SiGrid2

Eclipse - ide - install - Howto

Abstract. This document is targeted at developers willing to contribute to the SiGrid 2 project. It describes the setup of a development environment for SiGrid 2 using the popular eclipse ide on windows. Other OS like linux will work as well if paths are amended. This howto does _not_ describe the SiGrid framework itself. Refer to the diploma thesis of Holzapfel and Beck if you are new to the SiGrid Framework. This howto describes the basic setup of SiGrid needed to start developing. Thus, side projects as the rails based controller frontend and the connection via SOAP is left out.

1. Getting eclipse & installing eclipse

In this document we describe the setup using the "Eclipse IDE for Java EE Developers" in the current (as of 2009-10) stable version "eclipse-jee-galileo-SR1" (Version 1.2.1.20090918-0703, Build id: 20090920-1017) which is based on Eclipse 3.5 SR1. Originally SiGrid2 was develloped using eclipse 3.1, so most eclipse 3.x builds should work as well. We downloaded our copy from

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR1/eclipse-jee-galileo-SR1-win32.zip

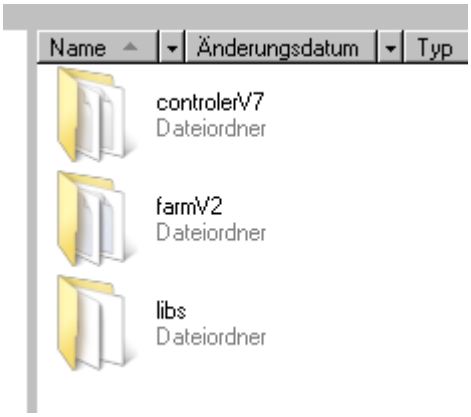
Install the ide according to Install-Instructions and choose an appropriate workspace. Create a Test-Project and ensure that everything is in place.

2. Building the projects

Sigrid2 is split into two projects: the “controller” part, which also hosts the basic objects referenced by the farm, and the “farm”-part, which contains the algorithms and the infrastructure needed to do distributed simulations.

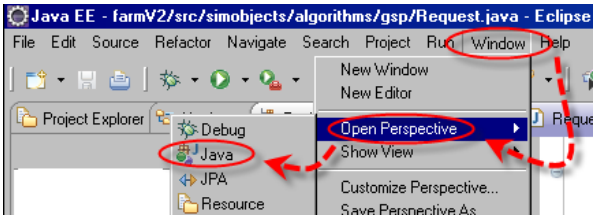
2.1. Extracting the zip Archives

Extract the Zip-File to a temporary location of your choice. After the file is extracted, three directories can be found a top Level:



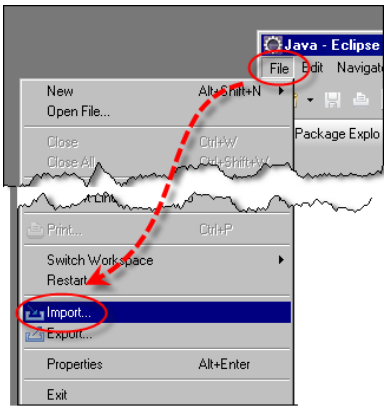
2.2.1 Importing and building the controller project

We start by importing the controller part. The following steps are using the Java Perspective of Eclipse. To choose this perspective, select Window -> Open Perspective -> Java.

