DTGov Guide

Chapter 1. Introduction to DTGov

1.1. Design Time Governance

The DTGov project layers Design Time Governance functionality on top of an S-RAMP repository. These two projects work together to provide the following:

- Store and Govern artifacts
- Custom Governance Workflows
- Integrated Governance Human Task Management

This guide will discuss the various pieces of functionality provided by DTGov and how to configure and use them.

1.2. Use Cases

In addition to a general framework for triggering business workflows based on changes to artifacts in the S-RAMP repository, the DTGov project focuses on the following specific Governance Use Cases:

- Deployment Lifecycle Management

This guide will not only discuss the generic governance capabilities provided by the DTGov project, but also the specific Use-Cases listed above.

Chapter 2. Getting Started

2.1. Prerequisites

The DTGov application is written in Java. To get started make sure your system has the following:

- Java JDK 1.6 or newer
- Apache Ant 1.7 or newer to use the installer
- Maven 3.0.3 or newer
- Overlord S-RAMP version 0.3.0.Final or newer

This Getting Started guide assumes you do not already have Overlord S-RAMP installed.

2.2. Download, Installation and Configuration

First, we recommend you download the following:

- *JBoss EAP 6.1* [http://www.jboss.org/jbossas/downloads]
- *ModeShape 3.2.0.Final* [http://www.jboss.org/modeshape/downloads/downloads3-2-0-final.html]
- *S-RAMP 0.3.0.Final* [http://www.jboss.org/overlord/downloads/sramp]
- *DTGov 1.0.0.Final* [http://www.jboss.org/overlord/downloads/dtgov]

Next, you must follow these steps to install and configure the application:

1. Unpack S-RAMP distribution
2. Copy the EAP download into the unpacked S-RAMP distro
3. Copy the ModeShape distribution into the unpacked S-RAMP distro
4. Run the S-RAMP installer
5. Unpack the DTGov distribution
6. Move the "target" folder from the S-RAMP distro to the unpacked DTGov distro
7. Copy the EAP download into the unpacked DTGov distro

8. Run the DTGov installer

9. Start JBoss

10. Populate the S-RAMP repository with DTGov seed data (ontology + workflow jar)

Some psuedo-shell code that might help

```
mkdir ~/overlord
cd ~/overlord
# Download JBoss EAP 6.1 (jboss-eap-6.1.0.zip)
#    From - http://www.jboss.org/jbossas/downloads
# Download the ModeShape EAP distro (modeshape-3.2.0.Final-jbosseap-61-dist.zip)
#    From - http://www.jboss.org/modeshape/downloads/downloads3-2-0-final.html
# Download S-RAMP distribution (s-ramp-0.3.0.Final.zip)
#    From - http://www.jboss.org/overlord/downloads/sramp
unzip s-ramp-0.3.0.Final.zip
cp jboss-eap-6.1.0.zip s-ramp-0.3.0.Final
cp modeshape-3.2.0.Final-jbosseap-61-dist.zip s-ramp-0.3.0.Final
cd s-ramp-0.3.0.Final
ant install
cd ..
mv s-ramp-0.3.0.Final/target dtgov-1.0.0.Final
cp jboss-eap-6.1.0.zip dtgov-1.0.0.Final
cd dtgov-1.0.0.Final
ant install
# Start JBoss (target/jboss-eap-6.1/bin/standalone.sh) - wait for startup to
 complete
ant seed
cd data
mvn package
```

2.3. Check your Installation

Now that everything is installed and running, you should be able to verify that everything is working by logging in to the S-RAMP Browser UI and verifying that you can see the DTGov seed data.

http://localhost:8080/s-ramp-ui (admin/overlord)

You should see something like this:

