

TUDIIR Toolkit Overview

David Urbansky, Klemens Muthmann

TU Dresden, Department of Systems Engineering, Chair Computer Networks, IIR Group, Germany

June 22, 2010

Abstract

This document explains basic functionalities of the TUDIIR Toolkit. Its intention is to explain (new) developers how to work with the toolkit and how to extend its capabilities.

Contents

1	IIR and this Toolkit	2
2	License	2
3	Toolkit Structure	2
3.1	Config Folder	3
3.1.1	apikeys.conf	3
3.1.2	classification.conf	3
3.1.3	crawler.conf	3
3.1.4	db.conf	4
3.1.5	general.conf	4
3.2	Data Folder	4
3.2.1	knowledgeBase	4
3.2.2	models	4
3.2.3	test	4
3.3	Documentation Folder	5
3.4	Exe Folder	5
3.5	Libs Folder	5
3.6	Src Folder	5
4	Management of the code base	5
4.1	Building the toolkit using Apache Maven	5
4.2	Regularly builds and tests using Hudson CI	6
4.2.1	The Continuous Integration Game	7
4.3	Reporting issues using Mantis	7
5	Conventions	7
5.1	Coding Standards	7
5.2	Tests	8
6	Example Usages	8
6.1	Hello Toolkit - Your first application using the TUDIIR Toolkit	8

6.2	Source Retriever	21
6.2.1	Basic Features	21
6.2.2	How To	21
6.3	Web Crawler	22
6.3.1	Basic Features	22
6.3.2	How To	22
6.4	FAQ Extractor	23
6.5	Fact Extraction	23
6.6	Web Page Classification	24
6.6.1	Basic Features	24
6.6.2	How To	24
6.7	Helpers	26
7	Reference Libraries	27
8	History	28

1 IIR and this Toolkit

Internet Information Retrieval (IIR) is a research domain in computer science that is concerned with the retrieval, extraction, classification, and presentation of information from the Internet. This toolkit provides functionality which is often needed to perform IIR tasks such as crawling, classification, and extraction of various information types.

2 License

The complete source code is licensed under the Apache License 2.0. All source files should include the following license snippet at the very top.

```
Copyright 2010 David Urbansky, Klemens Muthmann
Licensed under the Apache License, Version 2.0 (the "License"); you may not
use this file except in compliance with the License. You may obtain a copy of
the License at
```

```
http://www.apache.org/licenses/LICENSE-2.0
```

```
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

3 Toolkit Structure

The TUDIIR Toolkit is managed using subversion and is located in our SVN¹. The folder structure looks as follows.

```
toolkit
|- config
```

¹<https://141.76.40.86/svn-students/iircommon/toolkit/>

```

|- data
  |- knowledgeBase
  |- models
  |- test
|- documentation
  |- handout
  |- javadoc
  |- other
|- exe
|- libs
|- src
  |- main
    |- java
  |- test
    |- java
|- dev

```

3.1 Config Folder

The config folder contains configuration files for several components of the toolkit. The files are explained in the following sections.

3.1.1 apikeys.conf

The api keys that are used by the toolkit components are specified here. You may need to apply for API keys at the provider's page.

```

yahoo.api.key =
yahoo_boss.api.key =
hakia.api.key =
google.api.key =
bing.api.key =

```

3.1.2 classification.conf

Classification settings for the `tud.iir.classification.page.ClassifierManager`.

```

# percentage of the training/testing file to use as training data
page.trainingPercentage = 80

# create dictionary on the fly (lowers memory consumption but is slower)
page.createDictionaryIteratively = false

# alternative algorithm for n-gram finding (lowers memory consumption but is slower)
page.createDictionaryNGramSearchMode = true

# index type of the classifier:
# 1: use database with single table (fast but not normalized and more disk space needed)
# 2: use database with 3 tables (normalized and less disk space needed but slightly slower)
# 3: use lucene index on disk (slow)
page.dictionaryClassifierIndexType = 1

```

3.1.3 crawler.conf

Crawler settings for the `tud.iir.web.Crawler`. All settings can be set in Java code as well.

```
# maximum number of threads during crawling
maxThreads = 10

# stop after x pages have been crawled, default is -1 and means unlimited
stopCount = -1

# whether to crawl within a certain domain, default is true
inDomain = true

# whether to crawl outside of current domain, default is true
outDomain = true

# number of request before switching to another proxy, default is -1 and means never switch
switchProxyRequests = -1

# list of proxies to choose from
proxyList = 83.244.106.73:8080
proxyList = 83.244.106.73:80
proxyList = 67.159.31.22:8080
```

3.1.4 db.conf

Database settings for the tud.iir.persistence.DatabaseManager.

```
db.type = mysql
db.driver = com.mysql.jdbc.Driver
db.host = localhost
db.port = 3306
db.name = toolkitdb
db.username = root
db.password = rootpass
```

3.1.5 general.conf

General settings used by the tud.iir.control.Controller.

3.2 Data Folder

The data folder contains files that are used during runtime of several components.

3.2.1 knowledgeBase

The knowledge base folder contains the OWL ontology files used for the extraction tasks in the tud.iir.extraction package.

3.2.2 models

The models folder contains learned models that can be reused.

3.2.3 test

The test folder contains data that is used for running jUnit tests.

3.3 Documentation Folder

The documentation folder contains help files to understand the toolkit. This very document is located in the handout folder and a Javadoc can be found there too.

3.4 Exe Folder

The exe folder contains all runnable jar files in separate folders including a sample script to run the program and a readme.txt that explains the run options.

3.5 Libs Folder

The libs folder contains all referenced libs used by the toolkit.

3.6 Src Folder

The src folder contains all source files of the toolkit. You may need to put the log4j.properties file here in order to use custom logging settings.

4 Management of the code base

The TUDIIR toolkit is managed using Subversion (SVN) (see 3). It is build and tested automatically on a weekly basis using Apache Maven and Hudson CI. Bugs might be reported using the Mantis bugtracker. The project encoding must be set to UTF-8. To support unavailable libraries we manage our own Maven repository using Sonatype Nexus. How to use these components is explained in the next sections.

At the moment each of the systems manages its own user base so you need to register with each one individually. When you start working with the toolkit you might consider sending an E-Mail to the administrator to get access to Mantis, Nexus and Hudson. Provide your name, the name of your advisor and the reason why you need to work with the toolkit and you will get your logins.

4.1 Building the toolkit using Apache Maven

Apache **Maven needs to be installed** before building the toolkit. How to do this is explained here: Maven installation. There is one necessary manual step before you can start building the toolkit. You need to add our Nexus repository to your local maven settings. Do this by **locating or creating the settings.xml file in your local home folder: %YOUR_HOME_FOLDER%/.m2/settings.xml** and adding the following content:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
    http://maven.apache.org/xsd/settings-1.0.0.xsd">
<mirrors>
  <mirror>
    <!--This sends everything else to /public -->
    <id>nexus</id>
    <mirrorOf>*</mirrorOf>
    <url>http://www.effingo.de/nexus/content/groups/public</url>
  </mirror>
</mirrors>
<profiles>
  <profile>
    <id>nexus</id>
```

```

<!--Enable snapshots for the built in central repo to direct -->
<!--all requests to nexus via the mirror -->
<repositories>
  <repository>
    <id>central</id>
    <url>http://central</url>
    <releases><enabled>true</enabled></releases>
    <snapshots><enabled>true</enabled></snapshots>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>central</id>
    <url>http://central</url>
    <releases><enabled>true</enabled></releases>
    <snapshots><enabled>true</enabled></snapshots>
  </pluginRepository>
</pluginRepositories>
</profile>
</profiles>
<activeProfiles>
  <!--make the profile active all the time -->
  <activeProfile>nexus</activeProfile>
</activeProfiles>
<servers>
  <server>
    <id>nexus</id>
    <username>your-username</username>
    <password>your-password</password>
    <filePermissions>664</filePermissions>
    <directoryPermissions>775</directoryPermissions>
    <configuration></configuration>
  </server>
</servers>
</settings>

```

You need to set your username and your password in the server section. This can be obtained by sending an E-Mail to klemens.muthmann@tu-dresden.de stating who is your advisor and why you need to work with the toolkit. After completing the installation perform the following steps to build the toolkit using Maven:

1. Check out the code from SVN!
2. Open your favorite command line.
3. Change to the toolkits root folder.
4. Type `mvn clean install` to start the build process.

There also is an Eclipse plugin, that allows you to issue maven build from within Eclipse. It is quite beta so be careful with it.

4.2 Regularly builds and tests using Hudson CI

Currently the toolkit is build automatically every week using Hudson CI. Be careful to check in only working code or Hudson will send you and your advisor an E-Mail about broken code. If this happens try to fix your code as fast as possible and check in again. You can get details about the problem by logging into Hudson. You can get a login by sending an E-Mail stating your advisor and why you need to work with the toolkit to klemens.muthmann@tu-dresden.de

4.2.1 The Continuous Integration Game

Hudson supports a game that rewards people committing code that improves the toolkit and punishing people breaking it. The leader (the person having the most points) is highly valued by the toolkit committers community. The rules for the game are explained in detail in the next section.

Rules The rules of the game are:

- -10 points for breaking a build
- 0 points for breaking a build that already was broken
- +1 points for doing a build with no failures (unstable builds gives no points)
- -1 points for each new test failures
- +1 points for each new test that passes
- Adding/removing a HIGH priority PMD warning = -5/+5. Adding/removing a MEDIUM priority PMD warning = -3/+3. Adding/removing a LOW priority PMD warning = -1/+1.
- Adding/removing a violation = -1/+1. Adding/removing a duplication violation = +5/-5.
- Adding/removing a HIGH priority findbugs warning = -5/+5. Adding/removing a MEDIUM priority findbugs warning = -3/+3. Adding/removing a LOW priority findbugs warning = -1/+1
- Adding/removing a compiler warning = -1/+1.
- Checkstyle Plugin. Adding/removing a checkstyle warning = -1/+1.

