Customize Workflows Create custom workflow templates.

Note

Workflows are not supported with a DS repository. If you are using a DS repository for IDM data, you must configure a separate JDBC repository as the workflow datasource.

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

The ForgeRock Common REST API works across the platform to provide common ways to access web resources and collections of resources.

Chapter 1
BPMN 2.0 and Workflow Tools

Business Process Model and Notation 2.0 is the result of consensus among Business Process Management (BPM) system vendors. The Object Management Group (OMG) has developed and maintained the BPMN standard since 2004.

BPMN 2.0 lets you add artifacts that describe your workflows and business processes to IDM, for provisioning and other purposes. You can create workflow definitions using a text editor or the ForgeRock Workflow Editor UI. The ForgeRock Workflow Editor UI is an open-source project that is not supported or included in IDM. This visual workflow editor can be set up as a standalone *node* server or integrated directly with IDM.

Scripts inside BPMN 2.0 XML files have access to the following scripting variables:

- `openidm`
- `identityServer`
- `console`

For example, to log a message with Groovy:

```
console.log('my message')
```

Note

For more information about the graphical notations and XML representations for events, flows, gateways, tasks, process constructs, and more, see BPMN 2.0 Constructs.

IDM does not support the following constructs:

- Mule task
- Camel task

The following terms are reserved and cannot be used as variable names: `out`, `out:print`, `lang:import`, `context`, and `elcontext`.

Chapter 2

Enable Workflows

IDM embeds a Flowable Process Engine that starts in the OSGi container. Workflows are not active by default. IDM needs two configuration files to activate the workflow bundle:

- `workflow.json` specifies the configuration of the Flowable engine, including the data source that the engine will use.
- `datasource.jdbc-default.json` the default data source for Flowable.

To enable workflows:

When you enable workflows in the Admin UI, IDM creates the `workflow.json` file in your project's `conf/` directory.

1. Log in to the Admin UI.
2. From the navigation bar, select Configure > System Preferences.
3. On the System Preferences page, click the Workflow tab.
4. Enable the display of workflows, and click Save.
5. Optionally, "Configure the Workflow Engine".
6. "Configure the Workflow Data Source".

Configure the Workflow Engine

IDM creates the default `workflow.json` file with the following structure:

```
{
    "useDataSource" : "default",
    "workflowDirectory" : "&{idm.instance.dir}/workflow",
    "userResource": {
        "path": "managed/user",
        "queryFilter": "/userName eq \"${username}\""
    },
    "groupResource": {
        "path": "managed/group",
        "queryFilter": "/id eq \"${gid}\""
    }
}
```

`useDataSource`

Specifies the datasource configuration file that points to the repository where Flowable should store data.

By default, this is the `datasource.jdbc-default.json` file. For information about changing the data store that Flowable uses, see "Configure the Workflow Data Source".

`workflowDirectory`

Specifies the location where IDM expects to find workflow processes. By default, IDM looks for workflow processes in the `project-dir/workflow` directory.

In addition to these default properties, you can configure the Flowable engine `history` level:

```
{
    "history" : "audit"
}
```

When a workflow is executed, information can be logged as determined by the history level. The history level can be one of the following:

- `none` This level results in the best performance for workflow execution, but no historical information is retained.
- `activity` Logs all process instances and activity instances, without details.
- `audit` This is the default level. All process instances, activity instances, and submitted form properties are logged so that all user interaction through forms is traceable and can be audited.
- `full` This is the highest level of history logging and has the greatest performance impact. This history level stores all the information that is stored for the `audit` level, as well as any process variable updates.

