Apache Commons, Logging, Ant

Ulrich.Schaefer@dfki.de

- purpose: open source projects for the WWW
- outstanding project: HTTP server (C++) runs more than 59% of all Web servers (as of December 2010)
- most application-oriented development in Java
- not-for-profit corporation
- membership: by invitation...
- but anyone can participate in projects

Apache Projects around Java and XML

- Maven: Java project management/comprehension tool
- Cocoon: XML-based web application framework
- Forrest: render content using XSLT, XML Schema etc.
- Struts: Java-, control layer based web applications
- XML: mother of
  - Xerces: XML parsers in Java and C++
  - Xalan: XSLT stylesheet processors in Java and C++
  - Axis: SOAP implementation (object communication via HTTP)
  - FOP (XSL formatting objects in Java), ...

- Jakarta: (Smaller) Apache Java projects and tools
  - Lucene: text search engine
  - BSF: scripting within Java (Python, javascript...)
  - Tomcat: servlet container (java server pages)
  - Velocity: template engine (MVC)
  - Commons: useful libraries
  - some projects have now 'graduated': ant, logging, tomcat, lucene

Apache Jakarta Commons

- each tool in a separate jar archive
- Codec: Soundex, Levenshtein, Base64, hexadec.
- Digester: map XML to Java classes and vice versa
- IO: useful tools for I/O, e.g., FileUtils (next slide)
- Net: FTP, NNTP, SMTP, Rlogin, POP3 protocols
- Math: linear algebra, statistics, numerical analysis
- Logging: wrapper for Log4j, JDK14 logging etc.
- ...

org.apache.commons – two examples


```
// transcode

import org.apache.commons.io.FileUtils;

String text = FileUtils.readFileToString
                (file, "iso-8859-1");

FileUtils.writeStringToFile(file, text, "utf-8");


// compute Levenshtein distance

import org.apache.commons.lang.StringUtils;

if ( StringUtils.getLevenshteinDistance("Maier",
    "Meyer") <= 2 ) System.out.println("1
  Meier");
```

logging.apache.org:

log4cxx

log4net

log4perl

log4php

log4j

Logging

- statements inserted in program code describing what's going on, exceptions, warnings, errors
  - debugging/tracing
  - error location, exception handling
  - alerting in servers (e.g., via email, Fax, SMS)
- since JDK 1.4: java.util.logging
  - poor presentation framework
  - laborious, too late...

Good logging APIs

- filter logging statements (e.g., log only severe errors in a production system vs. detailed tracing during development)
- manage output destinations: files, network, database, console, ... ("Appender")
- configure output formats: formatted text , HTML, XML, ... ("Layout")
- don't waste processing time

- installation: download log4j.jar

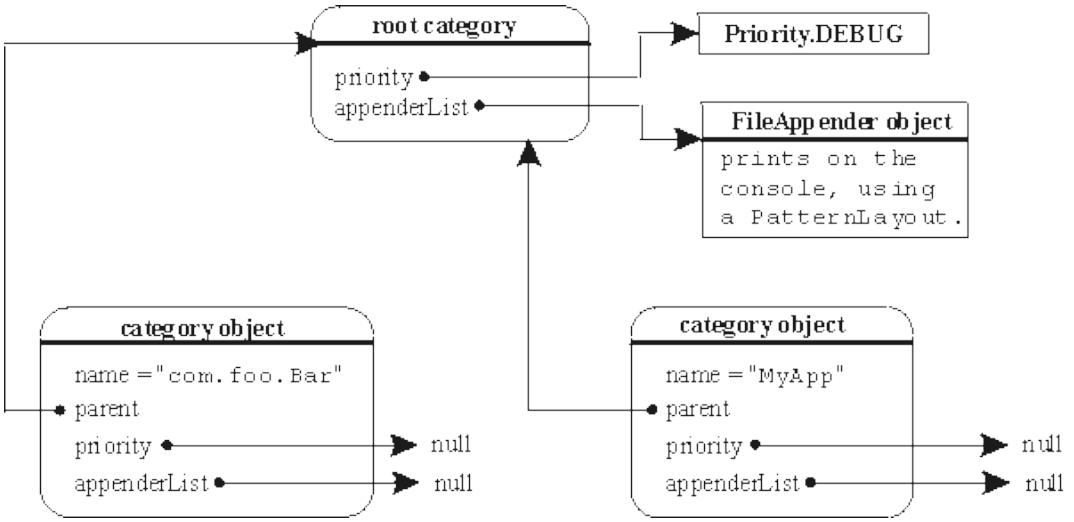
```
import org.apache.log4j.Logger;

static Logger logger = Logger.getLogger
(javaKurs2.vorlesung7.class.getName());

...   // later, log statements in methods:

if (x==null) logger.debug("x has Null value.");

logger.fatal("file system is full.");
```