4.3 Reporting issues using Mantis

To report issues with the toolkit or to view issues assigned to you, you need to register with Mantis. To do this send an E-Mail to the administrator at klemens.muthmann@tu-dresden.de and provide your name, your advisor and the reason why you are working on the toolkit (i.e. thesis topic or research fellow). You will receive an E-Mail (not automatically so it might take some time) providing a link where you can set your password. If you successfully set a password for your account you can login on: Mantis.

5 Conventions

5.1 Coding Standards

To keep the code readable and easy to understand for other developers, we use the following coding guidelines.

1. All text is written in (American) English (color instead of colour).
2. Variables and method names should be camel cased and not abbreviated (*computeAverage* instead of *comp_avg*).
3. There must be a space after a comma.
4. There must be a line break after opening { braces.
5. Static fields must be all uppercase. There should be an _ for longer names (*STATIC_FIELD*).
6. Each class must have a comment including the author name.
7. Methods with very simple, short code do not need to have comments (getters and setters) all other methods should have an explaining comment with @param explanation and @return.

8. Avoid assignments (=) inside if and while conditions.
9. Statements after conditions should always be in braces ({}).

The following listing shows an example class with applied coding standards. Please also have a look at [15] for a quick overview of best practices.

```

1 /**
2  * This is just an example class.
3  * It is here to show the coding guidelines.
4  *
5  * @author Forename Name
6  */
7 public class ExampleClass implements Example {
8
9     // this field holds all kinds of brackets
10    private static final char[] BRACKET_LIST = {'(', ')'};
11
12    /**
13     * This is just an example method.
14     *
15     * @param timeString A string with a time.
16     * @return True if no error occurred, false otherwise.
17     */
18    public boolean computeAverageTime(String timeString) {
19        if (hours < 24 && minutes < 60 && seconds < 60) {
20            return true;
21        } else {
22            return false;
23        }
24    }
25 }

```

5.2 Tests

To guarantee that all components work as expected, we use jUnit tests. Before major check-ins to the repository, all jUnit Tests must run successfully. Run the `tudi.ir.control.AllTests.java` to make sure all components work correctly. After finishing a new component, new testing code must be written.

6 Example Usages

Learning to use a toolkit often does not work by reading the Javadoc. Therefore, we provide a simple example how to setup your first project using the toolkit, eclipse and maven. Afterwards we provide a short overview of some important modules and how you can use them. All examples are intended as entry points to facilitate the first use of the toolkit. Many components are too mighty to be explained exhaustively in small code snippets, make sure to read the Javadoc and more importantly the actual code for further information.

6.1 Hello Toolkit - Your first application using the TUDIIR Toolkit

Project creation: Create a new Maven Project using the New Project wizard of Eclipse (See Fig. 1, 2 and 3).

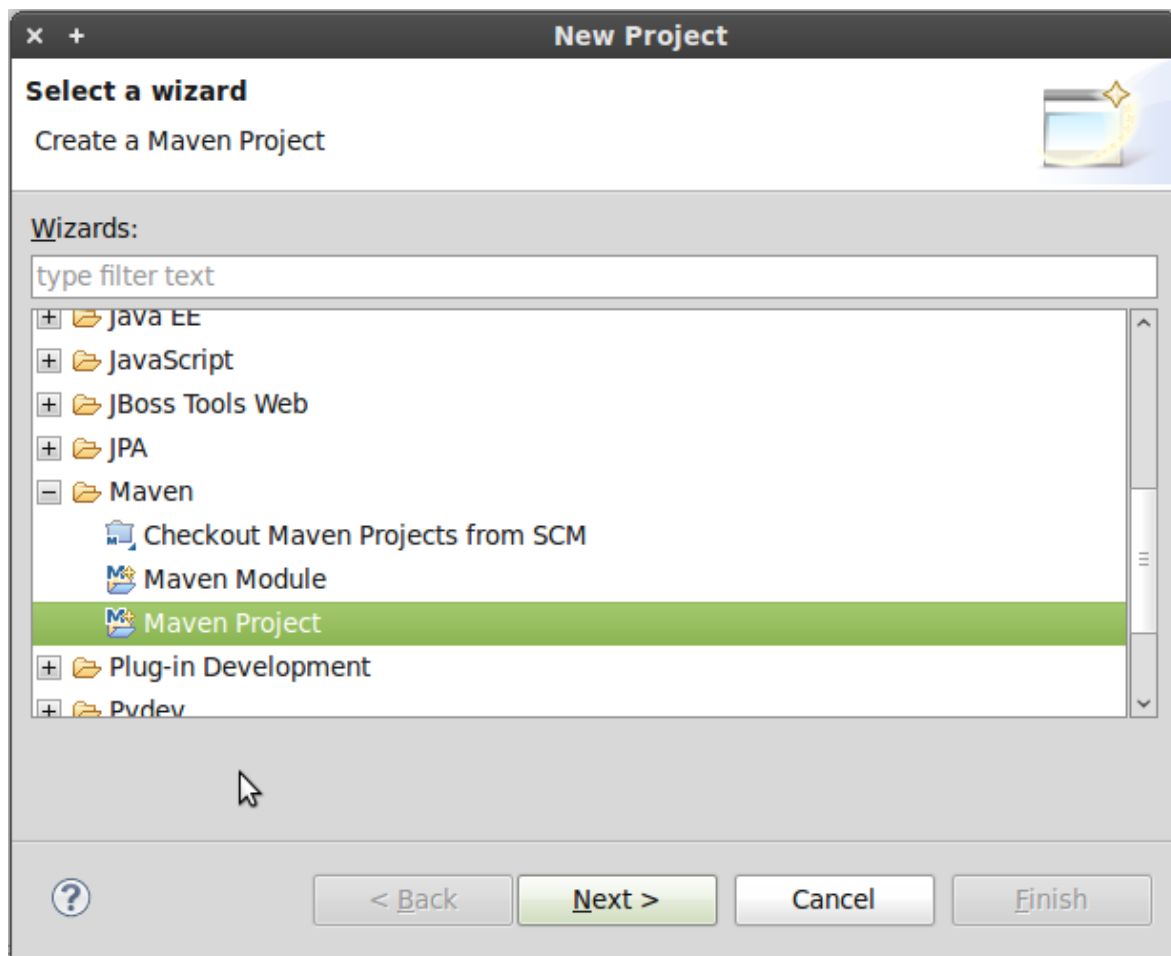


Figure 1: Create a new Maven Project.

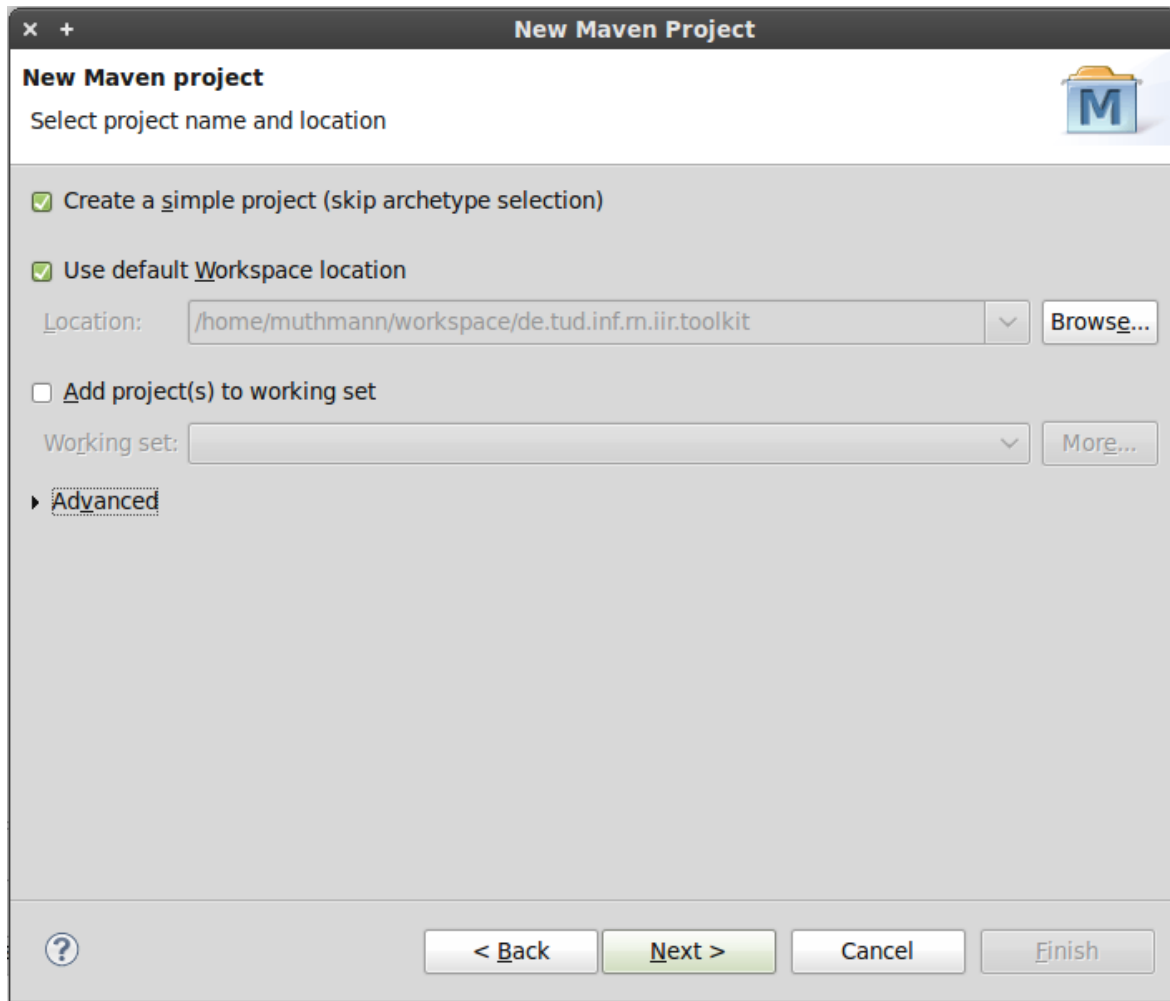


Figure 2: Choose to create a simple Maven Project.

