Contents

I Practices **3**

II Ant **4**

1 Overview **5**

2 Build File **6**
2.1 overview and a document structure 6
2.2 preamble 6
2.3 tags 6
2.3.1 project 6
2.3.2 target 7
2.3.3 task 7
2.4 list of tasks 7
2.4.1 delete 7
2.4.2 echo 7
2.4.3 ivy:retrieve 7
2.4.4 java 8
2.4.5 javac 8
2.4.6 jar 8
2.4.7 javadoc 8
2.4.8 mkdir 8
2.5 ant properties 8
2.5.1 predefined ant properties 9
2.6 ant-data types 9
2.6.1 fileset 9
2.6.2 filelist 10
2.6.3 patternset 10

3 Property File **12**

4 Resources **13**

III Ivy **14**

5 Overview **15**

6 Syntax **16**
6.1 structure 16
6.2 ivy-module 16
6.3 info 16
6.4 dependencies 17
6.4.1 dependency 17

7 Resources **18**

IV Graddle **19**

8 Overview **20**
8.1 Project 20

9 Resources **21**

Part I

Practices

Part II

Ant

Chapter 1

Overview

Another Neat Tool. Open and portable standard to automate build and deploy processes. Build files are in XML format. By convention they are called *build.xml*.

Chapter 2

Build File

2.1 overview and a document structure

By convention they are called *build.xml*. Structured wit following tags:

- **preamble** (optional)
- **project** - mandatory!
- **target** - at least 1 mandatory

2.2 preamble

Optional!

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

There should be no tab, blank line or whitespace before!

2.3 tags

2.3.1 project

This element is required! Attributes:

- name (optional)
- default (**mandatory!**). It specifies which target should be executed when command *ant* is not suplied with an argument.
- basedir; optional - location of the build file is taken as a root dir if not specified
- xmlns:ivy=“antlib:org.apache.ivy.ant defines an XML namespace for ivy. Mandatory in order to resolve dependencies with ivy.

2.3.2 target

`ant -p`

lists all available targets. **At least one is mandatory!** Attributes:

- name = mandatory
- depends - optional; points to other targets that have to be execute as a perequisites in order to execute this target
- description
- if; a condition that has to be true in order to execute the task.
- unless; adds the task to the dependency list of the specified Extension Point.

2.3.3 task

It is a named tag that actually does something. There is about 150 built-in tasks.

2.4 list of tasks

`http://ant.apache.org/manual/index.html`

2.4.1 delete

Deletes from a file system. Attributes:

- dir - **mandatory**.

2.4.2 echo

Prints to stdout.

2.4.3 ivy:retrieve

Executes ivy script. Needs XML namespace declaration in the project tag.

```
<project xmlns:ivy = "antlib:org.apache.ivy.ant" ...
    ...
    <target name = "resolve" ...>
        <ivy:retrieve />
    </target>
    ...
</project>
```

2.4.4 java

Runs java application. Example running executable jar:

```
<java jar = "dir-to/file.jar" fork = "true"/>
```

2.4.5 javac

Attributes:

- srcdir - **mandatory**
- destdir **mandatory**

2.4.6 jar

Packages an application. For startable jar-package additional nested task *manifest* is required Standard usage of *basedir*. Exmaple:

```
<jar destfile = "dir-to/file.jar" basedir = "root-of/java.class">
    <manifest>
        <attribute name = "Main-Class" value = "package.address.of.MainClass"\>
    </manifest>
</jar>
```

2.4.7 javadoc

2.4.8 mkdir

Attributes:

- dir **mandatory**

2.5 ant properties

```
<property name = "prop_name" value = "prop_value"/>
```

- There are 10 predefined, they can be also custom specified.
- **scope** - global.
- **lifecycle** - the lifecycle of immediate surrounding tag. If it is a project - then the property is set for all tasks. If it is within a task - the property is set globally, but just for this particular task.
- **modification and accessibility**
  - they are immutable
  - they value of any property can be used anywhere in the code using following syntax:

```
${prop_name}
```

See: BuildFile.tags.tasks.ivy:retrieve

2.5.1 predefined ant properties

- **ant.file**
  The full location of the build file.
- **ant.version**
  The version of the Apache Ant installation.
- **basedir**
  The basedir of the build, as specified in the basedir attribute of the project element.
- **ant.java.version**
  The version of the JDK that is used by Ant.
- **ant.project.name**
  The name of the project, as specified in the name atrribute of the project element.
- **ant.project.default-target**
  The default target of the current project.
- **ant.project.invoked-targets**
  Comma separated list of the targets that were invoked in the current project.
- **ant.core.lib**
  The full location of the Ant jar file.

  item **ant.home**
  The home directory of Ant installation.
- **ant.library.dir**
  The home directory for Ant library files - typically ANT_HOME/lib folder.