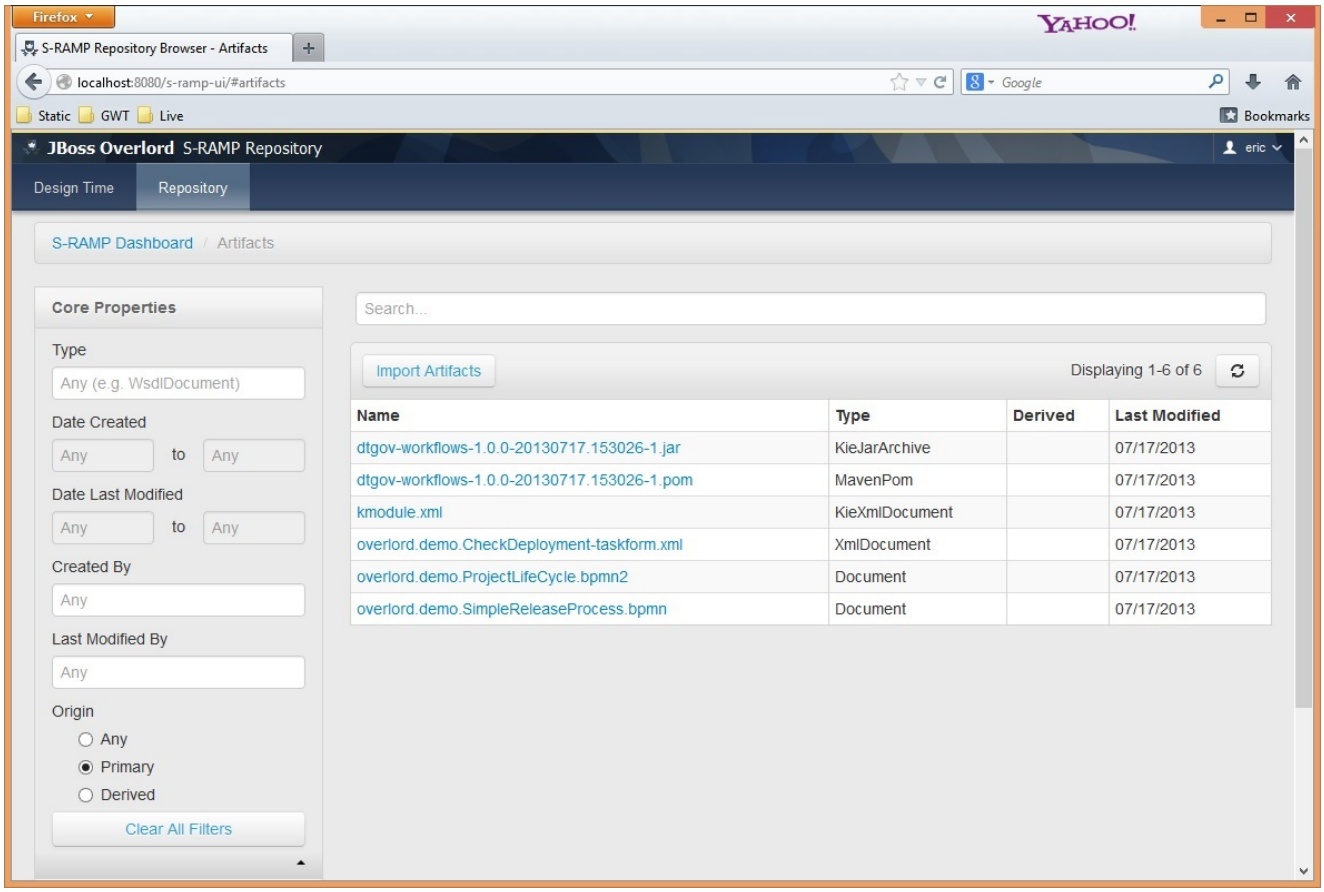

Figure 2.1. Screenshot of the DTGov data in S-RAMP

2.4. Get to Work

It's all installed, running, and checked? Now it's time to use the software! This guide will explain advanced configuration and usage, but you can get started by logging in to the DTGov User Interface:

http://localhost:8080/dtgov-ui (admin/overlord)

It's likely that users will need to customize the system based on their organization's specific work processes. The **Configuring** and **Governance Workflows** chapters should be helpful in describing how to customize the system.

Chapter 3. Configuring DTGov

3.1. Overview

DTGov has two configurations that can be modified to suit a particular deployment and business. Specifically, the back-end DTGov system (dtgov.war) has a configuration file as does the User Interface (dtgov-ui.war). This chapter describes these two configuration files so that users can configure DTGov for their particular deployment environment and organization's unique business processes.

3.2. Back-End Configuration

The configuration of the back-end system can be modified by making changes to an external configuration file found in the application server's **configuration** directory. In JBoss EAP the configuration file can be found here:

jboss-eap/standalone/configuration/dtgov.properties

The location of this file can be overridden by setting the following system property to be the full path to a properties file anywhere on the server's file system:

governance.file.name

This configuration file is used to control a number of settings, listed and described in the following section.