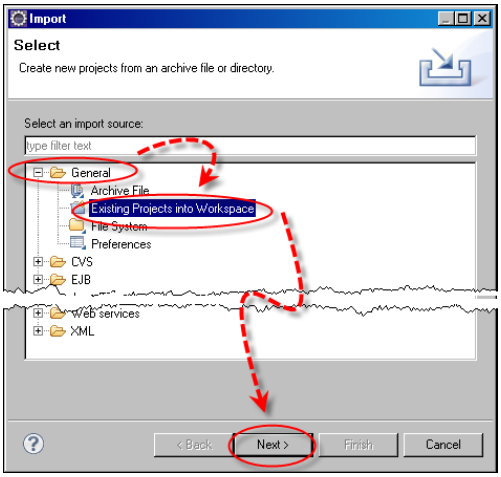


2.2.2 Importing sources into a new project

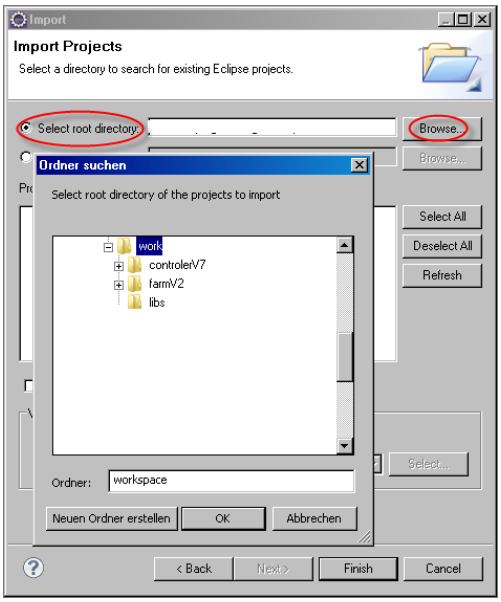
Then choose File -> Import



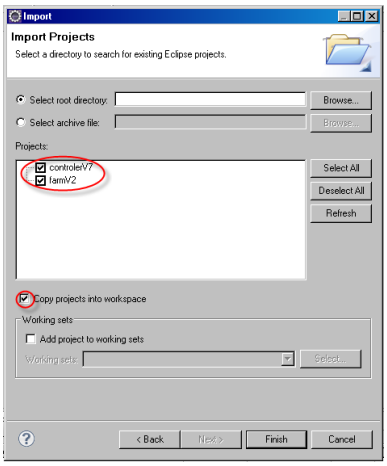
Choose “Existing Projects into Workspace”, then click “Next”



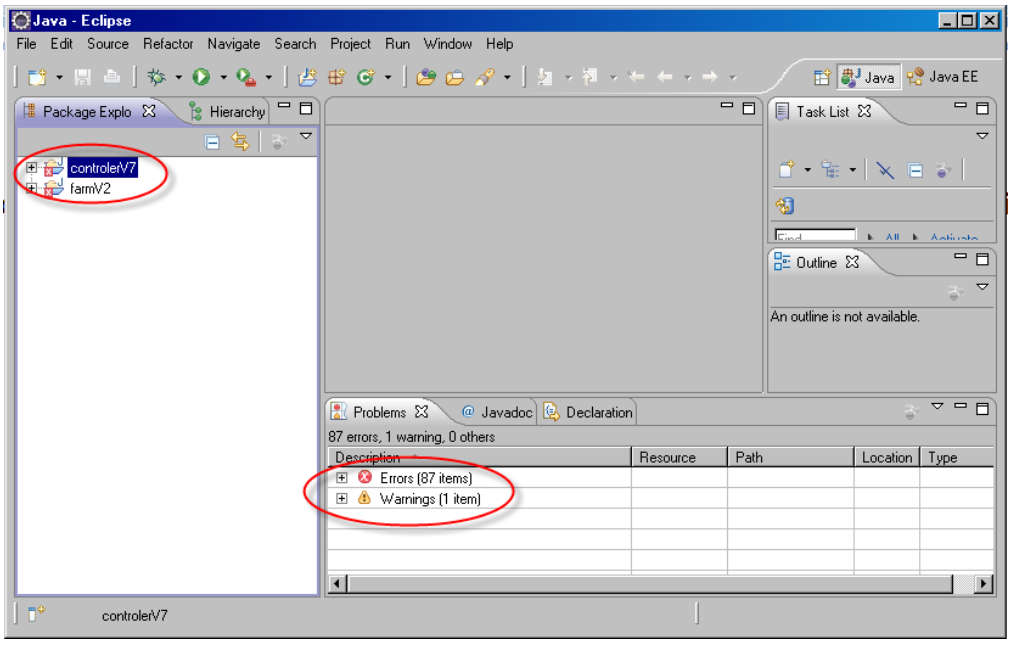
Navigate to the directory which contains the extracted folders



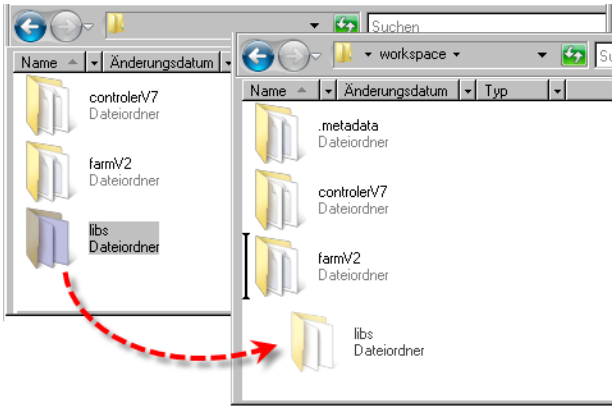
Select both entries in the following dialogue. Take care to tick “copy projects into workspace”