2.6 ant-data types

They are predefined tags. They are kind of built-in services.

2.6.1 fileset

Is used as a filter to include or exclude files that match a particular pattern. Attributes:

- id

- dir - uri of the root folder
- casesensitive

Tags:

- include
- exclude

They have an attribute *name*.

```
<fileset dir = "${some_dir}" casesensitive = "yes">
<include name = "**/*.java"/>
<exclude name = "**/*.class"/>
```

2.6.2 filelist

Explicit list of files.

- no wild cards
- can be applid to non-existing files, too

fileset **filters** files, *filelist* names them explicitly:

```
<filelist id = "list_id" dir = "${base_url}">
<file name = "file1_name"\>
<file name = "file2_name"/>
...
</filelist>
```

2.6.3 patternset

Meta characters:

- ? - exactly one character
- * - zero or more characters
- ** - zero ore more dirs

Tags:

- id - pattern identifier
- include, exclude - pattern definitions
- refid - actual use of pattern definition

Example. First a pattern is specified:

```
<patternset id = "java.src">
<include name = "**/*.java"/>
<exclude name = "**/*.class"/>
</patternset>
```

Then the pattern can be reused by its id:

```
<fileset dir = "${src} casesensitive = "yes">
<patternset refid = "java.src"/>
</fileset>
```

Chapter 3

Property File

By convention it is *build.properties* or *build.properties.ver*, where *ver* specifies a version, like *prod*, *test*. Should be placed at the same location as *build.xml* file. Contains a list of key-value pairs **without quotes!**. Example:

```
# My Project Details
location = local machine
buildversion = 4.2
```

Then we can use properties stored in the property file in *build.xml* by specyfying the value of *file* attribute in an ant property tag:

```
<property file = "build.property.test"/>
```

Comments:

```
# This is a comment!
# ...and this is another comment
location = my local machine
buildversion = 4.2
```

Chapter 4

Resources

- https://www.tutorialspoint.com/ant/index.htm

Part III

Ivy

Chapter 5

Overview

- resolves dependencies
- in XML
- maven2 repository
- *ivy.xml* - conventional config file name
- See: Ant.BuildFile.tags.tasks.ivy:retrieve for syntax how to refer from ant build file.

Chapter 6

Syntax

6.1 structure

Mandatory tags in bold, mandatory attributes in bold-red

- **ivy-module** **version**
  - **info** **organisation** **module**
  - **dependencies**
    - **dependency** **org** **name** **rev**
    - ...

6.2 ivy-module

```
<ivy-module version = "2.0">
   ...
<\ivy-module>
```

Mandatory! It's a preamble that states that this is ivy-file. Attributes:

- version **mandatory!**.

6.3 info

Mandatory. Custom values that identify the project. Attributes **(mandatory!)**:

- organisation - source owner, creator...
- module - usually application name or its part.

```
<info organisation = "msz" module = "my-app"\>
```

6.4 dependencies

Contains set of *dependency* tags.

6.4.1 dependency

Seperate entry for each dependency. Attributes (**mandatory**):

- org
- name
- rev

Value for each attribute can be found here:
`https://mvnrepository.com/`
Information provided as POM. The way to convert:

- groupID - org
- artifactId - name
- version - rev

Chapter 7

Resources

http://ant.apache.org/ivy/history/2.3.0-rc2/tutorial.html

Part IV

Graddle

Chapter 8

Overview

8.1 Project

This interface is the main API you use to interact with Gradle from your build file. From a Project, you have programmatic access to all of Gradle's features. During build initialisation, Gradle assembles a Project object for each project which is to participate in the build. A project is essentially a collection of Task objects. Each task performs some basic piece of work, such as compiling classes, or running unit tests, or zipping up a WAR file

Chapter 9

Resources

Official getting started tutorial:
`https://guides.gradle.org/creating-new-gradle-builds/`