Integrate your BPMS processes with Red Hat JBoss Fuse Service Works

Babak Mozaffari
Consulting Software Engineer, Systems Engineering, Red Hat
Red Hat JBoss BPM Suite
An overview of Red Hat JBoss BPM Suite 6.0
Business Process

What is a business process?

• A defined set of business activities that represents the steps required to achieve a business objective.

• It includes the flow and use of information and resources.

• Typically involves both machine and human interactions, integrates with various internal and external systems, and includes both static and dynamic flows that are subject to both business rules and technical constraints.
Business Process Management

• Business process management is a systematic approach to defining, executing, managing and refining business processes.
• BPMN (Business Process Model and Notation) 2.0 is a popular standard for business process management.
BPMN 2.0

• The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes.

• Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation.
Red Hat JBoss BPM Suite 6

• BPMS 6 is an open source BPM suite that combines business process management and business rules management, enabling business and IT users to create, manage, validate, and deploy business processes and rules.

• BPMS 6 provides advanced business process management capabilities compliant with the widely adopted BPMN 2.0 standard.
Red Hat JBoss BPM Suite 6

- BPMS 6 comes with a choice of modeling tools; it includes a business-user-friendly, web-based authoring environment as well as an Eclipse plugin for developers, to enable all project stakeholders to collaborate effectively and build sophisticated process and decision-automation solutions.

- The inclusion of Red Hat JBoss Business Rules Management System (BRMS) adds seamless integration with business rules and complex event processing functions to ease the development and facilitate the maintenance of processes in the face of rapidly changing requirements.
Red Hat JBoss BPM Suite 6

• Business activity monitoring (BAM) and process dashboards provide invaluable information to help manage processes.

• Process simulation helps refine business processes by enabling their analysis and assessment of the dynamic behavior of processes over time.

• Red Hat JBoss BRMS and Red Hat JBoss BPMS use a centralized repository where all resources are stored. This ensures consistency, transparency, and the ability to audit across the business.
Red Hat JBoss Fuse Service Works
An overview of Red Hat JBoss Fuse Service Works 6.0
Red Hat JBoss Fuse Service Works

• It is a platform for developing enterprise application integration (EAI) and service-oriented architecture (SOA) solutions.

• It consists of a service component framework, business rules/complex event processing, life-cycle governance, runtime governance and process automation.

• It is built on the same core as JBoss Fuse with enterprise messaging, Camel and CXF

• Users can use it to design, deploy, integrate and orchestrate business services.
Enterprise Application Integration (EAI)

- Defined by Wikipedia as an integration framework composed of a collection of technologies and services which form a middleware to enable integration of systems and applications across the enterprise.
EAI with JBoss Fuse Service Works

• The versatile **Enterprise Integration Pattern** (EIP) framework is implemented in routing and transformation processes for faster and more efficient integration solutions.

• A **high performance messaging** broker supports messaging patterns such as publish-subscribe, point-to-point and store-forward, and multiple cross language clients.

• **Service development** using the web services framework, which exposes integration assets as services and calls external services, supporting all major web services standards. It also supports RESTful calls.
Service-Oriented Architecture (SOA)

• Defined by OASIS as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.
SOA with JBoss Fuse Service Works

• A high performance messaging broker supports messaging patterns such as publish-subscribe, point-to-point and store-forward, and multiple cross language clients.

• Service development using the web services framework, which exposes integration assets as services and calls external services, supporting all major web services standards. It also supports RESTful calls.

• Structured service development through a lightweight framework that provides full lifecycle support for developing, deploying, and managing service-based applications.
SOA with JBoss Fuse Service Works

• The capability of an **automated registry with workflow** to allow you to manage the lifecycle of services from design, development and deployment by defining, exposing and enforcing rules or policies.

