EL4J

GettingStarted

Generated from Wiki page:
http://wiki.elca.ch/twiki/el4j/bin/view/EL4J/GettingStarted

Version	Date	Author(s)	Visa
1.24	03 Aug 2007 − 09:48	Main.DavidStefan, Main.MarcusDeluigi, Main.PhilippHOser, Main.StefanWismer, StefanWismer	

Table of Contents

1 GettingStarted

1.1 Getting Started

This guide is intended to help new EL4J developers (=developers working ON EL4J, not developers using EL4J!) with setting up the environment, downloading the sources and learning the ropes of EL4J and its tools. The contained http://wiki.elca.ch/twiki/el4j/pub/EL4J/GettingStarted/el4j.rar file can help you getting started quickly. It may later need some adjustments.

If you want to start your own project on top of EL4J, please have a look at the instruction of the page EL4J under "first steps"

1.1.1 Setting up your environment

This section will guide you through the installation of all necessary tools, which are needed for the EL4J development.

1.1.1.0.1 EL4J directory structure.

We recommend the following directory structure:

- Create a directory `Projects` for your projects, e.g. `D:\Projects`
- Create a directory `EL4J` in your `Projects` directory, e.g. `D:\Projects\EL4J`
- Create a directory `tools` in your `EL4J` directory, e.g. `D:\Projects\EL4J\tools`

1.1.1.0.2 JDK 1.5 SE

- Download the most recent update of JDK 5.0 (*Standard edition*) from http://java.sun.com/javase/downloads/index_jdk5.jsp
- Follow the instruction of the installation guide and install the Developer Kit to `C:\jdk<version>`, where `<version>` is your update version. The reason for this is that whitespaces in the path could lead to problems.
- At some time, the installation guide will ask you to install the JRE. Change the standard installation directory to `C:\jre<version>`
- Check your environment variables. You should have:
    - ♦ `JAVA_HOME` pointing to `C:\jdk<version>`
    - ♦ an entry in your `Path` variable pointing to `%JAVA_HOME%\bin` (add all entries in the path at the *beginning*)
- Go to `C:\jdk<version>\bin` and make a copy of `javaw.exe`. Name it `eclipse_javaw.exe`. You will find this very handy, because it will prevent you from killing Eclipse when killing Java jobs.

1.1.1.0.3 JAD

- Download the most recent version of JAD from http://www.kpdus.com/jad.html#download
- Copy the `jad.exe` file into your `C:\jdk<version>\bin` directory.

1.1.1.0.4 Cygwin

- Go to http://www.cygwin.com/ and download the latest version of Cygwin.
- Install it to `C:\cygwin`
- Check your environment variables. You should have:
    - ♦ an entry in your `Path` variable pointing to `C:\cygwin\bin`
- create a `.bash_profile` file in your home directory and add following lines:
    - ♦
    ```
    alias debugmaven='export MAVEN_OPTS="-Xmx1024M -Xss128k
    -XX:MaxPermSize=512M -Xdebug
    -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000
    ```

  `-Ddb.name=db2"'`
  - `alias runmaven='export MAVEN_OPTS="-Xmx1024M -Xss128k -XX:MaxPermSize=512M -Duser.language=en -Duser.region=US -Ddb.name=db2"'`

1.1.1.0.5 SVN

- Please check out the info under http://intranet.elca.ch/Business_Process/Utilities/Subversion/Subversion.php as there are some disturbing bugs in Svn clients (it's slowly getting better − YMA knows more about it)
- Download the correct version of Svn to your favorite directory, such as `C:\Subversion`.
- Check your environment variables. You should have:
  - `APR_ICONV_PATH` pointing to `C:\Subversion\iconv`
  - an entry in your `Path` variable pointing to `C:\Subversion\bin`
- Open a cygwin console and type in `svn --help` to check the correct installation of subversion.

1.1.1.0.6 Maven2

Note for internal developers: please follow the corresponding section in the InternalGettingStarted guide.

- Take the Maven zip file provided with this Getting Started Guide (from https://sourceforge.net/projects/el4j). We use our own patched version of Maven due to a few unresolved bugs in the standard version.
- Unzip it to `D:\Projects\EL4J\tools\maven`
- Check your environment variables. You should have:
  - `M2_HOME` pointing to `D:\Projects\EL4J\tools\maven`
  - an entry in your `Path` variable pointing to `%M2_HOME%\bin`
- Open a cygwin console and type `mvn -version` to check if Maven is working.
- Change to your home directory in your cygwin console (e.g. by `cd ~`) and generate a directory `.m2` (e.g. by `mkdir .m2`)

1.1.1.0.7 Eclipse

Note for internal developers: please follow the corresponding section in the InternalGettingStarted guide.

