Subscription Repository

Developer Guide

v.7.0

Legal Notices

Document id:

Table of Contents

In This Guide

This guide explains how to develop processes in order to customize the behaviour of subscription repository.

Audience

The audience for this guide is the Solutions Integrator (SI). The SI has a combination of some or all of the following capabilities:

- Understands and has a solid working knowledge of:
  - UNIX® commands
  - Windows® system administration
- Understands networking concepts and language
- Is able to program in Java™ and XML
- Understands security issues
- Understands the customer's problem domain

References

- Subscription Repository – Administration Reference 1.0

Manual Organization

This guide contains the following chapters:

- Chapter 1, "Introduction", provides a brief explanation about the purpose, the scope and the definitions involved in this document.
- Chapter 2, "External Processes", provides a description of implementation for new customized processes.

Conventions

The following typographical conventions are used in this guide.

Font	**What the Font Represents**
Italic	Book or manual titles, and man page names
	Provides emphasis
	Specifies a variable that you must supply when entering a command
	Parameters to a method
Bold	New terms
`Computer`	Text and items on the computer screen
	Command names
	Method names
	File and directory names
	Process names
	Window/dialog box names
	XML tag references
`Computer Bold`	Text that you must type
Keycap	Keyboard keys
[Button]	Buttons on the user interface
Menu Items	A menu name followed by a colon (:) means that you select the menu, then the item. When the item is followed by an arrow (->), a cascading menu follows

Install Location Descriptors

The following names are used throughout this guide to define install locations.

Descriptor	**What the Descriptor Represents**
`$JBOSS_HOME`	The install location for JBoss. The location is depending on the path defined in installation process (i.e. <drive>:\jboss).

1. Introduction

1.1. Purpose

This document is a manual for the delivery team of the Subscription Repository. It explains how to develop new customized processes for the Subscription Repository system.

1.2. Document Scope

This document is oriented for the delivery team of the Subscription Repository.

2. External Processes

The SR application provides the possibility of defining external processes to interact with the system. The main objective of this mechanism is to provide a way to develop external processes for migration, information export purposes or any other task required in a project. For this purpose, a Java API is provided (see `$JBOSS_HOME/srapi` folder) in order to define and develop their own processes.

2.1. Internal Architecture

Bulk processes are defined as java classes (implementing an interface provided with the application), in which the developer implements the logic to process the product information obtained from the system.

The java API is provided to the developer containing the bulk process interface and the basic classes for invoking the CRUD operations. That way any kind of migration process can be implemented. The indexing process in Solr is implemented internally using this solution.

There are two basic interfaces to extend: *SRSubscriptionProcess* and *SRSystemProcess*. First it´s normally used for all processes that manage subscriptions. The other manages other processes, like SQLGenerator.

You must implement the interface that needed for your purposes.

The execution of the subscription process consists of two main parts. The first one consists of executing the query to obtain the subscription objects to be processed. Then, for each subscription the method processSubscription defined by the user, has to be invoke.
In other side, system process "only" has a method ("run()") where developer has to implement all his functionality.

Hierarchy of SRProcess is:

- SRProcess
    a. SRSpecificationProcess (Restricted process):
        i. IndexationProcess.
    b. SRSubscriptionProcess:
        i. MigrationProcess
        ii. … (To implement).
    c. SRSystemProcess
        i. IndexationOnBatch
        ii. … (To implement).

2.2. How Implement Process

Develop an external process consists in create a Java class implementing the interface *SRSubscriptionProcess* or *SRSystemProcess*, so this new process will implement the following methods:

Method getName

This method returns the name of the process. This name will represent the process in the system. The signature for this method is:

```
public String getName();
```

Method getDescription

This method returns a human readable description of the process. The signature for this method is:

```
public String getDescription();
```

Method getType

This method indicates if the process is either a migration or an export process. The signature for this method is:

```
public ProcessType getType();
```

ProcessType is an enumeration object which contains the supported process types. Currently MIGRATION, EXPORT and INDEXATION (internal type) process types are supported.