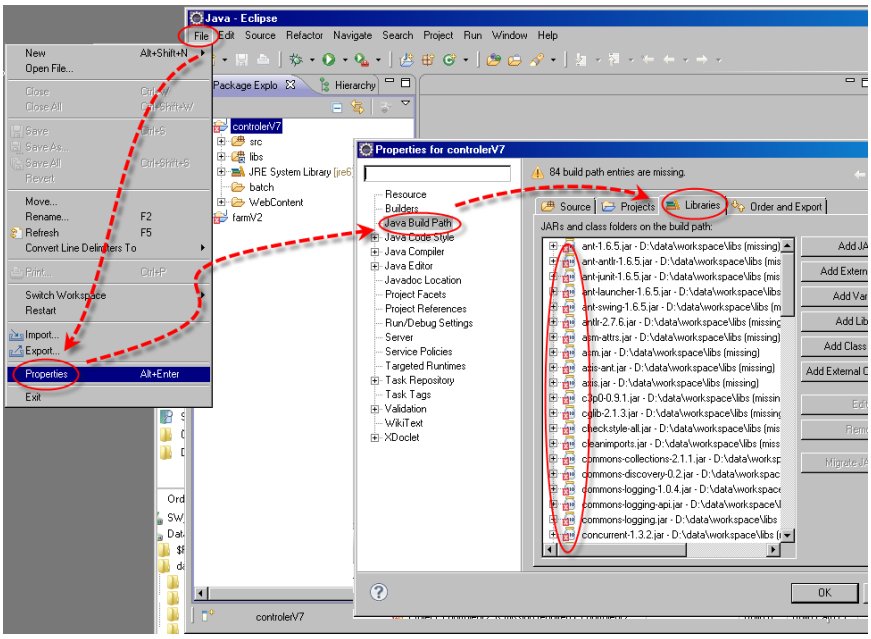
As soon as Eclipse is trying to build the Project, many errors are reported. This is because both projects require Java libraries that we have not included yet.



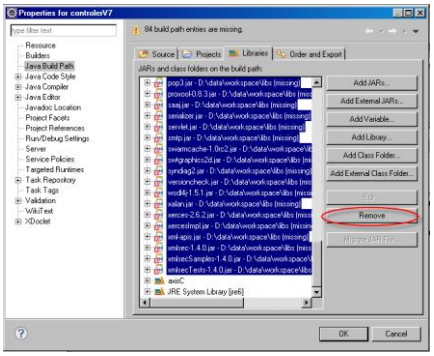
To fix this, we first copy the “libs” directory from the extracted archive to our workspace



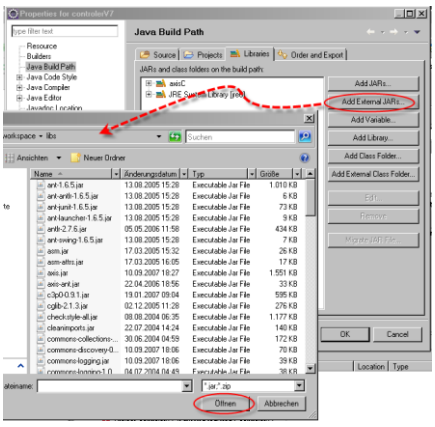
Then, back into eclipse we select File -> Properties -> Java -> Java Build Path -> Libraries, where we remove all libraries that are marked red ("… missing)



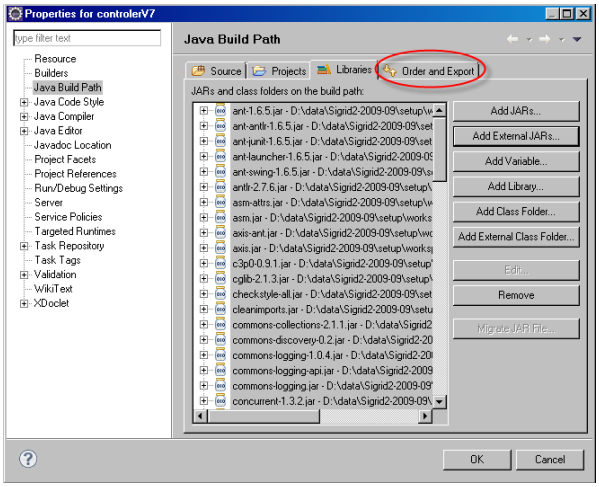
To do this, we mark those entries and click on "remove"