- run with java -classpath .:log4j.jar myclass -Dlog4j.configuration=file:/path/to/log4j.properties

configuration file

Loggers, LogLevels (priorities), Appenders

- Levels: `debug < info < warn < error < fatal`
- Loggers:



- root logger:

```
log4j.rootLogger=debug, appender1, ...
```

- for a specific package:

```
log4j.logger.package.name=warn, appender2, ...
```

Appenders

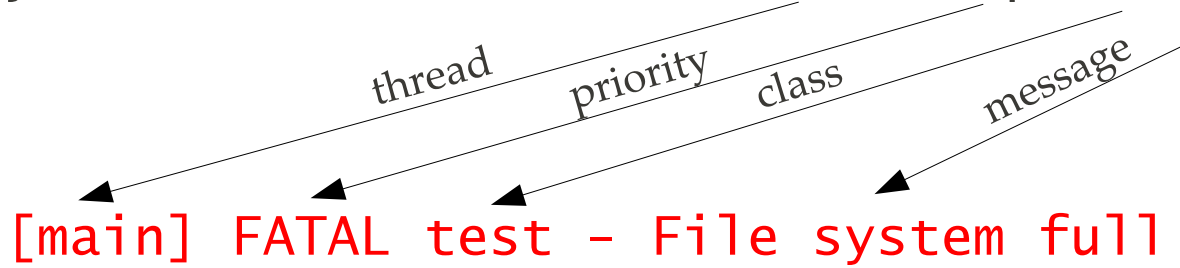
- ConsoleAppender
- FileAppender
  - RollingFileAppender
- SocketAppender
- SMTPAppender
- SyslogAppender
- NTEventLogAppender
- ...

Layouts

- SimpleLayout
- PatternLayout
- HTMLLayout
- XMLLayout

Sample log4j properties file: log to console


```
# Set root logger level to DEBUG and its appender to a1
log4j.rootLogger=DEBUG, a1

# a1 is specified to be a ConsoleAppender
log4j.appender.a1=org.apache.log4j.ConsoleAppender

# a1 uses a PatternLayout
log4j.appender.a1.layout=org.apache.log4j.PatternLayout
log4j.appender.a1.layout.ConversionPattern=[%t] %-5p %c - %m
  %n
```