• **Business transaction monitoring** to capture service activity information, define and collect metrics, and define alerts and SLAs.
Red Hat JBoss BPM Suite
Features and Capabilities of BPM Suite 6.0
BPM Suite Example Application

• Application developed entirely within the web process designer.
• Sample mortgage application processing.
• Uses various BPMS features including data modeler, rules, task and process forms, user tasks, web service task, decision gateways, swimlanes and so on.
• Released alongside the BPMS Reference Architecture and documented in detail.
• Released alongside the produce in the Customer Support Portal.
Business Central - Repositories

• BPMS uses asset repositories (knowledge stores) that contain projects.
• Each asset repository uses a Git repository to store its data.
• Business Central provides a front-end to allow users to manage these repositories and their content.
• Business Central allows access to the Git repositories through the Git and SSH protocols. Pushing content back is only possible through SSH for security.
Business Central - Projects

• Each BPMS repository contains one or multiple projects.
• BPMS projects are Maven projects.
• Each BPMS Maven project may depend on another Maven project or act as a dependency of other projects.
• Maven project object models (pom) or JAR files for a BPMS project may be accessed through REST calls.
Project General Settings

Project Name: MortgageApplication
Project Description: Mortgage application to demo basic features of BPMS 6

Group artifact version

Group ID: com.redhat.bpms.examples
Artifact ID: mortgage
Version ID: 1

Example: com.myorganization.myprojects
Example: MyProject
Example: 1.0.0
Graphical Data Modeler

• The BPMS web designer includes a web-based custom graphical data modeler that is accessed from the Tools menu.

• The data modeler allows Java classes to be created and modified without requiring Java development expertise.

• Drop-downs and fields are presented to create new types, fields and composite objects. User-friendly labels can be set for classes and their field names.

• Classes created by the data modeler are regular Java Beans with annotations added.
Process Design

• Business processes can be created with either the web process designer or the eclipse-based tooling.
• The web process designer links to and automatically opens related tools, such as the form designer and the data modeler.
• The web process designer is specially ideal for non-developers who do not routinely use a Java IDE.
• The web tooling allows processes to be started, input variables to be set and even test data to be provided.
User Tasks

• A User Task represents a piece of work the given user can claim and perform. It allows human interaction to be modeled in a business process.

• A User Task can be assigned to a particular actor, multiple actors, or to a group of actors. If assigned to multiple actors or a group of actors, it is visible in the Task Lists of all the actors and any of the possible actors can claim the task and execute it.

• Tasks can have any number of input and output variables that mapped from or to the variables of the parent business process.
Process and User Task Forms

• Processes can be started through a custom form where initial values may be provided for some or all process variables.

• User tasks may be viewed, claimed and completed through Business Central, in which case, task forms are used to display the input variables and obtain the values of the output variables from the user.

• Both process and user task forms are designed and edited with the form designer in Business Central.

• The form designer supports mapping fields to object graphs.
Business Rules

• Red Hat JBoss BPM Suite 6 comes with integrated Red Hat JBoss BRMS 6.

• Various formats and tools are available for rules, including but not limited to technical rules, guided business rules, decision tables, domain specific languages and so on.

• Processes can incorporate rules by using a Business Rule Task that evaluates a number of specified rules based on available data.
<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Credit Score (\geq)</th>
<th>Credit Score (&lt;)</th>
<th>Fixed Mortgage</th>
<th>Mortgage APR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>660</td>
<td></td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>660</td>
<td>680</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>680</td>
<td>700</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>700</td>
<td>720</td>
<td>30</td>
<td>4.75</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>720</td>
<td>740</td>
<td>30</td>
<td>4.625</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>740</td>
<td></td>
<td>30</td>
<td>4.5</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>660</td>
<td>15</td>
<td>9.25</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>660</td>
<td>680</td>
<td>15</td>
<td>4.25</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>680</td>
<td>700</td>
<td>15</td>
<td>4.25</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>700</td>
<td>720</td>
<td>15</td>
<td>4</td>
</tr>
</tbody>
</table>
Swimlanes and Business Continuity

• Assigning tasks to groups has the advantage of avoiding tight coupling between individuals and business processes that may need attention outside that individual's working hours.

• In this model, any member of a group of users is able to view all assigned tasks and claim a task to work on.

• Swimlanes allow a task to be assigned to a group, but to undergo assignment a single time for each business process instance.

• Once a task in a swimlane has been claimed by an actor, all other instances of the same task or other tasks in the same swimlane will automatically be assigned to that same actor.
Service Tasks

• The pluggable implementation allows users to create various custom and service tasks that may be reusable and even made available in the palette with a distinct custom icon.

• For convenience, Log, Email, REST and Web Service tasks have been provided out of the box.