Configure Workflow Email

Workflows can send an email using the following methods:

Flowable email tasks

To use workflow email tasks, add the email configuration to `workflow.json`. Example email configuration:

```
"mail" : {
    "host" : "mail.example.com",
    "port" : 1025,
    "username" : "username",
    "password" : "password",
    "useSSL" : false,
    "starttls" : true,
    "defaultFrom" : "workflow@example.com",
    "forceTo" : "overrideSendToEmail@example.com"
}
```

`scriptTask`

Emails sent using `scriptTask` in the *External Services Guide* utilize IDM's email client configuration. in the *External Services Guide*

Example script:

```
openidm.action("external/email", "send", { "to": "bob@example.com" }, { waitForCompletion: true });
```

Configure the Workflow Data Source

The Flowable engine requires a JDBC database. The connection details to the database are specified in the `datasource.jdbc-default.json` file. If you are using a JDBC repository for IDM data, you will already have a `datasource.jdbc-default.json` file in your project's `conf/` directory. In this case, when you enable workflows, IDM uses the existing JDBC repository and creates the required Flowable tables in that JDBC repository.

Important

If you are using a DS repository for IDM data, you must configure a separate JDBC repository as the workflow datasource. For more information, see "*Select a Repository*" in the *Installation Guide*.

To specify that Flowable should use a data source that is separate from your existing IDM JDBC repository, create a new datasource configuration file in your project's `conf/` directory (for example, `datasource.jdbc-flowable.json`) with the connection details to the separate data source. Then, reference that file in the `useDataSource` property of the `workflow.json` file (for example, `"useDataSource" : "flowable"`.

For more information about the fields in this file, see "JDBC Connection Configuration" in the *Object Modeling Guide*.

Custom Workflow Object Mapping

For custom object mapping, edit the default `workflow.json` configuration:

```
"userResource": {
    "path": "managed/user",
    "queryFilter": "/userName eq \"${username}\""
},
"groupResource": {
    "path": "managed/group",
    "queryFilter": "/id eq \"${gid}\""
}
```

Note

Do not replace `${username}` or `${gid}` in the queryFilter; for example:

- OK: `"queryFilter": "/callSign eq \"${username}\""`

- *NOT* OK: `"queryFilter": "/callSign eq \"${callsign}\""`

Chapter 3
Test Workflow Integration

IDM reads workflow definitions from the `/path/to/openidm/workflow` directory.

The `/path/to/openidm/samples/provisioning-with-workflow/` sample provides a workflow definition (`contractorOnboarding.bar`) that you can use to test the workflow integration. Create a `workflow` directory in your project directory and copy the sample workflow to that directory:

```
cd project-dir
mkdir workflow
cp samples/provisioning-with-workflow/workflow/contractorOnboarding.bar workflow/
```

Verify the workflow integration by using the REST API. The following REST call lists the defined workflows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/workflow/processdefinition?_queryFilter=true"
```

The sample workflow definition that you copied in the previous step is named `contractorOnboarding`. The result of the preceding REST call is therefore similar to the following:


```
{
  "result": [
    {
      "_id": "contractorOnboarding:1:5",
      "_rev": "1",
      "candidateStarterGroupIdExpressions": [],
      "candidateStarterUserIdExpressions": [],
      "category": "Examples",
      "deploymentId": "1",
      "description": null,
      "eventSupport": {},
      "executionListeners": {},
      "graphicalNotationDefined": false,
      "hasStartFormKey": true,
      "historyLevel": null,
      "ioSpecification": null,
      "key": "contractorOnboarding",
      "laneSets": [],
      "name": "Contractor onboarding process",
      "participantProcess": null,
      "processDiagramResourceName": "contractorOnboarding.contractorOnboarding.png",
      "properties": {},
      "resourceName": "contractorOnboarding.bpmn20.xml",
      "revisionNext": 2,
```

```
      "startFormHandler": null,
      "suspended": false,
      "suspensionState": 1,
      "taskDefinitions": null,
      "tenantId": "",
      "variables": null,
      "version": 1
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```


For more information about the above workflow, see "*Provision Users With Workflow*" in the *Samples Guide*.

For more information about managing workflows over REST, see "*REST Endpoints and Sample Commands*" in the *REST API Reference*.

