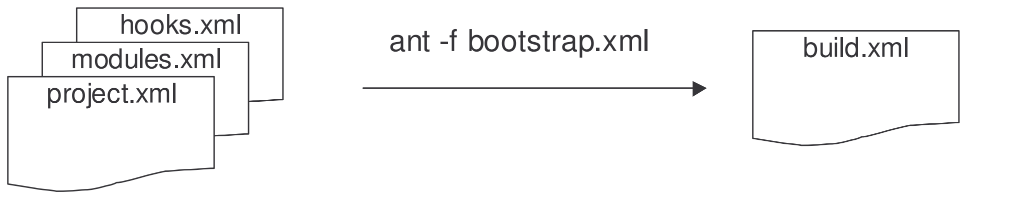
EL4Ant Quick Reference Sheet (http://el4ant.sourceforge.net)

Project description
(typically split in files)

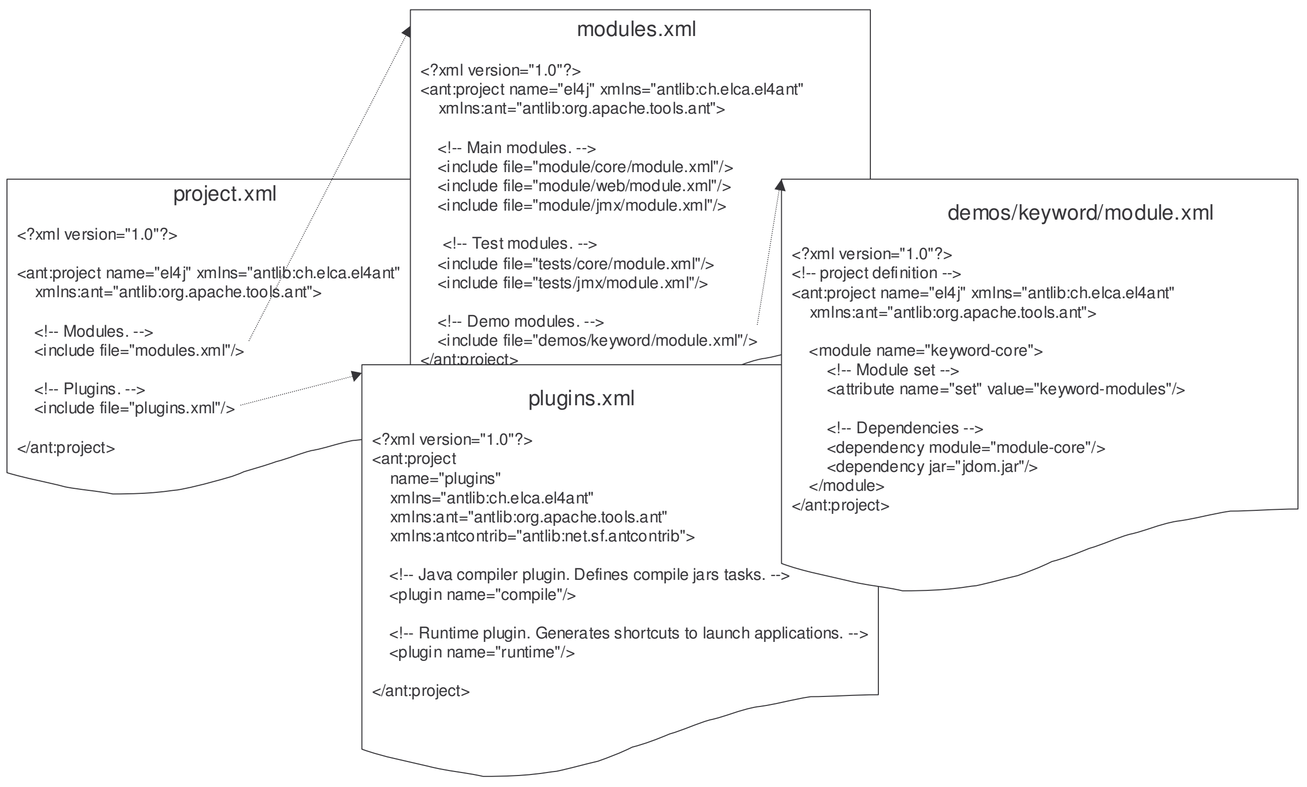
Standard Ant build file



Typically, the first step when working with EL4Ant is to generate a standard `build.xml` file from a project description.

Concept	**Description**
Module	Collection of source code, configuration, jar files and dependencies to other modules. A module is a directory in the file system and is (by default) described in the `module.xml`.
Target	An operation of the `build.xml` file that can be invoked on the level of Ant.
Attribute	An attribute is a name=value pair. EL4Ant and all its plugins are configured with attributes.
Plugin	A plugin can extend the basic functionality of EL4Ant. See the list of existing plugins below.
Module set	Each module defines what module sets it is contained in (sample sets are `tests` or `framework`). A set classifies modules in logical groups to apply targets.
Dependencies	Dependencies specify what jar files and other modules are required to run. Dependencies are transitive and the dependency management also takes care of the classpath.
Hook	Hooks allow adding specific behavior during the execution of build system targets. For example, one can add special treatment before or after compilation or just after the deployment target.
Execution unit (eu)	With execution units, one can partition a module into subsets. Sample executions units are `interface` and `implementation` or `client` and `server`. Executions units (eu) are fully optional.
Project	A project is a collection of *modules*, *plugins, execution units* and *hooks* that are used to generate a standard build.xml file.