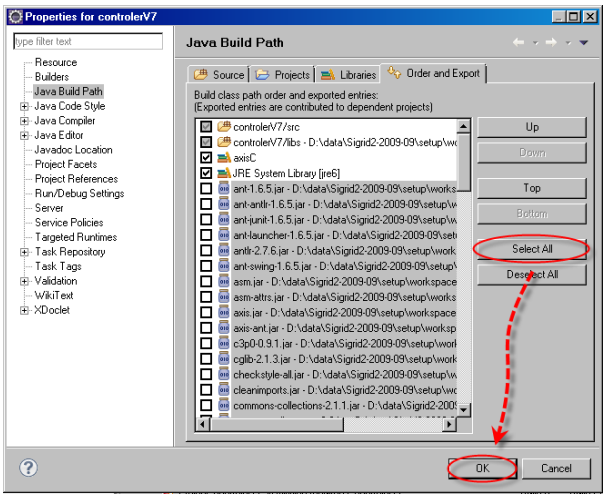
Then, in the same dialogue, we add all libraries contained in the folder we just copied.
-> Click on "Add external JARs" and navigate to the "lib" folder
-> Select all files and click on "open"



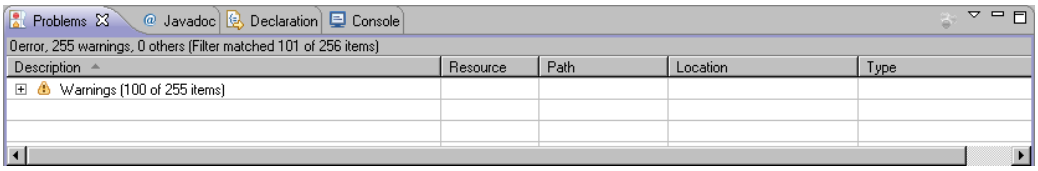
Make sure all entries have been imported and change to the tab “Order and export”



Include all libraries by choosing “select All” and “ok”



The project is now being rebuild by the eclipse IDE. Now, the errors have disappeared:



(A message “this project needs to migrate to WTP metadata” can be safely ignored, as we do not use the webservice-functionality here)

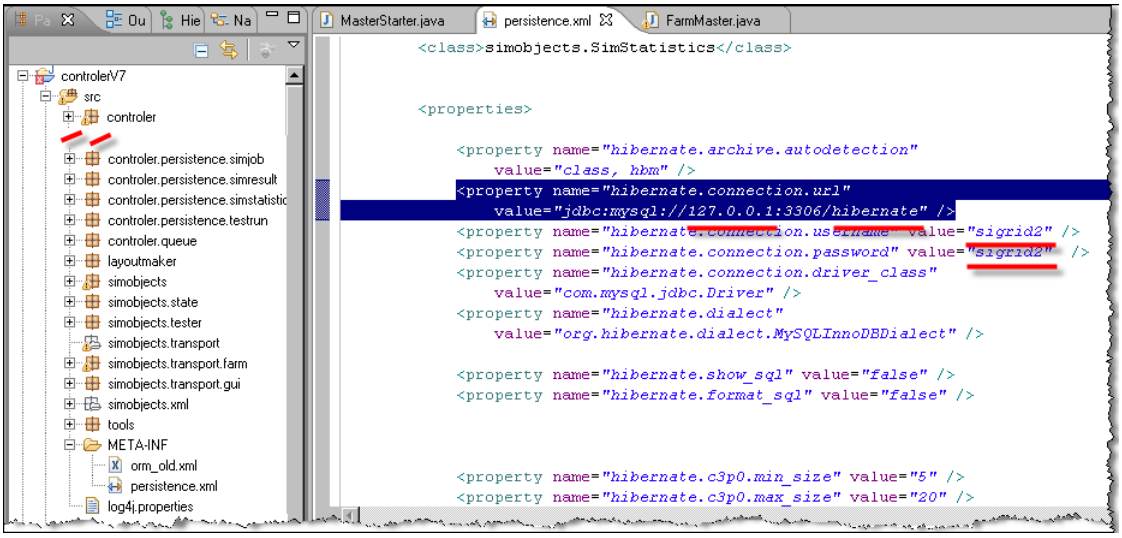
3. Database setup

Sigrid uses hibernate (https://www.hibernate.org/) as Object-Relational-Mapper (ORM). During development, we used a MySQL 5.0 database (http://www.mysql.de/). Other databases like Postgres (http://www.postgresql.org/) are also possible.

