SWAMP - Adminguide

How to administrate the SWAMP platform

1. What is SWAMP?

SWAMP is a platform that drives workflows. Workflows are, especially in a professional or business environment, processes that involve many people. Typically processes require input of distinct people in a more or less fixed sequence. Some people urgently need to be informed about progress in the process, others may want to be informed.

Unfortunately, processes are not fix in the real world and they are very hard to be defined. Process definitions change as fast as the opinions of the people taking part.

SWAMP wants to help to provide as much help as software can provide to improve processes. Software can not talk to people and try to motivate them, but software can help people to organise their daily work and give them a more free head for interesting jobs by taking over the organisation of boring parts.

Tasks and work sequences that are always of a recurring, similar type and that consist of unchanging subtasks may be defined as a workflow: A chronological sequence of similar tasks, that may or may not depend on each other. These subtasks are handled by one or more persons. The Information about ongoing and already done tasks needs to be spread.

SWAMP wants to accept the challenge to be a platform to drive workflows. As mentioned above, processes that are described by workflows change rapidly. That makes it expensive and inefficient to 'hardcode' workflows. Workflows need to be easy defineable for people that are not programmers. SWAMP offers a XML based language (basically a dtd ;-) to define workflows. Once a workflow is defined by one person, it can be used in SWAMP. The target is that no programmer need to touch the SWAMP core for new workflows that come from a new XML file.

2. Overview of the SWAMP architecture

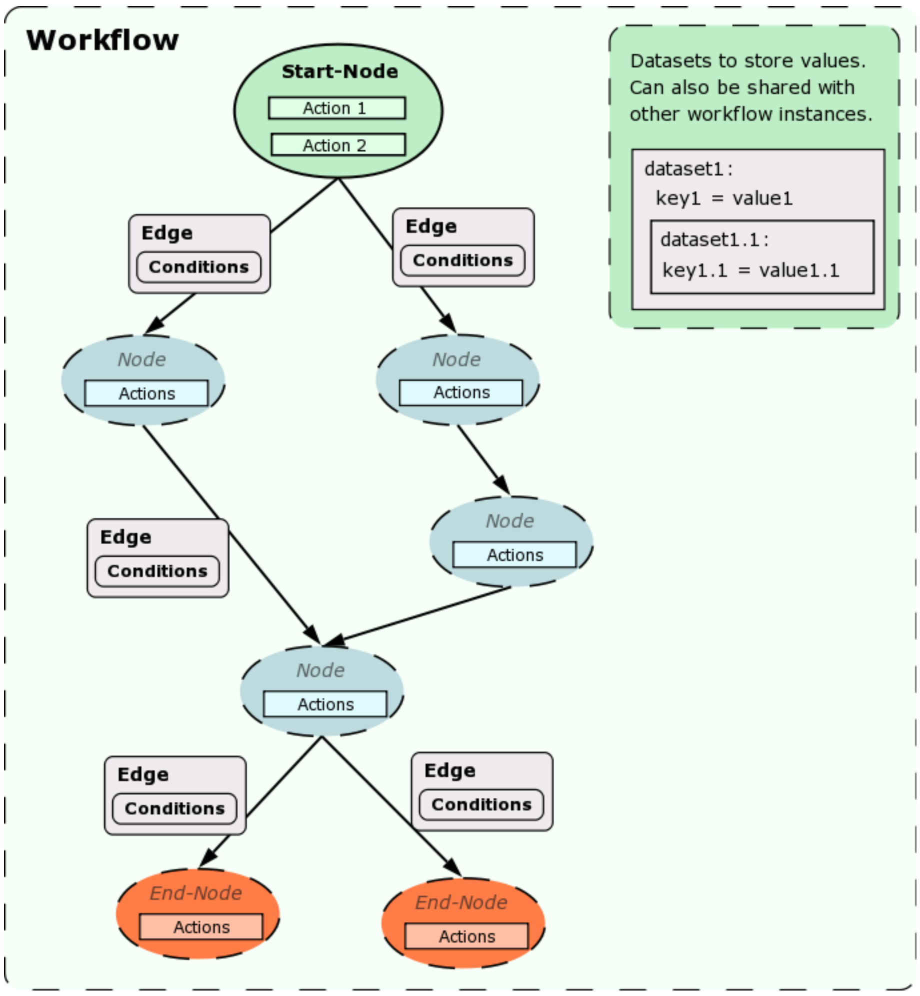
2.1. The SWAMP workflow model

A workflow is quite similar to the world. There are a lot of villages, connected to each other by streets. Think of a craftsman on a walk. He was born in one village and on some day he starts on a walk to the world. Probably and hopefully he will find something to do in every village and after he finished the job he continues his way along one of the ways to the next village. The way he takes may depend on the result of the job he did in the last village or on other events.

Thats quite similar to how workflows are processed in SWAMP: In SWAMP the workflow's state engine acts as the craftsman, the villages are nodes. Roads that connect villages are edges in SWAMP that connect nodes. And finally, the jobs are similar to tasks in SWAMP.

A workflow is a compound object consisting of Nodes and Edges, which in turn are compound objects. The workflow definitions are read in from XML files, and a workflow template object is created for each workflow bundle. When a new workflow instance is requested , the template object is responsible for the creation of the workflow object. The workflow will run based on the template that it was created from. When a new version of a workflow-type was created, the running workflow instances are not affected. Only new workflow instances will be based on the new workflow definition. This mechanism assures that running workflows will not break on updates of the workflow definition.

A workflow has the following graph-like structure:



Each workflow has one or more datasets connected to it where it stores values that are needed for the workflow. These datasets can also be shared across workflow instances, to be able to create dependencies between different workflows.

2.2. Actions and Tasks

Whenever a node is entered, its included actions are triggered. Actions represent anything that has to be done, in any subsystem. Once an action is triggered, a task object is created that consists of:

- a reference to the action
- an (empty or pre-set) result set that fits the action
- The user/group to whom this task is assigned to

An action is the template for a task, which means that a task is a concrete instance of an action with a person/group assigned to it. Another possibility are system-tasks that are automatically done by the system, such as sending notifications, changing values of the dataset, starting subworkflows and so on. When a Task is done, it sends out an event, and thus drives the workflow on into the path where there is a condition waiting for that event.