Chapter 4
Create Workflows

For more information about the graphical notations and XML representations for events, flows, gateways, tasks, process constructs, and more, see BPMN 2.0 Constructs.

IDM does not support the following constructs:

- Mule task
- Camel task

1. Create a workflow definition, and save it with a `bpmn20.xml` extension.
2. Package the workflow definition file in a Business Archive File (`.bar`). A `.bar` file is similar to a `.zip` file, but with a different extension.
3. Copy the `.bar` file to the `openidm/workflow` directory.
4. Invoke the workflow using a script (`openidm/script/`), or directly, using the REST interface. For more information, see "*Invoke Workflows*".

   You can also schedule the workflow to be invoked repeatedly, or at a future time.

Workflow Definition Comparison

Versions of IDM prior to 7.0 used the Activiti workflow engine. If you are upgrading from one of these versions, your current workflow definitions will continue to work in compatibility mode, but new definitions must be written for the new engine, Flowable. The following overview shows the main differences between the old and new workflow definitions:

Note

You can view additional upgrade information in the Flowable Migration Guide.

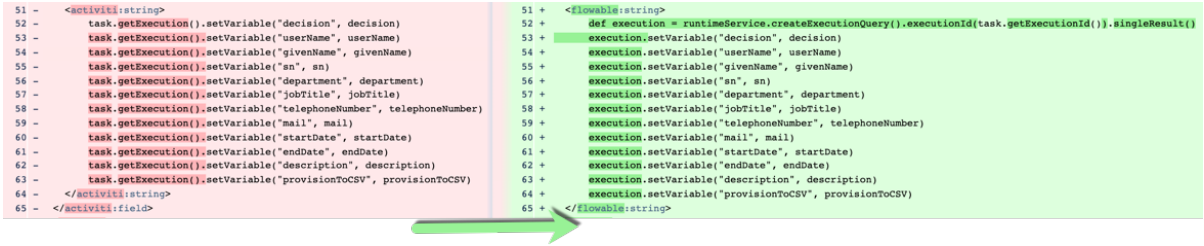
1. Change all occurrences of `activiti` to `flowable`. Update examples:
   - Namespace

     ```
     xmlns:flowable="http://flowable.org/bpmn"
     ```

   - Elements

```
<flowable:formProperty id="givenName" name="First Name" type="string"
 required="true"></flowable:formProperty>
<flowable:formProperty id="sn" name="Last Name" type="string"
 required="true"></flowable:formProperty>
<flowable:formProperty id="department" name="Department" type="string"></flowable:formProperty>
```

2. Change `task.getExecution()`, per the changes to the sample file `contractorOnboarding.bpmn20.xml`:



Chapter 5
Query Workflows

The workflow implementation supports filtered queries that let you query the running process instances and tasks, based on specific query parameters. To perform a filtered query, send a GET request to the `workflow/processinstance` context path, including the query in the URL.

For example, the following query returns all process instances with the business key `"newOrder"`, as invoked in the previous example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/workflow/processinstance?_queryId=filtered-
query&processInstanceBusinessKey=newOrder"
```

Any workflow properties can be queried using the same notation; for example, `processDefinitionId=managedUserApproval:1:6405`. The query syntax applies to all queries with `_queryId=filtered-query`. The following query returns all process instances that were started by the user `openidm-admin`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/workflow/processinstance?_queryId=filtered-query&startUserId=openidm-admin"
```

You can also query process instances based on the value of any process instance variable, by prefixing the variable name with `var-`. For example:

```
var-processvariablename=processvariablevalue
```

Chapter 6
Invoke Workflows

You can invoke workflows and business processes from any trigger point within IDM, including reacting to situations discovered during reconciliation. Workflows can be invoked from script files, using the `openidm.create()` function, or directly from the REST interface.