New Maven Project
Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

► **Advanced**

Figure 3: Enter detail information about your new Maven Project

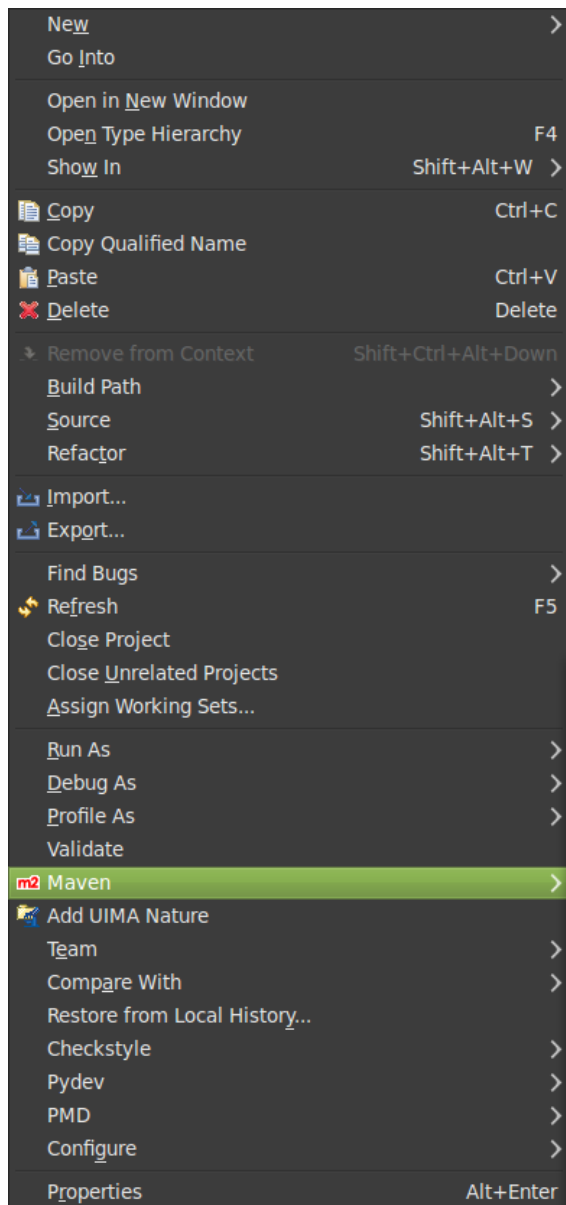


Figure 4: Maven Project Context Menu. Choose *Maven*

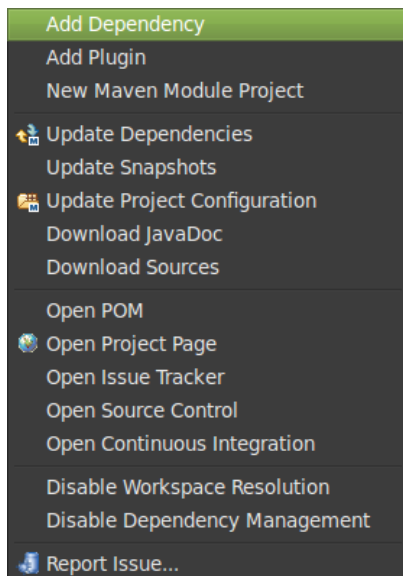


Figure 5: Create a new Maven project

Adding the toolkit dependency to the project: Right click on the new project. In the context menu that appears choose *Maven* → *Add Dependency* (See Fig. 4 and 5). A search interface appears (See 6). If you followed the steps in Section 4.1 and installed the toolkit to your local Maven repository, you can type *toolkit* in the search interface (See Fig. 7 and add the dependency with a double click on the *de.tud.inf.rn.iir.toolkit* entry.

Note: If you need to add further dependencies in the future you can use the same steps. The Maven plugin adds the dependency to your pom.xml file, which should look like Fig. 8.

Configuring your project Since Maven by default still uses Java 1.4 (very conservative), but the toolkit depends on Java 1.6 (very visionary) you need to configure the Maven Java compiler plugin to use Java 1.6. This is shown in Fig. 9. The same step is necessary to tell Eclipse that it should use Java 1.6 instead of 1.4. For this purpose open the project properties via the context menu for example. In the tree to the left choose *Java Compiler* and change all three entries to 1.6. This is shown in Fig. 10 and Fig. 11.

Writing your first toolkit code Now you can start to write your first code. Your project will already contain the default Maven directory structure. Do not change this structure since Maven depends on it². As usual we will start with a very simple "Hello World" application. Create a new package *de.tud.inf.rn.iir.toolkit* and a new class *HelloToolkit* containing a *main* method. In this main method you can add actual toolkit code. We used the first example as described in Section 6.2.2 and search Bing for the term "Hello Toolkit". The example is also shown in Fig. 12. For the code to work you need three additional files that are already in the config folder in the toolkit project. These are "crawler.conf", "apikey.conf" and "feeds.conf". You need to copy them to *src/main/resources/config*. The folder *src/main/resources* usually contains all resources that are not Java files but are required by your code. All files are shown in Fig. 13, 14 and 15. Fig. 16 shows the final directory and file structure of your project. With the structure from Fig. 16 you can run your project. Just open your projects context menu again, choose *Run As* → *Maven Clean*, then open context menu again choose *Run As* → *Maven Install* and finally choose *Run As* → *Java Application*.

²Of course you can change Mavens behaviour via the pom.xml but this requires additional configuration not covered by this document. Refer to the Maven documentation under <http://maven.apache.org/> for further information.