2.3. Events and Conditions

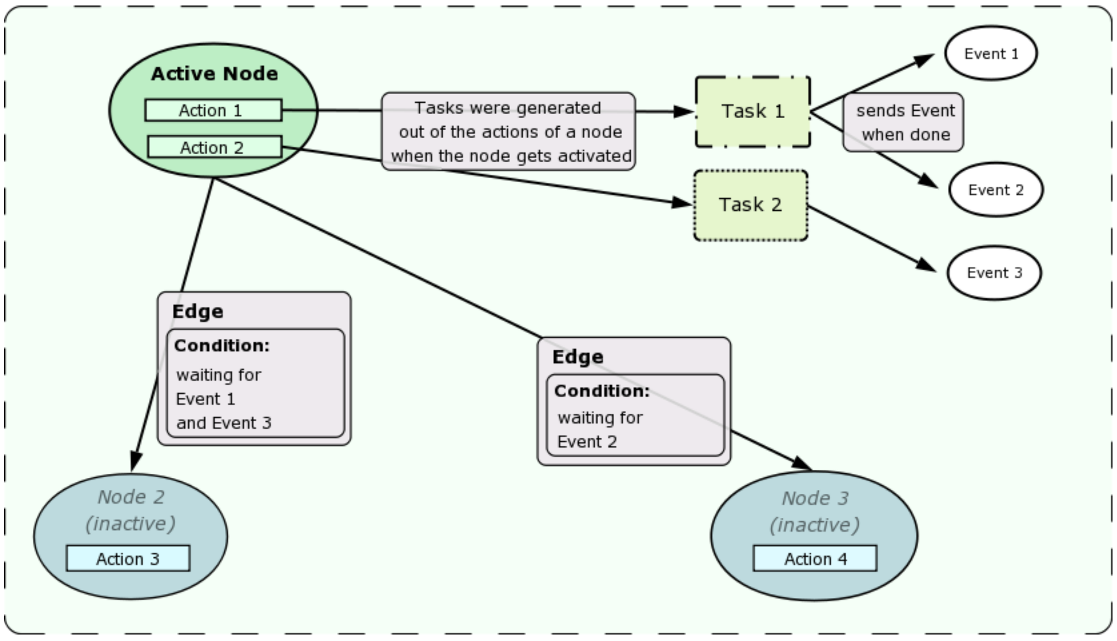
A node stays active until one or more of the edges leading out of it can be followed. Which means that the edge's condition must be met. Conditions are constructed out of the boolean operators AND, OR and NOT and the following atomic conditions: Either use only empty conditions to leave a node or only non-empty ones, mixing both doesn't make any sense at all. The empty conditions are used for modelling convenience.

- EventCondition: waits until the workflow receives a specific event.
- DataCondition: waits for a specific databit to be set to a certain value. Dataconditions can use regexps to evaluate the content of a databit, or just wait for the change of a databit-value.
- SubsfinishedCondition: waits for attached subworkflows to be finished.
- Empty: edges that should always be followed immediately are realized as an EventCondition with the special event "none".

If several different edges with compound conditions leave a node, they are followed (i.e. the node is left) according to these precedence rules:

- Empty edges are followed immediately. If there are more than one, all of them are followed, so that the workflow can split its thread and afterwards has more than one active node at a time.
- On entering a node, all Conditions are checked. If this leads to (simple or compound) conditions of one ore more edges to resolve to true, these edges are followed.
- If the node is still active after 1. and 2., incoming events and data changes are processed in the order they occur. As soon as one or more simple or compound conditions resolve to true, the respective edge(s) are followed and the node is left

and won't receive any further events.


Active Node
Action 1
Action 2
Tasks were generated out of the actions of a node when the node gets activated
Task 1
Task 2
sends Event when done
Event 1
Event 2
Event 3
Edge
Condition: waiting for Event 1 and Event 3
Edge
Condition: waiting for Event 2
Node 2 (inactive)
Action 3
Node 3 (inactive)
Action 4

When a node is left, but still has active tasks that were not yet done, these tasks get cancelled.

2.4. Data

Data is organized in datasets, which in turn can contain other datasets and databits. A databit is a single key-value item with a datatype assigned to it to be able to verify user input. Available datatypes so far are:

- boolean (true/false, rendered by a checkbox in webSWAMP)
- number (integer)
- string (a text, rendered by a dynamically growing input field in webSWAMP)
- text (a text, rendered by text field of static size)
- person (comma-seperated list of loginnames and/or mail adresses)
- date (a date string in the format: yyyy-mm-dd)
- datetime (a date string in the format: yyyy-MM-dd, HH:mm)
- bugzilla (a id referencing bug in a bugzilla installation)
- enum (a list of possible values, rendered by a dropdown box in webSWAMP)
- multienum (a list of more than one possible values, rendered by a multi-select box in webSWAMP)

- comment (a comment field which will automatically create the possibility to post replies to it)

2.5. Users, Groups and Roles

The security concept of SWAMP is based on two role-systems. Roles can be stored in the database backend and roles can be defined in the workflow definition files. The workflow definition can also reference roles from the database. The systemwide "admin" role is defined as a group in the database. The admins are superusers of the SWAMP server and are allowed to do everything such as clearing the workflow cache, reloading workflow definitions and administrative tasks. This system is based on the database tables dbUsers, dbGroups, dbPermissions, dbUsers_Groups and dbGroups_Permissions which connect the permissions to certain groups and add users to groups.

The second system takes care of permissions inside workflows. That means who is allowed to do a special task, start or cancel workflows etc. These roles are initially defined in the workflow template, and may be overwritten in each single workflow instance. Standard roles are:

- "user": is allowed to see a workflow and its content in the GUI
- "owner": is the person who started this workflow-instance and has all rights for this instance.
- "admin": is the admin of that workflow-template / -instance and has all rights for this instance. He is able to perform "admin"-action on that workflow, e.g. restarting, de-/activate certain nodes manually
- "starter": is allowed to start workflows of that type

additional roles may be defined in each workflow template to be available in the workflow.

3. Creating a Workflow Definition

3.1. Overview

The definition files are stored in the well-known file format XML. As widely known, XML allows to define hierarchical structures very powerfull. Moreover, XML files are very good to verify and to parse. However it is not so comfortable and straightforward to edit XML files in a text editor. But since that improves with good syntax highlighting capabilities of modern editors and finally if somebody writes a GUI to create the definition files (which is relative easy again) this disadvantage was considered to be ok for SWAMP.

This chapter tries to explain how to write a workflow definiton file for SWAMP. All shown snippets are taken out of the workflow "Example", that is included in the SWAMP release. Thus it should be easy to learn how to define a workflow and to play around with it.

Note: The system expects all textfiles to be UTF-8 encoded to be able to handle special characters.

