JSpider

User Manual

version 0-5-0-dev
http://j-spider.sourceforge.net

Part

1

Overview

I. Introduction

This section will introduce JSpider and give some general information about the project.

A. What is JSpider?

JSpider is an open source project, written entirely in Java. (J2SE).
It is an implementation of a highly flexible, configurable web robot engine.

B. Definition of terms

This is a list of terms used throughout the document with their definition:

Base Site
The base site is the site that functions as the starting point of the spidering process. When you start JSpider, the URL given to start the spidering from will determine the base site. The base site is the site of which the base URL is part of.

Base URL
The base URL is the URL given to JSpider to start the spidering process from (at startup time).

Parsing
Used as a term for examining the content of a web page to find links to other resources in it.

Spidering
Spidering is the process of fetching resource from a web server, reading the content and looking for references to other resources to fetch.
This way, the whole site (and other sites) can be 'discovered'

C. License

JSpider is distributed under the LGPL license.
More information can be found at http://www.opensource.org.
The license itself comes with the JSpider distributions and is also accessible at http://www.opensource.org/licenses/lgpl-license.php.

D. What can I do?

JSpider is a piece of software that's evolving constantly. Well, great, but what can I do to make it even better?

1. Using JSpider

Well, as this is a free piece of software, you could simply start by using it. We hope that the countless hours of thinking, designing, implementing, testing and refactoring JSpider will pay off.
We really hope that this project will be of use to many of you out there.

2. Giving feedback

We can't create a top product without feedback from the user base. If you have any comments, ideas, success or failure stories regarding JSpider, please post them on our mailing lists or the forums. Of course, any questions you might have are welcome and will be answered as soon as possible by the people who know the ins and outs of the software.

3. Posting on mailing lists

A great way of communicating is mail. JSpider has different mailing lists to discuss its working, ask questions about the usage, configuration and workings, and another subject you can think of.
The procedure to subscribe and post to the mailing lists, as well as the information on how to browse the archives can be found in the reference section.

4. Forums

The sourceforge project site also contains forums that can be used to ask questions and discuss the future of the project.
The URL of the JSpider forums can be found in the reference section at the end of this manual.

5. Reporting bugs

If you face bugs in JSpider while using it, or find that it behaves another way than it should be, you can fill out a bug report on the sourceforge project site.
This makes sure that your issue will attract the attention of the developers and will be handled appropriately.

Bug reporting URLs can be found at the end of this manual

6. Submitting feature requests

You'll probably encounter situations where you find that JSpider is lacking functionality, or doesn't support something out-of-the-box that would be beneficial to many people.
If you think of any feature, any idea related to JSpider, please submit it to the project site on sourceforge as a feature request. Even if you doubt how and if your idea is implementable, it doesn't cost anything to voice it to the world.
The location at which you can find the bug tracking is given at the end of this document, in the reference section

7. Submitting patches

If you are a developer, you might want to dive into the JSpider code right away.
If you implement some new functionality, or fix a bug, you can contribute this to the project.
Instructions on how to do this can be found in the developer's manual.

II. Concepts

This section will introduce you in the concepts of JSpider. When you understand the different components and their interaction, you ll be able to configure JSpider to adapt it to your needs in any situation.

A. JSpider global design

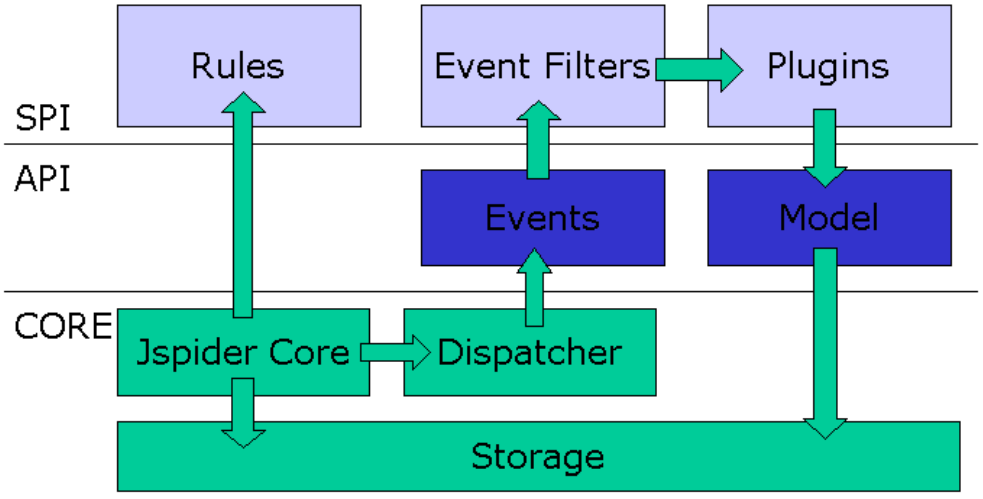
JSpider is an engine, not an application. Although an out-of-the box installation is very useful already, it is designed to be easily extended and configured.
This means that the engine (the core) of JSpider is the most important part. Actual functionality will be added afterwards, and will be invoked by the engine.

1. Main components

The main parts that can be distinguished are:

- the engine core
- a model, describing all spidered resources, sites, etc...
- an event model notifying of what s going on
- components implementing a SPI (Service Provider Interface)

JSpider overview



2. JSpider engine core

The main part of JSpider is the **engine core**, that implements the most basic functionality delivered by JSpider.

3. SPI components

The real application functionality that will be of use to you will be implemented by the **components** that adhere to certain SPI interfaces.
JSpider comes with a number of implementations of these components, and you can also implement your own.
There are actually used to extend and add functionality to JSpider.

There are several types of components that can be used by JSpider. The most important ones are:

- Rules
- Plugins
- Event Filters

Rules

Rules decide which resources should be fetched and/or processed by JSpider. By construction a set of rules on a global or per-website basis, you can define JSpider's behaviour and scope.
There are a lot of rule implementations that come with JSpider, and you can also develop your own.

Rules are executed one after another, until a decision is made.
The decision types that can be taken are:

- *don't care*
  The rule states it doesn't apply to the situation and doesn't interfere with the decision

- *accept*
  The rule checked the situation and decides that the URL should be accepted for spidering or parsing (depending on the situation in which it was used)
  Other rules can override this decision, however.

- *ignore*
  The rule checked the situation and decided that the resource should be skipped for processing (fetching or parsing, depending on the situation presented). This decision is not

vetoable anymore by another rule and ends the decision chain.

- *forbidden*
  The rule checked the situation and decided that the processing of the resource (fetching or parsing, depending on the situation) is forbidden. This is a stronger version of ignore, and also ends the decision chain.

With every resource, the results of the spidering and parsing decision chain is saved, so you can always trace what rule caused a certain resource (not) to be fetched or parsed.

Plugins

Plugins are components that have access to the data structures exposed by JSpider, and are notified of certain events happening. They can then take appropriate actions.
These actions can be anything you can think of:

- Writing a report file
- Displaying a message on the console
- Writing a fetched resource to disk
- Sending a mail to someone
- Etc...

By implementing your own plugin, you can add functionality to JSpider. You could, for instance, construct a configuration in which JSpider tests a certain site for 404 errors (link errors), and send an e-mail with all error links to the webmaster.
Another usage would be to mirror a website on your local disk: for this purpose, you would enable a plugin that writes every fetched resource into a file on your harddisk.