Method isTransactional

This method indicates if the process has to be transactional (if some error occurs, all the changes made in the system will be undo), or not transactional (if a problem occurs all previous changes will remain in the system). The signature for this method is:

```
public Boolean isTransactional();
```

Method getDefaultQuery (only for *SRSubscriptionProcess*)

During the process invocation you can specify a query indicating the group of subscriptions you want to work with. Anyway, with this method there is the possibility of defining a default query to be used if no query is specified in the invocation. The signature for this method is:

```
public Query getDefaultQuery();
```

Method init

This method is invoked when you start the process and is the initialization method. The signature for this method is:

```
public void init(Map<String, String> params);
```

It receives a hashtable containing the process params (these params are optional).

Method processSubscription (only for *SRSubscriptionProcess*)

Once the query is executed, this method is invoked for each subscription obtained. The signature for this method is:

```
public SROperationResult processSubscription(SRSession session, SRSubscription subscription);
```

The method receives the subscription object and an instance of the class SRSession. With the subscriptions object you get all information about the subscription and with the SRSession object you can interact with the system (CRUD operations).

The method returns a SROperationResult object to the system, it indicates whether the method has been executed correctly, or some error has occurred.

If some error has raise, it also contains an object representing this error (SROperationError).

Method endHandler

This method is executed after the process in case everything has worked properly. The signature for this method is:

```
public void endHandler();
```

Method run (only for *SRSystemProcess*)

This method is executed when to process main logic, so you should fill with your process code. The signature for this method is:

```
public SROperationResult run() throws Exception;
```

Method errorHandler

This method is executed when an error (severity Mayor) occurs during the process execution. The signature for this method is:

```
public void errorHandler();
```

Method createHistoryEntries (only for *SRSubscriptionProcess*)

This method will return true if you want that SR save history of each modified subscription. Normally, this value will be syncronized with SR configuration:

```
public boolean createHistoryEntries();
```

2.3. Java interfaces

2.3.1. SRProcess

```
/**
 * Bulk processors must implement this interface.
 *
 */
public interface SRProcess
{
  /**
   * Enumeration class with the supported process types. Currently <code>MIGRATION</code>, <code>EXPORT</code> and
   * <code>INDEXATION</code> process types are supported.
   *
   */
  public static enum ProcessType {
    /** Migration process. Migration processes are mutually exclusive. */
    MIGRATION,
    /** Export process. */
    EXPORT,
    /** Indexation process. For Solr indexing. */
    INDEXATION,
```

```
    /** System process. */
    SYSTEM
  }

  /**
   * Returns the name of the process. This name will represent the process in the system.
   *
   * @return the process name.
   */
  public String getName();

  /**
   * Returns a human readable description of the process.
   *
   * @return human readable description of the process.
   */
  public String getDescription();

  /**
   * This method should return the type of process being implemented. Currently <code>MIGRATION</code> and
   * <code>EXPORT</code> process types are supported.
   *
   * @return the type of process being implemented.
   */
  public ProcessType getType();

  /**
   * This method should return <code>true</code> if the implemented process is meant to be transactional. If the process
   * is considered transactional and some error occurs during execution, all changes made by it to the system will be
   * undo.
   *
   * @return <code>true</code> if the process is meant to be transactional.
   */
  public boolean isTransactional();

  /**
   * Initializes the processor with the given parameters.
   *
   * @param params
   *          a parameter map with initialization values.
   */
  public void init(Map<String, String> params);

  /**
   * This method is invoked after the subscription processing ends.
   */
  public void endHandler();

  /**
   * This method is invoked if an error occurs during process execution.
   *
   * @param error
   *          information about the raised error.
   */
  public void errorHandler(SROperationError error);

}
```

2.3.2. SRSystemProcess

```
/**
 *
 * Specific interface to define methods that implement logic of generic process.
 *
```

```
 */
public interface SRSystemProcess extends SRProcess {
            /**
             * Generic method to implement by processes.
             *
             * @return SROperationResult object with operation result content.
             * @throws Exception
             */
            public SROperationResult run() throws Exception;

            /**
            * Generic method to check if the process is executed in online or unplugged mode.
            * @return true if it´s online
            */
            public boolean isOnline();

}
```