• For example, to use the Web Service Task, provide the required data mapping between process variables to the service parameter and from the service result, and provide the service location details.
<table>
<thead>
<tr>
<th>From Object</th>
<th>Assignment Type</th>
<th>To Object</th>
<th>To Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Url</td>
<td>is equal to</td>
<td></td>
<td><a href="http://localhost:8080/jboss-mortgage-demo-ws/CreditService?WSDL">http://localhost:8080/jboss-mortgage-demo-ws/CreditService?WSDL</a></td>
</tr>
<tr>
<td>2 Namespace</td>
<td>is equal to</td>
<td></td>
<td><a href="http://mortgage.examples.bpms.redhat.com/">http://mortgage.examples.bpms.redhat.com/</a></td>
</tr>
<tr>
<td>3 Interface</td>
<td>is equal to</td>
<td></td>
<td>CreditServiceService</td>
</tr>
<tr>
<td>4 Mode</td>
<td>is equal to</td>
<td></td>
<td>SYNC</td>
</tr>
<tr>
<td>5 Operation</td>
<td>is equal to</td>
<td></td>
<td>getCreditScore</td>
</tr>
<tr>
<td>6 ssn</td>
<td>is mapped to</td>
<td>Parameter</td>
<td></td>
</tr>
<tr>
<td>7 Result</td>
<td>is mapped to</td>
<td>creditScore</td>
<td></td>
</tr>
</tbody>
</table>
Error Handling

• Business errors can be thought of as error conditions that arise from invalid input data or unexpected results. Errors like these are best handled in the process design by using decision gateways and user tasks to take the necessary manual steps.

• Calling a web service introduces new risks. The external service may be down and nonfunctional for unexpected reasons. Various communication, network and server errors may result in an invalid response.

• An intermediate error catching event can redirect the process through a special flow to handle a technical error.
Process Simulation

• Business process simulation is a valuable analytic instrument for the purpose of assessing the behavior of business processes over time.

• This can help estimate the number of external calls, plan for human resources to handle the user tasks and otherwise assist in sizing and estimating technical and business requirements.

• The web process designer can show all the possible process paths, as calculated for a process. The number of paths depends on the number of decision points and the potential complexity of the process.
Running Process Simulation

• Before running process simulation, various simulation properties need to be set up for the process. This includes a probability value for each sequence flow that leaves a diverging gateway.

• Other simulation properties include the cost of executing a node per time unit, the minimum and maximum processing time that's envisioned for a node and how it may be distributed.

• For user tasks, staff availability and the number of working hours also effect process simulation.

• Simulation runs a number of processes at the given pace.
Business Activity Monitoring

• The Dashbuilder application included in BPM Suite 6 allows the creation of reports, panels, pages and entire workspaces for the purpose of business activity monitoring.

• The BPMS database is set up as a source of data by default but other external data sources may also be defined and added.

• A dashboard called jBPM is preconfigured with sample reports on BPMS processes and tasks; it queries the BPMS database.

• New reports can be created based on the BPMS database or in combination with other business data sources.
Red Hat JBoss Fuse Service Works
Features and Capabilities of Fuse Service Works 6.0
Fuse Service Works Capabilities

• Enterprise Service Bus (ESB)
• Service delivery
• Service governance
• Business transaction monitoring
Enterprise Service Bus

Fuse Service Works includes an ESB, consisting of:

- **Apache Camel**, to provide a modern enterprise integration pattern-based development model
- **Apache ActiveMQ**, as a high-performance messaging broker
- **Apache CXF**, a web services framework
Service delivery

• Fuse Service Works provides full life cycle support for developing, deploying and managing service-oriented applications.

• This is provided by **SwitchYard**, a lightweight service delivery framework, based on **Service Component Architecture (SCA)**.

• SwitchYard supports distributed and cross-functional teams with code and model-driven tooling, and enables developers to focus on the actual business logic by separating the underlying implementation.
Service governance

Life cycle management for services, from design and development through deployment, is based on the Overlord project, helping to define, expose and enforce rules or policies.
Business transaction monitoring

**Overlord** also provides the infrastructure to capture service activity information and then provides necessary metrics and alerts to help meet business and service level agreements (SLAs) and optimize the integration solutions.
SCA Assembly Model

• Service Component Architecture (SCA) provides a programming model for building applications and solutions based on a Service Oriented Architecture (SOA).

• SCA provides a model for the composition of services and creation of service components.