Event Filters

Event Filters can select the events that have to be handled by the system as a whole or a particular plugin.

4. API components

The JSpider API consists out of the following type of objects

- an object model
- an event system

Object model

The **model** is an object model that represents everything that JSpider encounters while spidering:

- Sites
- Resources (URLs)
- Content
- References between resources
- Cookies
- Etc…

This model can be accessed from within the components in order to look up data, write a report, calculate statistics, etc…
The model is backed by the storage component.

Event system

The **event system** is a group of event classes that will be used to notify all interested plugins of certain events happening during the spidering progress.

There are three types of events in JSpider:

- *engine events*
  spidering started, stopped, configuration chosen, …
- *spidering events*
  site discovered, resource spidered, fetch error, …
- *monitoring events*
  give information about the spidering progress and the thread pool occupation.

B. JSpider applications

JSpider is useable in two forms:

- A standalone application (JSpider itself)
- A set of useful tools (JSpider-tool)

1. JSpider application

The main JSpider application is a web robot that ll spider (fetch) web resources, parse the result content, search for new links, and fetch those also.
This whole process is configurable, and only resources that apply to certain rules are spidered and/or parsed.

You can easily limit the spidering process to:

- a certain web site
- a certain group of websites
- a certain part of a web site
- resources with a certain content type
- resources that are referenced from a certain other site
- etc…

The possibilities are virtually unlimited.

During and after this process, reports can be written to disk, fetched resources can be downloaded to a local folder, errors can be reported, etc…

2. JSpider-tool

The JSpider-tool is a tool that can do several things for you, all based on JSpider. Instead of spidering everything that s within the boundaries of the configured rules, it is aiming towards a single web resource (URL).

Tools exist to:

- Download a file
- Show the contents of a web resource
- Show all info about a web resource
- Show all HTTP Headers given by the server
- Etc…

While this functionality is implemented in a way that it depends on the JSpider core, it can be used separately from the main application.

More information on how to use these applications can be found in the section on the usage of JSpider.

C. Event system

The **JSpider event model** is a group of java event classes that represent events that can occur while spidering websites.

These events include:

- An event telling that a new site was found
- An event telling a certain web page was fetched
- An event telling the robots.txt file for a certain site was interpreted
- An event telling that a web page wasn't found on the server
- An event telling a resource is skipped for processing because access to it wasn't allowed by some rule
- ...

These events will be **generated in the JSpider core**, and the JSpider event dispatcher will **dispatch them towards the plugins**.

Each plugin can then take appropriate action for each event. A plugin that is interested in finding 404 errors, for example might use the notification of a webpage-not-found-event to write the URL of the page in a file, along with the page that referred to it.

1. Types of events

Different types of events can be distinguished, along different criteria:

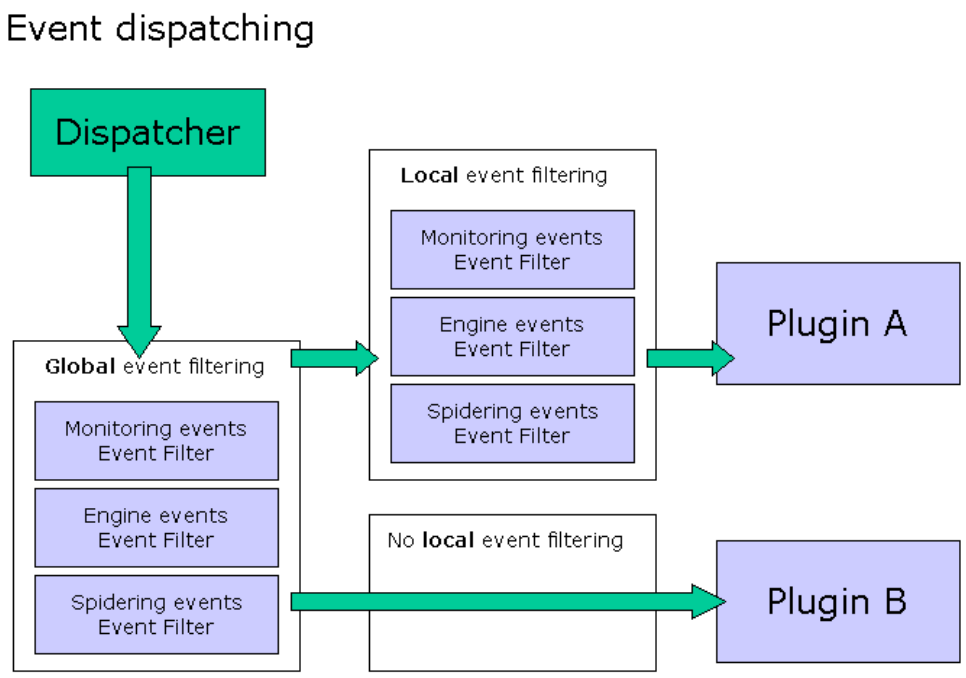
- Whether the event is a **Engine**-related, **spidering**-related, or a **monitoring** event.
- Whether an event is **filterable** or will be dispatched no matter what
- Whether the event expresses an **error** situation or not.

These differences can be used to filter, select and interpret events. They can be used inside plugins to determine the appropriate action to take.

2. Event Dispatching

It's important to understand what happens between the raising of an event during spidering by the JSpider core, and the moment it eventually arrives in a plugin.

The process of dispatching is done as follows:



As you can see, the event dispatching starts at the dispatcher. This component will throw the events to be dispatched through the dispatching chain.

The first component met is the **Global event filtering**. This is a group of filters (one for each type of event – engine, spidering, monitoring), that will let or let not pass a certain event.
If an event is filtered out at this point, it will never arrive at any plugin.

The second component exists per plugin: it is the **local event filtering**, which functions in the same way as the global event filtering, except that is put right in front of a particular plugin, so events that are filtered out will not reach that particular plugin, but may be let through to another one.

It is also possible to disable the global event filtering as a whole, or the local event filtering for a certain plugin. (As plugin B in the example picture).
This results in all events being passed through.

By **customizing the event filtering chain**, you can configure JSpider to focus on the aspect of the spidering process you are

interested in. If you're using JSpider to genere error reports for a website, you're probably only interested in error events. If you're automating the JSpider process, you're probably not interested in monitoring events telling how much load there is upon the thread pools, etc…

3. Event list

In order to get a better insight of the JSpider event system, here is an (incomplete) **overview of all event classes**: (Note that all events are in packages under net.javacoding.jspider.api.event):

The **engine**-related events:

```
engine.SpideringStartedEvent
engine.SpideringStoppedEvent
engine.SpideringSummaryEvent
```

The **monitoring** events:

```
monitor.MonitorEvent
monitor.SchedulerMonitorEvent
monitor.ThreadPoolMonitorEvent
```

