Developers Guide

Activiti Developers Guide

Building a distribution from the source

Dependencies

JDK 5+ : Make sure that you do not use code constructs that require JDK 6 or higher.
Maven 2.0.9
Ant 1.7.1

Source Code

https://svn.codehaus.org/activiti (see http://xircles.codehaus.org/projects/activiti/repo for more links including anonynmous access for non-committers).

Building the distribution

1. Check out the latest version from trunk: '*svn co* http://svn.codehaus.org/activiti/activiti/trunk/'
2. Go into the *distro* subfolder and run the following ant command: *'ant clean distro'*
3. The distribution can now be found in the *target* folder.

Setup scripts in the distro

After an distribution is unzipped (or directly in the *distro/target/activiti-version* directory), a number of scripts are offered to our users as well. Those scripts can be found in *[activiti.home]/setup/build.xml* and can optionally use settings from *[user.home]/.activiti/build.properties*

As an example, here's the content of my *[user.home]/.activiti/build.properties*:

```
activiti.home=/Users/tombaeyens/Documents/wo
rkspace/activiti/distro/target/activiti-5.0.
alpha1-SNAPSHOT
downloads.dir=/Users/tombaeyens/Downloads
tomcat.enable.debug=true
```

activiti.home, is used in the setup/build.xml. It will work straight from the sources. So you can add the setup from the sources to your ant view in your IDE.

downloads.dir, is used in the setup/build.xml. It allows you to configure a custom location for your directory to download/find tomcat. If you don't specify this, the default is relative to the setup directory (../downloads). So if you use the setup/build from the sources, you might end up downloading tomcat into your sources. If that still would happen, don't check it in! 😉

tomcat.enable.debug is used in setup/build.xml when Tomcat is installed. If the property is specified (whatever value), the parameter 'jpda' will be added to the start commands in Tomcat's 'startup.sh' and 'startup.bat' scripts. That will cause the remote debugging service to listen on port 8000.

skip.deploy.activiti.modeler=true set this property if the Activiti Modeler fetching from net and deployment must be skipped.
Unless before a release or explicitly working on the Activiti Modeler, it's advised to set this property since sometimes a version matching the current trunk version isn't uploaded yet, causing the demo setup to fail.
mvn.executable=mvn.bat on windows you have to set this property to make the calls from the ant build scripts (distro/build.xml and qa/build.xml) to maven work.

activiti.modeler.download.url: To work with Activiti Modeler or Activiti Cycle on trunk, set the property *activiti.modeler.download.url* in your *[user.home]/.activiti/build.properties* to, e.g., http://activiti.org/downloads/activiti-modeler-5.0.beta1.war if you want to use a previous release or something like file:///home/falko/svn/activiti-modeler/dist/activiti-modeler.war if you want to use an own build of the Activiti Modeler.

linux.browser=echo: Set this poperty to echo to prevent the browser from being opened. You also use a different browser than Firefox through this property.

[user.home]/.activiti/tomcat-users.xml To enable the automatic redeployment targets in *qa/build.xml*, put a *tomcat-users.xml* in your *[user.home]/.activiti* directory with the following content:

```
  <?xml version="1.0" encoding="utf-8"?>
  <tomcat-users>
    <role rolename="manager"/>
    <user username="activiti"
password="activiti" roles="manager"/>
  </tomcat-users>
```

Eclipse IDE Setup

You'll need to install the Maven and Subversion (SVN) plugins.

In the "Galileo - [http://download.eclipse.org/releases/galileo|http://download.eclipse.org/releases/galileo]" update site, which should be available automatically, install "Collaboration --> Subversibe SVN Team provider". After rebooting and adding an SVN repository, you'll be asked automatically to install one of the polarion connectors for SVN. Just take the latest version of the polarion connectors. In case that doesn't happen automatically install a polarion connector manually from "Subversive SVN Connectors Site [http://community.polarion.com/projects/subversive/download/eclipse/2.0/update-site/|http://community.polarion.com/projects/subversive/download/eclipse/2.0/update-site/]"

Install the Maven plugins from [http://m2eclipse.sonatype.org/sites/m2e|http://m2eclipse.sonatype.org/sites/m2e]

First check out the activiti root from SVN as one project. Then import existing projects and navigate to the modules directory. All the module projects should then be found and can be imported in one go.

In order to have BPMN code completion and validation, import BPMN's XML Schemas from *activiti-engine/src/main/resources/org/activiti/impl/bpmn/parser* into the Eclipse XML Catalog, which can be found in Preferences --> XML --> XMLCatalog.

Coding style

- In the eclipse directory of the root activiti source directory are some files that should be imported as eclipse preferences. The filenames indicate the dialog that can be used to import them. Those eclipse files indicate the general coding style guidelines.** indents are 2 spaces (no tabs)** sun style braces formatting
- By default, our member field access should be set to **protected**

Commits

As much as possible try to group all related changes into single commits.