- Go to http://www.eclipse.org/ and download the latest version of Eclipse.
- Unzip it to `D:\Projects\EL4J\tools\eclipse`.
- If you like, you can set a shortcut. After creating the shortcut, right click on it and set the target to `D:\Projects\EL4J\tools\eclipse\eclipse.exe -vm C:\jdk<version>\bin\eclipse_javaw.exe -Duser.language=de -Duser.region=CH -vmargs -Xmx384M`

Finally, open a cygwin console and test if the configuration above made correctly. Enter the following commands:

- java −version
  - Must print out the version number of a Java 5 JDK or newer.
- javac −version
  - Must print out the same version number as above.
- mvn −version
  - Must print out the version "2.1−SNAPSHOT" or newer.
- echo $MAVEN_OPTS
  - Must print something like `-Xmx1024M -Xss128k -XX:MaxPermSize=512M -Duser.language=en -Duser.region=US -Ddb.name=db2`.
    If not, execute `export MAVEN_OPTS="-Xmx1024M -Xss128k -XX:MaxPermSize=512M -Duser.language=en -Duser.region=US -Ddb.name=db2"`

1.1.2 Setting up EL4J

By now, you should have all necessary tools and you will download the sources for EL4J now, set up Eclipse and the Maven2 repository for development.

Note for internal developers: after reading this section, please follow the corresponding section in the InternalGettingStarted guide.

1.1.2.0.1 Download sources

- Open a cygwin console and change to your `EL4J` directory.
- Check out the external repository with `svn co https://el4j.svn.sourceforge.net/svnroot/el4j/trunk/el4j external`

1.1.2.0.2 Change settings

- Go to `D:\Projects\EL4J\external\etc\m2\` in your explorer and copy the `settings.xml` file to your `~/.m2` directory
- Change the `settings.xml` file in your `~/.m2` directory to the following
    - Change the value of the `localRepository` tag to `D:/m2repository`
    - Remove the comments around the `proxy` tag
    - Remove the comments around the `el4j.root`, `el4j.external` and `el4j.internal` tags (under profile `el4j.general`)
    - Change the value of `el4j.root` to `D:/Projects/EL4J`
    - The value of `el4j.external` should be `${el4j.root}/external` (same holds for the internal for internal developers).
    - Change the value of `el4j.project.home` to `${el4j.root}`

- Open Eclipse and go to `Window -> Preferences` and there to `Checkclipse`
- Add `D:\Projects\EL4J\external\etc\checkstyle\checks.xml` as the `Checkstyle Configuration File`
- Add `D:\Projects\EL4J\external\etc\checkstyle\checks.properties` as the `Checkstyle Properties File`

1.1.2.0.3 JDBC Drivers

Note for internal developers: skip this section.

If you are not inside the ELCA network you will have to download the Database JDBC Drivers for Oracle and Derby/DB2 yourself, because we are not allowed to distribute them. To download and deploy these drivers into your local repository, do the following:

- Download the Oracle driver (ojdbc14.jar) from "http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/". Note: You can also use the 10g driver for version 9 database.
- Download the DB2/Derby JDBC driver (db2jcc.jar and db2jcc_license_c.jar) from "http://www.ibm.com/developerworks/db2/downloads/jcc/".
- Open a console and change to the directory where you have saved the downloaded jar files.
- Execute the following commands:
    - `mvn install:install-file -DgroupId=com.oracle -DartifactId=ojdbc14_g -Dversion=10.2.0.1.0 -Dpackaging=jar -Dfile=ojdbc14_g.jar`
    - `mvn install:install-file -DgroupId=com.ibm -DartifactId=db2jcc -Dversion=20040819 -Dpackaging=jar -Dfile=db2jcc.jar`
    - `mvn install:install-file -DgroupId=com.ibm -DartifactId=db2jcc_license_c -Dversion=20040819 -Dpackaging=jar -Dfile=db2jcc_license_c.jar`

Note that the used file names of the jar files in the commands above may depend on the downloaded version. The used version number (`10.2.0.1.0`) is the one used in `module-database`. Leave this version number as it is to avoid dependency conflicts.

1.1.2.0.4 Build project

- Go to `D:\Projects\EL4J\external` in your cygwin console and type `mvn clean install`. This will probably take a while (~ 30min) and will build the external part of the EL4J framework.
- Type `mvn eclipse:clean eclipse:eclipse -DdownloadSources=true` for Maven2 to generate Eclipse project files.