• In the SCA Assembly Model, SCA Domain includes composites which contain assemblies of service components and the connections and related artifacts, which describe how they are linked together.
Composite

• A composite is displayed as a light blue rectangle and represents the boundary between what is inside and what is outside an application.

• A SwitchYard application consists of exactly one composite that has a name and a targetNamespace.

• The target namespace value qualifies a name and makes it unique within a SwitchYard runtime.
Component

A component is a modular container for application logic and consists of the following:

• Zero or one component service definitions
• Zero to many component reference definitions
• One implementation, which is the actual logic for providing or consuming services
Component Implementation

SwitchYard provides multiple implementation options including:

• **Bean**, where a CDI Bean is used to provide a service or consume other services using annotations

• **Camel**, EIP-style routing and service composition using the XML or Java DSL in Apache Camel

• **BPMN 2**, service orchestration and human task integration expressed as BPMN 2 and executed using jBPM

• **BPEL**, web service orchestration using the OASIS standard

• **Rules**, decision services based on Drools
Component Service

- Used to expose the implementation as a service
- Java interface, WSDL, or ESB interface (named data types)
- Can only be invoked by components in the same application
- Promoted to a composite service to be exposed externally
Composite Service

• Represents a service that is visible to other applications
• Can only be realized by promoting a component service
• May have an interface that is different from the component service
• May require transformation to and from the type of the component service interface
Service Binding

- Used to define an access method for a composite service
- Multiple bindings may allow a single service to be accessed in different ways
- Typically represents a protocol or transport adapter such as SOAP, JMS, and REST
- The SCA binding is an exception, which allows services across applications in the same runtime to be wired together in memory
Component Reference

• Allows a component to consume other services
• May consume a service offered by another component in the same application
• May leverage a composite reference to consume a service outside the application
Composite Reference

• Allows a component reference to be wired to a service outside the application

• Reference bindings are used with composite references to specify the communication method for invoking the external service

• Available reference bindings are the same as service bindings

• There can only be one binding for each composite reference
FSW / BPMS Integration
Integration Fuse Service Works and BPM Suite
Fuse Service Works – Integration Capabilities

• As an integration platform, various protocols are supported
• Out of the box support for multiple inbound protocols
• Out of the box support for multiple outbound protocols
• Extensible to support new integration methods
BPM Suite – Outbound Integration Capabilities

• Out of the box service tasks:
  • WS Task for SOAP / HTTP calls
  • REST Task for JSON / HTTP calls
  • REST Task for XML / HTTP calls

• Custom development:
  • Synchronous and Asynchronous custom service tasks
  • Implement the WorkItemHandler interface
  • Add icon and make new task available in service task palette
BPM Suite – Web Service Task

• Simply drag and drop WS task from the palette
• Configure the task based on the Web Service WSDL:
  • Url: WSDL URL
  • Namespace: WSDL target namespace
  • Interface: service name
  • Mode: SYNC or ASYNC
  • Operation: WSDL operation
BPM Suite – Web Service Task

• In DataInputSet, set correct **Parameter** type for Web Service
• In DataOutputSet, set correct **Result** type for Web Service
• In Assignments:
  • Map to **Parameter** from a process variable or a constant
  • Map from **Result** to a process variable
BPM Suite – REST Task

• Simply drag and drop REST task from the palette
• Configure the task based on the service:
  • Url: service location
  • Method: HTTP method, may be GET, POST, PUT and DELETE
• REST services that use the GET and DELETE methods do not expect a message and react to the URL
BPM Suite – REST Task

• REST services that use the POST and PUT methods typically require the following two DataInputSet items added:
  • **ContentType**: For example, application/json or application/xml
  • **Content**: the actual message content
BPM Suite – REST Task

• The service response is return in String form as a DataOutputSet item called **Result**. This can be mapped to a process variable.