Before committing, run the following check to see if all is OK.

```
mvn -Pcheck clean install
```

In the commit message, start with the jira issue, a space and then the commit text like in this example

```
ACT-826 Fixed all problems in the world
```

Running the test suite using Spring configuration

Use

```
mvn -Pcheckspring clean install
```

to run the test suite on spring configuration only or

```
mvn -Pcheck,checkspring clean install
```

to combine running the engine testsuite on an Activiti config as well as on a Spring config

If you run the checkspring profile, the spring module will copy the engine tests over to the spring module before compiling and running the tests. Those tests are deleted in the 'package' build phase

If you need to debug spring configuration test cases, just execute a

```
mvn -Pcheckspring clean test
```

in the activiti-spring module, then refresh your IDE and the classes and spring configuration will be in your activiti-spring project ready to be debugged.

Checkin when test is failing

In some situations, it might be practical to check in while some tests are still failing. In that situation, make sure that the tests are excluded in the modules pom.xml and that you have created a JIRA issue for it. Reference the jira issue from the pom next to the excluded test. And reference to the pom in the jira issue.

QA

Build file qa/build.xml contains a number of targets for driving the QA. It also contains convenience targets for developers to do integration testing.

More about the QA and CI infrastructure can be found here: QA and CI Guide

Demo setup

To run smoke tests on webapps, the quickest way to get started is using the target test.demo.setup in qa/build.xml That will also startup h2 and tomcat. So to rebuild, you can use the target test.demo.setup.refresh That target will first shutdown tomcat and h2 and then do a full new demo test setup.
Building the javadocs

Use target build.javadocs in qa/build.xml

Debugging ant task

http://www.vitorrodrigues.com/blog/2009/07/10/debugging-ant-tasks-in-eclipse/

GIT pointers

We decided not to use GIT for now. These pointers are here for future reference when we would evaluate GIT again.

Currently there is a GIT repository fork from SVN, for experimental purposes. If you prefer to use GIT let Tom know and we can take that into account when deciding whether to stick with it.

To use GIT as a committer you will probably want to use ssh to authenticate automatically. Codehaus requires DSA tokens (not RSA) so you may need to create those first:

```
$ ssh-keygen -t dsa
... <accept the defaults>
```

Ensure that your `~/.ssh` directory is only readable by you (`chmod 700 ~/.ssh` should do it). You will also probably need a `~/.ssh/config` that contains these lines:

```
Host git.codehaus.org
 User git
 Hostname git.codehaus.org
 PreferredAuthentications publickey
 IdentityFile ~/.ssh/id_dsa
```

Typical initial setup:

```
$ git config --global user.name "My Name"
$ git config --global user.email
my.email@my.address.com
$ git config --global core.autocrlf input
# All the --global above can be set locally
if you prefer
$ git clone
ssh://git@git.codehaus.org/activiti-git.git
activiti
$ cd activiti
$ git branch -a
... (list of branches)
```

There are plenty of online resources for git. A good one is [Pro Git|http://progit.org/book]. IntelliJ has good IDE support. Eclipse has some very active development in [EGit|http://www.eclipse.org/egit/], but only basic features currently (things are moving very fast). Most people find `gitk`or `gitx` (Mac only) very useful for visualization (launch with `--all` for best results).

As a rule with GIT you do not make changes directly to remote branches, but rather you create your own local branch and merge changes from there to the remote branch (via a local mirror). Typical workflow

```
# create a new local branch to work from
$ git checkout -b feature/my-new-idea

# edit source code...
$ git status
... (list of changes)

# add all your changes to local index (N.B.
-A might not always be appropriate)
$ git add -A .

# commit locally
$ git commit -m "My nice changes"
```

Keep doing that until you are happy. If you want to tidy up your commits and send them as a batch instead of individual changes look at `git rebase -i ...`. Now you are ready to share your work. If you want to stay on a branch and collaborate with someone, push up your branch

```
# if it is new:
$ git push origin feature/my-new-idea

# if you already shared it
$ git push
```

To checkout someone else's remote branch for collaboration:

```
# Make sure you have all the remote changes
$ git fetch

# Or just make sure the branch is there
$ git branch -a
...

# Checkout the branch
$ git checkout origin/feature/my-new-idea
```

When you are ready to merge with the remote master (dev trunk):

```
# get changes from remote repository
$ git fetch

# merge changes onto master (= dev trunk for
everyone else)
$ git checkout master
$ git merge origin/master

# merge master onto your features (assuming
you are ready to try and push it
# up to the remote repository)
$ git merge feature/my-new-idea

# push changes up to remote repository and
move back to local branch to
# continue work
$ git push
$ git checkout feature/my-new-feature

# optionally rebase to keep history clean
$ git rebase master
```

There are many combinations of commands that achieve the same end result as that last sequence. Your preferred workflow might be different, and of course any time there are conflicts it will have to change (GIT is quite good at telling you what it thinks is wrong in a conflict).