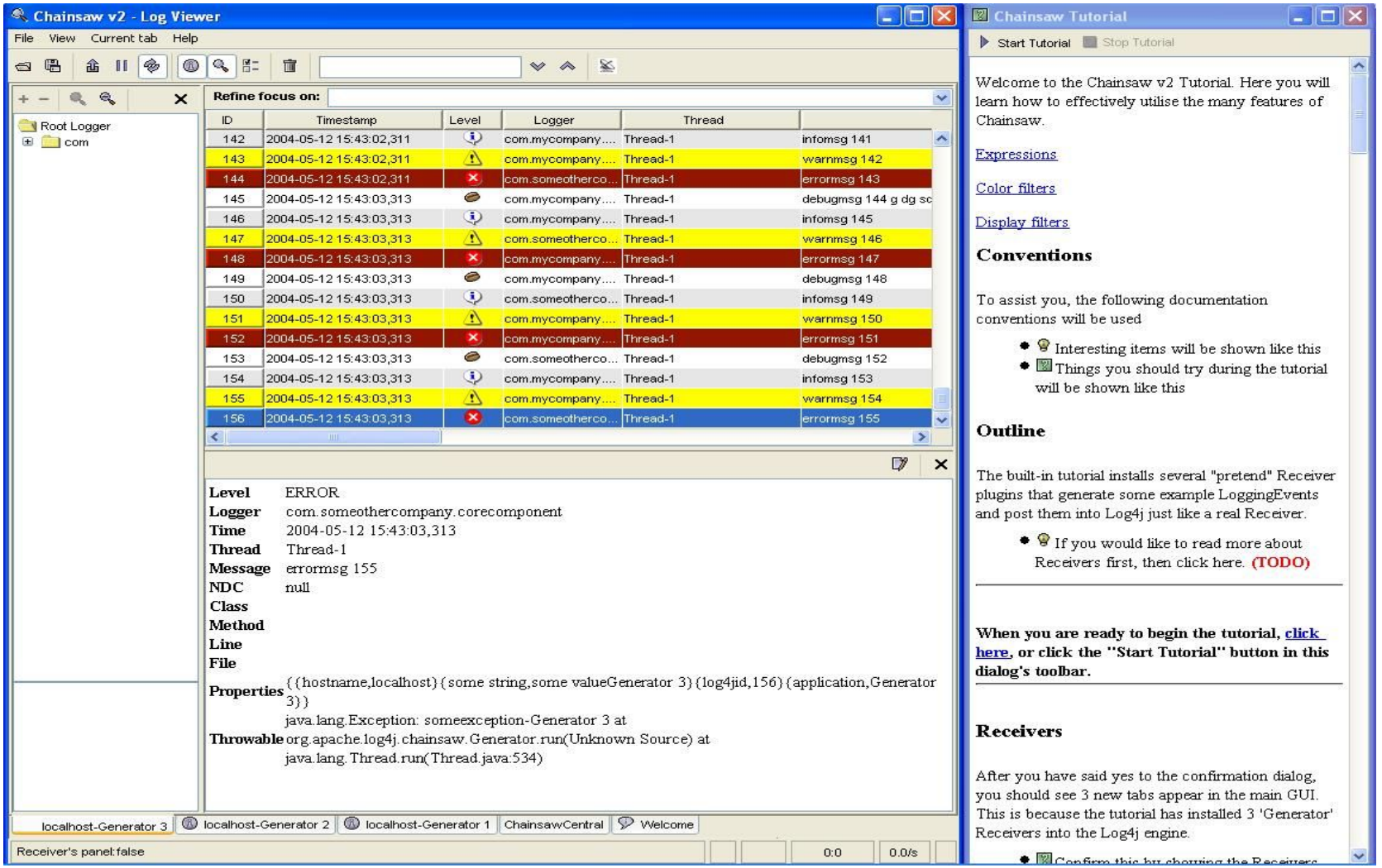
Sample log4j properties file: log to HTML


```
# Log INFO to HTML file
log4j.rootLogger=INFO, h1
log4j.appender.h1=org.apache.log4j.FileAppender
log4j.appender.h1.layout=org.apache.log4j.HTMLLayout
log4j.appender.h1.file=log.html

# Log only messages of level WARN or above in the
# specified package
log4j.logger.javaKurs.uebung10=WARN
```

Sample log4j properties file: log to rolling file


```
# DEBUG log to console and to rolling file
log4j.rootLogger=DEBUG, c1, r1
log4j.appender.c1=org.apache.log4j.ConsoleAppender
log4j.appender.c1.layout=org.apache.log4j.PatternLayout
# Pattern to output the caller's file name and line number
log4j.appender.c1.layout.ConversionPattern=%5p [%t] (%F:%L)
   - %m%n
log4j.appender.r1=org.apache.log4j.RollingFileAppender
log4j.appender.r1.File=example.log
log4j.appender.r1.MaxFileSize=10MB
# Keep one backup file
log4j.appender.r1.MaxBackupIndex=1
log4j.appender.r1.layout=org.apache.log4j.PatternLayout
log4j.appender.r1.layout.ConversionPattern=%p %t %c - %m%n
```