Here, we assume that you have already set up und configured a MySQL-database. In the following, we use the username “sigrid2” and the password “sigrid2” to connect to a

database called “hibernate” on the host localhost (IP=127.0.0.1). You can (and should) use different credentials. If you do so, you need to edit the file persistence.xml (in the subdir controlerV7/src/MEATA-INF).

```
<property name="hibernate.connection.url" value="jdbc:mysql://127.0.0.1:3306/hibernate" />
<property name="hibernate.connection.username" value="sigrid2" />
<property name="hibernate.connection.password" value="sigrid2"  />
<property name="hibernate.show_sql" value="false" />
<property name="hibernate.format_sql" value="false" />
```



3.1. Creating User and Database

To create user and database, you can use the following SQL commands:

```
mysql> create database `hibernate`;
mysql> create user sigrid2@'localhost' identified by 'sigrid2';
mysql> grant all on hibernate.* to sigrid2@'localhost';
```

After this, the following command-line shoul provide access to the database:

```
mysql -u sigrid2 -psigrid2 hibernate
```

3.2. Creating and filling the required tables

To fill the database with the required tables and data, you can use the file “sigrid.sql”, which can also be found in the project archive. The following command-line can be used.

```
mysql -u sigrid2 -psigrid2 hibernate < sigrid.sql
```

If we now connect to the database, all tables should be shown:

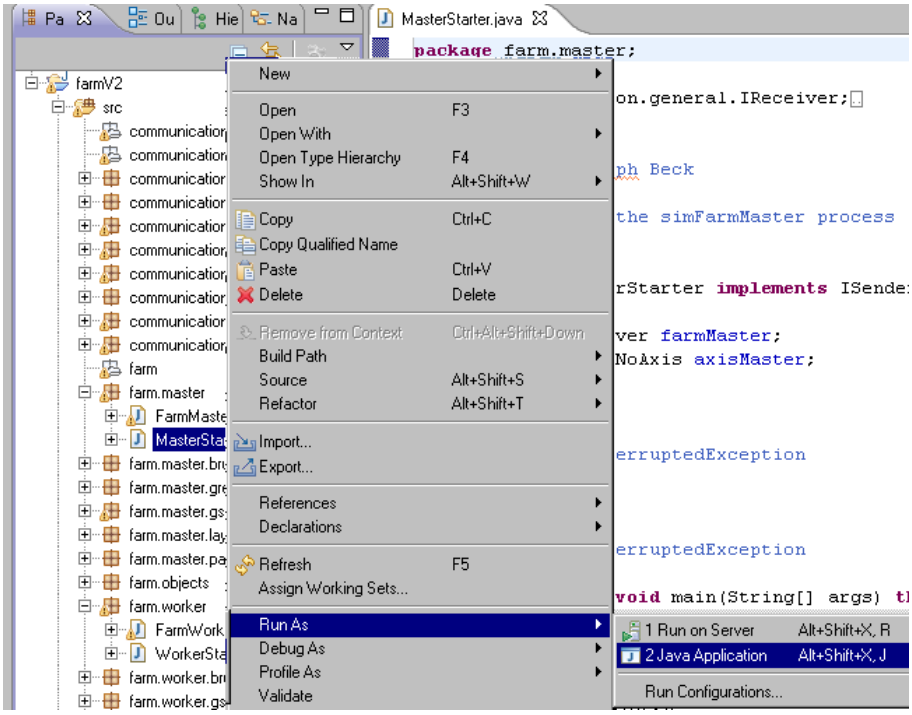
```
mysql -u sigrid2 -psigrid2 hibernate
mysql> show tables;
+--------------------------+
| Tables_in_hibernate      |
+--------------------------+
| algorithm                |
| layout                   |
| layout_jobs              |
| layout_server            |
| simfarm                  |
| simfarm_simjob           |
| simjob                   |
| simresult                |
| simstatistics            |
| simstatistics_statistic  |
| testrun                  |
+--------------------------+
11 rows in set (0.00 sec)
```

4. Testing the setup

We can now test the setup by starting the application. Here, the communication with the controller is done by direct object-reference, not via the SOAP/Axis connection.

4.1. Creating a launch configuration for the farmMaster-Process

We will now start the FarmMaster-Process, which coordinates the processes of farmWorker(s) and controller. First, locate the MasterStarter Class in the farm.master package of farmV2. Select the class in the package explorer. Add a run task by right-clicking and choosing “Run As -> Java Application”