• `ReadTimeout` and `ConnectTimeout` provide further fine tuning of the `http` (Apache HttpClient) behavior.
BPM Suite – REST Task

• By default, the REST Task is not configured to authenticate
• The REST Task may be reconfigured for security by editing the work item handler configuration file within Business Central:
  • WEB-INF/classes/META-INF/CustomWorkItemHandlers.conf
• This file configures the public default constructor of the REST work item handler:
  • "Rest": new org.jbpm.process.workitem.rest.RESTWorkItemHandler()
BPM Suite – REST Task

• You may set up your environment to use a different constructor.
• To use Basic Authentication, use the constructor that takes the username and password:
  • "Rest": new org.jbpm.process.workitem.rest.RESTWorkItemHandler("myID", "myPassword")
• To use Form-based Authentication, use the constructor that takes the username and password:
  • "Rest": new org.jbpm.process.workitem.rest.RESTWorkItemHandler("myID", "myPassword", "http://local/passwd")
BPM Suite – REST Task

• Configuring the username, password or in the case of form-based authentication, the authentication URL in the property file affects all instances of all your processes that use the task.

• These values will be ignored and replaced by equivalent DataInputSet variables which may in turn be set to constant values in the process, or mapped from a process variable:
  • Username
  • Password
  • AuthUrl
BPM Suite – Inbound Integration Capabilities

• BPMS provides a REST API out of the box for incoming calls

• Individual application components accept slightly different implementations of this REST API:
  • Knowledge Store REST API: to manage the content
  • Deployment REST API: to deploy or undeploy kieModule jar files
  • Runtime REST API: calls to the Execution Server and the Process Execution Engine, Task Execution Engine, and Business Rule Engine. They allow you to request and manipulate runtime data.
BPM Suite – Runtime REST API

• Runtime REST API calls include:
  • Simple process calls
  • Simple task calls
  • History calls
  • Command executions
BPM Suite – Runtime REST API

• All calls are synchronous and return JAXB objects
• Except the Execute calls, all other REST calls can either use JAXB or JSON
• To use JSON, the JSON media type ("application/json") should be added to the ACCEPT header
• Parameters are defined as query string parameters for GET calls
• To pass a Map parameter, you can submit key-value pairs to the operation using a query parameter prefixed with the keyword map_ keyword, e.g. ?map_age=5000
BPM Suite – Simple Process Calls

These are simple GET or POST calls that accept either JSON or XML. Examples include:

- Start a process instance (accepts query map parameters):
  - POST /runtime/{deploymentId}/process/{procDefID}/start
- Return process instance details:
  - GET /runtime/{deploymentId}/process/instance/{procInstanceId}
- Signal process instance (accepts query map parameters):
  - POST /runtime/{deploymentId}/process/instance/{procInstanceId}/signal
BPM Suite – Simple Task Calls

These are simple GET or POST calls that accept either JSON or XML. Examples include:

• List task summary data:
  • GET /task/query

• Retrieve task content in JAXB XML form:
  • GET /task/content/{contentID}

• Complete task (accepts query map parameters):
  • POST /task/{taskId}/complete
BPM Suite – History Calls

These are simple GET or POST calls that accept either JSON or XML. Examples include:

• Return all logs of process instance:
  • GET /runtime/{deploymentId}/history/instance/{procInstanceId}

• Return variables of process instance:
  • GET /runtime/{deploymentId}/history/instance/{procInstanceId}/variable

• Return logs of all process instances:
  • GET /runtime/{deploymentId}/history/instance
BPM Suite – Command Execution

• Available through both REST and JMS API
• Advanced usage, including batch execution
• Only accepts XML (representing JAXB Command classes)
• Client API helps create and send commands
BPM Suite – REST Client API

• Use the appropriate remote factory:
  • org.kie.services.client.api.RemoteRestRuntimeFactory
  • org.kie.services.client.api.RemoteJmsRuntimeEngineFactory

• Obtain the abstract RuntimeEngine from the factory, for example:

```java
String applicationContext = "http://localhost:8080/business-central";
String deploymentId = "com.redhat.bpms.examples:mortgage:1";
URL jbpmURL = new URL( applicationContext );
RemoteRestRuntimeFactory remoteRestSessionFactory =
    new RemoteRestRuntimeFactory( deploymentId, jbpmURL, userId, password );

RuntimeEngine runtimeEngine = remoteRestSessionFactory.newRuntimeEngine();
```
BPM Suite – REST Client API

• The transport will be transparent to the client code in the following steps
• Get the KIE Session and call the required API

```java
KieSession kieSession = runtimeEngine.getKieSession();
Map<String, Object> processVariables = new HashMap<String, Object>();
Application application = new Application();
application.setDownPayment(100000);
processVariables.put("application", application);
String processId = "com.redhat.bpms.examples.mortgage.MortgageApplication";
kieSession.startProcess( processId, processVariables );
```
BPM Suite – Command Execution Client API