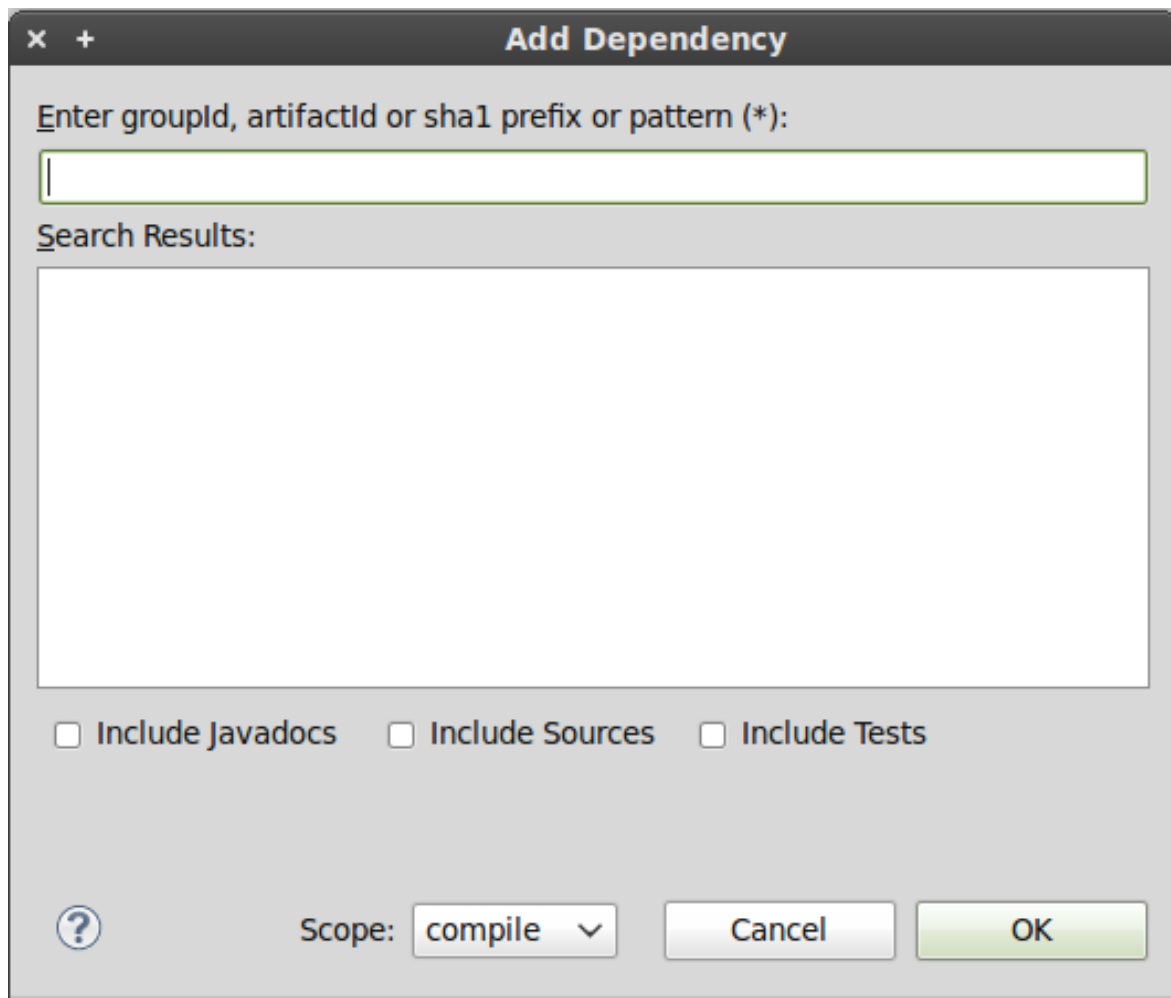
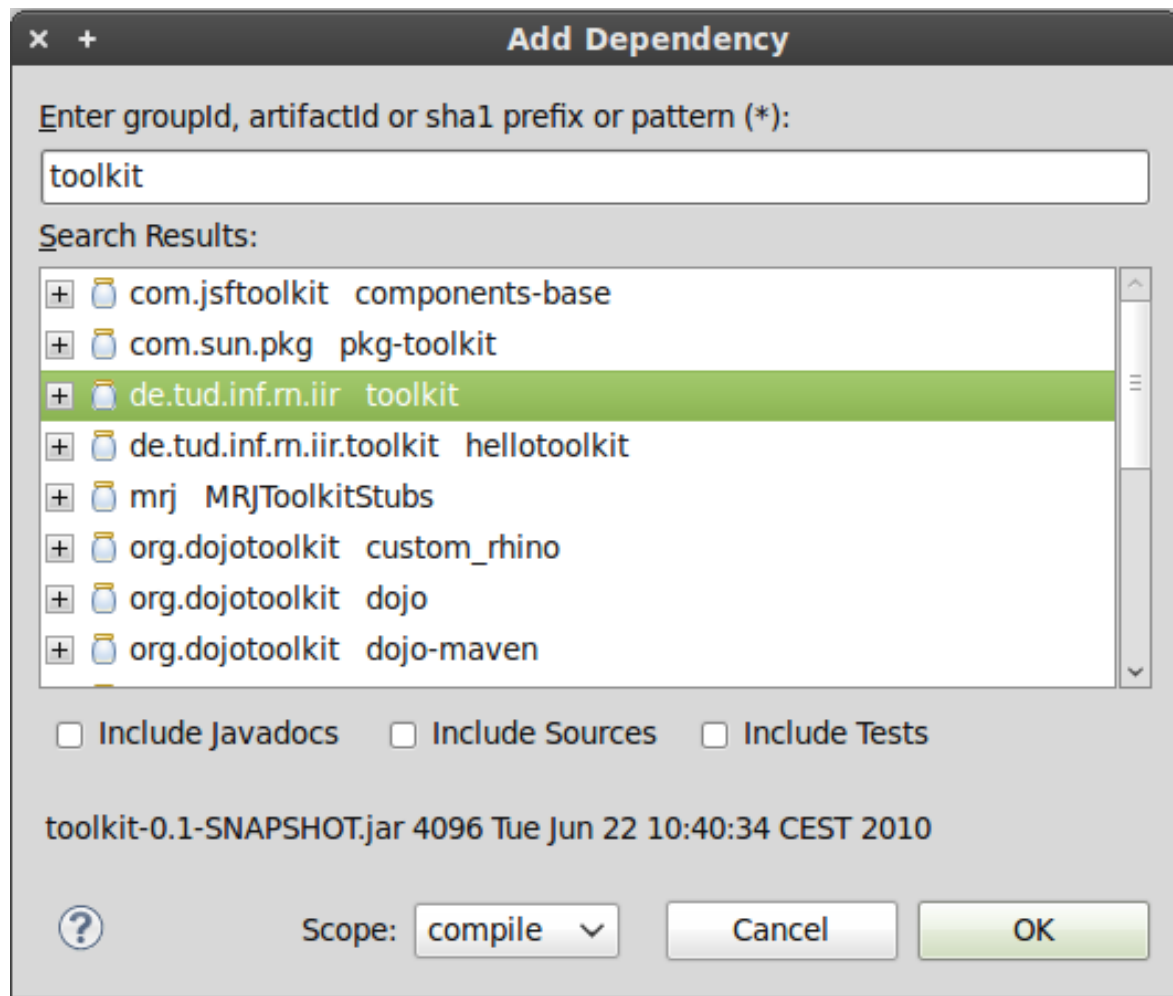


Figure 6: Search interface for Maven dependencies.

Figure 7: Search results for *toolkit*

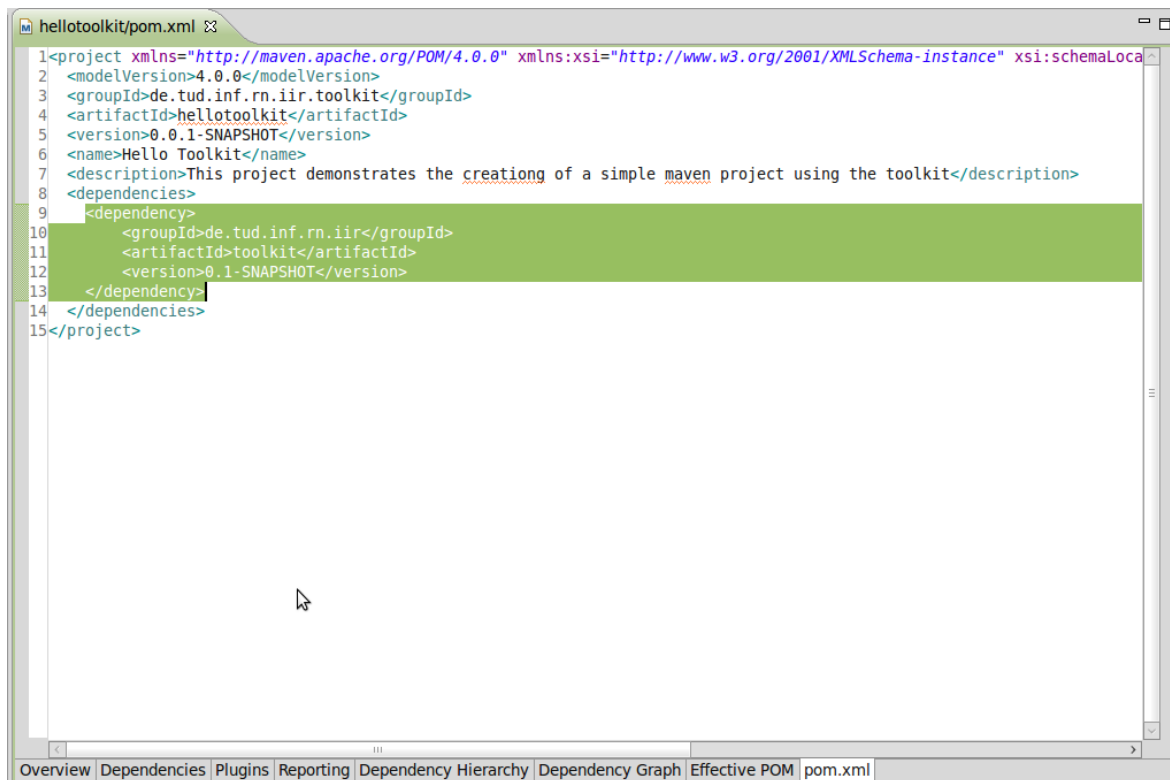


Figure 8: POM with added dependency on the TUD IIR Toolkit.