3.3. Back-End Configuration Properties

```
# S-RAMP Connection details
sramp.repo.url
sramp.repo.auth.provider
sramp.repo.user
sramp.repo.password
sramp.repo.validating
sramp.repo.auth.saml.issuer
sramp.repo.auth.saml.service

# Location of the DTGov WAR
governance.url
# Frequency with which to poll S-RAMP for query matches
governance.query.interval
# Location in JNDI of the email service
governance.jndi.email.reference
# "From" information to use when sending email (domain and address)
governance.email.domain
governance.email.from
```

```
# RHQ connection info
rhq.rest.user
rhq.rest.password
rhq.base.url

# BPM connection info
governance.bpm.user
governance.bpm.password
governance.bpm.url

# JAAS user used to invoke DTGov provided services
governance.user
governance.password

# Deployment targets configured for the DTGov deployment service
governance.targets

# Mapping of S-RAMP query to governance workflow
governance.queries

# Location of the DTGov UI
dtgov.ui.url

# S-RAMP
s-ramp-wagon
dtgov.s-ramp-wagon.snapshots
dtgov.s-ramp-wagon.releases

# DTGov Workflow maven info
dtgov.workflows.group
dtgov.workflows.name
dtgov.workflows.version
dtgov.workflows.package
```

In particular, the **governance.targets** and **governance.queries** configuration properties bear additional explanation. Please see the **Governance Workflows** chapter for more information on how to use these properties to configure the DTGov Deployment Service and the Governance Workflow Queries, respectively.

3.4. User Interface (UI) Configuration

The DTGov user interface can also be configured for a specific deployment and business environment. The configuration of the UI can be modified by making changes to an external configuration file found in the application server's **configuration** directory. In JBoss EAP the configuration file can be found here:

jboss-eap/standalone/configuration/dtgov-ui.properties

The location of this file can be overridden by setting the following system property to be the full path to a properties file anywhere on the server's file system:

dtgov-ui.config.file.name

This configuration file is used to control a number of settings, listed and described in the following section.

3.5. UI Configuration Properties

```
# S-RAMP API connection endpoint
dtgov-ui.s-ramp.atom-api.endpoint
# What kind of authentication to use (class name)
dtgov-ui.s-ramp.atom-api.authentication.provider
# Only used when the provider is basic auth
dtgov-ui.s-ramp.atom-api.authentication.basic.username
dtgov-ui.s-ramp.atom-api.authentication.basic.password
# Only used when the provider is SAML bearer token auth
dtgov-ui.s-ramp.atom-api.authentication.saml.issuer
dtgov-ui.s-ramp.atom-api.authentication.saml.service
# Whether to validate the S-RAMP connection
dtgov-ui.s-ramp.atom-api.validating

# Task API connection endpoint
dtgov-ui.task-api.endpoint
# Implementation of a task client
dtgov-ui.task-client.class
# Authentication to use when invoking the task API
dtgov-ui.task-api.authentication.provider
# Only used when using basic auth
dtgov-ui.task-api.authentication.basic.username
dtgov-ui.task-api.authentication.basic.password
# Only used when using saml bearer token auth
dtgov-ui.task-api.authentication.saml.issuer
dtgov-ui.task-api.authentication.saml.service

# Deployment lifecycle base classifier
dtgov-ui.deployment-lifecycle.classifiers.base
dtgov-ui.deployment-lifecycle.classifiers.initial
# Classifier to use when querying for all deployments
dtgov-ui.deployment-lifecycle.classifiers.all
dtgov-ui.deployment-lifecycle.classifiers.in-progress
# This next one is a prefix for any property that will indicate a possible
 classifier stage that
# should be displayed in the UI.  In the dtgov ui configuration file,
 multiple properties would
# be specified that begin with this prefix and have a value of the format
 {label}:{classifier}
dtgov-ui.deployment-lifecycle.classifiers.stage
```

```
# And another one that is a prefix for any property that will indicate a
 possible deployment type
# that should be displayed in the UI.  In the dtgov ui configuration file,
 multiple properties would
# be specified that begin with this prefix and have a value of the format
 {label}:{type}
dtgov-ui.deployment-lifecycle.types

# S-RAMP UI integration properties
dtgov-ui.s-ramp-browser.url-base
```

In particular, the **dtgov-ui.deployment-lifecycle.classifiers.stage** and **dtgov-ui.deployment-lifecycle.types** properties require further explanation. See the following sections for details.