Chainsaw: A log4j GUI for viewing logs



Chainsaw: starting logging

- start chainsaw GUI as server (independently from application):

```
java -classpath log4j.jar -Dchainsaw.port=4445
  org.apache.log4j.chainsaw.Main
```

- log4j configuration (logging application as client):

```
log4j.rootLogger=DEBUG, cs
log4j.appender.cs=org.apache.log4j.net.SocketAppender
log4j.appender.cs.RemoteHost=localhost
log4j.appender.cs.Port=4445
log4j.appender.cs.LocationInfo=true
```

log4j performance

- time-critical parts of log4j have been rewritten many times
- `System.out.println()` may be extremely slow!
- cost of a log request = method invocation + 1 integer comparison: some nanoseconds
- actual logging is about 100 µseconds (format log output and send to appender destination)

Log4j Literature

- Apache logging entry page: http://logging.apache.org/
- Online tutorial (also part of log4j zip distribution) http://logging.apache.org/log4j/docs/manual.html
- Log4j Wiki: http://wiki.apache.org/logging-log4j/Log4JProjectPages

Building projects platform-independently with Apache Ant

http://ant.apache.org/

Ant

- "graduated" Jakarta project
- purpose: automate routine development tasks
- platform-independent (pure Java)
- modular targets define routine build tasks like compiling, cleaning, generating doc, building jar, testing, preparing distribution packages etc.
- ant resolves dependencies between targets and performs only necessary tasks

Ant Installation

- unpack zip file from http://ant.apache.org/bindownload.cgi
- set JAVA_HOME to JDK root, set ANT_HOME to the location where Ant was unzipped
- define shortcut or alias for `${ANT_HOME}/bin/ant` (Unix) or `%ANT_HOME%\bin\ant.bat` (Windows)
- ant is a script calling `java -cp ant-launcher.jar org.apache.tools.ant.launch.Launcher`
- ant is included in Eclipse JDT

C/Unix pendant: make

- `make [-f Makefile] target property=value`
  - only Unix and cygwin, slow on multiple Java tasks
  - heavily relies on other Unix tools

- `ant [-f build.xml] target -Dproperty=value`
  - targets may run in same JVM → faster for Java projects
  - many built-in tools, works platform-independently

Ant syntax

- XML syntax, no namespace required (cf. XSLT)
- **<project>** (root element): each build file defines a single project consisting of one or more targets
- **<targets>** defines a routine, called with <antcall>
- ***<tasks>*** runs built-in command (*task*=javac, copy, ...)
- **<properties>**: variables, parameters (string values)
- **<fileset>**, **<patternset>**: bunches of files, matches

An ant build file defines a project

- default build file is build.xml, otherwise specify using the -f command line option
- use `-Dproperty=value` to define properties
- a default target can be defined for a project
- ant `-projecthelp` lists targets available in project
- Eclipse JDT: select targets graphically, define run configurations etc.

Targets are user-defined routines

- define with `<target name="name">` *body* `</target>`
- a target is composed of tasks that are executed in sequence, execution stops on errors
- call a target from commandline by name or from other targets via `<antcall target="name">`
- specify default target for a project as `<project name="xxx" default="defaulttargetname">`
- dependencies are defined using the depends attribute: `<target name="t3" depends="t1,t2">`

Sample ant build file


```
<?xml version="1.0" encoding="iso-8859-1"?>
<project name="lecture7" default="init"
         basedir="/home/uschaefer/javakurs">
  <description>This project ... </description>
  <property name="src" value="${basedir}/src"/>
  <property name="bin" value="${basedir}/bin"/>
  <target name="init" description="create dirs">
     <mkdir dir="${src}"/>
     <mkdir dir="${bin}"/>
  </target>
</project>
```

task

Tasks

- tasks = predefined commands for use in targets
- take attributes for parameters, e.g.

```
<copy file="text.txt" todir="/some/other/dir"/>
```

- some tasks also take nested (structured) elements:

```
<copy todir="../dest/dir">
  <fileset dir="src">
    <exclude name="**/*.java"/>
  </fileset>
</copy>
```

- it is possible to define additional tasks in Java!