3.2. The XML Workflow Definition File

The rules for the syntactical rules for a workflow definition are specified in the workflow DTD, located at conf/dtds/workflow.dtd. If a workflow definition is malformed and does not validate against the DTD, or has other semantic errors, SWAMP will refuse to load it. The *workflow definition file* starts with an XML header consisting of an *XML declaration* and a *document type declaration*:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE workflow SYSTEM "../../../dtds/workflow.dtd">
```

This way the *root* resp. *top-level element* is also defined and it is called `workflow`. The `workflow` element requires the attributes `name` (has to be the same as the directory the workflow-version is stored in), `version` (the version of this workflow) and `leastSWAMPVersion` (the minimum version of SWAMP this workflow requires to run on).

If the workflow is a subworkflow, you additionally have to set the attributes parentwf (the parent workflow name) and parentwfversion (the version of the parent workflow). With this additional information the workflow verifier is able to some checks on the workflow.

```
<workflow name="ExampleWorkflow" version="0.1" leastSWAMPVersion="1.2">
```

A raw view on the structure of the workflow definition looks like this:

```
<workflow ... >
 <metadata>
  <!-- Meta information of that workflow -->
  <roles>
   <!-- definition of roles needed in that workflow -->
  </roles>
 </metadata>

 <node ...>
 <!-- definition of nodes including actions and conditional
  edges to other nodes -->
 </node>

 <dataset ...>
 <!-- definition of the workflows dataset -->
 </dataset>
</workflow>
```

Now, we are going to show the details of each workflow element, and describe the example implementation as in the ExampleWorkflow workflow.

3.2.1. Metadata + Role definitions

The `metadata` element contains the mandatory elements `templatedescription`, `description` and `roles`.

```
<metadata>
 <templatedescription>
 Workflow for Testing Issues
 </templatedescription>
 <description>Workflow for Testing Issues</description>

 <roles>
  <role name="owner" restricted="true" type="databit">
   <rolevalue>testdataset.roles.owner</rolevalue>
  </role>
  <!-- @type is set to "value" by default -->
  <role name="admin" restricted="true">
   <description>Admins</description>
   <rolevalue>swamp_user, swamp_admin</rolevalue>
  </role>
  <role name="starter" restricted="true" type="reference">
   <rolevalue>user</rolevalue>
  </role>
  <role name="user" restricted="false" type="databit">
   <rolevalue>testdataset.roles.user</rolevalue>
  </role>
  <!-- reference the group with name "supporter" from the database:  -->
  <role name="supporter" restricted="true" type="dbreference">
   <description>Support-Team</description>
   <rolevalue>supporter</rolevalue>
  </role>
 </roles>
```

```
</metadata>
```

`templatedescription` contains the general description of this workflow type, whereas `description` means the description of a single workflow instance and may be dynamically modified with script content as described later.

The shown role definitions are a minimum set of required roles. The standard roles `owner`, `admin`, `starter` and `user` have to be defined in each workflow. These roles were already explained in the previous chapter. Each role has a flag `restricted` with which you simply can turn off that role, means every logged in user automatically has that role and is allowed to do actions that require that role. If `restricted` is set to `true`, you have two options:

- Specify a `rolevalue` as the example does for the role `admins`. This way all workflow instances share the admin definition of the template, and if you want to add a new admin to all workflows of that type, just add the new admin to the template and reload the template. These roles are called "static" roles in SWAMP.
- Specify a target databit (`roledatabit`) where the usernames of the users the are assigned to that role are stored. This way the members of that role can be changed individually for each workflow instance.

The `owner` role is automatically set to the user that has started the workflow. To change the users that are assigned to a role in a workflow instance simply edit the corresponding Databit.

Merging roles:

```
<role name="mergerole" restricted="true">
 <description>Merged role</description>
 <roleref>owner</roleref>
 <roleref>admin</roleref>
</role>
```

To merge 2 or more roles into a new one use the `roleref` element.

3.2.2. Nodes

A raw overview of a node definition looks like this:

```
<node [type="start"] name="start">
 <description>Mandatory start node</description>

 <!-- duedate  definition -->
 <duedate ... />
 <!-- definition of a milestone -->
 <milestone ... >
 <!-- definitions of included actions -->
 <action... >
 <!-- definitions of edges leaving the node -->
 <edge ...>
</node>
```

Each workflow must exactly have one node with the attribute type="start", because this node gets activated on workflow creation and starts the process. Normal nodes do not have the attribute type set, where nodes with type="end" mark endpoints of a workflow, and signal that the workflow is finished. When an endnode is reached, all remaining active tasks will get canceled, and the workflow will disappear from the "running" list of workflows. The name attribute is the unique identifier of a node.

Nodes can include a milestone, that marks a point of significant progress and makes it easier to track the progress of a complex workflow in the GUI. A node can have any amount jobs to be done, in SWAMP jobs are called actions. If a node is entered, all included actions will get activated. At least a node can define edges and conditions that define which edge should be taken at which situation.

3.2.3. Milestones

Milestones can help to generate a linear list of points in the workflow that show the progress of a process in a simple way and hide the workflow complexity. Milestones can be rendered nicely in the workflow list pages of webSWAMP and give a fast overview of the workflows progress. An example milestone definition looks like this:

```
<milestone name="m1" weight="5">
 <description>Milestone 1 reached</description>
</milestone>
```

The description will be displayed to the user in the GUI, and the weight-factor gives the milestones an order to pretend a linear order that is often not given in complex workflow scenarios.

3.2.4. Duedates

Each node can be marked with a "duedate" that means a date when the node should be left. This is usually done when the contained tasks were done, depending on the attached conditions of the leaving edges. Workflows with duedates that are near their target time, or already late can get displayed with yellow and red color in the

GUIs workflow lists to show that something is stuck. The XML definition:

```
<duedate databit="testdataset.duedate1"/>
```

`databit` contains the path to the databit where the duedate value is set. Usually this value is set there by a preceding dataedit action or by any automatic mechanism.

3.2.5. Actions - The tasks that have to be done

There are two classes of available actions: system-actions and user-actions.

- Systemactions are jobs that are done automatically by the system, such as sending notifications, starting subworkflows or doing scripted actions.
- A useraction needs the interaktion of the assigned user/role in the GUI. This can be entering data, making a decision or just clicking "OK" at some point of the workflow.

The actions have some attributes in common: The attribute `name` always contains a unique identifier for that action, and the `description` element contains a description for that action. The user-actions have the following attributes in common:

- `role` contains the name of the assigned role. The definition of roles was described earlier in this chapter. This attribute is not mandatory. But if you don't specify a role to an action, you are not able to send notifications to the assigned role, or restrict the execution of the action to the role members.
- `restricted` (boolean) defines whether the execution of this action is only allowed to users that are in the configured `role` or if every valid workflow user may act this action. This attribute in not mandatory and set to false by default.
- `notificationtemplate` contains the path to the mail-notification template. When the action gets activated the users of the assigned role will get notified.
- `mandatory` is a hook for the GUI to distinguish between "maintenance" tasks and important tasks. Only mandatory tasks are shown in the workflow lists. Default is "true".

3.2.6. Dataedit (user-action)

A dataedit-action wants the user to enter/edit data of certain fields in the workflows dataset. Example code:

```
<dataedit name="dataedit1" eventtype="DATAEDIT2_OK">
 <description>Please fill in the fields.</description>
 <longdesc>Workflows-Threads are unitet now</longdesc>
 <field path="testdataset.product.product_name" mandatory="yes" />
 <field path="testdataset.roles.manualtask_owner" mandatory="yes" />
</dataedit>
```

Here, the user gets a view where he has to edit 2 fields of the dataset. The `mandatory` tag says if the user may leave the field blank. If the entered values do not fit the datatype of the field, they will not get saved and the user will get an errormessage. When no errors were reported, the event "DATAEDIT2_OK" will get sent to the workflow and the task is done.

3.2.7. Manualtask (user-action)

Manualtask is a simple acknowledgement of the user. The Description as in the elements `description` and `longdesc` will be shown, and when the user clicks on "OK" the defined event will get sent out. This task can be used to acknowledge that some work has been done outside the system, or a manager can give his ok for the workflow to continue...

```
<manualtask name="manualtask3" eventtype="UNITE">
 <description>Go on in the Workflow</description>
</manualtask>
```

3.2.8. Decision (user-action)

A Decision presents the user with a description and some possible options. Each answer is connected with an event that will get sent when the user has made a choice.

```
<decision name="decision" >
 <description>Testen der CDs</description>
 <question>Please test the CD</question>
 <answer eventtype="PATH1">Take Path 1 in this Workflow.</answer>
 <answer eventtype="PATH2">Take Path 2 in this Workflow.</answer>
 <answer eventtype="PATH3">Take Path 3 in this Workflow.</answer>
</decision>
```

3.2.9. Notification (system-action)

The notification-action sends out notification at any point of the workflow progress. Based on a notification-template, there can be any amount of recipients be added, like the following snippet shows:

```
<notification name="notify_owner"
  msgtemplate="notifications/notification1">
 <recipient recipientemail="please_change@swamp.swamp" eventtype="none"/>
 <recipient dbit="testdataset.roles.user" eventtype="none"/>
 <recipient recipientrole="user" eventtype="none"/>
 <recipient recipientname="swamp_user" eventtype="none"/>
</notification>
```

The recipients can be configured directly by their mail-adress, their SWAMP username, all users that have a special role in that workflow or all usernames, mail-adresses that are included in a certain databit.

3.2.10. Customtask (system-action)

A `customtask` allows the instantiation of a custom java class at runtime. Thus it is possible to invoke a piece of code at any point of a workflow. The usage of this action is not recommended, as the java class has to be compiled and must be available to the tomcat classloader. It cannot be contained in a workflow resource bundle. Please try to use a `scriptaction` when possible.

```
<customtask name="custom_test" eventtype="none"
 class="de.suse.swamp.custom.CustomActionExample"
 function="customTest" >
 <description>Calling CustomActionExample.customTest()</description>
</customtask>
```

The invoked method must have the following signature:

```
public Boolean customTest(Integer wfid, Integer userId) throws Exception
```

A customaction can be used for calling external programs on the server. An example on how to do this is included in the de.suse.swamp.custom.CustomActionExample class.

3.2.11. Startsubworkflow (system-action)

This action starts a new workflow and attaches it as a subworkflow to the workflow from where it was called. A subworkflow may be a complex sub-process in a workflow that happens one or moe times. A subworkflow is defined the same way as a normal workflow, as it just a normal workflow but has a reference to its parent workflow and sends an event to its parent workflow on finish. Definition looks like this:

```
<startsubworkflow name="startsub" subname="Example" subversion="0.1">
 <description>Starting Subworkflow</description>
</startsubworkflow>
```

When a subworkflow is started, a reference to the datasets of its parent workflow is given to him. This way the subworkflow has access to its parent workflows data. In addition it is checked if there are databits in the subworkflows default-dataset that have the same name as databits in the default-dataset of the root workflow. If there are any, their content is copied from the root workflow to the subworkflow.

3.2.12. Sendevent (system-action)

The sendevent action can send out events dependant on time contraints. With this action we are able to implement a reminder after a certain time, or take any other action in the workflow if a node hasn't been left in time.

```
<sendevent name="reminder" eventtype="DELAY_1D">
 <triggerdate databit="System.path2.enterDate"
  offset="+1d"
  onlyweekdays="true"/>
</sendevent>
```

In this example the event "DELAY_1D" will get sent if the node isn't left 1 day after it was activated. The "databit" attribute can reference a normal databit that contains a date value, or like in this case a special "pseudo-" databit that contains workflow values. More details about these system-databits can be found in one of the next sections. The "offset" parameter has the format "+[0-9][m|h|d]" which stands for the amount of minutes, hours and days.

Note: if you use time triggers for intervals shorter than 30 minutes, you have to reconfigure the scheduler thread to run more often. This can be done in the database table TURBINE_SCHEDULED_JOB.

The sendevent action does not need to set a triggerdate, if none is set, the event is send immediately. It is also possible to send events to other workflows with this action, for example:

```
<sendevent name="reminder" eventtype="DELAY_1D">
 <targetwfs>
 #foreach ($subwf in $wf.getSubWorkflows(true))
  $subwf.getId(),
 #end
 </targetwfs>
</sendevent>
```

This action will send the Event DELAY_1D to all attached subworkflows. The element targetwfs expects to have a comma seperated list of workflow ids, that can be generated by velocity scripting.

3.2.13. Scriptaction (system-action)

A scriptaction allows you to invoke Velocity (http://jakarta.apache.org/velocity/) and Groovy (http://groovy.codehaus.org) scripts at a certain point of the workflow. These scripts are executed in a special environment where they have some limited access to workflow objects. This is an example:

```
<scriptaction name="script_example" language="velocity">
 <description>Setting comment</description>
 <script>
      $wf.getDatabit("testdataset.comment").setValue("TADAA")
      </script>
</scriptaction>