3.6. Configuring UI Deployment Stages

The DTGov user interface has a page that allows users to see a list of all deployments being tracked. That page allows the user to filter the list of deployments based on the environments in which the deployment is…deployed. In other words, the UI page allows the user to show only the deployments that are currently deployed in, for example, the DEV environment. Since different organizations have different numbers and names for these environments, the actual filter options are configurable. An example will prove useful:

```
dtgov-ui.deployment-lifecycle.classifiers.stage.dev=Development:http://
www.jboss.org/overlord/deployment-status.owl#InDev
dtgov-ui.deployment-lifecycle.classifiers.stage.qa=QA:http://www.jboss.org/
overlord/deployment-status.owl#InQa
dtgov-ui.deployment-lifecycle.classifiers.stage.stage=Staging:http://
www.jboss.org/overlord/deployment-status.owl#InStage
dtgov-ui.deployment-lifecycle.classifiers.stage.prod=Production:http://
www.jboss.org/overlord/deployment-status.owl#InProd
```

If the above configuration is used (in the **dtgov-ui.properties** file) then the UI will show four possible environments that the user can use to filter deployments (dev, qa, stage, prod). The format for the value of each entry is:

Label : **Classifier**

The Label will be shown in the UI (in the filter drop-down) and the Classifier will be used when performing the S-RAMP query to retrieve the filtered list of deployments.

3.7. Configuring UI Deployment Types

The DTGov user interface's deployment listing page also allows users to filter by the type of deployment (Java Web Application, SwitchYard Application, etc). Since different organizations will likely work with varying technologies, the Deployment Type filter is configurable. For example:

```
dtgov-ui.deployment-lifecycle.types.switchyard=SwitchYard Application:ext/
SwitchYardApplication
dtgov-ui.deployment-lifecycle.types.jar=Java Archive:ext/JavaArchive
dtgov-ui.deployment-lifecycle.types.war=Java Web Application:ext/
JavaWebApplication
```

In the above example, the user would be able to filter by SwitchYard Application, Java Archive, and Java Web Application. The format for each entry is:

Label : **S-RAMP Artifact Type**

The Label will be shown in the UI (in the filter drop-down) and the S-RAMP Artifact Type will be used when performing the S-RAMP query to retrieve the filtered list of deployments.

Note: the list of Deployment Types is also used in the **Add Deployment** dialog when adding a new deployment. In this case, the S-RAMP Artifact Type is used when adding the deployment to the repository.

This configuration works in conjuction with the Deployment Service described in the **Deployment Management** chapter of this guide. The classifiers specified when configuring Deployment Targets should be represented here.

Chapter 4. DTGov and S-RAMP

4.1. Overview

DTGov integrates tightly with a compliant S-RAMP repository, and it is recommended that the Overlord S-RAMP implementation is used. The S-RAMP repository is used as the storage mechanism for all artifacts that DTGov is interested in (e.g. Deployments). This chapter describes this integration as well as how it is configured.

DTGov is integrated with S-RAMP via the Atom based REST API that all S-RAMP repositories expose. The repository is leveraged in a number of ways, including:

- Storage of all artifacts
- Monitor for changes to trigger business workflows (described in another chapter)
- Managing deployments

4.2. Configuration Properties

A number of configuration properties drive the integration between DTGov and S-RAMP. In particular note that the DTGov back-end and the DTGov User Interface each have their own separate configuration. This is because the back-end and UI are separate applications that can be independently deployed.

DTGov Back-End Configuration.

```
# S-RAMP Connection details
sramp.repo.url
sramp.repo.auth.provider
sramp.repo.user
sramp.repo.password
sramp.repo.validating
sramp.repo.auth.saml.issuer
sramp.repo.auth.saml.service
```

DTGov User Interface Configuration.

```
# S-RAMP API connection endpoint
dtgov-ui.s-ramp.atom-api.endpoint
dtgov-ui.s-ramp.atom-api.authentication.provider
dtgov-ui.s-ramp.atom-api.authentication.basic.username
dtgov-ui.s-ramp.atom-api.authentication.basic.password
dtgov-ui.s-ramp.atom-api.authentication.saml.issuer
```

```
dtgov-ui.s-ramp.atom-api.authentication.saml.service
dtgov-ui.s-ramp.atom-api.validating
```

Here is an example of how the back-end configuration might look:

```
sramp.repo.url=http://localhost:8080/s-ramp-server/
sramp.repo.auth.provider=org.overlord.sramp.governance.auth.BasicAuthenticationProvider
sramp.repo.user=dtgov
sramp.repo.password=DTG_PASSWORD
sramp.repo.validating=true
```

The above configuration uses BASIC authentication when connecting to the S-RAMP repository. It will connect to S-RAMP on localhost (port 8080).