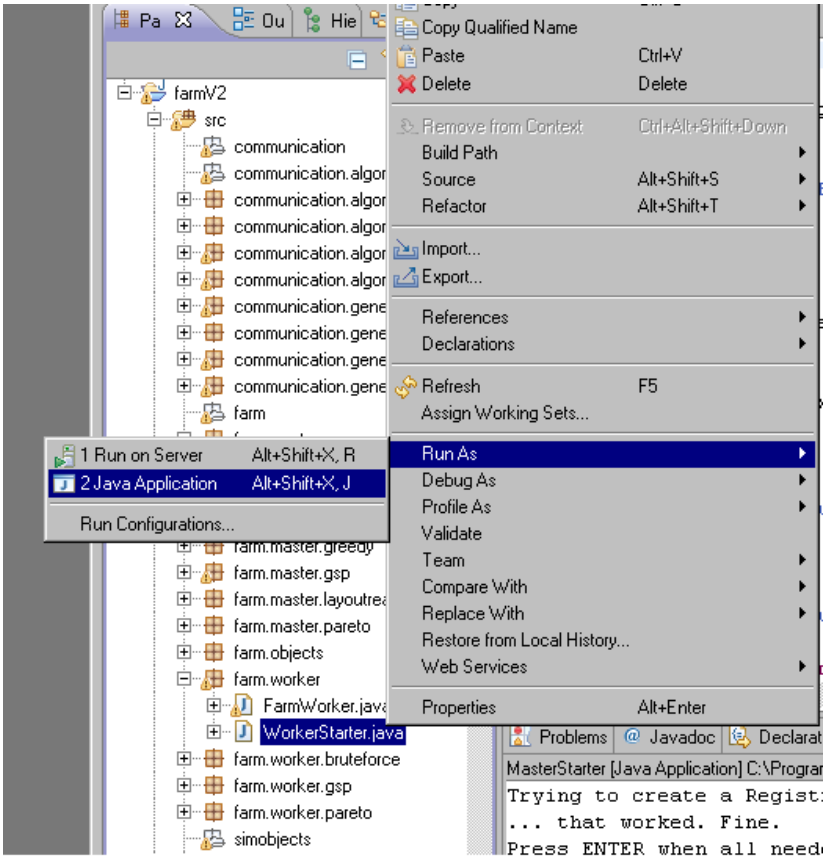


The FarmMaster-Process is now booting up and showing in the console window:

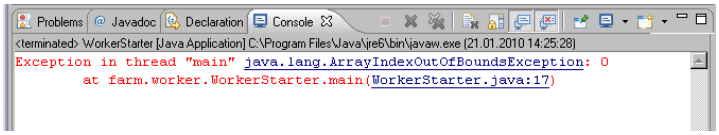
The FarmMaster-Process binds to the localhost network interface by default. If network connections are controlled by a firewall, make sure that connections to the RMI Port are allowed for the java-processess.

4.1. Creating a launch configuration for the farmWorker-Process

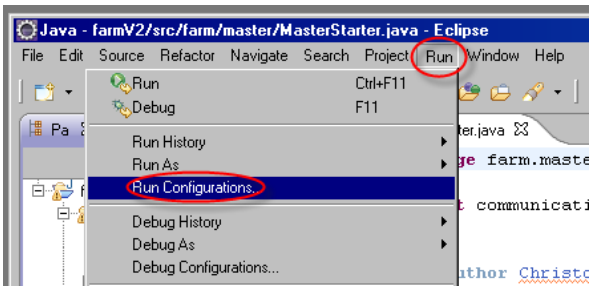
Now, locate the WorkerStarter Class in the farm.packeage package of farmV2. Again, select the class in the package explorer. Add a run task by right-clicking and choosing “Run As -> Java Application”.



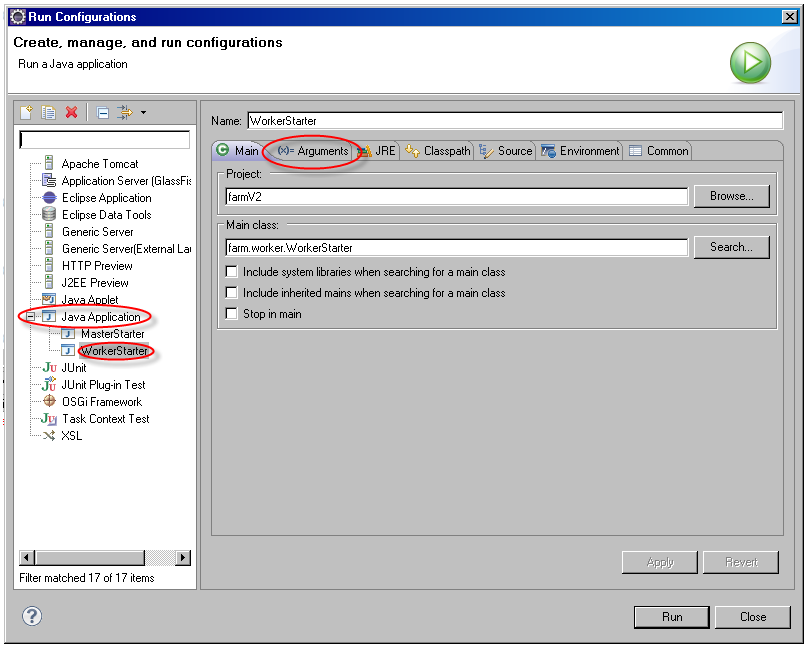
An error is shown in the console window:



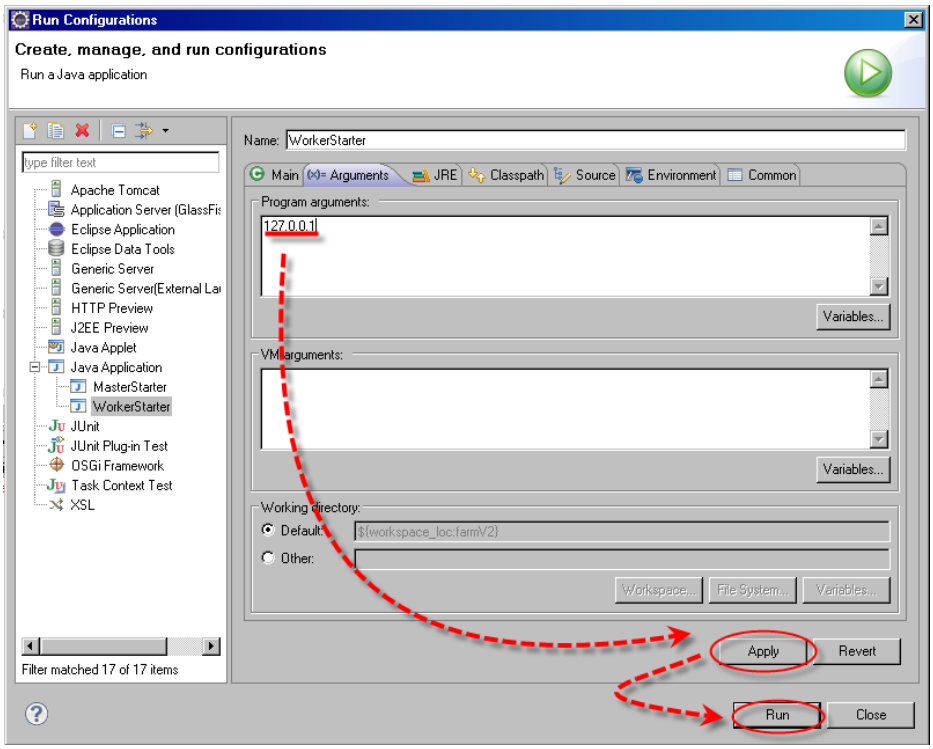
This happens because the WorkerStarter needs a Parameter on the command line: The IP-Address of the host, the FarmMaster-Process is running on. To fix this, we need to modify the “run-configuration” which has be automatically created by the eclipse-ide during the last step. Chose “Run->Run Configruations” from the main menue:



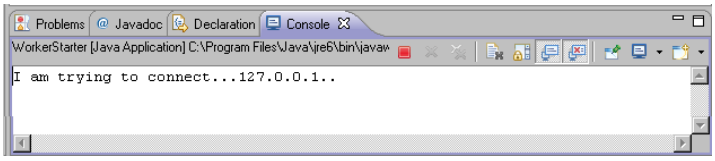
Locate the entry “WorkerStarter” in the “Java Application” branch of the configuration tree and activate the tab “Arguments”



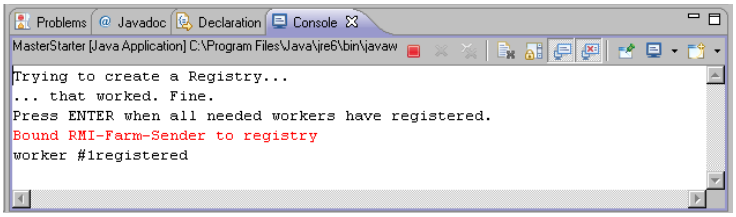
There, enter the address of the host the Master is running on (here 127.0.0.1) click on “Apply”, then “Run”



Now the FarmWorker boots up successfully:



The log of the FormMaster shows the connect:



We can add more worker at taste. When done, activate the MasterWorker Console and click on enter. The Farm-Process is now binding to the controller and ready for simulation.

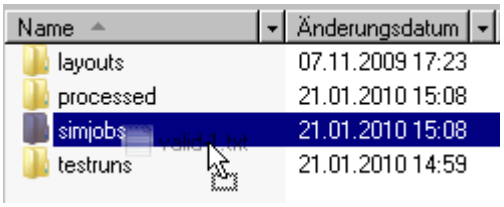
The log also shows the directory which can be used for batch-processing: Any valid Layout or Testrun files placed here will be processed. We use this directory for the final run test.