<property>, Properties

- properties have string values
- similar to XSLT variables, values are immutable: whatever sets the value first, wins (∃ no unset)
- define: `<property name="name" value="value"/>`
- use `${name}` to refer to value
- passing via commandline (`-Dprop=value`)
  - overwrites external properties declared in a target with `<property file="build.properties"/>`
  - overwrites globally defined properties in build file

Built-in properties (examples)

- ant-specific:
  - `${basedir}`: basedir (default: same dir as build file)
  - `${ant.file}`: absolute path of build file
  - `${ant.project.name}`: name of project
- all Java system properties such as
  - `${user.name}`: user login name
  - `${os.name}`: name of operating system
  - `${java.version}`: JRE/JDK version number

Passing properties as parameters in <antcall>

- pass parameters in <param> when calling:

```
<antcall target="do_it">
    <param name="when" value="now"/>
<antcall>
```

- in the called target, no declaration is needed as in XSLT's <xsl:param>, just use ${when} to get the value there

<fileset>

- part of many file-based tasks
- attributes e.g. file, dir, includes(file), excludes(file)
- sub-elements: include(sfile), exclude(sfile), patternset
- cf. ant manual/Concepts and Types/Core Types: ∃more...

```
<fileset dir="${src}" includes="**/*.java"/>
<fileset dir="${src}" casesensitive="yes">
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</fileset>
```

Filename, directory patterns

- different from and less powerful than full regex
- * matches zero, one or more characters (like Unix)
- ? matches exactly one character, e.g. P?st
- ** matches zero or more directories, e.g. src/**/*.java
- ** at the end of a path additionally matches any file, e.g. src/**
- / at the end is a shorthand for /**, e.g. test/ = test/**

<patternset>

- defines a group of filename patterns that can later be referenced by the `refid` attribute, e.g. in a classpath, `<fileset>` or directory-based task
- attributes: id, includes(file), excludes(file)
- sub-elements: include(sfile), exclude(sfile), patternset

```
<patternset id="java.files">
  <include name="classes/**/*.class"/>
  <include name="src/**/*.java" if="includesrc"/>
</patternset>
```

Using <patternset>s in <fileset>s

- `<patternset>` with attribute `refid`
- no $ to indicate value (as opposed to variable access!)

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

- Documentation: see 'Concepts and Types' in the Ant manual

<javac> task

- for all following tasks, check Ant manual / AntTasks / Core Tasks
- <javac> compiles java sources (only if necessary)

```
<javac srcdir="${src}"
       destdir="${classes}"
       includes="mypackage/p1/**,mypackage/p2/**"
       excludes="not/this/package/**"
       classpath="additional.jar"
       listfiles="yes" />
```

<java> task

- run java classes either in same Java VM as ant (default) or in new one (fork="yes")

```
<java fork="no" classname="my.testclass">
  <classpath>
    <pathelement location="${classes}"
    <pathelement location="${runjar.dir}/
                           log4j-1.2.8.jar"/>
  </classpath>
  <arg value="4711"/>
</java>
```

<javadoc> task

- generates Javadoc in specified destination directory

```
<javadoc packagenames="de.usaar.coli.*"
         sourcepath="${src}"
         destdir="${apidoc}"
         windowtitle="Saarland University"
         version="yes">
```

<exec> task

- executes external command – this is platform-dependent!

```
<exec dir="${src}" executable="cmd.exe"
      os="Windows XP" output="dir.txt">
  <arg line="/c dir"/>
</exec>

<exec dir="${src}" executable="ls"
      os="Linux" output="dir.txt">
  <arg line="-l"/>
</exec>
```

<xslt> task (synonymous with <style>)

- calls XSLT transformer (JAXP/Xalan in JDK1.4)
- example:

```
<xslt    in="inputfile.xml"
        out="outputfile.html"
      style="transform.xsl">
  <param name="xslparam1" expression="42"/>
  <outputproperty name="method" value="html"/>
</xslt>
```

- Example (cf. JavaCC lecture; requires JavaCC)

```
<javacc
  javacchome="c:/program files/JavaCC"
  target="src/Parser.jj"
  outputdirectory="javaccgenerated"
  static="true"          // STATIC option
  unicodeinput="true"    // UNICODE_INPUT option
/>
```

File and directory operations

- `<mkdir>` create directory
- `<copy>` copy files, directories
- `<move>` move files, directories
- `<delete>` remove files, directories
- `<loadfile>` read file into a property
- `<dirname>`, `<basename>` compute paths
- `<(un)jar>`, `<(un)tar>`, `<(un)zip>`, `<g(un)zip>`, `<b(un)zip2>` (un)compress files
- `<apply>` apply executable to filesets etc.