1.1.2.0.5 Include EL4J into Eclipse

You could add all EL4J modules into your workspace, but this is very confusing. Therefore, we recommend the following:

- Open Eclipse with your `D:\Projects\EL4J\workspace` workspace.
- Go to Window -> Preferences
- Go to Java -> Build Path -> Classpath Variable
- Add a new variable with the name `M2_REPO` and the path `D:\m2repository`
- Add another variable, set the name to `EL4J_HOME` and the path to `D:\Projects\EL4J`

- Close the Preferences window and find the little triangle in the uppermost right corner of your `Package Explorer`.
- Go to `Configure Working Sets...`.
- Create the working sets `modules`, `applications`, `demos`, `tests` and `plugins`.
- Click on the triangle again and choose `Top Level Elements -> Working Sets`.

- Go to File -> Import and choose Existing Projects into Workspace
- Click on Next and on the next page on Browse next to "Select root directory". Go to `D:\Projects\EL4J\external\framework\modules`, click on Ok and then Finish.
- Move the projects in your `modules` set.
- Repeat the same with
    - ♦ `D:\Projects\EL4J\external\framework\tests` (for `tests`),
    - ♦ `D:\Projects\EL4J\external\applications\demos` (for `demos`)
    - ♦ `D:\Projects\EL4J\external\applications\templates\common` (for `applications`).
    - ♦ `D:\Projects\EL4J\external\maven` (for `plugins`).

- Additionally you can exclude external libraries. Click on the triangle and on `Filters...`. Choose `Libraries from external` there.

1.1.3 EL4J introduction

1.1.3.1 Maven2 introduction

This section will give you a brief introduction to maven. It will explain you the basic terms of Maven and the use of `archetypes`. Maven2 is a tool to manage software projects. Maven2 is able to manage a project's build, reporting and documentation based on a project object model called `POM`.

1.1.3.1.1 Structure of a Maven Project

EL4J is built with Maven and consists of several subprojects. Each of these subprojects (called `artifacts` in Maven) has the following structure:

- `src` directory containing the source files
- `pom.xml` file with the description of the `artifact` for Maven
- `.settings` directory as well as a `.classpath` and `.project` file if you invoke `mvn eclipse:eclipse`
- `target` directory if you invoke `mvn install`

`Artifacts` are hierarchically structured having a `root pom.xml` file, in our case `D:\Projects\EL4J\external\pom.xml`. The are linked with help of a `parent` tag that a `pom.xml` file can have.

1.1.3.1.1.1 Maven commands

There are only two Maven commands you will need at the beginning.

The first one is `mvn clean install`, which will do the following to the artifact and any child artifact

- `clean` deletes existing `target` directories in the artifact directory
- `install` compiles all sources in the `src` directory into a `artifactName.jar` file, runs JUnit tests, if there are any, creates the `target` directory, copies the jar file in the `target` directory. Moreover it copies the jar file into the local repository, in our case `D:\m2repository`

Note: To make changes on your artifact effective, you always have to invoke `mvn clean install`*. This will cause Maven to deploy the jar file into the local repository.*

The second command is of the form `mvn <plugin>:<goal>`. You will need the Maven Eclipse plugin to generate Eclipse project files for your projects. You do this using the command `mvn eclipse:clean eclipse:eclipse −DdownloadSources=true`. For further instructions on how to import a project into Eclipse, please read the `Eclipse` section under `Setting up EL4J`.

1.1.3.1.1.2 Dependencies & repositories

Now, go to your `D:\Projects\EL4J` directory in a cygwin console and

- invoke `mvn archetype:create −DarchetypeGroupId=ch.elca.el4j −DarchetypeArtifactId=EL4JArchetypeCore −DarchetypeVersion=1.2 −DgroupId=ch.elca.el4j −DartifactId=myFirstProject −DremoteRepositories=http://el4.elca−services.ch/el4j/maven2repository`. This will generate you a new Maven project
- Change to `cd myFirstProject`. As you see, you can find the `src` directory and the `pom.xml` file typical for a Maven project.
- Take a look at the `pom.xml` file:
    - ♦ You will see that our `pom.xml` file doesn't have a parent, because it's the top level pom of an independent project.
    - ♦ There are `dependencies` to `junit` and `module−core`. The first one is needed to run the tests of our projects (you'll see them later) and the second is the Core Module of EL4J. It's there because we want to build our project upon the EL4J framework.

Maven tries to resolve dependencies from the local repository, i.e. it checks if you have a jar file with the same `groudId`, `artifactId` and `version` in your local repository. If this is not the case, Maven will try to download these artifacts from the remote repositories to your local repository.
As you can easily see, Maven will have to download the artifacts from the remote repository only for the first time and will look it up in the local repository afterwards.