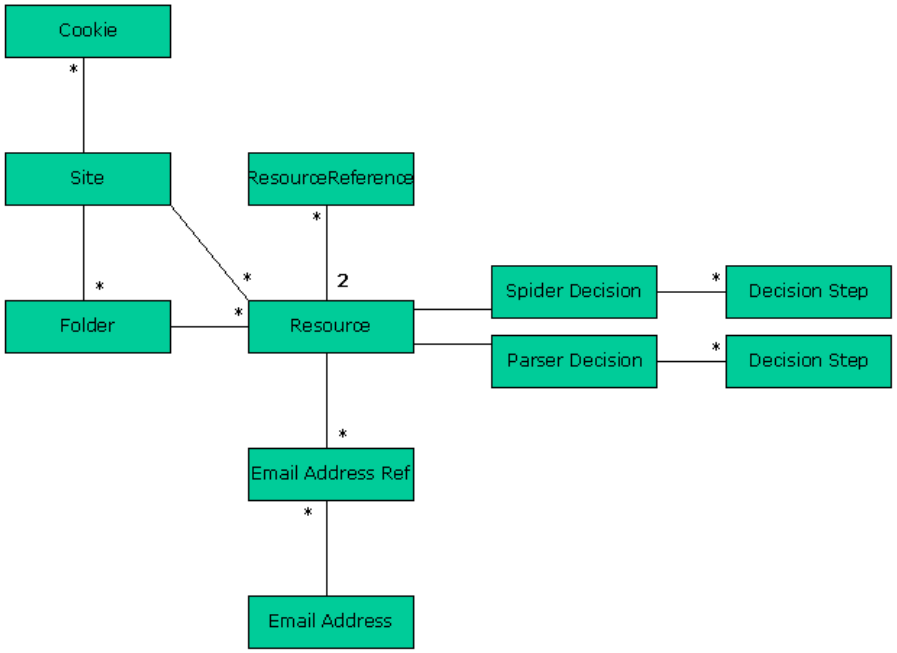
And finally the **spidering**-related events:

```
folder.FolderDiscoveredEvent

resource.EMailAddressDiscoveredEvent
resource.EMailAddressReferenceDiscoveredEvent
resource.MalformedBaseURLFoundEvent
resource.MalformedURLFoundEvent
resource.ResourceDiscoveredEvent
resource.ResourceFetchedEvent
resource.ResourceFetchErrorEvent
resource.ResourceForbiddenEvent
resource.ResourceIgnoredForFetchingEvent
resource.ResourceIgnoredForParsingEvent
resource.ResourceParsedEvent
resource.ResourceReferenceDiscoveredEvent

site.RobotsTXTFetchedEvent
site.RobotsTXTFetchErrorEvent
site.RobotsTXTMissingEvent
site.RobotsTXTSkippedEvent
site.SiteDiscoveredEvent
site.UserAgentObeyedEvent
```

D. Object model

The second part of the public API of JSpider consists out of an object model. This object model represents all entities found during the spidering process.

Object Model



The entities presented here are:

- Site
- Resource
- Folder
- Cookie
- Resource Reference
- Email Address
- Email address reference
- Decision (one for the spidering, one for the parsing decision)
- Decision Step

The public model interfaces and classes are found under the package net.javacoding.jspider.api.model.

1. Sites

The site is a type of object in JSpider that aggregates all data about resources that make up a web site. The unique combination of a hostname/port makes up a site.

The status of a site is dependent on the stage of processing it is in:

- STATE_DISCOVERED
- STATE_ROBOTSTXT_HANDLED

- STATE_ROBOTSTXT_ERROR
- STATE_ROBOTSTXT_UNEXISTING
- STATE_ROBOTSTXT_SKIPPED

As you can see, the state of a Site is highly dependent on the processing of the robots.txt file.

2. Resources

The most important object type in JSpider is a **Resource**.
It is the representation of a web page, a URL.

An important concept you should know about is the **status** a resource can have. JSpider knows different statuses for resources:

- *Discovered*
  Means that the resource is found in a web page that links to it, but no processing has been done yet

- *Fetch_Ignored*
  Specifies that some rule decided that the resource should be skipped for fetching (it will not be retrieved from the webserver, so we won't know whether it actually exists).

- *Fetch_Error*
  Used when we tried to fetch the resource, but this resulted in an error situation.

- *Fetch_Forbidden*
  Used when our rules (custom, robots.txt) tell the resource may not be fetched. Mostly used for robots.txt disallow rules.

- *Fetched*
  Used when the resource has been retrieved, but no intepretation has been done yet.

- *Parse_Ignored*
  Used when rules decided that the resource that was fetched from the server shouldn't be inspected to find links to other web pages.
  This will be the final stage for images and binary files, for example, since it's no use parsing them to find other links.

- *Parse_Error*
  Specifies that we tried to parse the resource, but this resulted in a severe error (rare).

?? *Parsed*
Specifies that the web page was parsed (inspected to find other links in it). This is the final stage for web pages.

During the parsing process, all resources will loop their lifecycles. According to rules, and encountered errors, they can end up in some specific state.

E. Spidering process

While this topic is strictly about JSpider internals, it might help to explain these concepts a bit.

The design of JSpider is based upon a principle of micro-tasks.
Every piece of work that should be carried out by JSpider is made a task.
For instance:

- The task of **fetching** a web page
- The task of **parsing** a web page an finding links in it
- The task of **deciding** whether a certain resource needs to be fetched (according to rules, robots.txt, etc…)
- The task of **deciding** whether a certain resource needs to be parsed (according to rules)
- The task of interpreting a **robots.txt** file
- …

All these tasks can produce new tasks:

- The parsing of a web page can find new URLs, which should be decided upon whether to fetch them or not
- …

These tasks are scheduled in two different groups, according to their type: we distinguish **Thinker** tasks and **Spider** tasks.

Spider tasks are tasks that go out on the network/internet and fetch some data (web page, robots.txt).

Thinker tasks are tasks like parsing a web page, deciding whether a resource should be fetched or parsed, interpreting a fetched robots.txt file, etc…

This distinction will become important, as these two types of tasks are carried out by **two different thread pools**. (see later in the configuration section).

Part

2

Installation

III. Prerequisites

JSpider has some prerequisites to run:

- J2SE 1.3+ Runtime
- XMLParser (Xerces, …) installed (comes with JDK1.4)

If you're planning to build from CVS installation, you'll also need:

- a CVS client
- Ant 1.5.2 (http://ant.apache.org)
- JDK 1.3+

IV. Binary Installation

The easiest way to install JSpider, is to download a binary JSpider distribution. These distributions are released from time to time when the codebase is considered stable enough, and a considerable amount of new functionality or bug fixes has been implemented since the previous release.

A. Downloading

The first step to do when installing a binary distribution is downloading the installation.

Our download page can be found at:
http://sourceforge.net/project/showfiles.php?group_id=65617
It is always recommended to use the latest (stable) release.

We offer our downloads in several flavours: simple binary downloads, or packages with the source included.
You can download them as a zip (jar) archive or a tar.gz file.

> **!** The filenames given are just examples.
> Since the distribution file contains the JSpider version, you should follow the examples using the filename of the actual file you **downloaded**.

B. Unpacking

The process of unpacking is dependent on the type of file you downloaded. For a zip (remember to use the appropriate name):

```
jar –xvf jspider-0-5-0-dev.zip
```

Will unzip the archive to it's current location. Of course you can also use WinZip, or any other program you're comfortable with.

For a tar.gz file:

```
gunzip jspider-0-5-0-dev.tar.gz
tar –xvf jspider-0-5-0-dev.tar
```

C. Basic configuration

Before you can test your JSpider installation, you should set your proxy server in the jspider.properties file (only have to do this if you want to access a resource on the internet and when you're behind a proxy server).