The user interface configuration might look something like this:

```
dtgov-ui.s-ramp.atom-api.endpoint=http://localhost:8080/s-ramp-server
dtgov-ui.s-ramp.atom-
api.authentication.provider=org.overlord.dtgov.ui.server.services.sramp.SAMLBearerTokenAuthe
dtgov-ui.s-ramp.atom-api.authentication.saml.issuer=/dtgov-ui
dtgov-ui.s-ramp.atom-api.authentication.saml.service=/s-ramp-server
dtgov-ui.s-ramp.atom-api.validating=true
```

The above configuration connects to S-RAMP on localhost (port 8080) and uses SAML bearer token authentication.

4.3. Authentication

Both the UI and the back-end support pluggable authentication mechanisms. Out of the box DTGov provides implementations for BASIC authentication and SAML Bearer Token authentication. If the S-RAMP repository is protected by some alternative form of authentication, another implementation of the authentication provider can be created. In both cases, the authentication provider must implement the following interface:

org.overlord.sramp.client.auth.AuthenticationProvider

The DTGov back-end provides the following authentication provider implementations:

1. **BASIC** - *org.overlord.sramp.governance.auth.BasicAuthenticationProvider*
2. **SAML Bearer Token** - *org.overlord.sramp.governance.auth.SAMLBearerTokenAuthenticationProvider*

The DTGov user interface provides the following authentication provider implementations:

1. **BASIC** - *org.overlord.dtgov.ui.server.services.sramp.BasicAuthenticationProvider*

2. **SAML Bearer Token** - *org.overlord.dtgov.ui.server.services.sramp.SAMLBearerTokenAuthenticationProvider*

Chapter 5. Governance Workflows

5.1. Overview

One of the most important features of the Overlord: DTGov software is the ability to trigger Governance Workflows based on changes detected in the S-RAMP repository. This chapter discusses this functionality, including:

1. How to create a workflow
2. Using DTGov supplied supporting Governance Services
3. How to deploy a workflow
4. Configuring a workflow to execute (trigger) when repository content changes

5.2. Creating Workflows

Overlord: DTGov integrates tightly with the jBPM business process management system. This allows DTGov to utilize any business process that is compatible with jBPM 6. The tooling available to author jBPM compatible business processes is varied and extensive (and is outside the scope of this document). One possibility is using the Eclipse based BPM tools. Another alternative is using the web based Drools authoring tools.

For additional information about how to create jBPM processes, please consult the *jBPM and Drools documentation* [http://www.jboss.org/jbpm].

5.3. Deploying Workflows

All of the workflows and supporting files (images, task forms, etc) should be bundled together into a KIE archive. A KIE archive is simply a JAR with a particular structure assumed by jBPM. For example, your archive file structure might look something like this:

```
META-INF/kmodule.xml
SRAMPPackage/HttpClientWorkDefinitions.wid
SRAMPPackage/com.mybusiness.deploy.EARLifeCycle.bpmn2
SRAMPPackage/com.mybusiness.deploy.WARLifeCycle.bpmn2
SRAMPPackage/com.mybusiness.validate.NewSchemaReview.bpmn2
SRAMPPackage/run-build-install.png
SRAMPPackage/user-properties.png
SRAMPPackage/audio-input-microphone-3.png
```

What are all these files?