1.1.3.2 EL4J project structure

An EL4J project will have a typical structure:

- `src`
    - `main`
        - `java` This is where all the source (i.e. java) files go to.
        - `resources` This is where all additional files go to like configuration files.
        - `env`
            - `env` This is where the `env.properties` file goes to. If you invoke `mvn clean install` it will be copied to the `target` directory and will be accessible in the progam with help of `module-env`
    - `test` This is where all test files go to. It has the same structure as `main`, but is there for testing.
        - `java`
        - `resources`
        - `env`
            - `env`

We recommend you to go on with reading some of the additional material now.
Alongside, try to play around with the `myFirstProject` a little bit. Try, to import the project into eclipse. Add then a `env.properties` file to your project, add a new dependencies to `module-env` from EL4J and use the class `EnvPropertiesUtils` from `module-env` to read out some properties you create.

1.1.4 Reading

Note for internal developers: please follow the corresponding section in the InternalGettingStarted guide.

For reading material, take a look at http://el4j.sourceforge.net/documentation.html

1.1.5 Initial development

By now, you should

- Have a local copy of the EL4J repository (don't forget to update now and then with `svn up`)
- This copy of EL4J should compile with `mvn clean install` without errors
- Have set up Eclipse to work with EL4J
- Understand the basic concepts of Maven and be able to include new dependencies and use them
- A basic understanding of Spring, especially about the `Application Context`, about the use of configuration files and IoC?.

If so, you're ready for the next step − go directly into EL4J! You will learn the structure of the EL4J framework, get to know some of the EL4J Demos and learn how to debug a project.

1.1.5.0.0.1 The EL4J framework

First, the EL4J framework has following structure:

- `applications`
    - `templates` Contains the two examples `keyword` and `refdb` out of which we create our templates
    - `demos` Demos that explain a specific functionality of the framework.
- `etc` Contains additional content like the checkclipse files, log4j configuration, etc.
- `framework`
    - `modules` The framework modules of EL4J external
    - `tests` (Integration) Tests, which test two or more (framework) modules.
- `maven`
    - `archetypes` The archetype you used earlier
    - `helpers` Some helpers you don't have to worry about now

        - `plugins` Maven plugins that were developed by the EL4J team
- `sandbox` The place where we try out new things
- `site` Configuration and additional documents for the website generation
- `skin` The "skin" of the website
- `src` Source folder for the website generation. This will hopefully be removed in the future.

1.1.5.0.0.2 EL4J Demos

Note for internal developers: for additional material, see the web application template section in the InternalGettingStarted guide.

EL4J comes with a few demos that show how to use a specific feature of EL4J (like the statistics functionality). You will find them all in your demo working set in Eclipse. They are all executable. Please read the corresponding `README.txt` files for further instructions.
You could try to take a closer look at the `Benchmark Demo`. How is remoting done in EL4J? What kind of protocols does EL4J support? What is `Implicit Context Passing`?

1.1.5.0.0.3 Developing with Eclipse

Eclipse should only be used to write code and test small parts of the project. Most other development tasks should be executed wit Maven, especially the unit tests due to the following reasons:

- Eclipse projects *do not separate compile and test scope* as Maven does. This can be dangerous, for example if the directory test resources contain Spring bean xml files in the “mandatory” directory.
- Maven does always have dependent jar artifacts as jar files in the classpath. In Eclipse, depending to execution level/directory of the goal “mvn eclipse:eclipse”, some dependencies are in classpath as jar and some directly as directory with its classes. The test classes itself are always in classpath via directory.
- Eclipse has its own compiler. There are some cases, for example with Java 5 syntax, that tests work only if the classes are compiled with the Eclipse compiler. If they are compiled with a Sun’s compiler, the tests fail. At the end tests should work with both compilers (so using the stricter compiler (as with maven) improves compatibility).

1.1.5.0.0.4 Debugging

Maven allows you to debug any executed command in Eclipse. To do so you have to:

- Call `debugmaven` in your cygwin console
- Set your breakpoints in Eclipse
- Invoke the Maven command that you want, e.g. `mvn clean install`
- Go to `Run -> Debug...` in Eclipse.
- There, create a new `Remote Java Application`
- Set the `Connection Properties` Host: `localhost` and Port: `8000`
- Click on "Apply" and "Debug"

More info on this can be found under DebuggingHowTo. *Note for internal developers: You can debug a Maven command at the Leaffy Server by changing the Host to `leaffy` as well*