Edit the conf/default/jspider.properties file and look for these lines:

```
jspider.proxy.use=false
jspider.proxy.host=
jspider.proxy.port=
jspider.proxy.authenticate=false
jspider.proxy.user=
jspider.proxy.password=
```

Adapt them as needed (see section on configuration for more information about this)
If you're going to test JSpider only on your local network (or even a webserver on your own machine), you can simply skip this step for now.

> **!** IMPORTANT NOTE: If you have a Java version lower then 1.4 (JDK1.3, ...), you'll have to add an XML Parser to your system in order for JSpider to work!
> See the section about "Environment Configuration" for more details on this.

This is all that is needed to start off with JSpider. Once you get a good understanding of how the system works, you will be able to fine-tune every aspect of it.

D. Testing

Now you've got your JSpider instance, CD into the 'bin' folder.

Start JSpider and spider the site of your choice (in the example, the webserver on the localmachine).
Since we don't specify anything more, the 'default' configuration will be used (all the settings in the files under the conf/default folder):

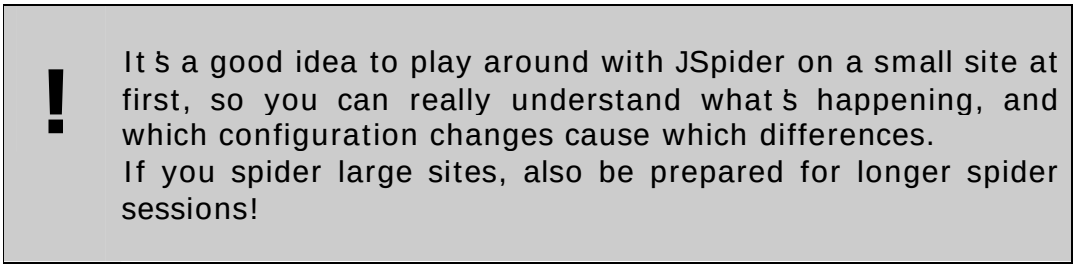
On Windows:

```
Jspider http://localhost
```

On unix:

```
./jspider.sh http://localhost
```

JSpider should start spitting out many lines in your console. After the spidering has stopped, you'll find many reports in the 'output' folder.



It's a good idea to play around with JSpider on a small site at first, so you can really understand what's happening, and which configuration changes cause which differences.
If you spider large sites, also be prepared for longer spider sessions!

V. Building from CVS

The JSpider project has regular releases, both of stable (production) releases, as well as 'development' releases which introduce new features and bug fixes between two major releases.

If you want the very latest changes, you can build your own JSpider from CVS source.

This section explains how to get the source from the CVS system, build a JSpider distribution, and test it.

> **!** The CVS HEAD version of JSpider may be unstable, and new features may be undocumented. If you're not going to develop on it, or don't have a very good reason to use the latest CVS version, you're probably better off with the latest release download.

Anyway, if you decide to build from CVS, here's how to do so:

This explanation assumes a CVS console client being installed on the system and put on the PATH environment variable. If you use a CVS GUI client, you should be able to extract the necessary information from the explanation.
(Also see the appendices section).

A. Setting the CVSROOT

The CVSROOT for the JSpider project must be set first.
On Unix:

```
export CVSROOT=
  :pserver:anonymous@cvs.j -spider.sourceforge.net:/cvsroot/j -spider
```

On Windows:

```
set CVSROOT=
  :pserver:anonymous@cvs.j -spider.sourceforge.net:/cvsroot/j -spider
```

after that, you can use 'cvs login' (with empty password) to verify you didn't make a typo in the CVSROOT, and the CVS server is accessible:

```
cvs login
```

(give empty password)

Normally, this should return without an error.

B. Checking out

You now can checkout the JSpider sources.

There are two ways to do this:
- ?? a CVS checkout
- ?? a CVS export

A **checkout** keeps the reference to the CVS repository. Use this if you're thinking of making modifications to the sources. This way, you can create a diff file to have your changes patched.

If you're only interested in building the actual sources, and you're not going to change them, you can use the **export** function to get the sources without any CVS administrative files.

To do a checkout, use:
```
cvs checkout -P jspider-main
```
To do an export, use:
```
cvs export -r HEAD jspider-main
```

You can also checkout or export an older version of jspider like this:
```
cvs checkout -P -r jspider-0-1-0-dev jspider-main
```
or
```
cvs export -r jspider-0-1-0-dev jspider-main
```

You should see a listing of all files being copied to your hard drive now under a folder 'jspider-main'. CD into this folder and you're ready to continue...

C. Basic configuration (optional)

If you copy the file '**base.user.properties**' to '**user.properties**' and fill in the right values for your proxy server, your JSpider instance will be built with these settings already in each and every configuration.
While you can skip this step, it's quite handy to do so, otherwise you'll have to edit the jspider.properties file in every configuration after the build is done.

D. Building from source

You're now ready to build JSpider. We have an Ant script to do so, which can be queried for it's targets using:

```
ant –projecthelp
```

This will return something like this:

```
Buildfile: build.xml
Main targets:

 buildComplete     buildSimple + documentation
 buildDistro       buildComplete + functionalTests + packaging
 buildSimple       builds a runnable and testable distribution with
                    technical tests
 testFull          runs the JUnit technical and functional tests
 testFunctional    runs the JUnit functional tests
 testInteractive   runs the JUnit tests interactively (swing ui)
 testTechnical     runs the JUnit technical tests

Default target: buildSimple
```

The ant targets for building JSpider are:

- *buildSimple* (default)
  This one compiles JSpider, runs a basic testsuite and prepares a distribution in 'dist/prepared'

- *buildComplete*
  This one does the same as buildSimple, but adds documentation (javadoc, todo list, junit test report, …)

- *buildDistro*
  This one does the same as buildComplete, but runs a complete online test suite also (tests JSpider in a real-life, functional way against resource on http://j-spider.sourceforge.net)
  It also creates distributable packages (zip, tar.gz – both bin and src) in 'dist/packaged'.

For now, we only need a simple build, so we're going to:

```
ant
```

You'll see the code compile, and the technical (simple class-level) junit tests to be carried out.

After this process has finished, you'll find a complete JSpider instance in a subfolder 'dist/prepared'.

If you have skipped the previous step of configuring your proxy server before building, now is the time to edit the jspider.properties file in every subfolder of 'dist/prepared/conf' to do so…

E. Running the test suite

Each time you build JSpider, the most basic test suite, the **technical** tests, will be run.

There is, however, a more thorough test suite, the **functional** tests, which test JSpider as a whole by letting it spider well-known resources on http://j-spider.sourceforge.net, and verify the results with the expectations.

The ant targets used to test JSpider are:

- *TestTechnical*
  Runs the most basic test suite, simply testing JSpider classes.

- *testFunctional*
  Runs a set of functional tests, for which JSpider needs to be configured (proxy server), and an internet connection needs to be available.

- *testFull*
  Runs both the technical and the functional tests

- *testInteractive*
  Which starts the JUnit GUI interface for some interactive testing.

After testing the JSpider instance with:

```
ant testFull
```

We will generate the JUnit test reports with:

```
ant generateJUnitDocs
```

You'll now find the results of the tests in 'stage/doc/junit', as 'junit-noframes.html' or 'index.html' (framed version).