• The Client API can also help with command execution:

```java
JAXBCommandsRequest jaxbCommandsRequest = new JAXBCommandsRequest();
jaxbCommandsRequest.setDeploymentId(deploymentId);

List<Command<?>> commands = new ArrayList<Command<?>>();
StartProcessCommand startProcessCommand = new StartProcessCommand();
startProcessCommand.setProcessId(processId);
startProcessCommand.setParameters(processVariables);
commands.add(startProcessCommand);

jaxbCommandsRequest.setCommands(commands);

String xmlRequest = new JAXBSerializationProvider().serialize(jaxbCommandsRequest);
```
Calling BPMS from FSW

• Given native BPMN support, some of the BPM Suite libraries are available in Fuse Service Works
• Fuse Service Works is missing some BPMS libraries, notably those used for remote access and client API
• Fuse Service Works will provide better support for remote calls to BPM Suite in later releases
• As a practical matter, the use of BPMS libraries from the FSW 6.0 environment to make remote calls is challenging
Calling BPMS from FSW

• To avoid the challenges and library conflicts in FSW 6.0, it's easiest to use simple Java, including DOM and JAXB libraries, and basic libraries such as Apache HTTP Client and RESTEasy.

• The goal is to produce the request in simple XML and use HTTP POST to send the request to BPMS.

• When using custom classes in the request (e.g., process variables), these custom classes must be serialized to XML.

• Third-party libraries may help serialize custom classes that are not JAXB compliant.
Serializing JavaBeans classes to XML

Using only JDK libraries to parse JavaBeans:

```java
BeanInfo beanInfo = Introspector.getBeanInfo( bean.getClass() );
for( PropertyDescriptor descriptor : beanInfo.getPropertyDescriptors() )
{
    String fieldName = descriptor.getName();
    if( "class".equals( fieldName ) )
    {
        continue;
    }
    Object value = descriptor.getReadMethod().invoke( bean, new Object[] {} );
    if( value != null )
    {
        Element child = document.createElement( fieldName );
        element.appendChild( child );
        if( isPrimitiveOrWrapper( descriptor.getPropertyType() ) )
            child.setTextContent( value.toString() );
        else
            addBeanProperties( child, value );
    }
}
```
Serializing JavaBeans classes to XML

• Recursion can be used to serialize nested JavaBeans
• The top-level object is mapped to a <value> element

```xml
<value xsi:type="application" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <amortization>30</amortization>
  <applicant>
    <income>100000</income>
    <name>Amy</name>
    <ssn>333224449</ssn>
  </applicant>
  <downPayment>100000</downPayment>
  <property>
    <address>12301 Wilshire</address>
    <price>500000</price>
  </property>
</value>
```
Generating command-request XML

• The request can be created with the primitive DOM library
• The request XML document has the same consistent wrapper:

```java
documentBuilderFactory.setNamespaceAware( true );

Element commandRequest = document.createElement( "command-request" );

Element deploymentIdNode = document.createElement( "deployment-id" );
deploymentIdNode.setTextContent( deploymentId );
commandRequest.appendChild( deploymentIdNode );

Element ver = document.createElement( "ver" );
ver.setTextContent( "1" );
commandRequest.appendChild( ver );
```
Generating start-process XML

• The start-process request is an element inside command-request

• When starting a process, the initial process variable values are passed as a parameter withing the request

```java
Element commandRequest = getCommandRequest( deploymentId, processId );
Element startProcess = commandRequest.getOwnerDocument().createElement( "start-process" );
startProcess.setAttribute( "processId", processId );
commandRequest.appendChild( startProcess );
addParameters( startProcess, params );
```
Adding parameters