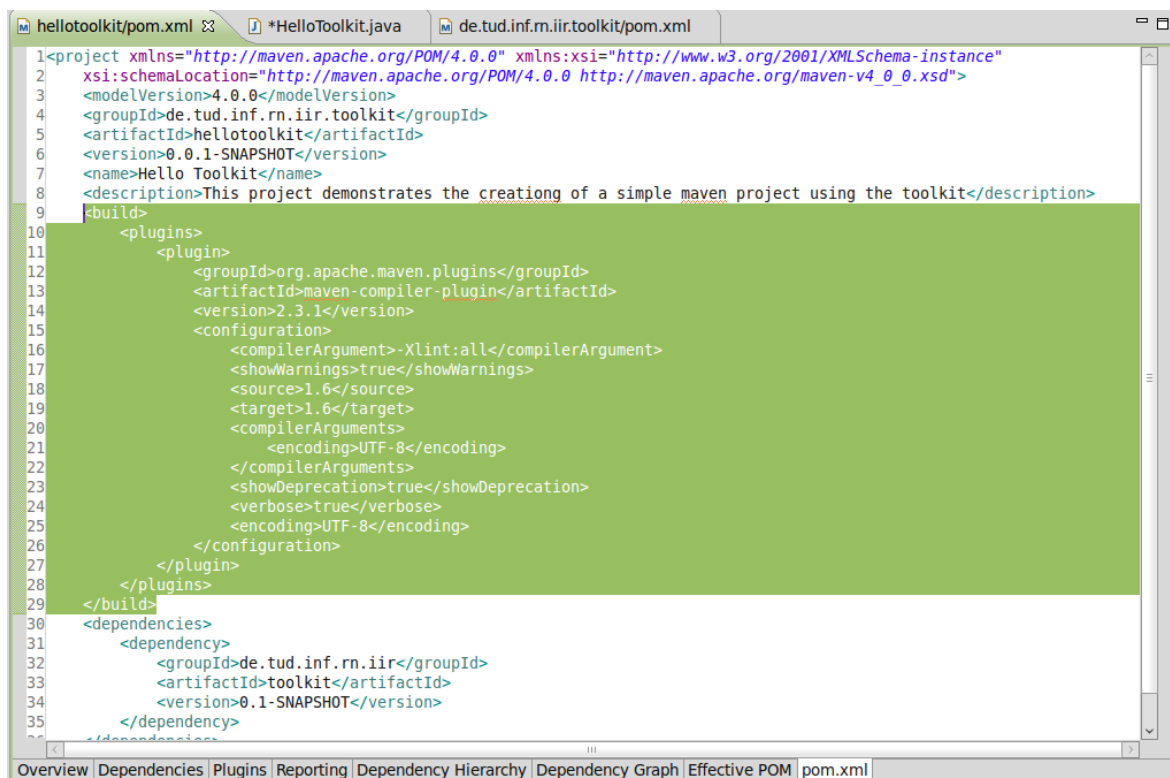


Figure 9: Configure the Maven Java compiler to use Java 1.6 instead of 1.4

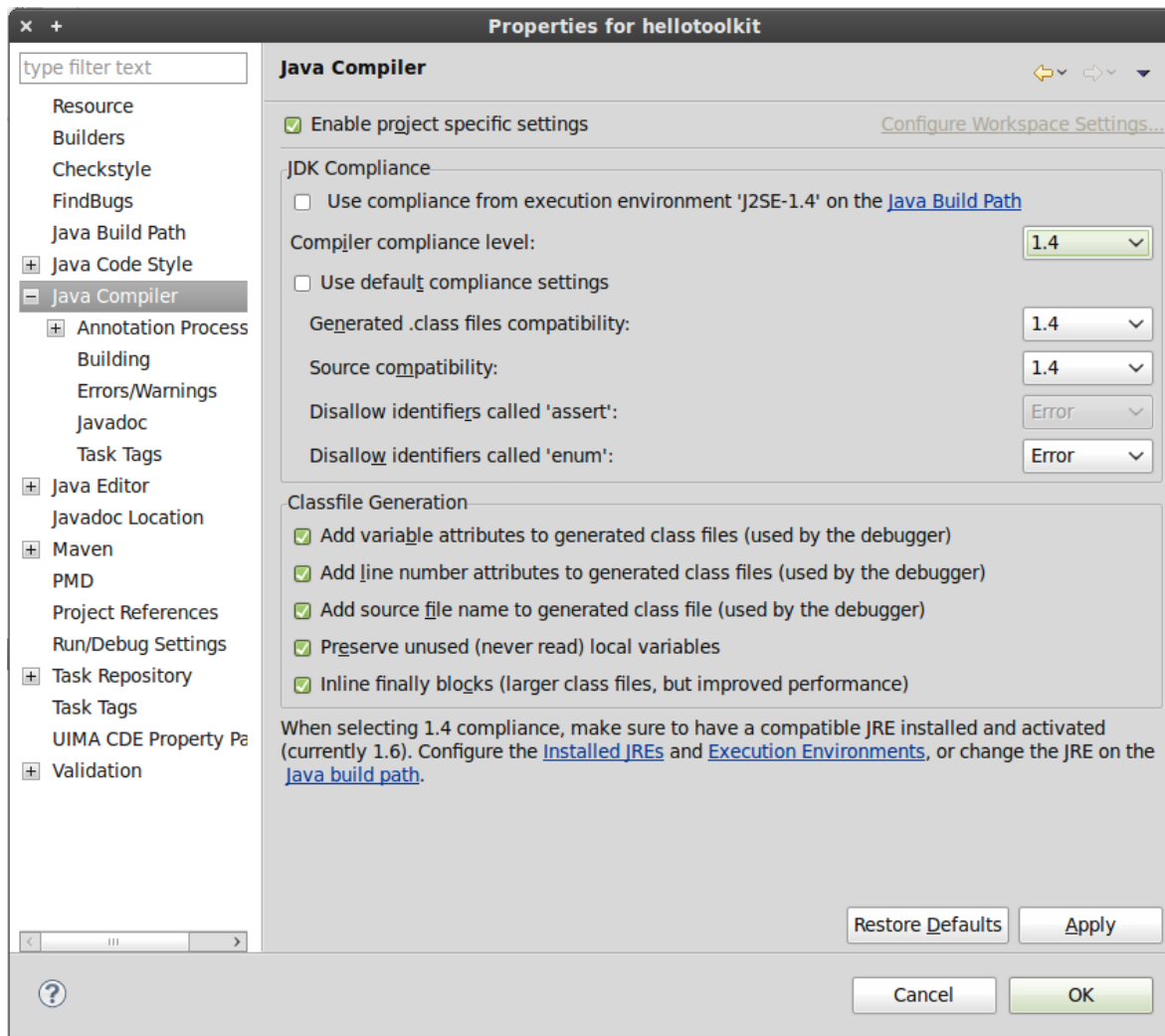


Figure 10: Eclipse uses Java 1.4 by default for Maven projects

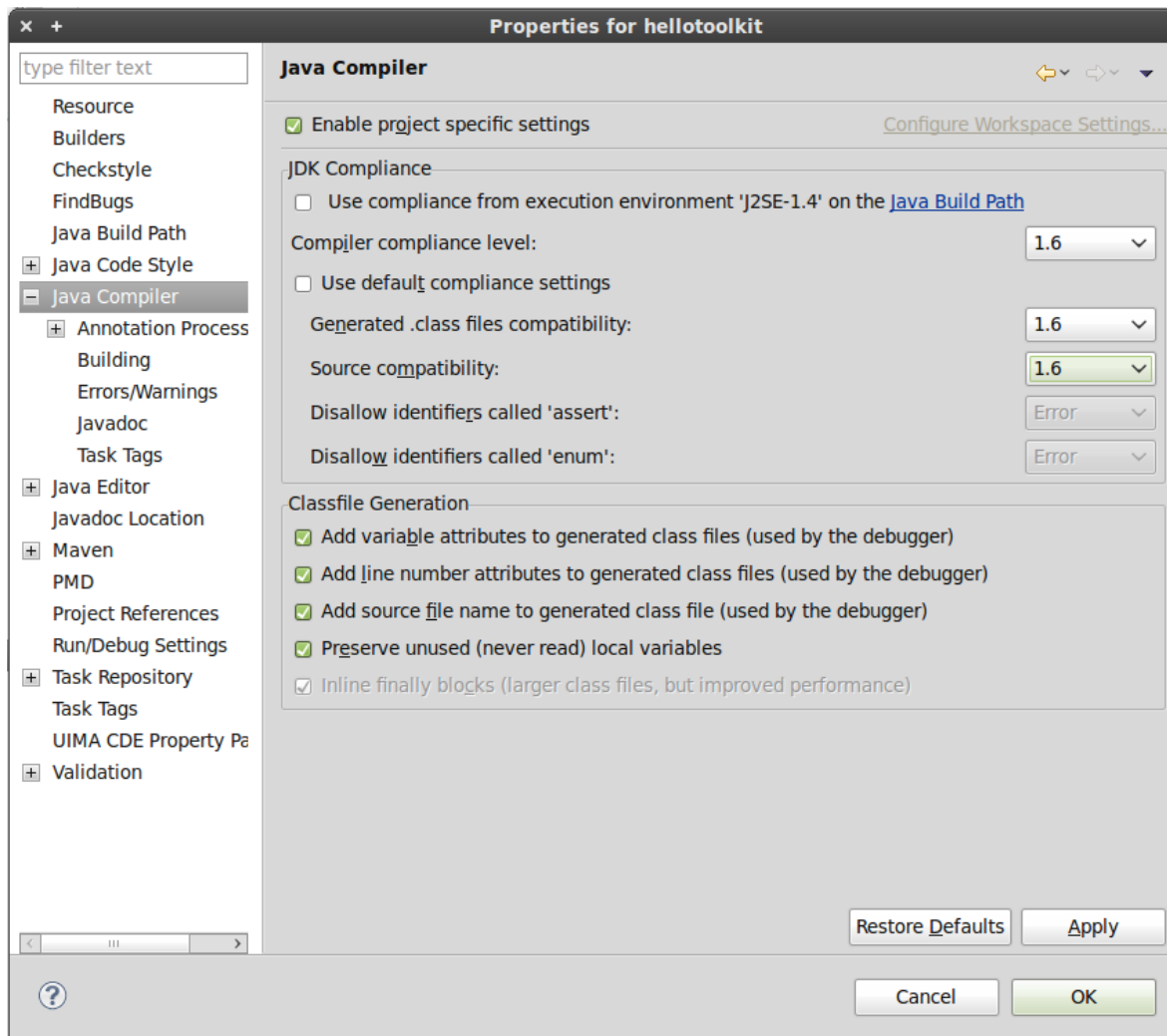
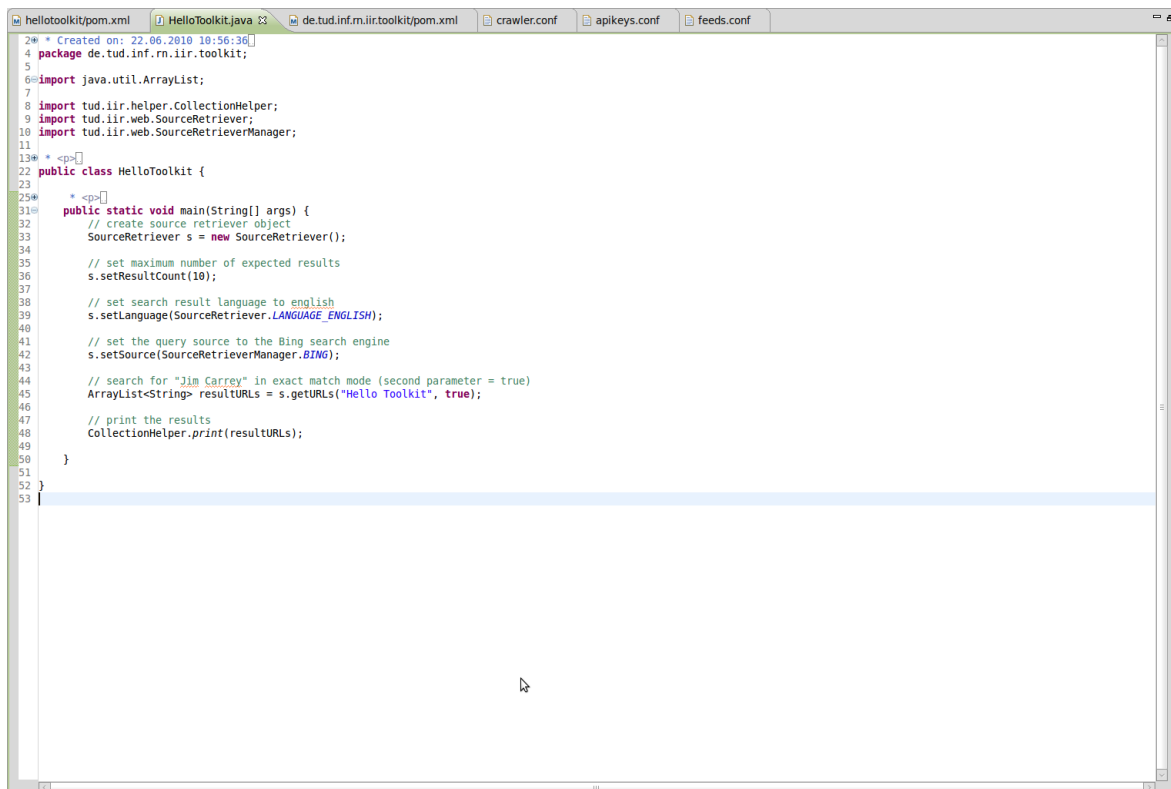