> **!** **IMPORTANT NOTE**: The **functional** tests are dependent on a bunch of server-side scripts and web pages to test upon. Since these change with the JSpider sources itself, only the testcase files for the **latest JSpider version** (CVS HEAD) are online. This will **cause functional tests of older versions to fail**, while the tests succeeded at the moment of release in the past.
> If you checkout the jspider-site module for the correct version and install the scripts locally, you can test against an older version.

F. Using JSpider

When your freshly built JSpider instance passed it's tests, you can start using it: CD into the 'dist/prepared/bin' folder and launch it:

Windows:

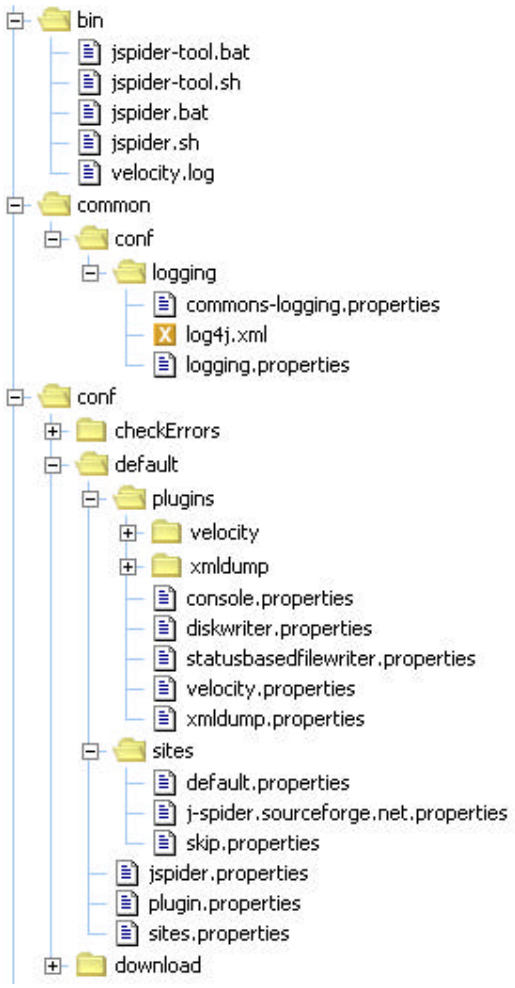
```
jspider http://localhost
```

Unix:

```
./jspider.sh http://localhost
```

VI. Folder Overview

Once you have your JSpider installed, you ll find a folder structure on your hard drive that looks like the picture:



You can use this picture as a reference for the rest of this document, as all files and folders we ll be talking about are presented here.

Most of the files are needed for configuration purposes, so we ll explain how they can be used to customize JSpider in the following sections.

For now, this is an overview of the folders found in your JSpider instance:

- **bin**
  folder containing all the JSpider startup scripts, for JSpider and JSpider-tool, both for windows and unix environments.

- **common**
  Files that are shared over all configurations
  - *Conf*
    Configuration files that are common to all configuration sets.

- **conf**
  Folder with all configuration sets
  - *default*
    Default configuration set which will be used when none is specified
  - *checkErrors*
    Out-of-the-box configuration that spiders a site looking for errors (404, 500, ...)
  - *download*
    Out-of-the-box configuration that spiders a site and downloads it to the local disk.
  - *Unittest*
    Configuration that's used when testing JSpider. Not to be used manually.

- **lib**
  Folder containing all needed libraries (jar files)

- **output**
  Folder in which all output files will be written by default (reports, downloads, ...)

- **src** (only if you downloaded the source distribution)

While this list is not exhaustive, it should give you a good overview of what files and folders are part of JSpider. We'll talk about all important files in the configuration section.

Part

3

Usage

VII. Starting JSpider

JSpider is started from the command prompt via a **startup script**. This script makes sure that the needed properties are set, that the right classpath is used (with all additional libraries on it), and that the JSPIDER_HOME variable is propagated to the main class.

These startup scripts can be found in the {JSPIDER_HOME}/bin folder.

If the JSPIDER_HOME environment variable is not set, it will be set to '..', since JSpider assumes to be started from within the bin directory then.

A. Windows

On windows, the syntax to start JSpider is like this:

```
jspider url [configuration]
```

This will execute the jspider.bat batch file.

Examples:

```
jspider http://localhost
```

Will start JSpider spidering the web server on the local machine with the default configuration (since none is specified).

```
jspider http://localhost checkErrors
```

Will start JSpider spidering the web server on the local machine with the configuration found under the folder 'conf/checkErrors'.

B. Unix

On unix, the syntax to start JSpider is like this:

```
./jspider.sh url [configuration]
```

This will execute the jspider.sh shell script.

Examples:

```
./jspider.sh http://localhost
```

Will start JSpider spidering the web server on the local machine with the default configuration (since none is specified).

```
./jspider.sh http://localhost checkErrors
```

Will start JSpider spidering the web server on the local machine with the configuration found under the folder 'conf/checkErrors'.

C. Configurations

The concept of multiple configurations allows you to create a separate configuration per environment or purpose for which you use JSpider.

Some out-of-the-box configurations come with JSpider:

- default
- checkErrors
- download

You can simply create your own by adding an extra folder for your custom configuration to the '/conf' folder and putting the necessary files in it.
(It's a good idea to start with a copy from another working configuration and customize that one).

In the new sections in this manual, we'll be looking into some scenarios in which you can use JSpider, and explain the configuration set up for those purposes.

VIII. Scenario: Checking a site for errors

In this section, we'll dive into a real-life usage of JSpider. We'll be using it to check a site of errors.
You might want to keep an eye on the Configuration part of this manual while reading this information.

A. Goal

It's a hard task to keep a web site up-to-date. All the time, new content is added, outdated content is removed, etc…

Another fact which adds to the complexity of maintaining a website, is that a site doesn't stand on it's own. You probably have a bunch of links to related sites.
Keeping track of changes in these sites (removed content, …) is important as well, as it may outdate links on your site.

Without a decent tool, it is fairly impossible and surely time-consuming to keep a site of any decent size clean and error-free.

JSpider can help you in this process, as it can automatically traverse your site, checking each link found on it, also to external sites.

It can generate reports that pinpoint the exact problems (which resources linked to an unexisting resource, which web pages resulted in an internal server error, etc…)

In minutes, JSpider can check your whole site for errors, and generate these reports for you. This dumb and repetitive checking task would cost you hours or days otherwise, and isn't as thorough when done by a human.

What we want to do with JSpider is:

- Check all links in our site
- Check all references to resources on other sites
- Write a report of all errors found
- Write reports of each error type encountered

B. Configuration

There is an out-of-the-box JSpider configuration that helps you with exactly this problematic: 'checkErrors'.

You can start JSpider using this configuration by typing:

```
jspider [host] checkErrors
```

In this section, we'll discuss this configuration, in order for you to understand how it is constructed to aid you in the process of finding errors on sites.
You might want to make small changes to adapt it to your specific situation.

1. Global configuration

We'll start of with the global configuration. Have a look in your 'conf/checkErrors' folder. The files involved in the global configuration are:

You'll find the file '**jspider.properties**', which contains the global configuration for this setup.
Open it with your favorite text editor, and have a look through it while we explain each part:

Proxy configuration

Well, this is something you should fill in. If you need a proxy server to connect to the internet, you should make sure that these properties are filled in with the correct values for your situation:

```
jspider.proxy.use=false
jspider.proxy.host=
jspider.proxy.port=
jspider.proxy.authenticate=false
jspider.proxy.user=
jspider.proxy.password=
```

See the configuration section for more information on these properties.