5. Final test with a TestRun-File

To test the setup, use the TestRun file “Valid-1.txt” in the subdir “testruns” of the batch directory. The file is also contained in the archive:

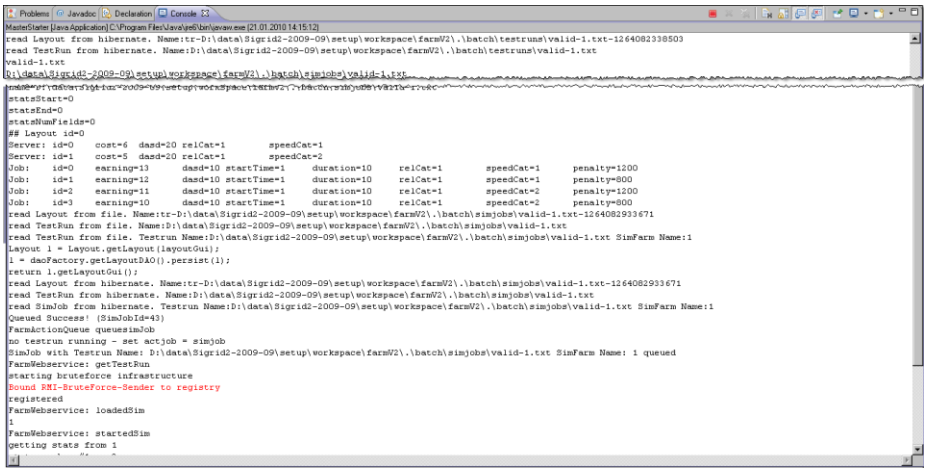
```
SimJob:   farmid=1
TestRun:  algorithmid=1

Server:   id=0 cost=6 dasd=20 relCat=1 speedCat=1
erver:    id=1 cost=5 dasd=20 relCat=1 speedCat=2

Job:      id=0 earning=13 dasd=10 startTime=1 duration=10 relCat=1 speedCat=1 penalty=1200
Job:      id=1 earning=12 dasd=10 startTime=1 duration=10 relCat=1 speedCat=1 penalty=800
Job:      id=2 earning=11 dasd=10 startTime=1 duration=10 relCat=1 speedCat=2 penalty=1200
Job:      id=3 earning=10 dasd=10 startTime=1 duration=10 relCat=1 speedCat=2 penalty=800
```

Place the TestRun file in the subdir “simjobs” of the batch directory:



As soon as the batch process discovers the new directory content, the file is parsed und the Layout is simulated with the given Algorihm.

This is shown in the Console of the FarmMaster Process:



On finishing the simulation, the maximal earning found by the algorithm is displayed:

```
starting bruteforce infrastructure
Bound RMI-BruteForce-Sender to registry
registered
FarmWebservice: loadedSim
1
FarmWebservice: startedSim
getting stats from 1
state worker #1 : -2
Statscomplete: true
state worker #1 : -2
all worker statscomplete
FarmWebservice: addSimStatistics
in finished sim ****
UNBIND
*****************
earning: 2400
FarmWebservice: addSimResult
finishedSim, receiverID=0
FarmWebservice: finishedSim
```

Also, a log file is generated in the “log” subdirectory of the workspace, which is named as the SimJob-File prepended by the epoche-seconds string of the moment it was started:

```
## Layout id=49
Server:   id=0      cost=6    dasd=20   relCat=1  speedCat=1
Server:   id=1      cost=5    dasd=20   relCat=1  speedCat=2
Job:      id=0      earning=13          dasd=10   startTime=1         duration=10         relCat=1  speedCat=1
          penalty=1200
Job:      id=1      earning=12          dasd=10   startTime=1         duration=10         relCat=1  speedCat=1
          penalty=800
Job:      id=2      earning=11          dasd=10   startTime=1         duration=10         relCat=1  speedCat=2
          penalty=1200
Job:      id=3      earning=10          dasd=10   startTime=1         duration=10         relCat=1  speedCat=2
          penalty=800
#Simjob id44
#Simjob queuedMs12910
#Simjob startedMs1264083842378
#Simjob finishedMS1264083842442
#####
#Testrun id48
#Testrun nameD:\data\Sigrid2-2009-09\setup\workspace\farmV2\.\batch\simjobs\valid-1.txt
#Testrun algorithmbf
#SimResult Earning2400
(…)
x -
x -
- x
- x
```

The last lines show the Job-server mapping in the following format:

(…)
x –
x –
– x
– x

	Server 1	Server 2
Job 1	x	-
Job 1	x	-
Job 1	-	x
Job 4	-	x