• The process variable is provided as an item within a parameter

```java
Document document = command.getOwnerDocument();
Element parameterElement = document.createElement("parameter");
command.appendChild(parameterElement);
for (Entry<String, Object> entry : params.entrySet()) {
    String beanName = entry.getKey();
    Object beanObject = entry.getValue();
    Element item = document.createElement("item");
    item.setAttribute("key", beanName);
    parameterElement.appendChild(item);
    Element value = document.createElement("value");
    value.setAttributeNS("http://www.w3.org/2001/XMLSchema-instance", "xsi:type", beanName);
    item.appendChild(value);
    addBeanProperties(value, beanObject);
}
return parameterElement;
```
Start Process XML request

```xml
<?xml version="1.0" encoding="UTF-8"?>
<command-request>
  <deployment-id>com.redhat.bpms.examples:mortgage:1</deployment-id>
  <ver>1</ver>
  <start-process processId="com.redhat.bpms.examples.mortgage.MortgageApplication">
    <parameter>
      <item key="application">
        <value xsi:type="application" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <amortization>30</amortization>
          <applicant>
            <income>100000</income>
            <name>Amy</name>
            <ssn>333224449</ssn>
          </applicant>
          <downPayment>100000</downPayment>
          <property>
            <address>12301 Wilshire</address>
            <price>500000</price>
          </property>
        </value>
      </item>
    </parameter>
  </start-process>
</command-request>
```
Creating a RESTEasy ClientRequest

• Given application URL and credentials, create a ClientRequest
• Apache HTTP Client must provide preemptive authentication

```java
BasicHttpContext localContext = new BasicHttpContext();
HttpClient preemptiveAuthClient = createPreemptiveAuthHttpClient( userId, password, 5, localContext );
ClientExecutor clientExecutor = new ApacheHttpClient4Executor( preemptiveAuthClient, localContext );
ClientRequestFactory requestFactory =
    new ClientRequestFactory( clientExecutor, new URI( applicationContext ) );

return requestFactory.createRelativeRequest( relativeURL );
```
HTTP Client with Preemptive Authentication

```java
BasicHttpParams params = new BasicHttpParams();
HttpConnectionParams.setConnectionTimeout( params, timeout * 1000 );
HttpConnectionParams.setSoTimeout( params, timeout * 1000 );
DefaultHttpClient client = new DefaultHttpClient( params );
if( userName != null && !"".equals( userName ) )
{
    client.getCredentialsProvider().setCredentials(
        new AuthScope( AuthScope.ANY_HOST, AuthScope.ANY_PORT ),
        new UsernamePasswordCredentials( userName, password ) );
    BasicScheme basicAuth = new BasicScheme();
    String contextId = UUID.randomUUID().toString();
    localContext.setAttribute( contextId, basicAuth );
    client.addRequestInterceptor( new PreemptiveAuth( contextId ), 0 );
}
String userAgent = "org.kie.services.client (" + idGen.incrementAndGet() + "/" + hostname + ")";
HttpProtocolParams.setUserAgent( client.getParams(), userAgent );

return client;
```
HTTP request interceptor

```java
private static class PreemptiveAuth implements HttpRequestInterceptor {
    private final String contextId;

    public PreemptiveAuth(String contextId) {
        this.contextId = contextId;
    }

    public void process(final HttpRequest request, final HttpContext context) throws Exception {
        AuthState authState = (AuthState)context.getAttribute(ClientContext.TARGET_AUTH_STATE);
        if (authState.getAuthScheme() == null) {
            AuthScheme authScheme = (AuthScheme)context.getAttribute(contextId);
            CredentialsProvider credsProvider = (CredentialsProvider)context.getAttribute(ClientContext.CREDS_PROVIDER);
            HttpHost targetHost = (HttpHost)context.getAttribute(ExecutionContext.HTTP_TARGET_HOST);
            Credentials creds = credsProvider.getCredentials(new AuthScope(targetHost.getHostName(), targetHost.getPort()));
            authState.setAuthScheme(authScheme);
            authState.setCredentials(creds);
        }
    }
}
```
Sending the REST request

• The RESTEasy ClientRequest object is used to send the request
• The request is constructed as org.w3c.dom.Element

```java
ClientRequest restRequest =
    getClientRequest( applicationContext, "/runtime/" + deploymentId + "/execute", userId, password );

StringWriter xmlWriter = new StringWriter();
Transformer transformer = TransformerFactory.newInstance().newTransformer();
Element xmlElement = Serializer.getStartProcess( deploymentId, processId, vars );
transformer.transform( new DOMSource( xmlElement ), new StreamResult( xmlWriter ) );

String jaxbRequestString = xmlWriter.toString();
restRequest.body( MediaType.APPLICATION_XML, jaxbRequestString );
ClientResponse<Object> response = restRequest.post( Object.class );

return Status.OK.equals( response.getResponseStatus() );
```