Other

The rest of the configuration can be kept as-is, as it is completely parallel to the default configuration.
This means:

- Threading: 5 spiders, 1 thinker
- Logging via commons-logging

- User Agent left default
- Spider rule: only spidering HTTP urls
- Parser rule: only parsing text/html mime type web pages

2. Per-site configuration

Open the 'conf/checkErrors/sites.properties' file.
It should look like this:

```
jspider.site.config.base=base
jspider.site.config.defaul t=default
```

This configuration is quite simple. We only make the difference between the site we are testing for errors (the **base** site) and any **other site** that might be referenced by ours.

According to this configuration file,our base site will have it's configuration in conf/checkErrors/sites/base.properties.
The configuration for all the other sites is described in the file conf/checkErrors/sites/default.properties.

TIP: If you have several sites that are interlinked, you can add these with the 'base' configuration, so they will be treated just like the base site.
All other sites (not yours) will still be handled via the 'default' site configuration.

We'll discuss those two site configurations in-depth (the most important part of the configuration will be the rules each time):

Site Configuration 'base'

This configuration will be assigned to the site(s) we're actually testing. This means we're going to spider them completely, checking each and every link.

The rules found in this file (near the bottom):

```
site.rules.spider.count=0

site.rules.parser.count=0
```

Well, isn't this simple? No extra rules for the base site (the global rules still apply).
So, a web page will be **spidered** if it's part of a site assigned the 'base' configuration, and has a URL with the 'http' protocol (global rule).

It will be **parsed** afterwards if it has a content type of "text/html" (global rule).

If you want to restrict access of JSpider to some parts of your site (that shouldn't be checked for errors), this is your place to do so.

For example, changing to this:

```
site.rules.spider.count=1

site.rules.spider.1.class=
  net.javacoding.jspider.mod.rule.ForbiddenPathRule
site.rules.spider.1.config.path=/apidocs
```

Will prevent JSpider from spidering your 'apidocs' folder on your website.

Other things that you might want to change in this file are:

- The throttling, to speed up the spidering process
- The proxy usage, if your site is on the LAN
- Robots.txt handling, to ignore your robots.txt file
- The User Agent, to access your site with another user agent

Site Configuration 'default'

Every other site (not the one(s) we're testing for errors) will be assigned the 'default' configuration.

At first, you'd have the feeling that you would simply have to put the 'site.handle' property to false.
However, this leads to a problem: if a link to any other site is found on your site, the linked resource will be skipped, and you'll never know whether your site links to a valid resource, or whether your link is dead.

Have a look in the conf/checkErrors/sites/default.properties site configuration file.
You'll find these rules:

```
site.rules.spider.count=1
site.rules.spider.1.class=
  net.javacoding.jspider.mod.rule.ExternallyReferencedOnlyRule

site.rules.parser.count=1
site.rules.parser.1.class=
  net.javacoding.jspider.mod.rule.RejectAllRule
```

Now this is a bit more interesting. What does this mean?

A web page of another site (not the base site) will only be **spidered** if:

- Its URL begins with the http protocol (global rule in jspider.properties)
- It is **referenced from an external site**. Only if the link to the resource is found on another site, we will spider the resource.
  This prevents in-site spidering on the site.
  Hey, we're not going to check someone else's site for errors, they should download their own copy of JSpider and do it theirselves!

A web page on another site (not the base site we're checking for errors) will NEVER be **parsed** :
The RejectAllRule tells that none of the resources in this type of site should be parsed.

This way, we prevent spidering someone else's site!

Please note that this way, we only spider web pages on external sites that we link to in direct: this way, we can make sure we have no dead links on our site.

3. Plugin Configuration

OK, now we taught JSpider to spider our own site completely, and only checking web pages on external sites if we link to these in direct.

But now we want JSpider to report it's findings during the spidering process: let's configure some plugins!

In the conf/checkErrors/plugin.properties, you'll find:

```
jspider.filter.enabled=true
jspider.filter.engine=
  net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
jspider.filter.monitoring=
  net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
jspider.filter.spider=
  net.javacoding.jspider.mod.event filter.ErrorsOnlyEventFilter

jspider.plugin.count=3
jspider.plugin.1.config=console
jspider.plugin.2.config=filewriter
jspider.plugin.3.config=statusbasedfilewriter
```

As you can see in the event filtering configuration, we let pass all engine events, all monitoring events (we want to see on the console how JSpider is doing), but only spider events that report **errors**.

We also assign 3 plugins to be used:

- Console, which we'll use to see the JSpider progression
- FileWriter, which will write down all our errors in a file report
- StatusBasedFileWriter, which will generate nice reports on each problem encountered (list with 404 errors, list with 500 errors, ...)

Let's have a look at the configuration of the plugins in detail:

Console plugin

The console plugin is the easiest. Open the file conf/checkErrors/plugins/console.properties, and you'll find:

```
plugin.class=net.javacoding.jspider.mod.plugin.console.ConsolePlugin

plugin.filter.enabled=false

plugin.config.prefix=[Plugin]
plugin.config.addspace=true
```

Nothing suprising... we'll just print out every event that passes, prefixed by [Plugin].

If you want JSpider not to be so verbose, simply turn on the event filtering and only let pass the events you are interested in!

Filewriter plugin

Remember that in the global event filtering (plugin.properties), we've filtered all spidering events so that only error events passed.

We're now going to write these errors down in a file report:

```
plugin.filter.enabled=true

plugin.filter.engine=
  net.javacoding.jspider.mod.even tfilter.Allow None EventFilter
plugin.filter.monitoring=
  net.javacoding.jspider.mod.eventfilter.Allow  None EventFilter
plugin.filter.spider=
  net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter

plugin.class=
  net.javacoding.jspider.mod.plugin.filew riter.FileWriterPlugin

plugin.config.filename=./error -report.out
```

The notion of JSpider progression is not wanted in our report, and neither are the events about the engine. So, we filtered out these in the filewriter's plugin local event filtering (see bold text)

The spidering events (those that passed the global event filtering – the error ones) are not filtered any further, all of them are used.

The filewriter plugin has one parameter, the **filename**. After the spidering process, you'll find this file in your JSpider's output folder.

StatusBasedFileWriter Plugin

Finally, we're configuring the StatusBasedFileWriter: open the file conf/checkErrors/plugins/statusbasedfilewriter.properties and you'll find:

```
plugin.class=net.javacoding.jspider.mod.plugin .statusbasedfilewriter.
StatusBasedFileWriterPlugin

plugin.filter.enabled=false
```

Which is very simple…
This plugin will create a file per http status encountered in the output folder:

- 404.out for all 'Not Found' errors
- 500.out for all 'Internal Server Errors'
- etc…

C. Example

OK! We have a complete JSpider configuration (checkErrors). Now let's use it and find some errors on a site!

For this particular purpose, we've set up a very small site for you to test JSpider upon.
You can find it at http://j-spider.sourceforge.net/samplesite.
Go have a look there with your favorite browser, you'll simply find a few interlinked web pages, one with an e-mail address on it, and one with a dead link.