2.3.3. SRSubscriptionProcess

```
package com.hp.om.sr.pm;

import com.hp.om.sr.bll.SRSession;
import com.hp.om.sr.component.SRSubscription;
import com.hp.om.sr.process.Scope;

/**
 *
 * Specific interface to define processes that operarate with subscriptions.
 *
 */
public interface SRSubscriptionProcess extends SRProcess {
             /**
              * Processes the given subscription.
              *
              * @param session
              *          the {@link SRSession} in which context the processor is run.
              * @param subscription
              *          the {@link SRSubscription} to be processed.
              * @return the process operation result.
              */
             public SROperationResult processSubscription(SRSession session, SRSubscription subscription);

             /**
              * Retrieve the default query for this processor.
              *
              * @return the default query for this processor.
              */
             public Scope getDefaultQuery();

             /**
              * Set to true if you want that update or delete operations will create copies of original subscription in history.
              *
              * @return true if update or delete operations will be saved in history.
              */
             public boolean createHistoryEntries();
}
```

2.4. Where Deploy Process

Deploy the class in the directory defined in parameter 'Dynamic Loading Folders' (see Administration Reference) of the SR installation (into a jar file). The process is immediately available, without any reset of system.

2.5. Examples

Here are some examples that demonstrating how to create external processes.

To reproduce these examples, you can create these specifications and subscriptions using the operations of the web services.

Invoke createSpecification in webservice operations as follow:

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:spec="http://www.sr.om.hp.com/Specifications">
  <soapenv:Header/>
  <soapenv:Body>
    <spec:createSpecification>
      <spec:customerFacingServices>
        <spec:customerFacingService>
          <spec:name>DSLService</spec:name>
          <spec:version>1.0</spec:version>
          <spec:description>DSL service</spec:description>
          <spec:creationDate>2010-08-31T12:00:00</spec:creationDate>
          <spec:updateDate>2001-09-02T15:00:00</spec:updateDate>
          <spec:characteristics>
            <spec:characteristic>
              <spec:name>upSpeed</spec:name>
              <spec:type>Integer</spec:type>
            </spec:characteristic>
            <spec:characteristic>
              <spec:name>downSpeed</spec:name>
              <spec:type>Integer</spec:type>
            </spec:characteristic>
          </spec:characteristics>
        </spec:customerFacingService>
      </spec:customerFacingServices>
      <spec:resources>
        <spec:resource>
          <spec:name>DSLRouter</spec:name>
          <spec:version>1.0</spec:version>
          <spec:description>DSL router</spec:description>
          <spec:creationDate>2010-08-31T12:00:00</spec:creationDate>
          <spec:updateDate>2001-09-02T15:00:00</spec:updateDate>
          <spec:characteristics>
            <spec:characteristic>
              <spec:name>manufacturer</spec:name>
              <spec:type>String</spec:type>
            </spec:characteristic>
            <spec:characteristic>
              <spec:name>model</spec:name>
              <spec:type>String</spec:type>
            </spec:characteristic>
            <spec:characteristic>
              <spec:name>version</spec:name>
```

```
            <spec:type>String</spec:type>
          </spec:characteristic>
        </spec:characteristics>
      </spec:resource>
    </spec:resources>
    <spec:products>
      <spec:product>
        <spec:name>DSLProduct</spec:name>
        <spec:version>1.0</spec:version>
        <spec:description>DSL product</spec:description>
        <spec:creationDate>2010-08-31T12:00:00</spec:creationDate>
        <spec:updateDate>2001-09-02T15:00:00</spec:updateDate>
        <spec:customerFacingServices>
          <spec:specificationRef>
            <spec:name>DSLService</spec:name>
            <spec:version>1.0</spec:version>
          </spec:specificationRef>
        </spec:customerFacingServices>
        <spec:resources>
          <spec:specificationRef>
            <spec:name>DSLRouter</spec:name>
            <spec:version>1.0</spec:version>
          </spec:specificationRef>
        </spec:resources>
      </spec:product>
    </spec:products>
  </spec:createSpecification>
 </soapenv:Body>
</soapenv:Envelope>
```

Invoke createSubscription in webservice operations as follow:

```
<soapenv:Envelope                                    xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:sub="http://www.sr.om.hp.com/Subscriptions">
  <soapenv:Header/>
  <soapenv:Body>
    <sub:createSubscription>
      <sub:product>
        <sub:name>DSLProduct</sub:name>
        <sub:version>1.0</sub:version>
        <sub:productCode>SN-0001</sub:productCode>
        <sub:customerCode>Michael</sub:customerCode>
        <sub:creationDate>2010-08-31T12:00:00</sub:creationDate>
        <sub:updateDate>2001-09-02T15:00:00</sub:updateDate>
        <sub:customerFacingServices>
          <sub:customerFacingService>
            <sub:name>DSLService</sub:name>
            <sub:version>1.0</sub:version>
            <sub:characteristics>
              <sub:characteristic>
                <sub:name>upSpeed</sub:name>
                <sub:value>10</sub:value>
              </sub:characteristic>
              <sub:characteristic>
                <sub:name>downSpeed</sub:name>
                <sub:value>10</sub:value>
              </sub:characteristic>
            </sub:characteristics>
          </sub:customerFacingService>
        </sub:customerFacingServices>
        <sub:resources>
          <sub:resource>
```

```
            <sub:name>DSLRouter</sub:name>
            <sub:version>1.0</sub:version>
            <sub:characteristics>
              <sub:characteristic>
                <sub:name>manufacturer</sub:name>
                <sub:value>Alcatel</sub:value>
              </sub:characteristic>
              <sub:characteristic>
                <sub:name>model</sub:name>
                <sub:value>Alcatel-DSL</sub:value>
              </sub:characteristic>
              <sub:characteristic>
                <sub:name>version</sub:name>
                <sub:value>1.1</sub:value>
              </sub:characteristic>
            </sub:characteristics>
          </sub:resource>
        </sub:resources>
      </sub:product>
    </sub:createSubscription>
  </soapenv:Body>
</soapenv:Envelope>
```

2.5.1. Export Process

This example shows how to recover all names of the subscriptions whose customer is passed as parameter to the init method. (A defaultQuery is defined to recover all subscriptions). This example is not transactional.

```
package com.hp.om.sr.cp;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import com.hp.om.sr.bll.SRSession;
import com.hp.om.sr.component.SRSubscription;
import com.hp.om.sr.pm.SROperationError;
import com.hp.om.sr.pm.SROperationResult;
import com.hp.om.sr.pm.SROperationResult.Code;
import com.hp.om.sr.subscriptions.Query;

public class ShowCustomerSubscriptions implements com.hp.om.sr.pm. SRSubscriptionProcess
{
  /**
   * Customer name.
   */
  private String customerName = "";
  /**
   * List to contain the names of the subscriptions.
   */
  private List<String> subscriptionsNames = new ArrayList<String>();

  /**
   * Returns the name of the process. This name will represent the process in the system.
   *
   * @return the process name.
   */
  public String getName()
  {
    return " ShowCustomerSubscriptions";
```

```
}

/**
 * Returns a human readable description of the process.
 *
 * @return human readable description of the process.
 */
public String getDescription()
{    return " Recover all names of the subscriptions whose customer is passed by parameter ";  }

/**
 * This method should return the type of process being implemented.
 * @return the type of process being implemented.
 */
public ProcessType getType()
{
  return ProcessType.EXPORT;
}

/**
 * This method should return <code>true</code> if the implemented process is meant to be transactional. If the process
 * is considered transactional and some error occurs during execution, all changes made by it to the system will be
 * undo.
 *
 * @return <code>true</code> if the process is meant to be transactional.
 */
public boolean isTransactional()
{
  return false;
}

/**
 * Retrieve the default query for this processor.
 *
 * @return the default query for this processor.
 */
public Query getDefaultQuery()
{
  // All subscriptions
  Query query = new Query();
  return query;
}

/**
 * Initializes the processor with the given parameters.
 *
 * @param params
 *          a parameter map with initialization values.
 */
public void init(Map<String, String> params)
{
  if (params != null && !params.isEmpty()) {
    customerName = (params.containsKey("CUSTOMERNAME")) ? params.get("CUSTOMERNAME") : null;
  }
}

/**
 * Processes the given subscription.
 *
 * @param session
 *          the {@link SRSession} in which context the processor is run.
 * @param subscription
 *          the {@link SRSubscription} to be processed.
```

