Developer's Guide

[Source code hosting](#_s20or58wo6km)

[Licensing](#_m49ktewduwnx)

[Code formatting](#_wt2g0je6vptb)

[Main tools used](#_4g08r05swtf5)

[Good practices](#_glc6d8vhqc63)

[Assigning project to issues](#_r5qh4gwtmoqj)

[Assigning severity to issues](#_9ojlxhb7kvhs)

[Assigning priority to issues](#_bch4zq4qjfb8)

[Assigning type to issues](#_zbs4lq59fcfl)

[Assigning milestone](#_8fnglrnqu2lp)

[Github usage](#_4vdbr59ia7cg)

[Git](#_atk5co1i93jl)

[General coding](#_yq66ycvobq4o)

[Coding rules](#_t8b2qlg8e1bs)

[Testing rules](#_ds5ewmv1fs5j)

[Testing platforms](#_1y9s35npevrx)

[Shortcuts for frequently used services](#_x6q41fq9tejv)

[Support related tasks](#_9ztmj4heerog)

[Email aliases](#_vi0sqgk0qnqx)

[The process of building and starting the proactive-server from sources on local machine](#_zdwr80p5ns8f)

[General description](#_180dydct9pd3)

[Where to get ready proactive distribution](#_jni8cxnlpdyk)

[Start proactive-server](#_lw3hzwv8np1z)

[Proactive distribution from sources](#_nwbdp2kwvrfk)

[Step by step explanation of building from sources](#_ov7ps1t669jc)

[Building Scheduling with Scheduler, Resource Manager and Studio portals](#_4vsmpwefens7)

[More examples for other services](#_rtgyyh99j5nv)

[Easy building from sources](#_9uopgdpww2rf)

[Useful notes for running proactive](#_wpza4o47mmpx)

[Debug mode](#_7igj4o6w6nxg)

[Build scheduler without tests](#_6zo6otn2j2eg)

# Source code hosting

Public projects related to ProActive are hosted on Github under the organization named 'ow2-proactive':

<https://github.com/ow2-proactive>

Private projects (e.g. R&D projects, customers related projects, customer specific branches) are hosted on Bitbucket under the organization named 'activeeon':

<https://bitbucket.org/activeeon>

The main repositories are:

* scheduling: our scheduler implementation
* scheduling-portal: it is the implementation of the scheduler and rm webapps, GWT based
  + it was decided recently that no effort should longer be put on the RM webapp interface
* programming: the underlying library used by scheduling for communications   
  (hope to be replaced in a near future by something like rabbitmq)
* connector-r: connector that allows to submit jobs to the scheduler using the R language
* connector-matlab-scilab: similar to connector-r but for matlab and scilab languages

# Licensing

Most of the code is under aGPL v3 license:

<https://tldrlegal.com/license/gnu-affero-general-public-license-v3-(agpl-3.0)>

# Code formatting

All ProeActive java projects should use the same code format, which is defined in the eclipse code formatting file that can b used in IntelliJ thanks to the Eclipse code formatting plugin.

For more details, check the coding-rules project in github

<https://github.com/ow2-proactive/coding-rules>

# Main tools used

* Gradle  
  2.3, version 2.4 is not working due to onejar dependency (fixed in master but seems there is now an issue with gradle 2.5)
* IntelliJ  
  Usually, we are using IntelliJ IDEA as IDE. If you need a license for the Ultimate Edition, please contact [Fabien Viale](mailto:fabien.viale@activeeon.com) who is currently the person in the team who manages the licenses for this tool.
* Java
  + Version 8 (JRE) for executing products which are released,
  + version 7 (JDK) for compiling sources up to release 6.2.0 included. Then, version 8.
* Yourkit
  + Should ask for an open source license

# Good practices

## Assigning project to issues

Please perform minimal testing to find out where the issue is coming from, web application or server side, in order to create the issue in the correct project. CLI tools can help to find out if it a client or server side issue.

## Assigning severity to issues

Below is given a description for the different level of priority used for issues. It should help to decide which level to assign to an issue on Github. When creating an issue, please select the appropriate level according the following description:

* **Critical**: Blocks client(s) or development.
* **Major**: Severe regression or limitation, race conditions, client request, severe difficulties for debugging.
* **Minor**: Easy workaround is exists, cleaner or faster way to do something, small limitation (wishlist).
* **Trivial**: Syntactic sugar, hides user mistakes, cosmetic problem.