<scriptaction name="script_example" language="groovy">
 <description>Setting comment</description>
 <script>
      wf.getDatabit("testdataset.comment").setValue("TADAA")
      </script>
</scriptaction>
```

The workflow object is available as ($)wf in the script context. It can be altered in any way, like changing a databit as in the example, activating- deactivating nodes. This assumes you know what you are doing, as it is easy to break a workflow this way. Other objects that are available in the script context are

- `uname`: the actual username
- `wf`: reference to the workflow object
- `bTools`: for triggering Bugzilla actions
- `hist`: an arraylist where messages can be appended that will be shown in the GUI afterwards. Usage: `hist.addResult(boolean isError, String result)`
- `scriptapi`: Object that allows to invoke the following methods:
  - `createSubWorkflow(String name, String version)`: create a subworkflow

- `sendEvent(String eventString, int wfid)`: send an event
- `getWfConfigItem(String name)`: retrieve property from workflow.conf
- `getSWAMPProperty(String name)`: retrieve property from defaults

- `executor`: Allows the execution of external programs and handling of their output and return code.

Example for calling external scripts from a scriptaction:

```
<scriptaction name="call_external" language="velocity">
 <description>Calling external a program</description>
 <script>
 $executor.setExecutable("/bin/ps")
 $executor.addArgument("-a")
 $executor.addArgument("-u")
 $executor.addArgument("-x")
 $executor.setExceptionOnError(true)
 $executor.execute()
 $wf.getDatabit("set.ps").setValue($executor.getStdout())
 </script>
</scriptaction>
```

3.3. The flow of the work - Edges and Conditions

When a node is entered (at workflow start this is the startnode) its edges get activated. That means, they check their associated conditions of being true. If it is true, the source node is left and the new node becomes active. That means, only edges that leave an active node are active and listen to events, changes of data etc. If more than one edges leave a node, and the condition of one of them resolves to true the source node and all leaving edges get deactivated.

```
<edge to="node1">

<!-- conditions for that edge -->

</edge>
```

Each edge needs to have a condition assigned to it which may be a compound condition consisting of AND/OR constructions. Examples for all possible condition types follow:

3.3.1. Event - condition

An eventcondition waits for an event to arrive and resolves to true when the event was received. An event is identified by an event string. For example:

```
<edge to="path2_is_late" >
 <event type="DELAY_1D"/>
</edge>
```

waits for the event "DELAY_1D" to arrive. If an edge has only one event-condition attached, like in the example, a short definition is available:

```
<edge to="path2_is_late" event="DELAY_1D" />
```

The event string "NONE" is a reserved event that defines that no event is required, and the condition will always immediately resolve to true on activation. This event is needed for modelling purposes, for example to split the workflow thread.

3.3.2. Data - condition

A datacondition waits for data to be changed, or to match against a given regular expression.

```
<edge to="node_product1">
 <data check="regexp"
  field="testdataset.product.product_name"
  value=".*SLES.*"/>
</edge>
```

This data-condition waits until the databit "testdataset.product.product_name" matches the regular expression ".*SLES.*". The change of the data has not neccessarily to happen within a dataedit-task, it can also happen by a user editing the workflows data directly or via another subsystem eg. the SOAP interface.

To make a data-condition watch for any change of a databit, use this code:

```
<edge to="node_product1">
 <data check="changed"
  field="testdataset.product.product_name" value=""/>
</edge>
```

This condition will always turn to true when the content of the databit gets changed. The value parameter has no effect in this case.

3.3.3. Subworkflowsfinished - condition

To wait until all appended subworkflows are finished the "subsfinished" condition can be used. Per definition a finished subworkflow sends the event "SUBWORKFLOW_FINISHED" to its parent workflow. The "subsfinished" condition is waiting for that event, checks if it was sent by the configured workflow type and version, and resolves to true if the finished subworkflow was the last running subworkflow. Example definition:

```
<subsfinished subname="Example" subversion="0.1"/>
```

It can be used to stop the main workflow at a certain point and wait until all started subworkflows finished.

3.3.4. Connecting conditions with AND, OR and NOT

Conditions can be nested in complex contructions by combining the already shown conditions with AND, OR and NOT operands. For example:

```
<edge to="node_product1">
 <and>
 <event type="DATAEDIT2_OK"/>
 <not>
 <data check="regexp" field="testdataset.product.product_name"
  value=".*LINUX.*"/>
 </not>
 </and>
</edge>
```

The AND and OR elements do both take 2 subelements, and the NOT element takes just one. That means, to connect for example 3 event conditions by AND you have to write your definition this way:

```
<edge to="node2">
 <and>
 <event type="DATAEDIT1_OK"/>
  <and>
  <event type="DATAEDIT2_OK"/>
  <event type="DATAEDIT3_OK"/>
  </and>
 </and>
</edge>
```

3.3.5. Datasets

Datasets contain are the workflows storage for data. They can store texts, dates and role assignments. Atm each workflow defines one root-dataset in its xml definition which has also to be referenced as "defaultdataset" in the workflow element. Datasets can be nested, and the elements that contain the actual data are called databits.

```
<dataset description="Roles" name="roles">

 <dataset ...>
 <databit ...>

</dataset>
```

Each databit has a datatype against which its content is verified on changes (type attribute). The visibility of databits and datasets can be set by the state attribute, for example to hide workflow-internal data from the user in the GUI. A databit is defined like this:

```
<databit name="admin" description="Workflow-Admins"
  type="person" state="read-write">
 <defaultvalue>swamp_admin</defaultvalue>
</databit>
```

The element defaultvalue sets the initial value of the databit. Databits can be referenced from various places within a workflow, eg. data-conditions, dataedit-actions, scriptactions, notificationtemplates and more. The notation for referencing a databit is:

```
datasetname.[datasetname.]databitname
```

3.3.5.1. System databits

To extend the power and flexibility of the previously shown workflow elements that use databits, some workflow information is also mapped into the databit "namespace".

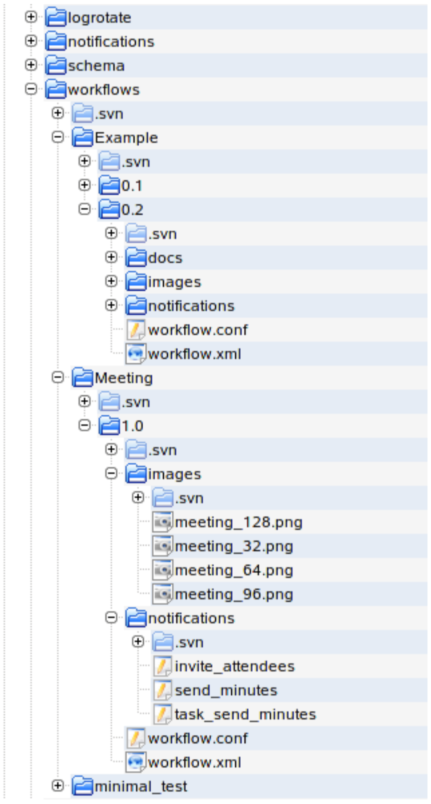
These special dynamic databits are referenced with a leading "System." as pseudo datasetname. Available system databits so far are:

```
System.<nodename>.dueDate
and
System.<nodename>.enterDate
```

to query the attached due date of a node, and to get the date when a node was entered. This can be used for example in the sendevent-action (see section "actions") to determine how long a node has been active. Also mechanisms to notify responsible persons if duedates are not met can be implemented that way.

4. Creating a workflow resource bundle

SWAMP learns about it’s workflows from workflow definition files that are read from the file system from a dedicated directory. Each workflow-type consists of a resource bundle that contains the workflow definition, configuration, mail-templates and icons.