Figure 11: configure Eclipse to use Java 1.6

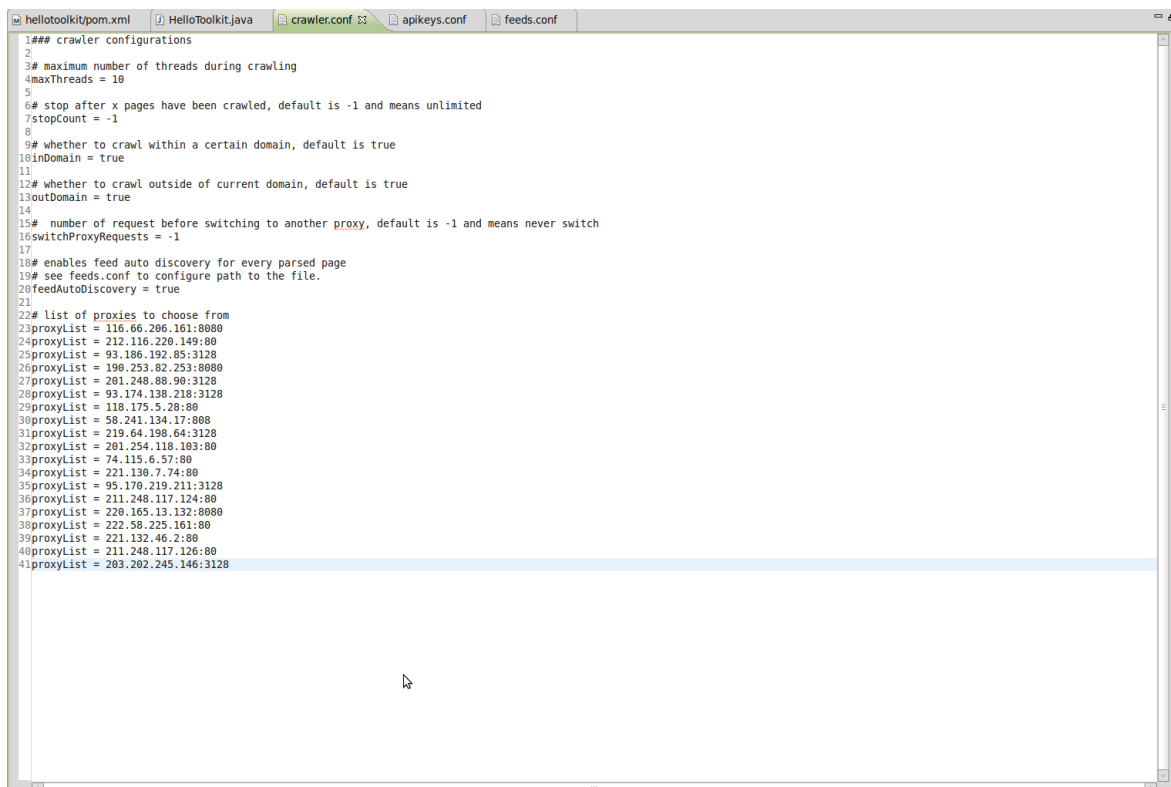


```

24 * Created on: 22.06.2010 10:56:36
4 package de.tud.inf.rn.iir.toolkit;
5
6 import java.util.ArrayList;
7
8 import tud.iir.helper.CollectionHelper;
9 import tud.iir.web.SourceRetriever;
10 import tud.iir.web.SourceRetrieverManager;
11
12 * <p>
22 public class HelloToolkit {
23
24 * <p>
31 public static void main(String[] args) {
32     // create source retriever object
33     SourceRetriever s = new SourceRetriever();
34
35     // set maximum number of expected results
36     s.setResultCount(10);
37
38     // set search result language to english
39     s.setLanguage(SourceRetriever.LANGUAGE_ENGLISH);
40
41     // set the query source to the Bing search engine
42     s.setSource(SourceRetrieverManager.BING);
43
44     // search for "Jim Carrey" in exact match mode (second parameter = true)
45     ArrayList<String> resultURLs = s.getURLs("Hello Toolkit", true);
46
47     // print the results
48     CollectionHelper.print(resultURLs);
49
50 }
51
52 }
53

```

Figure 12: "Hello Toolkit" Code



```

1### crawler configurations
2
3# maximum number of threads during crawling
4maxThreads = 10
5
6# stop after x pages have been crawled, default is -1 and means unlimited
7stopCount = -1
8
9# whether to crawl within a certain domain, default is true
10inDomain = true
11
12# whether to crawl outside of current domain, default is true
13outDomain = true
14
15# number of request before switching to another proxy, default is -1 and means never switch
16switchProxyRequests = -1
17
18# enables feed auto discovery for every parsed page
19# see feeds.conf to configure path to the file.
20feedAutoDiscovery = true
21
22# list of proxies to choose from
23proxyList = 116.66.206.161:8080
24proxyList = 212.116.220.149:80
25proxyList = 93.186.192.85:3128
26proxyList = 190.253.82.253:8080
27proxyList = 201.248.88.90:3128
28proxyList = 93.174.138.218:3128
29proxyList = 118.175.5.28:80
30proxyList = 58.241.134.17:808
31proxyList = 219.64.198.64:3128
32proxyList = 201.254.118.103:80
33proxyList = 74.115.6.57:80
34proxyList = 221.130.7.74:80
35proxyList = 95.170.219.211:3128
36proxyList = 211.248.117.124:80
37proxyList = 220.165.13.132:8080
38proxyList = 222.58.225.161:80
39proxyList = 221.132.46.2:80
40proxyList = 211.248.117.126:80
41proxyList = 203.202.245.146:3128

