EL4J

GettingStarted

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

Version	Date	Author(s)	Visa
1.57	17 Mar 2009 - 10:38	Main.AlexanderDeiss, Main.ClaudeHumard, Main.DavidBernhard, Main.DavidStefan, Main.MarcusDeluigi, Main.MartinZeltner, Main.PhilippHOser, Main.StefanWismer, PhilippHOser	

©ELCA Informatique SA, Switzerland 2009

Table of Contents

1 Getting started with EL4J 1
- 1.1 Setup EL4J 1
- 1.2 First steps with EL4J 1
- 1.3 EL4J demo applications 1
  - 1.3.1 Simple project 1
  - 1.3.2 GUI application template 1
  - 1.3.3 JSF application template 1
  - 1.3.4 Old web application template 1
- 1.4 Questions for people migrating from LEAF2 1
- 1.5 EL4J introduction 1
  - 1.5.1 Maven2 introduction 2
    - 1.5.1.1 Structure of a Maven project 2
    - 1.5.1.2 Maven commands 2
    - 1.5.1.3 Dependencies & repositories 2
  - 1.5.2 EL4J project structure 3
- 1.6 Reading 3
- 1.7 For Developers: Initial development 3
  - 1.7.1 Getting support 3
  - 1.7.2 The EL4J framework 3
  - 1.7.3 EL4J Demos 4
  - 1.7.4 Developing with Eclipse 4
  - 1.7.5 Debugging 4

1 Getting started with EL4J

1.1 Setup EL4J

Follow the steps in SetupEL4J carefully.

1.2 First steps with EL4J

CAVEAT: EL4J contains a lot of different technologies. Please contact POS in case you feel overwhelmed while you learn EL4J!

- IntroductoryReadingListForEl4J: What documents we recommend that you read to get into EL4J.
- General information about EL4J can be found at WebHome and AboutEL4J.
- CourseAboutEl4j

If you start a new project using EL4J see also

- MavenRepositoryForOwnProject

1.3 EL4J demo applications

1.3.1 Simple project

To start with a trivial program (1 class and 1 test) that contains a dependency to EL4J module core, follow the steps of the README.txt file in the current EL4J convenience zip.

1.3.2 GUI application template

Refer to SwingApplicationTemplate.

1.3.3 JSF application template

Refer to TenderTracker and ModuleJsf.

1.3.4 Old web application template

Refer to the old web application template.

1.4 Questions for people migrating from LEAF2

I want to migrate from LEAF2. What should I know?

- A comment on the migration plan from LEAF 2 to EL4J
- What is new for LEAF 2 users

For new projects starting with Spring or EL4J, please contact PhilippHOser to decide on the most appropriate use of the technologies. We can help you setting the environment up effectively. We also collect information on how other projects use EL4J and Spring (to give you interesting links to other parts). Ideally there is an experienced Spring/ EL4J programmer in the team of a new project.

1.5 EL4J introduction

This section gives you a short overview over the build system Maven 2 and the project structure of a typical EL4J application. At the end, you find links to additional documents.

1.5.1 Maven2 introduction

This section will give you a brief introduction to the Maven2 build system. 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`.

1.5.1.1 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.

1.5.1.2 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`.

1.5.1.3 Dependencies & repositories

Now, go to your `D:\Projects\EL4J` directory in a cygwin console and set up a trivial application as described in the README.txt file of the EL4J convenience zip file.

- 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 `groupId`, `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.

1.5.2 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.

1.6 Reading

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

1.7 For Developers: 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.

1.7.1 Getting support

More info can be found here: GettingSupport

1.7.2 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.

1.7.3 EL4J Demos

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`?

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

1.7.4 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 with 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).

1.7.5 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*

Note for internal developers: please see the corresponding section in the InternalGettingStarted#Reading guide for additional readings.