```
⊕ logrotate
⊕ notifications
⊕ schema
⊖ workflows
    ⊕ .svn
    ⊖ Example
        ⊕ .svn
        ⊕ 0.1
        ⊖ 0.2
            ⊕ .svn
            ⊕ docs
            ⊕ images
            ⊕ notifications
              workflow.conf
              workflow.xml
    ⊖ Meeting
        ⊕ .svn
        ⊖ 1.0
            ⊕ .svn
            ⊖ images
                ⊕ .svn
                  meeting_128.png
                  meeting_32.png
                  meeting_64.png
                  meeting_96.png
            ⊖ notifications
                ⊕ .svn
                  invite_attendees
                  send_minutes
                  task_send_minutes
              workflow.conf
              workflow.xml
    ⊕ minimal_test
```

Additionally the workflows are versioned so that the process can be changed at any time without breaking already running workflows.

All existing workflow subdirectories are scanned and parsed at the startup of SWAMP. Even while SWAMP is running, new workflow definitions or new revisions of existing workflow-types can be copied into the directory and SWAMP can be told to re-scan the directory for new workflow definitions.

4.1. Notification Templates

Notification templates are used to create the emails for users that have a new task assigned or that were created by notification-actions. On loading the workflow template, the system already checks if all needed template files are available. The path to a notification-template is relative to its workflow definition. For example when an action has configured the "notificationtemplate" attribute with the value "notifications/template.txt" the template is expected at conf/workflows/<workflowname>/<version>/notifications/template.txt. An example template looks like this:

```
subject=$wf.getName(): new Test-Notification
xheader=$wf.getName()
This is an example-notification of workflow: $wf.getName()

Owner of this workflow: $wf.getDatabitValue("testdataset.roles.owner")
Link to this workflow: $webswamp_link/wf/$wf.getId()


--
This Message was automatically generated by the the WebSWAMP-Server at
$app_link
```

The first line must start with "subject=" and contains the subject of the email. The second line defines an additional x-Header line in the email. The remaining part of the template contains the text that will be sent as message body. See the example for possible velocity replacements, all velocity constructs are useable here, including if-then-else conditions and loops.

4.2. Workflow configuration files

Each workflow searches for a config file named "workflow.conf" at its resource directory. If none is found, default values will be used as a fallback.

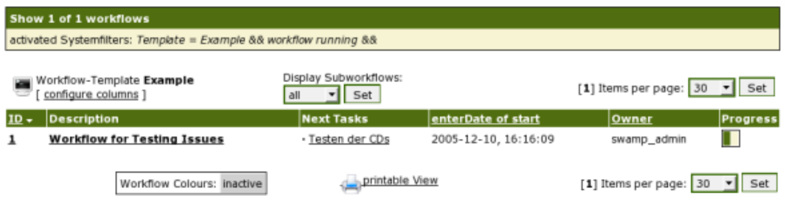
```
## config file for Workflow testworkflow.
## in this file some GUI options for webSWAMP are provided

logo=images/logo.jpg
icon=images/yast_system_22.png
icon_big=images/yast_system_64.png

# displayed columns, should be one line.
displayedcolumns_workflowview=column_workflowid,column_workflowdescriptio
 column_nexttasks,System.start.enterDate,testdataset.roles.owner, column_
```