```

Figure 13: Create a new Maven project

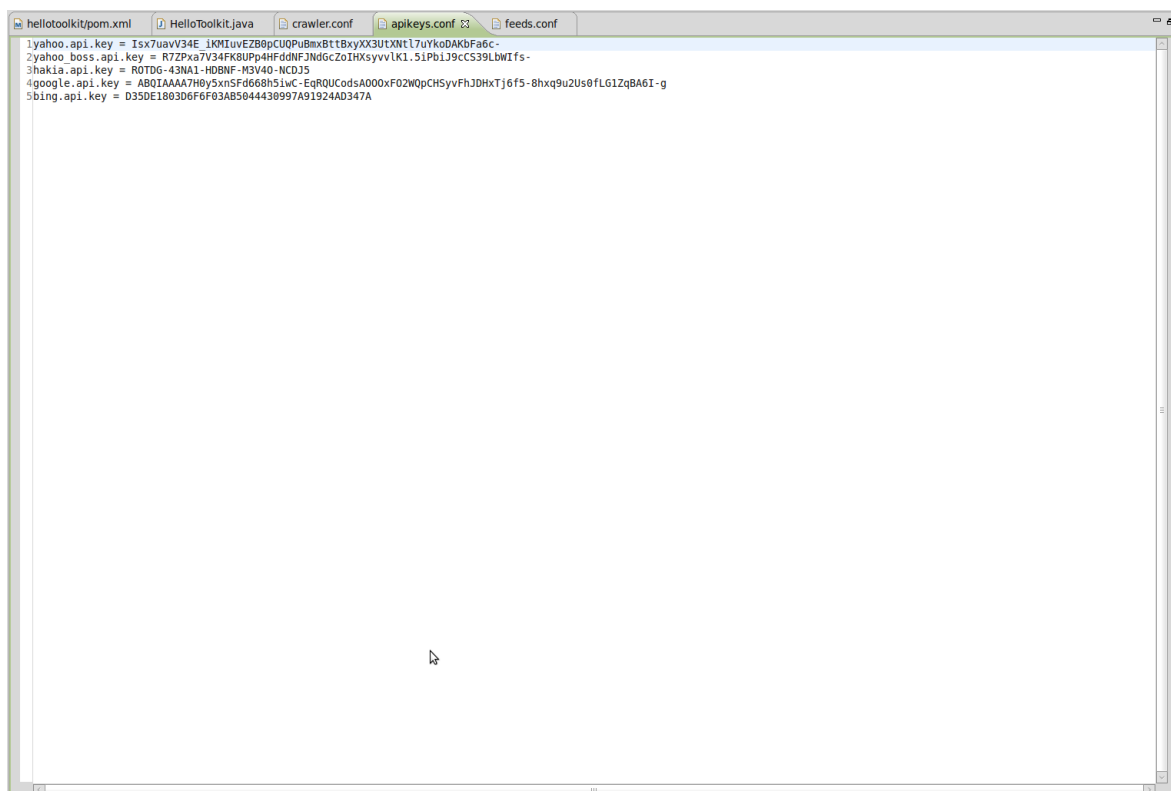


Figure 14: Create a new Maven project

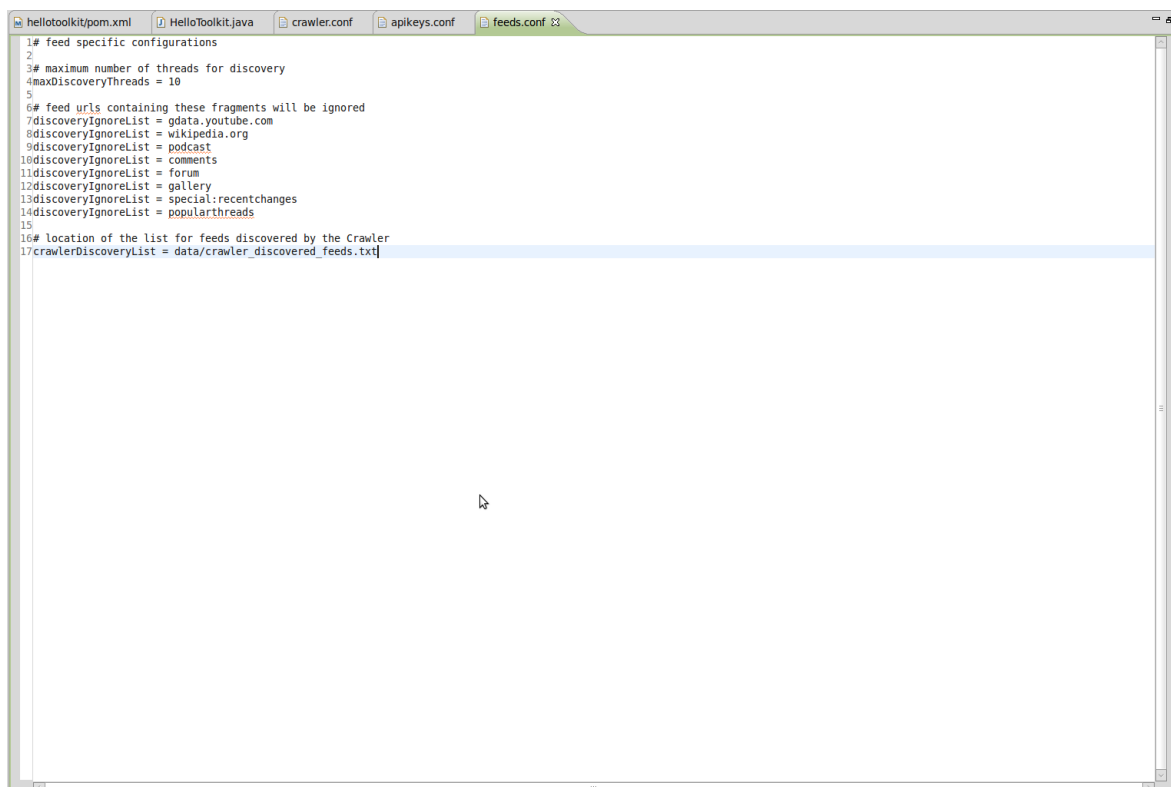


Figure 15: Create a new Maven project

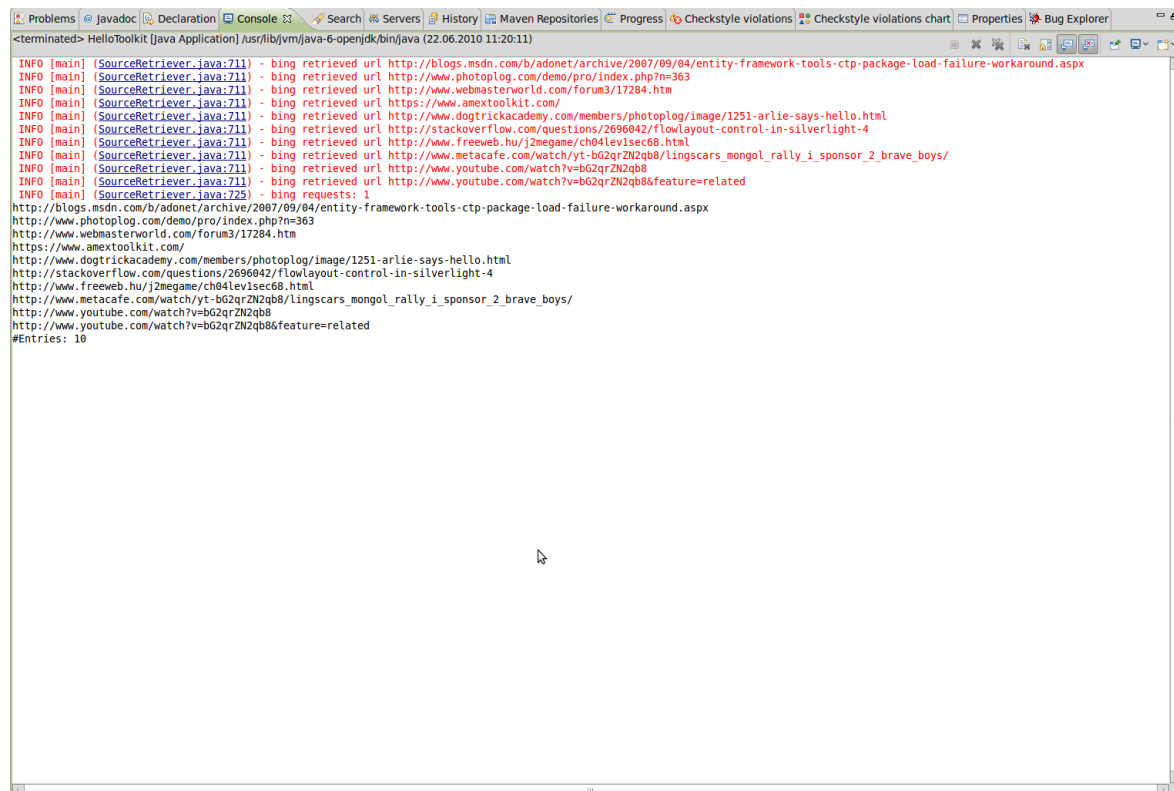


Figure 16: Create a new Maven project

6.2 Source Retriever

The source retriever is a module that can query a number of sources such as search engines and web pages with terms and retrieve matching results.

6.2.1 Basic Features

Basic features are:

- Query the Google search engine (unlimited queries, top 64 results only).
- Query Yahoo search engine (5000 queries per IP and day, top 1000 results).
- Query Bing search engine (unlimited)
- Query Hakia search engine.
- Query Twitter.
- Query Google Blog search.
- Query Textrunner web page.

Some of the search APIs require API keys which must be specified in the config/apikeys.conf file. See 3.1.1 for more information.

6.2.2 How To

The following code snippet shows how to initialize the source retriever and get a list of (English) URLs from the Bing search engine for the exact search “Jim Carrey”.

```

1 // create source retriever object
2 SourceRetriever s = new SourceRetriever();
3
4 // set maximum number of expected results
5 s.setResultCount(10);
6
7 // set search result language to english
8 s.setLanguage(SourceRetrieve.LANGUAGEENGLISH);
9
10 // set the query source to the Bing search engine
11 s.setSource(SourceRetrieverManager.BING);
12
13 // search for "Jim Carrey" in exact match mode (second parameter = true)
14 ArrayList<String> resultURLs = s.getURLs("Jim_Carrey", true);
15
16 // print the results
17 CollectionHelper.print(resultURLs);

```

6.3 Web Crawler

The web crawler can be used to crawl domains or just retrieve the cleansed HTML document of a single web page.

6.3.1 Basic Features

Basic functionalities include:

- Download and save contents of a web page.
- Automatically crawl in- and/or outbound links from web pages.
- Use URL rules for the crawling process.
- Extract title, description, keywords and body content of a web page.
- Remove HTML, SCRIPT and CSS tags.
- Find a sibling page of a given URL.
- Switch proxies after a certain number of requests to avoid being blocked.

6.3.2 How To

The following code shows how to instantiate a simple crawler that starts at <http://www.dmoz.org> and follows all in- and outbound links. The URL of each crawled page is printed to the screen. The crawler will use 10 threads, changes the proxy after every third request and stops after having crawled 1000 pages. Instead of setting the parameters using the code, we can also specify them in the `config/crawler.conf` file. See 3.1.2 for more information.

```

1 // create the crawler object
2 Crawler c = new Crawler();
3
4 // create a callback that is triggered for every crawled page
5 CrawlerCallback crawlerCallback = new CrawlerCallback() {
6     @Override
7     public void crawlerCallback(Document document) {
8         // TODO do something with the page

```

```

9         System.out.println(document.getDocumentURI());
10    }
11 };
12 c.setCrawlerCallback(crawlerCallback);
13
14 // stop after 1000 pages have been crawled (default is unlimited)
15 c.setStopCount(1000);
16
17 // set the maximum number of threads to 10
18 c.setMaxThreads(10);
19
20 // the crawler should automatically use different proxies
21 // after every 3rd request (default is no proxy switching)
22 c.setSwitchProxyRequests(3);
23
24 // set a list of proxies to choose from
25 List<String> proxyList = new ArrayList<String>();
26 proxyList.add("83.244.106.73:8080");
27 proxyList.add("83.244.106.73:80");
28 proxyList.add("67.159.31.22:8080");
29 c.setProxyList(proxyList);
30
31 // start the crawling process from a certain page,
32 // true = follow links within the start domain
33 // true = follow outgoing links
34 c.startCrawl("http://www.dmoz.org/", true, true);