This little playground can be used to test various JSpider configurations and to see the impact of configuration changes.

So, we'll start off spidering this site with the checkErrors configuration:

On windows:
jspider http://j-spider.sourceforge.net/samplesite checkErrors
On uniw:
./jspider.sh http://j-spider.sourceforge.net/samplesite checkErrors

1. Console output

This should give output on your console similar to this:

```
------------------------------------------------------------
JSpider startup script
JSPIDER_HOME=..
------------------------------------------------------------
JSpider v0.4.1 DEV (http://j -spider.sourceforge.net)
Build: 20030429
Started from .
[Engine] jspider.home=..
[Engine] default output folder=.. \output
[Engine] starting with configuration 'checkErrors'
```

This is the welcome message of our polite spider, echoing some information.

Next is the initialisation procedure for the plugins. You ll find the three plugins configured in the checkErrors configuration:

```
INFO  [core.impl.PluginFactory] Loading 3 plugins.
INFO  [core.impl.PluginFactory] Loading plugin configuration
'console'...
INFO  [mod.plugin.console.ConsolePlugin] Prefix s  et to '[Plugin] '
INFO  [core.impl.PluginFactory] Plugin not configured for local event
filtering
INFO  [core.impl.PluginFactory] Plugin Name    : Console writer
JSpider module
INFO  [core.impl.PluginFactory] Plugin Version : v1.0
INFO  [core.impl.PluginFa ctory] Plugin Vendor  :
http://www.javacoding.net
INFO  [core.impl.PluginFactory] Loading plugin configuration
'filewriter'...
INFO  [mod.plugin.filewriter.FileWriterPlugin] Writing to file:
./error-report.out
INFO  [core.impl.PluginFactory] Plugin uses lo  cal event filtering
INFO  [core.impl.PluginFactory] Plugin Name    : File writer JSpider
plugin
INFO  [core.impl.PluginFactory] Plugin Version : v1.0
INFO  [core.impl.PluginFactory] Plugin Vendor  :
http://www.javacoding.net
INFO  [core.impl.PluginFactory]  Loading plugin configuration
'statusbasedfilewriter'...
INFO  [mod.plugin.statusbasedfilewriter.StatusBasedFileWriterPlugin]
initialized.
INFO  [core.impl.PluginFactory] Plugin not configured for local event
filtering
INFO  [core.impl.PluginFactory] Plugi  n Name    : Status based
Filewriter JSpider plugin
INFO  [core.impl.PluginFactory] Plugin Version : v1.0
```

```
INFO  [core.impl.PluginFactory] Plugin Vendor  :
http://www.javacoding.net
INFO  [core.impl.PluginFactory] Loaded 3 plugins.
```

Then, the storage subsystemn is started:

```
INFO  [core.storage.StorageFactory] Storage provider class is 'class
net.javacoding.jspider.core.storage.memory.InMemorySto
```

The context takes care of user agent usage, proxy settings, etc...

```
INFO  [core.SpiderContext] default user Agent is  'JSpider v0.4.1 -dev
(http://j -spider.sourceforge.net)'
INFO  [core.task.SchedulerFactory] TaskScheduler provider class is
'class net.javacoding.jspider.core.task.impl.DefaultSche
```

The worker threads are created:

```
INFO  [core.Spider] Spider born  - threads: s piders: 5, thinkers: 1
[Plugin] Module : Console writer JSpider module
[Plugin] Version: v1.0
[Plugin] Vendor : http://www.javacoding.net
[Plugin] Spidering Started,
 baseURL = http://j -spider.sourceforge.net/sampleSite
INFO  [core.SpiderContext] using use rAgent 'JSpider v0.4.1 -dev
(http://j -spider.sourceforge.net)' for site 'http://localhos
```

Now, the spidering begins:

```
INFO  [core.SpiderContext] Using proxy for http://j  -
spider.sourceforge.net
INFO  [core.throttle.ThrottleFactory] Throttle provider class is
'class net.javacoding.jspider.core.throttle.impl.Distribut
[Plugin] Job monitor: 66% (2/3)  [S:50% (1/2) | T:100% (1/1)]
[blocked:0] [assigned:1]
[Plugin] ThreadPool Spiders occupation:20% [idle: 80%, blocked: 20%,
busy: 0%], size: 5
[Plugin] ThreadPool Th inkers occupation:0% [idle: 100%, blocked: 0%,
busy: 0%], size: 1
[Plugin] Job monitor: 85% (6/7)  [S:66% (2/3) | T:100% (4/4)]
[blocked:0] [assigned:1]
[Plugin] ThreadPool Spiders occupation:20% [idle: 80%, blocked: 20%,
busy: 0%], size: 5
[Plugin] Thread Pool Thinkers occupation:0% [idle: 100%, blocked: 0%,
busy: 0%], size: 1
[Plugin] Job monitor: 84% (11/13)  [S:60% (3/5) | T:100% (8/8)]
[blocked:0] [assigned:2]
[Plugin] ThreadPool Spiders occupation:40% [idle: 60%, blocked: 40%,
busy: 0%], size: 5
[Plugin] ThreadPool Thinkers occupation:0% [idle: 100%, blocked: 0%,
busy: 0%], size: 1
[Plugin] Job monitor: 84% (16/19)  [S:57% (4/7) | T:100% (12/12)]
[blocked:0] [assigned:3]
[Plugin] ThreadPool Spiders occupation:60% [idle: 40%, blocked: 60%,
busy: 0%], siz e: 5
[Plugin] ThreadPool Thinkers occupation:0% [idle: 100%, blocked: 0%,
busy: 0%], size: 1
```

JSpider finds our deliberately created dead link:

```
[Plugin] 404  - ERROR !!!http://j -
spider.sourceforge.net/sampleSite/unexisting.html
```

```
INFO  [mod.plugin.statusbased filewriter.StatusBasedFileWriterPlugin]
creating file for status '404'
[Plugin] Job monitor: 89% (17/19)  [S:71% (5/7) | T:100% (12/12)]
[blocked:0] [assigned:2]
[Plugin] ThreadPool Spiders occupation:40% [idle: 60%, blocked: 40%,
busy: 0%], size: 5
[Plugin] ThreadPool Thinkers occupation:0% [idle: 100%, blocked: 0%,
busy: 0%], size: 1
[Plugin] Job monitor: 95% (20/21)  [S:85% (6/7) | T:100% (14/14)]
[blocked:0] [assigned:1]
[Plugin] ThreadPool Spiders occupation:20% [idle: 80%, blocked: 20%,
busy: 0%], siz e: 5
[Plugin] ThreadPool Thinkers occupation:0% [idle: 100%, blocked: 0%,
busy: 0%], size: 1
[Plugin] Job monitor: 96% (24/25)  [S:87% (7/8) | T:100% (17/17)]
[blocked:0] [assigned:1]
[Plugin] ThreadPool Spiders occupation:20% [idle: 80%, blocked: 20%,
busy: 0%], size: 5
[Plugin] ThreadPool Thinkers occupation:0% [idle: 100%, blocked: 0%,
busy: 0%], size: 1
INFO  [core.Spider] Stopped spider workers...
INFO  [core.Spider] Stopped thinker workers...
```

When the process is finished, a summary is displayed:

```
 [Plugin]
SPIDERING SUMMARY :
known urls ............. : 8

  visited urls ........... : 7
    parsed urls ........... : 6
    parse ignored urls ..... : 1
    parse error urls ....... : 0

  not visited urls ....... : 1
    fetching ignored urls .. : 0
    forbidden urls ......... : 0
    fetch error urls ....... : 1

  not yet  visited urls .. : 0
[Plugin] Spidering Stopped
INFO  [core.Spider] Spidering done!
INFO  [core.Spider] Elapsed time : 7490
```

You see that we encountered 1 fetch error, and one resource that s not parsed to find new URLs in it (this will be the robots.txt file)

Now, this was only the logging part and the console plugin. These were configured to follow the progress while spidering.