```
sortby_workflowview=column_workflowid
direction_workflowview=ascending
```

In this file you can configure how the workflow will be presented in the GUI. The `logo` and `icon` values refer to images that get displayed in the GUI. The `displayedcolumns_workflowview=` line tells webswamp wich columns should be displayed when showin workflow lists. Possible values are all databit names including system databits, and some special columns:

- `column_nexttasks`: Shows the next tasks.
- `column_workflowdescription`: Shows the workflows description.
- `column_progress`: Shows a progress bar that indicates what milestones have already been reached.
- `column_workflowid`: Shows the workflow id with a link to the workflows detail page.
- `column_wficon`: Shows the workflows icon.
- `column_state`: Shows the workflows state (running/finished).

Please note that this just are the default values for displaying lists. Each user can change the configuration for himself to best fit his needs. The shown example workflow.conf file would result in a layout like this:



Show 1 of 1 workflows

activated Systemfilters: *Template = Example && workflow running &&*

Workflow-Template **Example**
[configure columns]

Display Subworkflows: all [Set]

[1] Items per page: 30 [Set]

ID	Description	Next Tasks	enterDate of start	Owner	Progress
1	**Workflow for Testing Issues**	· Testen der CDs	2005-12-10, 16:16:09	swamp_admin	▮

Workflow Colours: inactive

printable View

[1] Items per page: 30 [Set]

4.3. Workflow documentation and template files

Each workflow can bring its own documentation with it. The element `helpcontext` inside the node tags of a workflow definition contains the path to the corresponding help file. Available help contexts are displayed on the corresponding pages in webSWAMP and a complete listing of all available help pages is also available from the menu.

The documentation files are stored in the docs/ subdirectory of the workflow resource directory. Please look into the "Example" workflow for an example documentation file. The "head" file is used to display a headline on the documentation overview page.

The subdirectory templates/ of a workflow resource can contain custom Velocity template files that can be used to generate pages that are unique for that workflow. By default each workflow can provide the following templates that will replace the default ones when this workflow is displayed:

- `page_top.vm`: Will replace the default page header, for example to display a custom image for each workflow.
- `menutop.vm`: The navigation on the left is build from this file. This way, you can extend the navigation of a workflow with extra links to filtered views for example.
- `wflist_colours.vm`: If you configured special colours for workflows in the workflow list by using the `workflowlist_colour=` variable in the workflow.conf file you can add an explanation to the wflist_colours.vm file which will get included at the bottom of the workflow list.

4.4. Extending elements with velocity scripting

There are several elements of a workflow definition that can be extended by embedding velocity (http://jakarta.apache.org/velocity/) scripts. The velocity script inside the elements is evaluated at runtime, and thus provides dynamic content at the used place. Here is a list of elements that are aware of content with included velocity scripts, and what objects are provided to be used within the scripts.

- Every description and longdescription element from the workflow definition. (Provided references: The current workflow object as $wf and the corresponding workflow-template object as $wftemplate)
- Notificationtemplates (Provided references: The current workflow object as $wf, the corresponding workflow-template object as $wftemplate, if called from a task, the task as $task. Links to webswamp can be generated from the varibles $app_link and $secure_app_link, see example workflow)
- The content of a script action (Provided references: The current workflow object as $wf and a reference to the BugzillaTools object as $btools)
- Sendevent-action (Provided references: The current workflow object as $wf for evaluating the workflow ids the event should be sent to)

Velocity scripting also provides the possibility of if-then-else conditions and loops. A velocity userguide is available at
http://jakarta.apache.org/velocity/docs/user-guide.html
(http://jakarta.apache.org/velocity/docs/user-guide.html)

5. Setting up authentication with LDAP / other datasource

The SWAMP user backend can be connected to a central LDAP server for authentication and getting user data which is a common scenario in large companies. To switch to the LDAP authentication the setting of `AUTH_CLASS` in the conf/defaults file has to be changed to `de.suse.swamp.core.security.SWAMPLDAPUserManager`. The LDAP connection is configured by the additional `LDAP_` config values. When a user is requested the first time, he gets fetched from the LDAP server and gets stored in the SWAMP database. So we don't have to query the LDAP backend everytime. Authentication always happens directly against LDAP, so we don't store the users passwords in SWAMP.

Users that are available from the database and have a value in the passwordHash field will get authenticated from there. This is useful for adding additional users to SWAMP when you don't have admin access to the LDAP server.

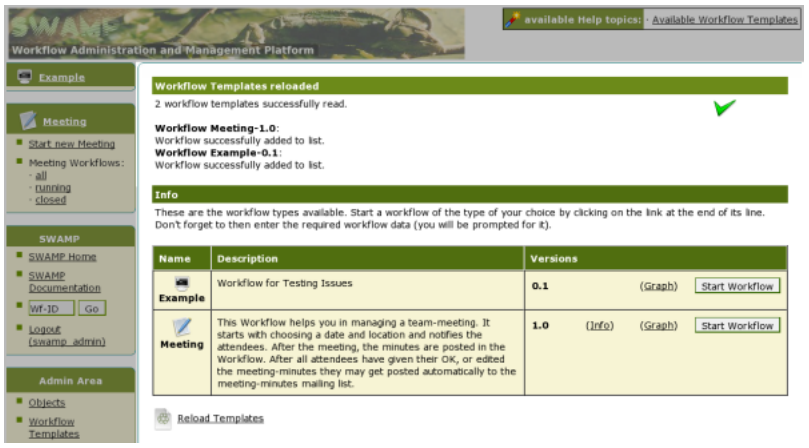
To implement another authentication method, you need to write a class that implements the interface `de.suse.swamp.core.security.UserManagerIface` and set it as `AUTH_CLASS` in the conf/defaults file.

6. WebSWAMP

WebSWAMP is the standard GUI for normal users to interact with SWAMP. All features are available here. The user can browse lists of workflows, edit data and work on tasks that are assigned to him. Administrative tasks that can be done from the web-frontend are reloading/installing workflow definitions, manually change a workflows state and clearing the workflow cache.

6.1. Reload / install workflow definitions

At the shown page (Admin Area -> Workflow Templates) you have the possibility to administrate the workflow definitions.



Workflow Templates reloaded

2 workflow templates successfully read.

Workflow Meeting-1.0:
Workflow successfully added to list.
Workflow Example-0.1:
Workflow successfully added to list.

Info

These are the workflow types available. Start a workflow of the type of your choice by clicking on the link at the end of its line.
Don't forget to then enter the required workflow data (you will be prompted for it).

Name	Description	Versions			
Example	Workflow for Testing Issues	0.1		(Graph)	Start Workflow
Meeting	This Workflow helps you in managing a team-meeting. It starts with choosing a date and location and notifies the attendees. After the meeting, the minutes are posted in the Workflow. After all attendees have given their OK, or edited the meeting-minutes they may get posted automatically to the meeting-minutes mailing list.	1.0	(Info)	(Graph)	Start Workflow

How to install / upgrade a workflow definition:

- Manually with access to the filesystem:

  Copy your new version to the workflow storage at workflows/${workflowname}/${workflowrevision} and tell SWAMP to reload the definitions either from WebSWAMP ("Workflow Templates"->"Reload Templates") or by reloading the complete SWAMP application.

- By uploading through the web interface:

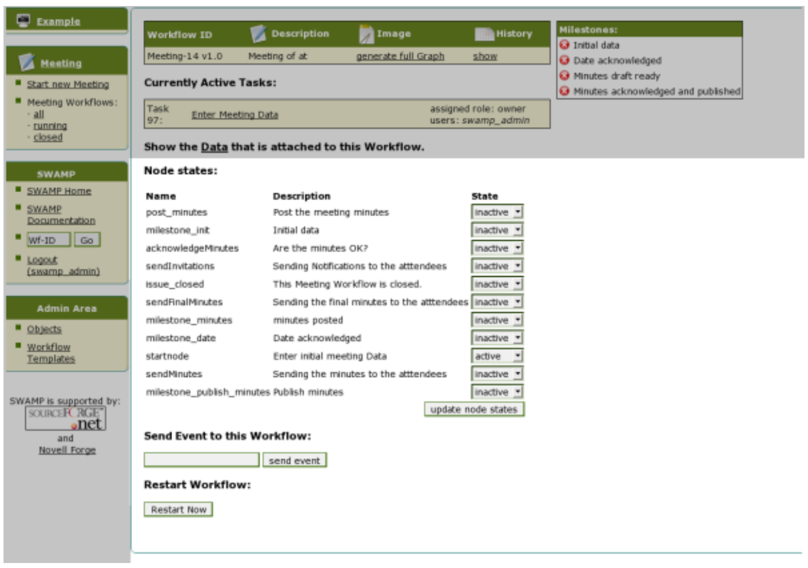
  To upload a new workflow / revision please go to the shown page and use the upload form at the bottom. You can upload a single workflow.xml definition file, or a zipped workflow resource bundle which can contain all elements, for example the workflow.conf file, icons, notification templates...

  After the upload the workflow will automatically get verified and can get installed if no fatal errors were detected.

Each workflow is read in from the resource directory and verified. If errors were detected when SWAMP reads the workflow and internally constructs the workflow template the system will refuse to add the workflow to the list. If non-fatal warnings were detected, they are displayed, but the workflow is added to the list.

6.2. Manually change a workflows state

Admins can manually change the internal state of a workflow. To do so, please open the workflow page of the workflow you want to change. Admins can now open the "admin console" and will get a menu similar to that on the screenshot.



Workflow ID	Description	Image	History
Meeting-14 v1.0	Meeting of at	generate full Graph	show

Milestones:
- Initial data
- Date acknowledged
- Minutes draft ready
- Minutes acknowledged and published

Currently Active Tasks:

Task 97:	Enter Meeting Data	assigned role: owner users: *swamp_admin*

Show the Data that is attached to this Workflow.

Node states:

Name	Description	State
post_minutes	Post the meeting minutes	inactive
milestone_init	Initial data	inactive
acknowledgeMinutes	Are the minutes OK?	inactive
sendInvitations	Sending Notifications to the atttendees	inactive
issue_closed	This Meeting Workflow is closed.	inactive
sendFinalMinutes	Sending the final minutes to the atttendees	inactive
milestone_minutes	minutes posted	inactive
milestone_date	Date acknowledged	inactive
startnode	Enter initial meeting Data	active
sendMinutes	Sending the minutes to the atttendees	inactive
milestone_publish_minutes	Publish minutes	inactive

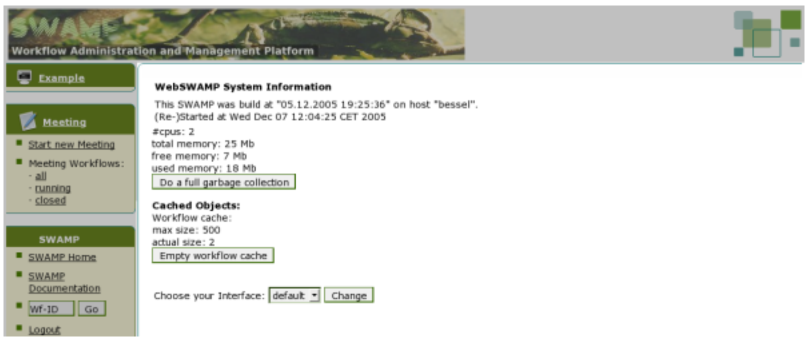
Send Event to this Workflow:

Restart Workflow:

Each node of the workflow is listed here, and can be set to active/inactive. You should have explicit knowledge of the workflow structure before changing the nodes states manually here, because a workflow can easily get out of its track when internal node states are changed in the wrong way. On this page admins have the additional possibility to send an event to the workflow manually.

6.3. Reset the workflow cache

Workflows are cached internally to save time when they are requested. The amount of cached workflows is configured in the "conf/defaults" config file. To empty the workflow cache (can be useful after direct editing of the database and other dirty tricks ;-)) please go to: Admin Area -> Objects.



WebSWAMP System Information