The following sample script extract shows how to invoke a workflow from a script file:

```
/*
 * Calling 'myWorkflow' workflow
 */

var params = {
  "_key": "myWorkflow"
};

openidm.create('workflow/processinstance', null, params);
```

The `null` in this example indicates that you do not want to specify an ID as part of the create call. For more information, see openidm.create(resourceName, newResourceId, content, params, fields) in the *Scripting Guide*.

You can invoke the same workflow from the REST interface with the following REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{"_key":"myWorkflow"}' \
"http://localhost:8080/openidm/workflow/processinstance?_action=create"
```

For more information, see "Workflows" in the *REST API Reference*.

There are two ways in which you can specify the workflow definition that is used when a new workflow instance is started.

- `_key` specifies the `id` attribute of the workflow process definition, for example:

  ```
  <process id="sendNotificationProcess" name="Send Notification Process">
  ```

  If there is more than one workflow definition with the same `_key` parameter, the latest deployed version of the workflow definition is invoked.

- `_processDefinitionId` specifies the ID that is generated by the Flowable Process Engine when a workflow definition is deployed; for example:

```
"sendNotificationProcess:1:104";
```

To obtain the `processDefinitionId`, query the available workflows, for example:

```
{
  "result": [
    {
      "name": "Process Start Auto Generated Task Auto Generated",
      "_id": "ProcessSAGTAG:1:728"
    },
    {
      "name": "Process Start Auto Generated Task Empty",
      "_id": "ProcessSAGTE:1:725"
    },
    ...
  ]
}
```

If you specify a `_key` and a `_processDefinitionId`, the `_processDefinitionId` is used because it is more precise.

Use the optional `_businessKey` parameter to add specific business logic information to the workflow when it is invoked. For example, the following workflow invocation assigns the workflow a business key of `"newOrder"`. This business key can later be used to query "newOrder" processes.

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{"_key":"myWorkflow", "_businessKey":"newOrder"}' \
"http://localhost:8080/openidm/workflow/processinstance?_action=create"
```

Access to workflows is based on IDM roles, and is configured in your project's `conf/process-access.json` file. For more information, see "Secure Access to Workflows" in the *Authentication and Authorization Guide*.

Chapter 7
Workflow Audit

The audit service logs workflow information in the *activity* event topic in the *Audit Guide* (default location=`openidm/audit/activity.audit.json`).

Example workflow audit events using the provisioning-with-workflow sample in the Samples Guide:

Each step shows the action performed along with the resulting audit data.

1. `user1` completes the *Contractor Onboarding Form*.

```
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-3871",
  "timestamp": "2020-05-06T17:39:52.021Z",
  "eventName": "workflow-create_process",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-3865",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "workflow/processinstance/6",
  "operation": "CREATE",
  "changedFields": [],
  "revision": null,
  "status": "SUCCESS",
  "message": "Process created. processDefinitionId = contractorOnboarding:1:5, processDefinitionKey =
 null, businessKey = null",
  "passwordChanged": false
}
```

2. `manager1` assigns the task to themself.

Note

`"changedFields":["/assignee"]` only displays when `conf/audit.json` contains the property `"watchedFields" : [ "assignee" ]`. For a complete list of fields that can be watched in this situation, see the API Descriptor for `UPDATE workflow/taskinstance/`.


```
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5748",
  "timestamp": "2020-05-06T17:43:18.058Z",
  "eventName": "workflow-update_task",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5744",
  "userId": "manager1",
  "runAs": "manager1",
  "objectId": "workflow/taskinstance/36",
  "operation": "UPDATE",
  "changedFields": [
    "/assignee"
  ],
  "revision": null,
  "status": "SUCCESS",
  "message": "Task updated",
  "passwordChanged": false
}
```