The **kmodule.xml** file is a jBPM artifact (it makes this a Kie Archive rather than just a plain old JAR file). This file should have the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://raw.github.com/droolsjbpm/droolsjbpm-
knowledge/master/kie-api/src/main/resources/org/kie/api/kmodule.xsd"
         xmlns="http://jboss.org/kie/6.0.0/kmodule" >
    <kbase name="SRAMPPackage">
        <ksession name="ksessionSRAMP"/>
    </kbase>
</kmodule>
```

Next, there is a folder in the archive that maps to the **kbase** element found in the **kmodule.xml** file. This folder contains all of the business process resources, primarily the BPMN2 files. There is a file called **HttpClientWorkDefinitions.wid** which contains the custom work items used by Governance Workflows. It might look something like this:

```
import org.drools.process.core.datatype.impl.type.StringDataType;
[
  // the HttpClient work item
  [
    "name" : "HttpClientDeploy",
    "parameters" : [
      "Url" : new StringDataType(),
      "Method" : new StringDataType(),
      "Uuid" : new StringDataType(),
      "Target" : new StringDataType()
    ],
    "displayName" : "Deploy",
    "icon" : "run-build-install.png",
  ],

  // the HttpClient work item
  [
    "name" : "HttpClientNotify",
    "parameters" : [
      "Url" : new StringDataType(),
      "DTGovUrl" : new StringDataType(),
      "Method" : new StringDataType(),
      "Uuid" : new StringDataType(),
      "Target" : new StringDataType(),
      "Group" : new StringDataType(),
    ],
    "displayName" : "Notify",
    "icon" : "audio-input-microphone-3.png",
  ],

  // the HttpClient work item
  [
    "name" : "HttpClientUpdateMetaData",
    "parameters" : [
      "Url" : new StringDataType(),
```

```
      "Method" : new StringDataType(),
      "Name" : new StringDataType(),
      "Value" : new StringDataType(),
      "Uuid" : new StringDataType(),
    ],
    "displayName" : "UpdateMetaData",
    "icon" : "user-properties.png",
  ]
]
```

This file also refers to some images files (useful for BPMN editors) which are also included in the package.

Once the workflows are built, they must be deployed into the S-RAMP repository so that the embedded version of jBPM can find them properly. It is recommended that the S-RAMP maven integration is used to do this. The best way is to put all of the business process resouces into a simple JAR style maven project. Then use the S-RAMP maven integration to **mvn deploy** the project directly into S-RAMP. Please see the Overlord: S-RAMP documentation for details on how this works. The result should be that your Governance workflow JAR (Kie Archive) is uploaded to the S-RAMP repository, complete with relevant maven properties set.

The embedded jBPM engine will pick up the Governance Workflows by pulling the Kie Archive out of the S-RAMP repository and using the content it finds within. It's worth noting that the maven information of the Kie Archive can be configured in the DTGov back end configuration file (dtgov.properties). The following properties control exactly what Kie Archive artifact the embedded jBPM engine will grab from S-RAMP:

```
dtgov.workflows.group=com.mybusiness
dtgov.workflows.name=governance-workflows
dtgov.workflows.version=1.0.7
dtgov.workflows.package=SRAMPPackage
```

5.4. DTGov Supporting Services

In order to make it a little easier to author interesting Governance Workflows, DTGov provides a set of useful Governance Services. A list of these services follows:

- **Deployment Service** - deploys a binary application artifact to a configured target
- **Meta-Data Update Service** - allows simple modification of an artifact's meta-data
- **Notification Service** - provides a simple way to send email notifications

These services can be invoked by using the work items defined above in the **HttpClientWorkDefinitions.wid** file.

Note: more information about the Deployment Service can be found in the **Deployment Management** chapter of this guide.

5.5. Query Configuration

Currently the only way to trigger the execution of a Governance Workflow is by configuring an S-RAMP query that will be used to monitor the S-RAMP repository for interesting changes. When changes are discovered, a new instance of the configured workflow is created and invoked. This section of the guide describes how to configure these query triggers.

All query triggers are defined in the Governance configuration file (dtgov.properties). The following is an example of this configuration:

```
governance.queries=/s-ramp/ext/JavaEnterpriseApplication[@maven.artifactId]|
com.mybusiness.deploy.EARLifeCycle.bpmn2|DeploymentUrl={governance.url}/
rest/deploy/{target}/{uuid}::NotificationUrl={governance.url}/rest/notify/
email/{group}/deployed/{target}/{uuid}
governance.queries=/s-ramp/xsd/XsdDocument|
com.mybusiness.validate.NewSchemaReview.bpmn2|
NotificationUrl={governance.url}/rest/notify/email/{group}/deployed/
{target}/{uuid}::UpdateMetaDataUrl={governance.url}/rest/update/{name}/
{value}/{uuid}
```

In the above example, two queries have been configured. The first is a query that will trigger the **EARLifeCycle** process whenever an EAR artifact is added to the repository. Note that only EAR artifacts added from Maven are targetted. The process will be passed two parameters:

1. DeploymentUrl
2. NotificationUrl

The second query will trigger the **NewSchemaReview** process whenever a new XML Schema document is added to the repository. This process will be passed two parameters:

1. NotificationUrl
2. UpdateMetaDataUrl

Chapter 6. Governance Human Tasks

6.1. Overview

Overlord: DTgov uses an embedded version of jBPM by default. However, human tasks can easily be included in Governance Workflows because the Task Inbox is integrated directly into the DTGov User Interface. Out of the box, this functionality should work seamlessly. However, it is possible to integrate a separate task system by providing an alternative (custom) Task API implementation.

6.2. Customizing the Task API

Simply put, the Task API system used by the DTGov user interface can be swapped out by setting the following property in the DTGov UI configuration file (dtgov-ui.properties):

dtgov-ui.task-client.class

This property must point to a fully qualified Java class that implements the following interface:

org.overlord.dtgov.ui.server.services.tasks.ITaskClient

Of course, any Governance Workflows that create Human Task instances must also point to the alternate task system. That configuration is out of the scope of this guide.

Chapter 7. Deployment Management

7.1. Overview

One of the most useful services provided by the Overlord: DTGov system is the Deployment Service. This is a service that makes it possible to deploy a binary artifact stored in the S-RAMP repository into a target runtime environment such as JBoss EAP. This Deployment Service can easily be invoked from a Governance Workflow and is often included as part of a Deployment Lifecycle business process.

7.2. Invoking the Deployment Service

Invoking the Deployment Service from a Governance Workflow should be a simple matter of using the **HttpClientDeploy** task defined in the **HttpClientWorkDefinitions.wid** file as described in the **Governance Workflows** chapter of this guide. Within a BPMN2 process, the XML markup might look something like this:

```
<bpmn2:task id="Task_1" drools:taskName="HttpClientDeploy"
 drools:displayName="Deploy" drools:icon="run-build-install.png"
 name="Deploy to DEV">
  <bpmn2:incoming>bpmn20:SequenceFlow_9</bpmn2:incoming>
  <bpmn2:outgoing>bpmn20:SequenceFlow_10</bpmn2:outgoing>
  <bpmn2:ioSpecification id="_InputOutputSpecification_10">
    <bpmn2:dataInput id="_DataInput_47" itemSubjectRef="__NameInputItem"
 name="Url"/>
    <bpmn2:dataInput id="_DataInput_48" itemSubjectRef="__NameInputItem"
 name="Method"/>
    <bpmn2:dataInput id="_DataInput_49" itemSubjectRef="__NameInputItem"
 name="Uuid"/>
    <bpmn2:dataInput id="_DataInput_50" itemSubjectRef="__NameInputItem"
 name="Target"/>
    <bpmn2:inputSet id="_InputSet_10" name="Input Set 10">
      <bpmn2:dataInputRefs>_DataInput_47</bpmn2:dataInputRefs>
      <bpmn2:dataInputRefs>_DataInput_48</bpmn2:dataInputRefs>
      <bpmn2:dataInputRefs>_DataInput_49</bpmn2:dataInputRefs>
      <bpmn2:dataInputRefs>_DataInput_50</bpmn2:dataInputRefs>
    </bpmn2:inputSet>
  </bpmn2:ioSpecification>
  <bpmn2:dataInputAssociation id="_DataInputAssociation_47">
    <bpmn2:sourceRef>DeploymentUrl</bpmn2:sourceRef>
    <bpmn2:targetRef>_DataInput_47</bpmn2:targetRef>
  </bpmn2:dataInputAssociation>
  <bpmn2:dataInputAssociation id="_DataInputAssociation_48">
    <bpmn2:targetRef>_DataInput_48</bpmn2:targetRef>
```