Further interesting targets

- `<echo>` print to console or to a file
- `<mail>` send email via SMTP
- `<get>` fetch files via http
- `<concat>` generate, append or list files
- `<tempfile>` generate temporary files
- `<input>` prompt for user input
- `<sql>` send SQL statements via JDBC

Example: parse config file, concatenate files

Task:

- treat lines in a configuration file as filenames (except comment lines starting with #)
- concatenate contents of all files into a new file

Solution:

- new target based on built-in tasks/concepts: `<loadfile>`, `<filterchain>`, `<fileset>`, `<concat>`

Sample config (input) file for the target


```
##################################################################
# Configuration file for the extended gazetteer (output.sgr)
##################################################################
###AREASCIENCE.GAZ is necessary for "en/en_nobel_domain.sgr"
#area_science.gaz
###CELL_PHONE.GAZ is necessary for "de/product.sgr"
### "en/en_product_name.sgr"
cell_phones.gaz
###CARS.GAZ is necessary for "de/product.sgr"
cars.gaz
###DANGEROUS_PLACES.GAZ
#dangerous_places.gaz
###GIVEN_NAME.GAZ is necessary for "{de,fr,en,nl}/person_names.sgr"
given_name.gaz
###LOCATION.GAZ is necessary for "{de,fr,en,nl}/location.sgr"
location.gaz
###NATIONALITY.GAZ is necessary for "{de,fr,en,nl}/organization.sgr"
nationality.gaz
###NUMEX.GAZ is necessary for "{de,fr,en,nl}/currency.sgr"
numex.gaz
###ORGANIZATION.GAZ is necessary for "{de,fr,en,nl}/organization.sgr"
organization.gaz ...
```

Target definition


```
<target name="concat-files-in-cfg-file" depends="init">
  <property name="gazetteer_config" value="gaz.cfg" /> ...

<!-- Extracts the files listed in ${gazetteer_config} into a property -->

  <loadfile property="gaz_files" srcFile="${gazetteer_config}">
    <filterchain>
      <striplinecomments>   <!-- strip comments -->
        <comment value="#" />
      </striplinecomments>
      <striplinebreaks />   <!-- strip line breaks -->
    </filterchain>
  </loadfile>

  <echo message="input files are ${gaz_files}" />

<!-- The contents of all files are written to file "${gazetteer_entries}" -->

  <concat destfile="${gazetteer_entries}">
    <fileset dir="${gaz.srcdir}/" includes="${gaz_files}" />
  </concat>

</target>
```

Control flow

- `<parallel>` execute contained tasks concurrently
- `<sequential>` (implicit), e.g.

```
<parallel>
  <wlrun ... >
  <sequential>
    <sleep seconds="30"/>
    <junit ... >
    <wlstop/>
  </sequential>
</parallel>
```