```
   * @return the process operation result.
   */
  public SROperationResult processSubscription(SRSession session, SRSubscription subscription)
  {
    if (customerName == null) {
      SROperationError error = new SROperationError("036", "Process stopped", SROperationError.Severity.MAJOR, null);
      return new SROperationResult(Code.ERROR, "Error - A init param with key CUSTOMERNAME must exist", error);
    } else {
      String subscriptionCustomer = subscription.getBean().getCustomerCode();
      if (customerName.equals(subscriptionCustomer)) {
        String subscriptionName = subscription.getBean().getName();
        subscriptionsNames.add(subscriptionName);
      }
      return new SROperationResult(Code.OK, "Name recovered correctly");
    }
  }

  /**
   * This method is invoked after the subscription processing ends.
   */
  public void endHandler()
  {
    for (Iterator<String> iterator = subscriptionsNames.iterator(); iterator.hasNext();) {
      String name = iterator.next();
      System.out.println(customerName + " subscription's with name : " + name);
    }
  }

  /**
   * This method is invoked if an error occurs during process execution.
   *
   * @param error
   *          information about the raised error.
   */
  public void errorHandler(SROperationError error)
  {
  }
}
```

Highlights:

- *Method getName*. Returns ShowCustomerSubscriptions as process name.
- *Method init*. Recovers the customer name passed by parameter.
- *Method getDefaultQuery*. All subscriptions to process.
- *Method processSubscription*. First, verifies that there is a customer passed by parameter and if there isn't then returns an error to the system. After recovers the customer name for each subscription to be processed and check if the customer name from the subscription is equal to the customer name passed by parameter. If the name matches, recovers the subscription name and adds it to the subscription name list.
- *Method endHandler*. Prints all subscription names.

An example of invoking the web service operation to execute this process is:

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope" xmlns:end="http://endpoint.sr.om.hp.com/"
xmlns:sub="http://www.sr.om.hp.com/Subscriptions">
  <soapenv:Header/>
  <soapenv:Body>
    <end:executeProcess>
      <name>ShowCustomerSubscriptions</name>
      <params>
```

```
          <entry>
            <key> CUSTOMERNAME</key>
            <value>Michael</value>
          </entry>
        </params>
      </end:executeProcess>
    </soapenv:Body>
</soapenv:Envelope>
```

2.5.2. Migration Process

This example is more complex, shows how to modify the value of a characteristic into a service for any subscription where the customer is Michael (in this example the query is specified during the process invocation to indicate the group of subscriptions to work).