```
    <bpmn2:assignment id="Assignment_1">
      <bpmn2:from xsi:type="bpmn2:tFormalExpression"
 id="FormalExpression_16">POST</bpmn2:from>
      <bpmn2:to xsi:type="bpmn2:tFormalExpression"
 id="FormalExpression_17">_DataInput_48</bpmn2:to>
    </bpmn2:assignment>
  </bpmn2:dataInputAssociation>
  <bpmn2:dataInputAssociation id="_DataInputAssociation_49">
    <bpmn2:sourceRef>ArtifactUuid</bpmn2:sourceRef>
    <bpmn2:targetRef>_DataInput_49</bpmn2:targetRef>
  </bpmn2:dataInputAssociation>
  <bpmn2:dataInputAssociation id="_DataInputAssociation_50">
    <bpmn2:targetRef>_DataInput_50</bpmn2:targetRef>
    <bpmn2:assignment id="Assignment_14">
      <bpmn2:from xsi:type="bpmn2:tFormalExpression"
 id="FormalExpression_17">dev</bpmn2:from>
      <bpmn2:to xsi:type="bpmn2:tFormalExpression"
 id="FormalExpression_18">_DataInput_50</bpmn2:to>
    </bpmn2:assignment>
  </bpmn2:dataInputAssociation>
</bpmn2:task>
```

The above task uses the **DeploymentUrl** and **ArtifactUuid** parameters that were passed in to the business process when it was invoked. It populates the inputs required by **HttpClientDeploy** including an input parameter named **Target**. The **Target** parameter maps to a configured Deployment Target. The target is a logical name and corresponds to a physical runtime environment configured in the DTGov configuration file (dtgov.properties). See the next section for details.

7.3. Configuring Deployment Targets

In order to make logical Deployment Targets available they must be configured in the DTGov configuration file. Typically an organization would configure three or four Deployment Targets, including:

- **dev** - the development environment
- **qa** - the test environment
- **stage** - the staging environment
- **prod** - the final production environment

Of course, any number of targets can be configured. Here is an example of how to configure the above four targets in the DTGov configuration file:

```
governance.targets=  dev|http://www.jboss.org/overlord/deployment-
status.owl#InDev|copy|/tmp/dev/jbossas7/standalone/deployments
```

```
governance.targets=   qa|http://www.jboss.org/overlord/deployment-
status.owl#InQa|copy|/tmp/qa/jbossas7/standalone/deployments
governance.targets=stage|http://www.jboss.org/overlord/deployment-
status.owl#InStage|copy|/tmp/stage/jbossas7/standalone/deployments
governance.targets= prod|http://www.jboss.org/overlord/deployment-
status.owl#InProd|copy|/tmp/prod/jbossas7/standalone/deployments
```

The format of each target is as follows:

LogicalName|Classifier|DeploymentType|TypeSpecificParams

- *LogicalName* : used in the BPMN process as described in the previous section as the value of **Target**
- *Classifier* : a classifier that should get added to the binary deployment artifact when deployed to the target environment (and removed when undeployed)
- *DeploymentType* : how to deploy to the environment. Valid values as of this writing include copy, rhq, as_cli, maven

Depending on the type of the deployment, additional parameters may be required. In the example above, the *copy* deployment type requires a folder on the server, which is where it will copy the deployment artifact.

Here are some examples of how to use the other deployment types:

```
governance.targets= rhq_example  |#Example|rhq|{rhq.user}::{rhq.password}::
{rhq.baseUrl}
governance.targets= cli_example  |#Example|as_cli|
asuser::aspassword::ashost::asport
governance.targets= maven_example|#Example|maven|scp://m2.example.com/m2/
snapshot-repository::false::true
```

7.4. Undeployment

Whenever the Deployment Service is used to deploy an artifact from the repository, it also annotates that artifact with relevant undeployment information. This annotation takes the form of another artifact in the repository of type **ext/UndeploymentInformation**. The annotation artifact will have a relationship named **describesDeployment** pointing from it back to the deployment artifact it annotates.

This undeployment information is used whenever an artifact needs to be undeployed. Undeploy of an artifact happens when a new version of that artifact is being deployed to a particular environment (deployment target). When this happens, the old version (whichever version is currently deployed in that environment) is undeployed in preparation of the new deployment.

Once the artifact is undeployed, its undeployment information artifact is deleted from the repository and any relevant classifier associated with the target environment is removed from the deployment artifact.

Note: please see the **Configuring DTGov** chapter for information about how to coordinate the configuration of the Deployment Service with the configuration of the DTGov User Interface (the Deployment Management UI).

Bibliography

Books

[walsh-muellner] Norman Walsh & Leonard Muellner. *DocBook - The Definitive Guide*. O'Reilly & Associates. 1999. ISBN 1-56592-580-7.