## Assigning priority to issues

Priority must not be assigned by developers.

* **High**
* **Medium**
* **Low**

## Assigning type to issues

* **Bug**: A problem which impairs or prevents the functions of the product
* **New feature**: A new feature of the product, which has yet to be developed.
* **Task:** A task that needs to be done.
* **Improvement**: An improvement or enhancement to an existing feature or task.
* **Story item**: Story are splitted in Story Item
* **Story**: = feature since we have story items? or a set of story = a feature ?
* **Task related bug**: A bug that is related to a task (ie this bug a no meaning outside of the development process)

## Assigning milestone

Please think to assign milestone once an issue or Pull-request is closed/merged. It is really important to automate release note in the future.

## Github usage

* Please create pull requests for commits that require a review and not for all commits
* Do not create an issue for each pull request. A pull request already contains the description of the issue that is solved (that's why pull requests and issues share the same id sequence on Github).

## Git

* Use the imperative, present tense: "change", not "changed" or "changes" for describing changes.

## General coding

* Do not forget to write tests for code which was edited and not yet tested.   
  Same for new features.

## Coding rules

* Functional tests on generated scripts should rely on the implementation proposed in functionaltests/dataspaces/TestSpecialCharacterFileName.java, where all interactions with the scheduler, rm, jobs, are achieved through ProcessBuilder instances.

## Testing rules

Tests must be systematically executed on a upgraded version of the code, before committing/pushing upgrades. There are 3 categories:

* Junit tests
* Functional tests (integration tests in reality)
* System tests
* Manual tests

# Testing platforms

Two platforms are currently running in order to test our products:

* <http://try.activeeon.com> the one running the latest stable version of the scheduler. It can be used for free by people who register on the website.
* <http://trydev.activeeon.com> the one running an unstable version (someone knows which one?). Mainly used for internal tests or demonstrations.

# Shortcuts for frequently used services

Some shortcuts have been created to redirect to services and URLs which are used frequently by developers. Below is the list of links you can use:

* <http://dev.activeeon.com>
* <http://github.activeeon.com>
* <http://ci.activeeon.com>
* <http://jenkins.activeeon.com>
* <http://bugs.activeeon.com>
* <http://issues.activeeon.com>
* <http://nexus.activeeon.com>
* <http://repository.activeeon.com>

Some shortcut also exists for old services which should no longer be used:

* <http://gerrit.activeeon.com>
* <https://jira.activeeon.com>
* <http://ci.lan>

# Support related tasks

For questions received by customers or potential customers, you have to put [support@activeeon.com](mailto:support@activeeon.com) in CC.

# Email aliases

The following email aliases exist:

* [activeeon-all@activeeon.com](mailto:activeeon-all@activeeon.com)
* [activeeon-dev@activeeon.com](mailto:activeeon-dev@activeeon.com)
* [activeeon-commercial@activeeon.com](mailto:activeeon-commercial@activeeon.com)

# The process of building and starting the proactive-server from sources on local machine

## General description

In order to be able to run the proactive-server on local you will need to have proactive-server distribution.

### Where to get ready proactive distribution

You can always download the latest nightly release from jenkins. (<http://jenkins.activeeon.com/view/NightlyRelease/job/nightly-release/>).

In this case you just download the distribution that you need from *artifact/build/distributions/*.

### Start proactive-server

When you unzipped the proactive distribution - you can start proactive-server.

You need to start the ./proactive-server script from /bin folder.

### Proactive distribution from sources

If you need to run the proactive-server from sources you will need to create the proactive-distribution by youself.

As proactive contains several projects, you need to follow general steps for each of them:

1. each project should be cloned from git
2. project should be built (usually with gradle command)
3. the build output should be copied to dist folder of main project (scheduling)

Next we will describe how to make this in simple steps

## Step by step explanation of building from sources

There are many micro-services inside proactive distribution, like:

* Resource Manager portal
* Scheduler portal
* Studio
* Workflow Catalog
* Cloud Watch
* Cloud Automation portal
* and others

We will concentrate only on the most indicative projects in order to have the basic functionalities:

- scheduling

- scheduling-portal

- rm-portal

- studio

- scheduling-api

You can build the other services following the same logic.