The CustomerFacingService name is DSLService, the characteristic is upSpeed and the new characteristic value is passed as parameters to init method. This example is transactional, so if a problem occurs, all previous changes made in the system will be undo.

```
package com.hp.om.sr.cp;

import java.util.Iterator;
import java.util.List;
import java.util.Map;

import com.hp.om.sr.bll.SRSession;
import com.hp.om.sr.component.SRSubscription;
import com.hp.om.sr.pm.SROperationError;
import com.hp.om.sr.pm.SROperationResult;
import com.hp.om.sr.pm.SRProcess;
import com.hp.om.sr.pm.SROperationResult.Code;
import com.hp.om.sr.subscriptions.CustomerFacingService;
import com.hp.om.sr.subscriptions.Product;
import com.hp.om.sr.subscriptions.Query;
import com.hp.om.sr.subscriptions.Subscription.Characteristics;
import com.hp.om.sr.subscriptions.Subscription.Characteristics.Characteristic;

public class ModifyUPSpeed implements SRSubscriptionProcess
{
  /**
   * Customer Facing Service to modify.
   */
  private String cfsName = "DSLService";
  /**
   * Name of the characteristic to modify.
   */
  private String characteristicName = "upSpeed";

  /**
   * New value of the characteristic, this value is passed by parameter.
   */
  private String characteristicValue = "";

  /**
   * Returns the name of the process. This name will represent the process in the system.
   *
   * @return the process name.
   */
  public String getName()
  {
```

```
  return "ModifyUPSpeed";
}

/**
 * Returns a human readable description of the process.
 *
 * @return human readable description of the process.
 */
public String getDescription()
{
  return "External process to modify a specific characteristic of a specific "
      + "Customer Facing Service for any subscription where the customer is Michael.";
}

/**
 * This method should return the type of process being implemented.
 *
 * @return the type of process being implemented.
 */
public ProcessType getType()
{
  return ProcessType.MIGRATION;
}

/**
 * This method should return <code>true</code> if the implemented process is meant to be transactional. If the process
 * is considered transactional and some error occurs during execution, all changes made by it to the system will be
 * undo.
 *
 * @return <code>true</code> if the process is meant to be transactional.
 */
public boolean isTransactional()
{
  return true;
}

/**
 * Retrieve the default query for this processor.
 *
 * @return the default query for this processor.
 */
public Query getDefaultQuery()
{
  return null;
}

/**
 * Initializes the processor with the given parameters.
 *
 * @param params
 *          a parameter map with initialization values.
 */
public void init(Map<String, String> params)
{
  if (params != null && !params.isEmpty()) {
    characteristicValue = (params.containsKey("characteristicValue")) ? params.get("characteristicValue") : null;
  }
}

/**
 * Processes the given subscription.
 *
 * @param session
```

```
   *          the {@link SRSession} in which context the processor is run.
   * @param subscription
   *          the {@link SRSubscription} to be processed.
   * @return the process operation result.
   */
  public SROperationResult processSubscription(SRSession session, SRSubscription subscription)
  {
    if (characteristicValue == null) {
      SROperationError error = new SROperationError("036", "Process stopped", SROperationError.Severity.MAJOR, null);
      return new SROperationResult(Code.ERROR, "Error - A init param with key characteristicValue must exist.", error);
    }
    int cont = 0;
    try {
      // Recover the customerFacingServices.
      Product.CustomerFacingServices customerFacingServices = subscription.getBean().getCustomerFacingServices();
      List<CustomerFacingService> CFSList = customerFacingServices.getCustomerFacingServices();
      for (Iterator<CustomerFacingService> iterator = CFSList.iterator(); iterator.hasNext();) {
        CustomerFacingService customerFacingService = iterator.next();
        if (customerFacingService.getName().equals(cfsName)) {
          // Recover the characteristics of the CFS.
          Characteristics characteristics = customerFacingService.getCharacteristics();
          List<Characteristic> characteristicList = characteristics.getCharacteristics();
          for (Iterator<Characteristic> iterator2 = characteristicList.iterator(); iterator2.hasNext();) {
            Characteristic characteristic = iterator2.next();
            // Modify the characteristic value if the name matches
            if (characteristic.getName().equals(characteristicName)) {
              characteristic.setValue(characteristicValue);
              cont++;
            }
          }
        }
      }
      // CRUD operation - Update the subscription
      session.updateSubscription(subscription);
    } catch (Exception e) {
      // Return an error if an exception is produced.
      SROperationError error = new SROperationError("036", "Process stopped", SROperationError.Severity.MAJOR, e);
      return new SROperationResult(Code.ERROR, "ModifyUPSpeed process has found an error.", error);
    }
    return new SROperationResult(Code.OK, "UpSeed modified in " + cont + " subscriptions corretly");
  }

  /**
   * This method is invoked after the subscription processing ends.
   */
  public void endHandler()
  {

  }
  /**
   * This method is invoked if an error occurs during process execution.
   *
   * @param error
   *          information about the raised error.
   */
  public void errorHandler(SROperationError error)
  {

  }

}
```

Highlights:

- *Method getName*. Returns ModifyUPSpeed as process name.
- *Method init*. Recovers the new value to the characteristic upSpeed.
- *Method getDefaultQuery*. No query defined, the group of subscriptions is mandatory defined in the process invocation.
- *Method processSubscription*. First, verifies that there is new characteristic value passed by parameter and if there isn't then returns an error to the system. After for each subscription to be processed iterates over the list of customer facing services, retrieves theirs characteristics and modifies the characteristic with name upSpeed. Finally, uses the session object to update the subscription (persist the changes). If an exception is produced, then an error is returned to the system.
- *Method endHandler*. Prints all subscription names.

