RoboCup

@Work

Getting Started with the Software

PART I: Installation and Deployment

To run and use the software successfully we recommend you to make sure that your system has the following:

- A *Java Runtime Environment* (JRE7 or newer is recommended) and *Eclipse SDK* (version 3.7.2 or newer is preferred).
- The *"ASTT-RoboCup-At-Work"* project's source files. You can get it from github repository (https://github.com/rnatar609/ASTT-RoboCup-At-Work.git).
- *ØMQ* (The Intelligent Transport Layer) with the language bindings (for C and C++ binding is done automatically with *ØMQ*).
- For Windows OS we need *Microsoft Visual C++ 2008* or newer to build *ØMQ* solution.

Here we will show you how to deploy the solution in both Windows and Linux (so we recommend you to follow the steps from A to Z to assure a successful implementation).

1) Windows

1. Open *Eclipse SDK and change the workspace to the project folder.*
   File→Switch Workspace→Other→Brows and navigate to the project's folder.

2. *ØMQ* integration*:-

   2.1. Get the software for Windows from this link (select Stable release 2.2 / windows source).
   http://www.zeromq.org/intro:get-the-software

   2.2. Extract the zipped source archive "*zeromq-2.2.0.zip".*

   2.3. Using the Microsoft VC++ open the *"\builds\msvc\msvc.sln"*
   File→Open→Project/Solution→navigate to the targeted file the click Open.

   2.4. Build the solution.

   2.5. *ØMQ* libraries will be in the *"lib"* subdirectory.

3. *ØMQ* Java binding (*jzmq*)**:-

3.1. Get the java binding from this link (use githup repository)
http://www.zeromq.org/bindings:java

3.2. Using the Microsoft VC++ open the *"\builds\msvc\msvc.sln"*
File→Open→Project/Solution→navigate to the targeted file the click Open.

3.3. Before building the solution be sure to of the following

3.3.1. ØMQ and JDK header file has to be on "include path"
Tools →Options →Projects and Solutions →VC++ Directories →Include files
Then add the following paths:

```
<jdk>\include\win32
<jdk>\include
<zeromq>\include
```

3.3.2. ØMQ libraries have to be on "library path"
Tools →Options →Projects and Solutions →VC++ Directories →Library files
Then add the following path:

```
<zeromq>\lib
```

3.4. Build the solution.

3.5. zmq.jar and the ØMQ java binding's libraries will be at "lib" subdirectory.

4. Finalize the ØMQ integration before run the project's modules

4.1. Add the "*libzmq.dll" which is in* ØMQ "*lib*" subdirectory to the System "PATH".

Right click Computer→Properties→Advanced system settings→from Advanced tab →Environment Variables→from System variables select Path→then click Edit→add semicolon ";" to the last entry→add the path to lib.

4.2. Now using Eclipse SDK, and for both of the project packages (RefereePkg and RobotPkg).

Right click→Build Path→Configure Build Path→navigate to Libraries tab→click on Add External JARs→navigate to jzmq\lib and select "*zmq.jar"*.

And from the same tab click on Add External Class Folder→navigate to jzmq\pre and click OK.

Now drop the JRE System Library[jre7] →select Native Library Location→click on Edit→for Location path click on External Folder→navigate to jzmq\lib then click OK→OK. And close the main window.

Be sure to do the same steps for both packages.

5. Run the project.

For Referee System drop the RefereePkg→src→RefereePkg→right click on the RefereeSystem.java→ select Run As→Java Application.

For Robot System drop the RobotPkg→src→java→right click on the RobotModule.java→select Run As→Java Application.

----

2) Linux

3. Open *Eclipse SDK and change the workspace to the project folder.*
   File→Switch Workspace→Other→Brows and navigate to the project's folder.

4. *ØMQ* integration*:-

   2.1. Get the software for Linux from this link (select Stable release 2.2 / POSIX tarball).
   http://www.zeromq.org/intro:get-the-software

   2.2. Make sure that *libtool*, *autoconf*, *automake* are installed.

   2.3. Check whether *uuid-dev* package*, uuid/e2fsprogs* RPM or equivalent on your system is installed.

   2.4. Unpack the *.tar.gz* source archive.

   2.5. Open Terminal and navigate to directory.

   2.6. Run "*./configure"*, followed by *make*.

   2.7. To install ØMQ system-wide run "*sudo make install"*.

   2.8. On Linux, run "*sudo ldconfig"* after installing ØMQ.

3. *ØMQ* Java binding (*jzmq*)**:-

   3.1. Get the java binding from this link (use githup repository) http://www.zeromq.org/bindings:java

3.2. Open Terminal and navigate to directory.

3.3. Run the following commands:

3.3.1. $*./autogen.sh*

3.3.2. $ *./configure*

3.3.3. $ *make.*

3.3.4. $ *sudo make install.*

3.3.5. $ *sudo ldconfig.*

4. Finalize the ØMQ integration before run the project's modules

4.1. Now using Eclipse SDK, and for both of the project packages (RefereePkg and RobotPkg).

Right click→Build Path→Configure Build Path→navigate to Libraries tab→click on Add External JARs→navigate to "*jzmq/src*" and select "*zmq.jar"*.

And from the same tab click on Add External Class Folder→navigate to "*jzmq/pre"* and click OK.

Now to add the "*libzmq.dll*" and "*libjzmq.dll*" to the j*ava.library.path* drop the JRE System Library[jre7] →select Native Library Location→click on Edit→for Location path click on External Folder→navigate to "*/usr/local/lib"* then click OK→OK. And close the main window.

Be sure to do the same steps for both packages.

5. Run the project.

For Referee System drop the RefereePkg→src→RefereePkg→right click on the RefereeSystem.java→ select Run As→Java Application.

For Robot System drop the RobotPkg→src→java→right click on the RobotModule.java→select Run As→Java Application.

----

* http://www.zeromq.org/intro:get-the-software

** http://www.zeromq.org/bindings:java

PART II: How to run the software

<u>Step 1</u>: *Referee / Server side*

- Run the referee system by running RefereeSystem.java class in RefereePkg.
- Once you run this class, a GUI window will pop-up.
- Then we have to load the configuration to the main GUI window by using the “load config” button and navigating to the configuration file “config.txt”.
- After the configuration is loaded, an arena map will appear on the GUI. A socket is created on the server side and it is now waiting for the client requests on the socket.
- Once this is done you can generate the task specification either by creating a new task specification or by using the existing one.
- Now the referee is ready to send the specification to the client once it is connected.

<u>Step 2</u>: *Client / Team side*

- We need to run the client which is implemented in three languages: Python, C++ and Java.
- For Java as an example, run the RobotModule.java class.
- Now client will try to connect to the server. When it gets connected it will send the team name to the server and waiting for the task specification from the server.

<u>Step 3</u>: *Referee / Server side*

- Once a connection is established between the server and the client, you can send the task specification by clicking on the specification button on the GUI panel.
- But to activate this button you first need to start the timer “setup time” by clicking the timer button.
- Only after this the send specification button is enabled and ready to send the specification to the connected client.
- Then the server will wait for acknowledgement from the client about the task specification reception.

<u>Step 4</u>: *Client / Team side*

- Once the client receives the task specification from the server it will send acknowledgement to the server and get terminated.

<u>Step 5</u>: *Referee / Server side*

- After receiving acknowledge the server will start the competition “run time”.
- Referee can finish the competition by clicking on the Competition finish button in GUI.