```
This SWAMP was build at "05.12.2005 19:25:36" on host "bessel".
(Re-)Started at Wed Dec 07 12:04:25 CET 2005
#cpus: 2
total memory: 25 Mb
free memory: 7 Mb
used memory: 18 Mb
```

Cached Objects:

```
Workflow cache:
max size: 500
actual size: 2
```

Choose your Interface: default

This page also offers the option to manually force a full garbage collection of the java virtual machine.

7. SOAPSWAMP

SOAPSWAMP is an extension to SWAMP that provides access to the SWAMP workflow engine by the SOAP interface. It is installed as an extra webapp in tomcat. It uses the axis framework (http://ws.apache.org/axis/) to provide the SOAP service. The WSDL webservice description can be fetched from:

```
http://<hostname>:8080/axis/services/swamp?wsdl
```

The WSDL contains all needed information about available methods, and can be used by some tools to automatically create client libs. SWAMP already ships a perl client module for connecting external systems to SWAMP (see next section). Methods available via the SOAP interface atm include:

- Reading / Writing data to a workflow
- Sending events to workflows
- Getting a list of workflow ids that match a certain criteria
- Read workflow information of a workflow instance
- Start workflow instances

All methods include parameters for username and password, as the SOAP interface is based on the same SWAMP interface as webSWAMP. Unauthorized access will be denied with an Exception.

8. Perl SOAP client

The perl SOAP client enables an easy integration of the SWAMP server into remote applications. It can be downloaded here (http://sourceforge.net/project/showfiles.php?group_id=68771) either as SUSE RPM or source tarball. After installing the perl module "SUSE::Swamp" should be useable from within your perl scripts. To view the module documentation please use:

```
perldoc SUSE::Swamp
```

All methods that are available via the SOAP interface are automatically mapped to perl functions. To display all available methods you can use this script:

```
#!/bin/bash -i
$WSDL='http://<swamp.server>:8080/axis/services/swamp?wsdl'
perl -I. -MSUSE::Swamp -e"print SUSE::Swamp->new('$WSDL')->generateDoc()"
```

```
 pod2man --name="SWAMP SOAP API" | nroff -man | less;
```

which will display a manpage with descriptions and signatures of the available methods.

A complete example script on how to use the module is included in its distribution. Usage looks like this:

```
#!/usr/bin/perl
use strict;
use SUSE::Swamp;
my $url = 'http://<swamp.url>:8080/axis/services/swamp?wsdl';
my $username = "swamp_user";
my $pwd = "swamp";

my $swamp = SUSE::Swamp->new($url);
my $version = $swamp->doGetProperty( "SWAMP_VERSION", $username, $pwd );
print "SWAMP server version: " . $version . "\n";

# create a workflow:
my $wfid = $swamp->createWorkflow( 0, "Example", $username, $pwd );
print "Created workflow with id: $wfid" . "\n";

# read data from the workflow
my $result = $swamp->doGetData( $wfid, "testdataset.reason", $username, $p
print "Value of testdataset.reason: $result" . "\n";
```

9. Updating / maintenance of the server

To update an existing SWAMP instance to a newer version you have to pay attention to some things. The complete state of the system is stored in the database backend. So, to be able to restore your system if something goes wrong you need a copy of your databese, and your workflow definitions. It is recommended to backup your database regulary, SWAMP already comes with a script that can do that for you. Call:

```
bin/db-dump.sh
```

to make a backup using `mysqldump` and `gzip`.

Every new SWAMP release comes with a file called `UPGRADING`. This file contains notes on changes to the database layout and workflow definition format in the new version. Thus you know how to convert your backed up state to the new version if there have been incompatible changes.

If you are running into space problems, or want to keep your backups small, you can safely emtpy the tables: `dbEventHistory`, `dbHistory` and `dbNotifications`. Your SWAMP will also run without he content of these tables, but will not be able to provide history information of old workflows.