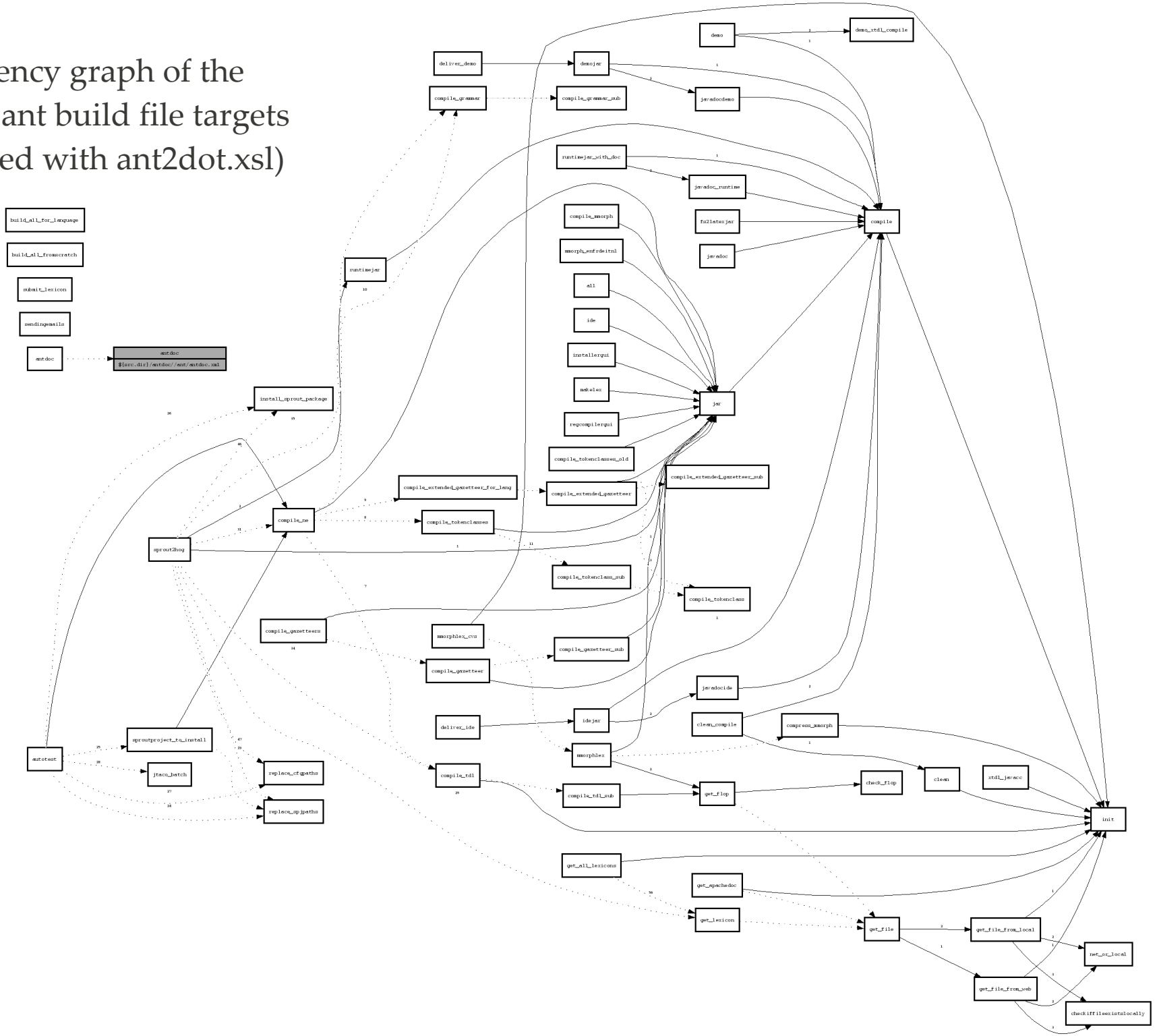
- `<waitfor>` wait inside <parallel>

```
<waitfor maxwait="30"
         maxwaitunit="second">
  <available file="errors.log"/>
</waitfor>
```

- Condition expressions:
  - `<and>, <or>, <not>, <available>, <uptodate>, <http>, <socket>, <os>, <equals>, <isset>, <contains>, <istrue>, <isfalse>, <filesmatch>, <checksum>`

- extensions are simply copied to ant lib directory
- ant contrib:
  - <if>, C compiler extensions, ...
- ant2dot.xsl generates visual dependency graph: http://ant2dot.sourceforge.net (example next slide)
- antdoc generates Javadoc-like descriptions of ant build files

Dependency graph of the SProUT ant build file targets (generated with ant2dot.xsl)



Ant Literature

- http://ant.apache.org
- zip archive including docs and runtime jars:
  http://ant.apache.org/bindownload.cgi
- comprehensive manual with examples:
  http://ant.apache.org/manual/index.html
- lots of articles, tutorials, etc:
  http://ant.apache.org/resources.html