An example of invoking the web service operation to execute this process is:

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope" xmlns:end="http://endpoint.sr.om.hp.com/"
xmlns:sub="http://www.sr.om.hp.com/Subscriptions">
  <soapenv:Header/>
  <soapenv:Body>
    <end:executeProcess>
      <name>ModifyUPSpeed</name>
      <query>
        <sub:customerCode>Michael</sub:customerCode>
      </query>
      <params>
        <entry>
          <key>characteristicValue</key>
          <value>150</value>
        </entry>
      </params>
    </end:executeProcess>
  </soapenv:Body>
</soapenv:Envelope>
```

3. Security and authorization

This topic covers the authorization configuration in the SR. There are several roles defined, one for each web service method. So, the administrator can manage users and associate roles, to grant o deny the access to any method.

Developer must follow the code style to ensure functionallity. Data security depends on role definition and configuration.

3.1. Roles

There is one role by each method.

Any modification or new role will be notified to administrators, because implies more changes in configuration files.

If you develop new web service method must add "@SRRolesAllowed" annotation, with new role. For instance, *getSRStatus* method (*ManagementEndPoint*):

```
@WebMethod(operationName = GET_STATUS)
@WebResult(targetNamespace = PROCESSES_TARGET_NAMESPACE)
@SRRolesAllowed(values = { "SRStatusRole" })
public com.hp.om.sr.process.ResultStatus getSRStatus(Void request) throws SOAPException
```

Administrators will add the new role, in `$JBOSS_HOME/server/default/conf/props/jbossws-roles.properties`, to desire users.

3.2. Securizing Solr connections

If you need to have Solr in only one server, without SR application, and configure Solr-Jboss service to use only secure connections. Then you must adapt SR application.

For example, you have two servers:

- First with Solr running into Jboss server that only listens in '443' port (secure connection).
- Second with SR running into Jboss server than only listens in '443' port (secure connection).

So, clients connect to second server through secure connection, and both servers communicate with secure connection. It´s not a normal deploy enviroment.

In this case, you should refactor "*SRSolrTransaction.innerBegin()*":

```
protected void innerBegin() throws Exception
{
  if (isStarted()) {
    throw new SRTransactionException("Transaction already in process").setErrorCode(System.SEARCH_ENGINE,
        Subsystem.SOLR, "082");
  } else {
    try {
      int solrUrlIndex = 0;
      if (server == null) {
        server = new SolrServer[SRContext.getSolrClusterSize()];
      }
      for (String solrUrl : SRContext.getSolrUrls()) {
```

```
        server[solrUrlIndex++] = new CommonsHttpSolrServer(solrUrl);
      }
      clear();
    } catch (Exception e) {
      new SRTransactionException("Could not stablish connection with Solr server").setErrorCode(System.SEARCH_ENGINE,
          Subsystem.SOLR, "083");
    }
    logger.debug("Connection from Solr server available");
  }
}
```

To:

```
protected void innerBegin() throws Exception
{
  if (isStarted()) {
    throw new SRTransactionException("Transaction already in process").setErrorCode(System.SEARCH_ENGINE,
        Subsystem.SOLR, "082");
  } else {
    try {
      int solrUrlIndex = 0;
      if (server == null) {
        server = new SolrServer[SRContext.getSolrClusterSize()];
      }
      for (String solrUrl : SRContext.getSolrUrls()) {
            DefaultHttpClient httpclient = new DefaultHttpClient();
            KeyStore trustStore  = KeyStore.getInstance(KeyStore.getDefaultType());
            FileInputStream instream = new FileInputStream(new File("sr.keystore"));

            try {
                trustStore.load(instream, "SRkeystorePassword".toCharArray());
            } finally {
                try { instream.close(); } catch (Exception ignore) {}
            }

            SSLSocketFactory socketFactory = new SSLSocketFactory(trustStore);
            Scheme sch = new Scheme("https", 443, socketFactory);
            httpclient.getConnectionManager().getSchemeRegistry().register(sch);

        server[solrUrlIndex++] = new CommonsHttpSolrServer(solrUrl, httpclient);
      }
      clear();
    } catch (Exception e) {
      new SRTransactionException("Could not stablish connection with Solr server").setErrorCode(System.SEARCH_ENGINE,
          Subsystem.SOLR, "083");
    }
    logger.debug("Connection from Solr server available");
  }
}
```

Actual SR connections are established through CommonsHttpSolrServer (using *SRConstants.DEFAULT_SOLR_URL*). If you desire connect SR with solr, over secure connection, you cannot use one url like "https …". Follow previous example and read HttpClient documentation, because you need to use HttpClient (it´s impossible with a basic CommonsHttpSolrServer).