3. `manager1` completes the task. Notice that `transactionId` is correlated to all managed/user, and other, operations.


```
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5868",
  "timestamp": "2020-05-06T17:43:22.138Z",
  "eventName": "activity",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "000000001edd9dc2",
  "status": "SUCCESS",
  "message": "create",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5871",
  "timestamp": "2020-05-06T17:43:22.141Z",
  "eventName": "activity",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/usermeta/cd237cca-913e-481e-9282-ba16c84b5131",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "0000000030b45c3e",
  "status": "SUCCESS",
  "message": "create",
```

```
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5876",
  "timestamp": "2020-05-06T17:43:22.145Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2/authzRoles/ee5bbbce-a020-45db-
ab41-66c80d84d8be",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "00000000fe6da3a7",
  "status": "SUCCESS",
  "message": "Relationship originating from managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2 via the
 relationship field authzRoles and referencing internal/role/openidm-authorized was created.",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5879",
  "timestamp": "2020-05-06T17:43:22.147Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2/manager/b58e5695-9e43-4e76-b89c-
e5d69d3bf52d",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "000000008dcca1b6",
  "status": "SUCCESS",
  "message": "Relationship originating from managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2 via the
 relationship field manager and referencing managed/user/038e65de-95ce-4180-94d3-4ea64bf25c6b was
 created.",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5882",
  "timestamp": "2020-05-06T17:43:22.149Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2/_meta/d9299603-b768-44b6-
a4c9-9b6441ca212e",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "0000000027b29fb4",
  "status": "SUCCESS",
  "message": "Relationship originating from managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2 via the
 relationship field _meta and referencing internal/usermeta/cd237cca-913e-481e-9282-ba16c84b5131 was
 created.",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5908",
  "timestamp": "2020-05-06T17:43:22.778Z",
  "eventName": "activity",
```

```
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/notification/12aa1698-bb1e-42c6-a92d-2e959c217ad0",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "000000004d025d75",
  "status": "SUCCESS",
  "message": "create",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5911",
  "timestamp": "2020-05-06T17:43:22.781Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/notification/12aa1698-bb1e-42c6-a92d-2e959c217ad0/target/eec80d30-
be1e-4c5d-9873-b4395373c833",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "00000000b4c7a701",
  "status": "SUCCESS",
  "message": "Relationship originating from internal/notification/12aa1698-bb1e-42c6-a92d-2e959c217ad0
 via the relationship field target and referencing managed/user/038e65de-95ce-4180-94d3-4ea64bf25c6b
 was created.",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5920",
  "timestamp": "2020-05-06T17:43:22.791Z",
  "eventName": "activity",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/notification/eec030e5-e520-4cf1-99c2-a9bbecb0627b",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "0000000033465ada",
  "status": "SUCCESS",
  "message": "create",
  "passwordChanged": false
}
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5923",
  "timestamp": "2020-05-06T17:43:22.794Z",
  "eventName": "relationship_created",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "user1",
  "runAs": "user1",
  "objectId": "internal/notification/eec030e5-e520-4cf1-99c2-a9bbecb0627b/
target/11eb13f8-991f-45ea-95dc-e8f0fd0b95c3",
  "operation": "CREATE",
  "changedFields": [],
  "revision": "000000005134a80d",
  "status": "SUCCESS",
```

```
  "message": "Relationship originating from internal/notification/eec030e5-e520-4cf1-99c2-a9bbecb0627b
 via the relationship field target and referencing managed/user/d736487d-c146-4a0e-b677-ebfd6805b1d2
 was created.",
  "passwordChanged": false
}
```


4. The audit service logs the `workflow-complete_task` event.


```
{
  "_id": "f24ac83b-200c-449d-b017-d12b9c6c9091-5926",
  "timestamp": "2020-05-06T17:43:22.827Z",
  "eventName": "workflow-complete_task",
  "transactionId": "f24ac83b-200c-449d-b017-d12b9c6c9091-5838",
  "userId": "manager1",
  "runAs": "manager1",
  "objectId": "workflow/taskinstance/36",
  "operation": "complete",
  "changedFields": [],
  "revision": null,
  "status": "SUCCESS",
  "message": "Task completed",
  "passwordChanged": false
}
```

Chapter 8
Custom Workflow Templates

The embedded workflow engine integrates with the default End User UI. For simple custom workflows, you can use the standard Flowable form properties, and have the UI render the corresponding generic forms automatically. For more complex functionality, including input validation, rich input field types, complex CSS, and more, you must define a custom form template.