Sample project description



project.xml

```
<?xml version="1.0"?>

<ant:project name="el4j" xmlns="antlib:ch.elca.el4ant"
    xmlns:ant="antlib:org.apache.tools.ant">

    <!-- Modules. -->
    <include file="modules.xml"/>

    <!-- Plugins. -->
    <include file="plugins.xml"/>

</ant:project>
```

modules.xml

```
<?xml version="1.0"?>
<ant:project name="el4j" xmlns="antlib:ch.elca.el4ant"
    xmlns:ant="antlib:org.apache.tools.ant">

    <!-- Main modules. -->
    <include file="module/core/module.xml"/>
    <include file="module/web/module.xml"/>
    <include file="module/jmx/module.xml"/>

     <!-- Test modules. -->
    <include file="tests/core/module.xml"/>
    <include file="tests/jmx/module.xml"/>

    <!-- Demo modules. -->
    <include file="demos/keyword/module.xml"/>
</ant:project>
```

plugins.xml

```
<?xml version="1.0"?>
<ant:project
    name="plugins"
    xmlns="antlib:ch.elca.el4ant"
    xmlns:ant="antlib:org.apache.tools.ant"
    xmlns:antcontrib="antlib:net.sf.antcontrib">

    <!-- Java compiler plugin. Defines compile jars tasks. -->
    <plugin name="compile"/>

    <!-- Runtime plugin. Generates shortcuts to launch applications. -->
    <plugin name="runtime"/>

</ant:project>
```

demos/keyword/module.xml

```
<?xml version="1.0"?>
<!-- project definition -->
<ant:project name="el4j" xmlns="antlib:ch.elca.el4ant"
    xmlns:ant="antlib:org.apache.tools.ant">

    <module name="keyword-core">
        <!-- Module set -->
        <attribute name="set" value="keyword-modules"/>

        <!-- Dependencies -->
        <dependency module="module-core"/>
        <dependency jar="jdom.jar"/>
    </module>
</ant:project>
```

Basic Targets

Targets work relative to the current directory: If you are in the directory of a module, the targets are launched only for this module. We assume here that the current project directory is `PROJECT`.
`ant -p` shows all available targets (this is standard ant).
There are many convenience targets to invoke targets just for a specific module (e.g. `jars.rec.module.module-core` invokes the `jars.rec.module` target for the module `module-core`; these convenience targets are omitted in the following).

jars.rec.module	Recursively compile the current module and all modules it depends on and generate the jars in `PROJECT/dist/lib`.
start.module.eu	Launch the default java class of the current module (as defined in its `module.xml` file). With a corresponding hook, one can pass arguments to the Java executable via `-Dargs="..."`
junit.start.all	Execute all junit tests (In more detail: call `start.module.eu` on all modules that have the `junit.runnable` attribute set).
javadoc	Generate javadoc in `PROJECT/dist/website/javadoc`
junit.report	Create a junit report for executed junit tests in `PROJECT/dist/website/junit`.
checkstyle	Checks the project with Checkstyle rules. The report is under `PROJECT/dist/website/checkstyle`.
website	Creates a project website in `PROJECT/dist/website`. It contains links to the generated project reports of junit, javadoc, checkstyle, …
clean.rec.module	Clean a module and its dependencies

Selected EL4Ant and EL4J* plugins

Plugin	Purpose	Sample ant targets
Eclipse	Generate project description and dependency files for eclipse.	None (is invoked by default during `ant -f bootstrap.xml`)
Binrelease	Generates binary releases for modules. Benefits: only 1 zip-file/module, versioned, typically tested, quicker.	`binrelease.module` generates a releasable module as a zip file in `dist/binaries`. `binrelease` generates all releasable modules as binary modules.
Resources	Adds a list of files (resources) to the classpath. Used e.g. to add configuration files to the classpath.	None (works implicitly during the postcompile hook of the compile target)
J2EE	Provides support for J2EE applications that run either in a servlet or in an EJB container. It allows creating packages of the specific format and delivers targets to control the different servers.	`create.war.module.eu` generates a war file for a module `deploy.war.module.eu` deploys the war file of a module. It starts the web container if necessary. `stop.web` stops the running Web container `create.ear.module.eu` generates an ear file for a module

Parallel*	Support for executing several targets in parallel, i.e. starting multiple JVMs. This feature is mostly used for distributed tests. Currently it supports to start a web or EJB server in parallel with the (client) JUnit tests.	There are no explicit targets. The following hooks are typically added before JUnit test executions:
		
		`runtime.hook.parallel-ejb.start`
		starts the EJB server in a separate process and invokes the target specified by `parallel-ejb.deploytarget`.
		`runtime.hook.parallel-ejb.stop`
		stops the EJB server
		`runtime.hook.parallel-web.start`
		same for the web
		`runtime.hook.parallel-web.stop`
		same for the web

Distribution*	Creates executable distributions (i.e. zip files) that don't have any EL4Ant or Ant dependencies.	`create.distribution.module` creates an executable distribution that contains all runnable execution units

*EL4J-Specific plugins