### Building Scheduling with Scheduler, Resource Manager and Studio portals

You will need to build those services from their associated repositories.

The ReleaseNumber in next explanation is depending on current version of release.

**Building Scheduling**

Let's start with the scheduling project:

git clone https://github.com/ow2-proactive/scheduling.git

cd scheduling

gradle build

If gradle is not installed on your system, you can try this instead:

./gradlew build

**Building Scheduling API**

Let's continue with the scheduling project:

git clone https://github.com/ow2-proactive/scheduling-api.git

cd scheduling-api

./gradlew clean build

The war file should be generated in:

scheduler-api-http/build/libs/scheduling-api-$ReleaseNumber-SNAPSHOT.war

We need to copy this file into the scheduling/dist/war folder and rename it to scheduling-api.war

**Building Scheduler and Resource Manager portals**

We will have to compile the other services and copy them into dist/war later on, so we need to generate the Scheduler and Resource Manager portals:

git clone https://github.com/ow2-proactive/scheduling-portal.git

cd scheduling-portal

./gradlew clean build

This should generate the following files:

- scheduler-portal/build/libs/scheduler-portal-$ReleaseNumber-SNAPSHOT.war

- rm-portal/build/libs/rm-portal-$ReleaseNumber-SNAPSHOT.war

We need to copy those 2 files into the scheduling/dist/war folder and rename them to (resp.):

- scheduler.war

- rm.war

**Building Studio**

Next is the Studio:

git clone https://github.com/ow2-proactive/studio.git

We need to copy the app/ folder into the scheduling/dist/war folder. Then, the app/ folder should be renamed to studio/

Finally we should have the following files inside the scheduling/dist/war folder:

- scheduler.war

- scheduling-api.war

- rm.war

- studio/

- getstarted/

- rest/

You should now be able to **run the proactive server** from within the scheduling/bin folder:

./proactive-server

### More examples for other services

Of course if you need to build particular service for developing you can do it with the same approach as before. Next we will show just few more examples.

**Building Workflow Catalog**

git clone https://github.com/ow2-proactive/workflow-catalog.git

cd workflow-catalog

./gradlew clean build

This should generate the workflow-catalog-$ReleaseNumber-SNAPSHOT.war file in the build/libs folder.

We need to copy this file into the scheduling/dist/war folder and rename it to workflow-catalog.war

**Building Cloud-automation portal**

You need to download repo: https://bitbucket.org/activeeon/cloud-automation

Then build:

grunt clean build

From dist folder of project copy to scheduling/dist/war/cloud-automation folder

## Easy building from sources

For making the process of building the proactive-distribution from sources easier you can use virtual links.

For example in unix:

cd ~/git/scheduling/dist/war

ln -s ~/git/forked/studio/app/ ~/git/forked/scheduling/dist/war/studio

ln -s ~/git/forked/scheduling-portal/scheduler-portal/build/libs/scheduler-portal-7.28.0-SNAPSHOT.war ~/git/forked/scheduling/dist/war/scheduler.war

ln -s ~/git/forked/scheduling-portal/rm-portal/build/libs/rm-portal-7.28.0-SNAPSHOT.war ~/git/forked/scheduling/dist/war/rm.war

ln -s ~/git/forked/scheduling-api/build/libs/scheduling-api-7.28.0-SNAPSHOT.war ~/git/forked/scheduling/dist/war/scheduling-api.war

## Useful notes for running proactive

### Debug mode

You can configure your idea to run in debug mode

ECLIPSE DEBUG MODE EXAMPLE:

-server -Dfile.encoding=UTF-8 -Dpa.rm.home=/Users/git/forked/scheduling -Dpa.scheduler.home=/Users/git/forked/scheduling -Djava.library.path=/Users/git/forked/scheduling/dist/lib -Djava.awt.headless=true -Dproactive.test.timeout=600000

### Build scheduler without tests

You should always run tests! But for scheduler project the tests take a lot of time. And in case if you really need to build scheduler without tests, you can do:

gradlew clean build - x check

For more information go into the documentation or ask your awesome colleagues ;)