The default workflows provided with IDM use the Vue JS framework for display in the End User UI. To write a custom form template, you must have a basic understanding of the Vue JS framework and how to create components. A sample workflow template is provided at `/path/to/samples/provisioning-with-workflow/workflow/contractorOnboarding.bar`. To extract the archive, run the following command:

```
jar -xvf contractorOnboarding.bar
inflated: contractorForm.js
inflated: contractorOnboarding.bpmn20.xml
```

The archive includes the workflow definition `contactorOnboarding.bpmn20.xml` and the corresponding JavaScript template `contractorForm.js` to render the workflow in the UI.

IDM Glossary

correlation query	A correlation query specifies an expression that matches existing entries in a source repository to one or more entries in a target repository. A correlation query might be built with a script, but it is not the same as a correlation script. For more information, see "*Correlating Source Objects With Existing Target Objects*" in the *Synchronization Guide*.
correlation script	A correlation script matches existing entries in a source repository, and returns the IDs of one or more matching entries on a target repository. While it skips the intermediate step associated with a `correlation query`, a correlation script can be relatively complex, based on the operations of the script.
entitlement	An entitlement is a collection of attributes that can be added to a user entry via roles. As such, it is a specialized type of `assignment`. A user or device with an entitlement gets access rights to specified resources. An entitlement is a property of a managed object.
JCE	Java Cryptographic Extension, which is part of the Java Cryptography Architecture, provides a framework for encryption, key generation, and digital signatures.
JSON	JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, see the JSON site.
JSON Pointer	A JSON Pointer defines a string syntax for identifying a specific value within a JSON document. For information about JSON Pointer syntax, see the JSON Pointer RFC.

JWT	JSON Web Token. As noted in the JSON Web Token draft IETF Memo, "JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties." For IDM, the JWT is associated with the `JWT_SESSION` authentication module.
managed object	An object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default configuration of a managed object is that of a user, but you can define any kind of managed object, for example, groups or roles.
mapping	A policy that is defined between a source object and a target object during reconciliation or synchronization. A mapping can also define a trigger for validation, customization, filtering, and transformation of source and target objects.
OSGi	A module system and service platform for the Java programming language that implements a complete and dynamic component model. For more information, see What is OSGi? Currently, only the Apache Felix container is supported.
reconciliation	During reconciliation, comparisons are made between managed objects and objects on source or target systems. Reconciliation can result in one or more specified actions, including, but not limited to, synchronization.
resource	An external system, database, directory server, or other source of identity data to be managed and audited by the identity management system.
REST	Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.
role	IDM distinguishes between two distinct role types - provisioning roles and authorization roles. For more information, see "Managed Roles" in the *Object Modeling Guide*.
source object	In the context of reconciliation, a source object is a data object on the source system, that IDM scans before attempting to find a corresponding object on the target system. Depending on the defined mapping, IDM then adjusts the object on the target system (target object).
synchronization	The synchronization process creates, updates, or deletes objects on a target system, based on the defined mappings from the source system. Synchronization can be scheduled or on demand.

system object	A pluggable representation of an object on an external system. For example, a user entry that is stored in an external LDAP directory is represented as a system object in IDM for the period during which IDM requires access to that entry. System objects follow the same RESTful resource-based design principles as managed objects.
target object	In the context of reconciliation, a target object is a data object on the target system, that IDM scans after locating its corresponding object on the source system. Depending on the defined mapping, IDM then adjusts the target object to match the corresponding source object.