SWAMP - developers guide

How to develop code for the SWAMP project

Note: Note: This document is still work in progress.

1. Setting up a development environment with eclipse

2. Debug a running SWAMP with eclipse

For debugging a Java Web Application (e.g. SWAMP) the Servlet Container can be run directly in Eclipse. You need the "Sysdeo Tomcat Plugin" for Eclipse from here: http://www.sysdeo.com/eclipse/tomcatPlugin.html (MIT License) After copying the Sysdeo Files to your Eclipse/plugins folder and restarting Eclipse there are some new Options available. At first, add the Tomcat Controls to your Menubar with Window->Customize Perspective->Commands->Tomcat. Then you have to configure the Tomcat startup options in Window->Preferences->Tomcat.

Now you can startup tomcat with the toolbar button. The output is captured in eclipse's console window. To start debugging simply set a beakpoint in the source file that is going to be debugged. For a short introduction into the Eclipse Debugger yuo may look here: http://www-106.ibm.com/developerworks/opensource/library/os-ecbug/?Open&ca=daw-ec-dr (http://www-106.ibm.com/developerworks/opensource/library/os-ecbug/?Open&ca=daw-ec-dr)

3. The Container

The central functionality of SWAMP is provided by the so-called managers. A manager is a Singleton that takes care of a specific subsystem. The managers are:

The *StorageManager* provides the abstraction level between data storage (currently database only) and the rest of the system. It provides the following methods for each object type Blubber we have:

- ArrayList getBlubbers()
- int storeBlubber(Blubber b)
- int loadBlubber(int blubberId)
- void removeBlubber(int blubberId)

Object types are: tasks, workflows (with all components), and data. Everything, actually.

The *WorkflowManager* reads in the workflow definitions and keeps track of the workflow template objects that are created from them. It provides a list of possible workflows, creates workflows on request and interacts with the StorageManager to store and reload workflows to and from storage. If you need access to one ore several workflow objects, get them from the WorkflowManager, not the StorageManager.

The *TaskManager* accepts tasks from the other subsystems or frontends, stores them and gives them back on request. When a task is done, the system that acted on it gives it back to the TaskManager with finishTask(Task), which then checks if everything is ok and performs the associated actions and clean-up work. Tasks containing a user action are stored until a frontend requests them, tasks for which other subsystems are responsible are handed over to them directly.

The *EventManager* accepts events from anywhere in the system and distributes them to any object that has registered itself as a listener to a specific event type. (Yet to be written.)

The *WorkflowReader* is used for reading in workflow definition files in XML.

> **Note:** This might be split into a WorkflowXMLReader and a WorkflowSwwReader class, where "sww" stands for "SWAMP workflow". swx would be SWAMP xml file etc.

4. How workflow tasks Work

4.1. What is a workflow task?

A workflow task consists of an action that has to be done within a workflow, and a result of this action. If it is an interactive or user task, it also has a user who is more or less responsible. It has a start and an end date, and a status.

4.2. What classes are involved in a task?

- the action, a subclass of de.suse.swamp.core.workflow.Action
- the result, a subclass of de.suse.swamp.core.workflow.Result (Actions and Results are paired)
- de.suse.swamp.core.tasks.WorkflowTask itself, which holds references to the action, the task, the user, the workflow...

4.3. What is the life cycle of a workflow task?

- When a node is entered, it loops over its actions and asks each one for a task (Action.getTask()). The Action class knows how to put together a workflow task for itself. The node then hands over the tasks to the TaskManager (TaskManager.addTask()). At this point, the task already has an empty result object.
- The TaskManager stores the task into whatever storage is there, updates its own internal data structures and logs that a task has arrived. If it is a non-interactive or system task, it calls the subsystem that is responsible for handling these. If it is an interactive or user task, it waits for a frontend to come and ask for it.
- A frontend (webapp, shell) calls one of the getTasks() methods of the taskmanager and gets back a list of tasks to display.
- The user selects one of these tasks to work on. The frontend gets the information it has to present to the user from the Action object, and puts the data the user creates by completing the task into the Result object.

- The frontend needs to know how to present and receive the data for each kind of Action/Result pair that exists in the system (i.e. Manualtask, Decision, DataEdit...).
- The frontend can call the validate(result) method from the Action class to check if the task is indeed completed. If this test is successful, it calls TaskManager.finishTask(taskid) for the task.
- The TaskManager asks the Task object (that has now a non-empty result object with validated data) for the events that should be sent, and hands these over to the EventManager (note: the frontend does not send any events any more!). It then logs whatever needs to be logged, marks the task as finished in storage and removes it from the list of active tasks.

5. SWAMP User Interface Style Guide

5.1. Buttons

- The default submit button is on the right below the form it is accociated with.
- Additional action buttons should be on the left side below the object they are associated with ordered by significance from left to right, the most significant button being the left-most.
- The reset button is on the left side below the form it clears. It is labeled "Clear Form". If there are additional action buttons for the same form the reset button is the right-most button of these.
- Texts on buttons use headline-style capitalization, where the first and last words of the text and all other major words are capitalized.

6. Implementation of Permissions and Roles

6.1. Workflow-type specific roles

Every call for a SWAMPAPI method is checked for sufficient permissions of the provided user. So we are securing access to Task- Workflow- and WorkflowTemplate Objects at this layer. If the user does not have the permission to read / write a requested Object an Exception will be thrown by the API that can be displayed in the GUI / SOAP etc. interface. To make a smooth GUI that does no give the user the option to request Objects he is not allowed to see the permissions can be checked by:

- `boolean Workflow.hasRole(String username, String role)`
- `boolean WorkflowTemplate.hasRole(String username, String role)`

for the Workflow roles, and for the general permissions with:

```
SecurityManager.isGroupMember(SWAMPUser user, String
groupName)
```

6.2. General SWAMP roles

We use a simple user / group / role system here, that uses the database tables dbUsers, dbGroups, dbPermissions, dbGroup_Permissions, dbUsers_Groups. For now we only have one group "swampadmins" with the permission "admin_permission" that is assigned to the admin of the swamp installation. This group is allowed to do maintenance actions, as reloading workflow definition files and emptying the workflow cache. Additionally members of the "swampadmins" group are automatically admins of all workflows.

7. The SWAMP API classes

8. Localization

SWAMP is localized using the commons-gettext library, which uses standard .po files. By calling `bin/generate-po.sh` .pot files will get created for webswamp and swamp-core at `conf/i18n/keys.pot` and `webapps/webswamp/conf/i18n/keys.pot`. Based on this .pot file you can create a translation file using a GUI editor, for example kbabel. When building SWAMP, the available .po files will automatically get installed. (You have to add the available languages to build.xml and webapps/webswamp/java/de/suse/swamp/modules/screens/Preferences.java at the moment)

To use localized texts in the code, you have to create an instance of: `de.suse.swamp.util.I18n` like this:

```
I18n i18n = new I18n(getClass(), "de.suse.swamp.webswamp.i18n.Webswamp");
```

(replace the second parameter with: `de.suse.swamp.webswamp.i18n.Swamp` when calling from swamp-core code)

Then, each usage of:

```
i18n.tr("text to translate", user)
```

will get trasnlated into the users preferred language and:

```
i18n.tr("text to translate")
```

will get translated to the configured default language of the system. If a text is not available in the .po file, the text from the method call will be returned.