```

6.4 FAQ Extractor

The FAQ extractor can extract question-answer pairs from several structured frequently asked questions pages on websites. The usage is quite simple as shown in the following listing.

```

1 // create a list of question answer pairs
2 ArrayList<QA> qas = null;
3
4 // the URL that contains the FAQ
5 String url = "http://blog.pandora.com/faq/";
6
7 // start extracting question and answers from the URL
8 qas = QAExtractor.getInstance().extractFAQ(url);
9
10 // print the extracted questions and answers
11 CollectionHelper.print(qas);

```

6.5 Fact Extraction

The Fact extractor can be used to detect facts in tables on web pages given a URL and optionally a small set of seed attribute names that help the extractor.

```

1 // the URL of the facts
2 String url = "http://en.wikipedia.org/wiki/Nokia_N95";
3
4 // the concept of the attributes
5 Concept c = new Concept("Mobile_Phone");

```

```

6
7 // a small list of seed attributes
8 HashSet<Attribute> seedAttributes = new HashSet<Attribute>();
9 int attributeType = Attribute.VALUESTRING;
10 seedAttributes.add(new Attribute("Second_camera", attributeType, c));
11 seedAttributes.add(new Attribute("Memory_card", attributeType, c));
12 seedAttributes.add(new Attribute("Form_factor", attributeType, c));
13
14 // detect the facts using the seeds from the URL
15 ArrayList<Fact> detectedFacts = null;
16 detectedFacts = FactExtractor.extractFacts(url, seedAttributes);
17
18 // print the extracted facts
19 CollectionHelper.print(detectedFacts);

```

6.6 Web Page Classification

The Web Page Classification module can be used to classify web pages by their URL or their full content.

6.6.1 Basic Features

The Web Page Classification module has the following basic features:

- Classify web pages by its URL only.
- Classify web pages by its full content.
- Use a combination of URL and full content for classification.
- Learn, test and reuse models.
- Simple one-category classification.
- Hierarchical classification.
- Multi-category classification (tagging).
- All algorithms are language independent.

6.6.2 How To

This section describes how to prepare training and testing data to learn a model and test the classifier.

Preparing the Training/Testing Data The data can be specified in a simple text file. There are three classification options, namely, one-category classification, hierarchical classification and multi-category classification. They all require a similar structure of the data.

One-Category Classification We write one URL and one category separated with a single space on each line. For example:

```

http://www.google.com search
http://www.fifa.com sport
http://www.oscars.com entertainment

```


Hierarchical Classification We write one URL and multiple categories separated with a single space on each line. The categories must be in the correct order, so the first category is the main one, all following are subcategories of each other. For example:

```
http://www.google.com search search_engine
http://www.fifa.com sport team_sports soccer
http://www.oscars.com entertainment movies awards usa
```

Multi-Category Classification We write one URL and multiple categories separated with a single space on each line. The order of the categories (tags) does not matter. For example:

```
http://www.google.com search image_search video_search
http://www.fifa.com soccer sport free_time fun ball_game results to_read
http://www.oscars.com entertainment movies films awards watch video stars
```

Building the Model The model is an internal representation of the learned data. After learning a model, a classifier can be applied to unseen data. We now have prepared the training and testing data so we can now learn the models. The results of the test will be printed to the console and written to a log file under data/logs. The classifier is saved as a lucene index or a database under the name “dictionary_Xclassifier_Y” where X is “url”, “fullpage” or “combined” and Y is 1 (one-category), 2 (hierarchical) or 3 (multi-category). The model will be written to data/models. How the file is saved can be configured in the config/classification.conf file. See 3.1.2 for more information.

```
1 // create a classifier manager object
2 ClassifierManager classifierManager = new ClassifierManager();
3
4 // use 80% of the data in the training/testing file as training data
5 // the rest is used for testing
6 classifierManager.setTrainingDataPercentage(80);
7
8 // specify location of training/testing file
9 String ttFile = "training-testing-file.txt";
10
11 // build and test the model
12 // the second parameter specifies that we want to use URL features only
13 // the third parameter specifies that we want to use
14 // multi-category classification (tagging)
15 // the last parameter is set to true in order to train not just test it
16 classifierManager.trainAndTestClassifier(ttFile,
17                                         WebPageClassifier.URL,
18                                         WebPageClassifier.TAG,
19                                         true);
```

Using the Model After we trained a model for a classifier we can apply it to unseen data. Let’s use the model we trained for a URL classifier with tagging:

```
1 // create the URL classifier
2 WebPageClassifier classifier = new URLClassifier();
3
4 // create a classification document
5 ClassificationDocument classifiedDocument = null;
6
7 // the web page to be classified
8 String url = "http://en.wikipedia.org/wiki/Computer";
9
```

```

10 // use the classifier to classify a web page to multiple categories
11 // using URL features only
12 classifiedDocument = classifier.classify(url, WebPageClassifier.TAG);
13
14 // print out classification results
15 System.out.println(classifiedDocument);

```

6.7 Helpers

The toolkit contains many helper functionalities for reoccurring tasks in the tud.iir.helper package. The following code snippet shows several sample usages of some of the functions.

```

1 // sort a map by its value in ascending order (2nd parameter = true)
2 Map m = CollectionHelper.sortByValue(map, true);
3
4 // reverse a list
5 List l = CollectionHelper.reverse(list);
6
7 // print the contents of a collection
8 CollectionHelper.print(collection);
9
10 // get the runtime of an algorithm and print it (2nd parameter = true)
11 long startTime = System.currentTimeMillis();
12 for (int i = 0; i < 10000; i++) {
13     int c = i * 2;
14 }
15 DateHelper.getRuntime(t1, true);
16
17 // (de) serialization of objects
18 FileHelper.serialize(obj, "obj.ser");
19 Object obj = FileHelper.deserialize("obj.ser");
20
21 // rename, copy, move and delete files
22 FileHelper.rename(new File("a.txt"), "b.txt");
23 FileHelper.copyFile("src.txt", "dest.txt");
24 FileHelper.move(new File("src.txt"), "dest.txt");
25 FileHelper.delete("src.txt");
26
27 // get files from a folder
28 File[] files = FileHelper.GetFiles("folder");
29
30 // zip and unzip a text
31 FileHelper.zip("text", "zipFile.zip");
32 String t = FileHelper.unzipFileToString("zipFile.zip");
33
34 // perform some action on every line of an ASCII file
35 final Object[] obj = new Object[1];
36 obj[0] = 1;
37
38 LineAction la = new LineAction(obj) {
39
40     @Override
41     public void performAction(String line, int lineNumber) {
42         System.out.println(lineNumber + ":" + line + obj[0]);
43     }
44 }

```

```

45 FileHelper.performActionOnEveryLine(filePath, la);
46
47 // round a number with a number of digits
48 double r = MathHelper.round(2.3333, 2);
49
50 // calculate n-grams of a string
51 Set<String> nGrams = StringHelper.calculateNGrams("abcde", 3);
52
53 // English singular word to plural
54 String p = StringHelper.wordToPlural("city");
55
56 // remove HTML tags
57 String r = StringHelper.removeHTMLTags("<a>abc</a>",
58                                     true, true
59                                     true, true);
60
61 // trim a string
62 String t = StringHelper.trim("_to_trim+++");
63
64 // get singular or plural of an English word
65 String plural = StringHelper.wordToPlural("city");
66 String singular = StringHelper.wordToSingular("cities");
67
68 // reverse a string
69 String r = StringHelper.reverse("abc");
70
71 // encode and decode base64
72 String e = StringHelper.encodeBase64("abc");
73 String d = StringHelper.decodeBase64(e);

```

7 Reference Libraries

The TUDIIR Toolkit makes excessive use of third party libraries. We do not intend to re-implement code but rather to built on it and create something superior. Here an incomplete list of libraries the toolkit uses:

- Alchemy API [11] to query the services of Alchemy.com.
- Apache Commons [1] for many standard tasks in string and number manipulation and more.
- Fathom [9] to measure readability of English text.
- iText [2] for creating PDF documents.
- Jena [3] for reading and writing ontology files.
- jFreeChart [4] for creating charts.
- jYaml [12] to read and write YAML files.
- Log4j [5] for logging.
- Lucene [6] for indexing and making learned models persistent.
- NekoHTML [7] to clean up the HTML of web pages in order to process them correctly.
- SimMetrics [8] to calculate similarities of strings.
- Twitter4j [10] to query the Twitter API.
- Weka [13] for machine learning.

8 History

The foundation of the toolkit code came out of the WebKnox project[14] that was started in 2008. The code is in development by students of the Dresden University of Technology. Contributors are:

- Christopher Friedrich
- Martin Gregor
- Philipp Katz
- David Urbansky
- Robert Willner
- Martin Werner

References

- [1] <http://commons.apache.org/>
- [2] <http://itextpdf.com/>
- [3] <http://jena.sourceforge.net/>
- [4] <http://www.jfree.org/jfreechart/>
- [5] <http://logging.apache.org/log4j/>
- [6] <http://lucene.apache.org/>
- [7] <http://nekohtml.sourceforge.net/>
- [8] <http://www.dcs.shef.ac.uk/~sam/simmetrics.html>
- [9] <http://www.representqueens.com/fathom/>
- [10] <http://twitter4j.org/en/index.html>
- [11] <http://www.alchemyapi.com/tools/>
- [12] <http://jyaml.sourceforge.net/>
- [13] <http://www.cs.waikato.ac.nz/ml/weka/>
- [14] <http://www.webknox.com>
- [15] <http://net.tutsplus.com/tutorials/html-css-techniques/top-15-best-practices-for-writing-super-r>