More interesting is the contents of the output folder by now. You ll find the following files:

- 404.out – contains all URLs of resources that resulted in a 404
- Our error report

Remember that all resources that didn't result in an error are not known, as the global event filtering only passed events telling about errors.

The contents of these files:

2. 404.out

```
http://j-spider.sourceforge.net/sampleSite/unexisting.html
  REFERED BY:
  http://j-spider.sourceforge.net/sampleSite/
```

Bingo! We've got our error. Now it's up to us to determine whether the referenced URL is wrong, or the resource is missing…
We also get the URL of the page that contains the dead link, to make it even easier to track down the problem …

3. Error-report.out

```
[Tue Apr 29 20:24:05 CEST 2003] Module : File writer JSpider plugin
[Tue Apr 29 20:24:05 CEST 2003] Version: v1.0
[Tue Apr 29 20:24:05 CEST 2003] Vendor : http://www.javacoding.net
[Tue Apr 29 20:24:05 CEST 2003] Spidering Started,
  baseURL = http://j-spider.sourceforge.net/sampleSite
[Tue Apr 29 20:24:09 CEST 2003] 404  - ERROR !!!
    http://j-spider.sourceforge.net/sampleSite/unexisting.html
[Tue Apr 29 20:24:13 CEST 2003] Spidering Stopped
```

This file is a simple report of all errors reported by JSpider during the spidering process.

And indeed, if you browse to the page with your web browser, you'll easily find the dead link.

Now, this example was really trivial, but if you're responsible for a web site that contains a few thousand pages, you'd be very happy seeing JSpider creating new error reports for you in minutes!

IX. Scenario: Downloading a site to local disk

In this section, we'll use JSpider to download a website to the local file system. This usage is supported by the out-of-the-box configuration 'download'.

A. Goal

We'll use JSpider to spider a complete site (without 'hopping' to other sites), and download each and every web page, image, stylesheet, binary file, etc... to local disk, so we can view it off-line.

This task is preconfigured in JSpider, the configuration for this can be found under the folder conf/download.
You might want to make small changes to this configuration, to adapt it to your own situation.

While reading the explanation, it's maybe a good idea to refer to the part about JSpider configuration from time to time.

B. Configuration

Have a look in the conf/download folder in your JSpider installation folder to see what configuration files are involved.

1. Global configuration

We'll start off with the global configuration: jspider.properties:

Make sure you have the proxy settings put to the right values for your situation. See the configuration section for more information on this.

The most interesting part of the global configuration is the part with the rules:

```
jspider.rules.spider.count=1
jspider.rules.spider.1.class=
  net.javacoding.jspider.mod.rule.OnlyHt tpProtocolRule

jspider.rules.parser.count=1
jspider.rules.parser.1.class=
  net.javacoding.jspider.mod.rule.TextHtmlMimeTypeOnlyRule
```

Well... nothing special here... we're only fetching HTTP resources, and only parsing text/html web pages.

2. Site-specific configurations

We'll distinguish two types of sites, as shows from the sites.properties file:

```
jspider.site.config.base=base
jspider.site.config.default=skip
```

This allows us to scope the spidering process to the base site only, and we'll ignore any other site that might be referenced from the base site.
After all, we want to download/mirror a site – not the whole Internet!

This also means we'll find a base.properties and skip.properties file in the conf/download/sites subfolder:

Site configuration 'base'

This site configuration was assigned to the base site, and has nothing special in it.
The only important property is:

```
site.handle=false
```

Which tells JSpider the site should be handled.

Another change you'll want to make, is to **decrease the throttle interval**, to speed up the site downloading progress. See the Site Configuration section for more information about this topic.

Site configuration 'skip'

Any other site is assigned the skip configuration, which is quite simple:

```
site.handle=false
```

This disables the handling of the site altogether, even the robots.txt will not be fetched. Any resource in this site will simply be ignored. This is because we only want to download a single site!

C. Example

Now, let's download our little sample site:

```
jspider http://j-spider.sourceforge.net/samplesite download
```

This will give you similar output on the console as the checkErrors example in the last chapter.

D. Sample output

After the spidering process is complete, you can find in your output folder for the downloaded site:

```
- j-spider.sourceforge.net
      - robots.txt
      - samplesite
            - index.html
            - some
                  - folder
                        - test.html
```

Which is the structure of the sample site. Or at least, the part that is not forbidden by the robots.txt file, but we'll look into that later.

While the downloaded version of some sites (dynamic content, absolute URLs in web pages, etc…) may not be perfect, the **quality of the downloads** will be improved in the future, with the diskwriter plugin rewriting the links to other resources intelligently on-the-fly, and converting any special extensions (.jsp, .php, .asp, …) to .html.

This example should put you in the right direction to create your own custom download configurations for JSpider.

X. Scenario: Playing around with JSpider

Well, this JSpider usage scenario doesn't serve a particular purpose besides giving you an idea on how the configuration of JSpider can be applied to influence the spidering process.

We'll start of with the default configuration on our little sample test site, make some modifications, and analyse the differences in the results.

A. The default configuration

As said, we'll simply start by spidering our little test site with the 'default' JSpider configuration.

1. Configuration

This will result in output delivered by the following plugins:

- Console
- Velocity (both trace and dump)
- XMLDump (altered velocity plugin for XML reporting
- StatusbasedFilewriter (shows fetched URLs per HTTP status)

(see plugin.properties – they're configured there, and plugins/*.properties – the per-plugin config files).

You can start the spidering process on the sample site like this:

2. Starting

On windows:

```
jspider http://j-spider.sourceforge.net/samplesite
```

On Unix:

```
./jspider.sh http://j-spider.sourceforge.net/samplesite
```

We'll not show the resulting output on your screen here, since this would be far too long and not very interesting (just look at yours).

3. Output

Maybe the most interesting file to have a look at after the spidering process finishes is **velocity-dump.out**, which gives an overview of all known resources, with their status. This are some snippets from the file. (Just have a look at the one generated by your JSpider for the rest of the information):

These are the URLs inside the site found during spidering:

```
[Site: http:// j-spider.sourceforge.net  - ROBOTSTXT_HANDLED *]
  http:// http://j -spider.sf.net/samplesite/forbidden/resource.html
  http:// http://j -spider.sf.net/samplesite/unexisting.html
  http:// http://j -spider.sf.net/samplesite/index.html
  http:// http:// j-spider.sf.net/robots.txt
  http:// http://j -spider.sf.net/samplesite/some/folder/test.html
  http:// http://j -spider.sf.net/samplesite/
  http:// http://j -spider.sf.net/samplesite
```

We see that there is a resource in the site that has been forbidden by the robots.txt rule:

```
http://j-spider.sourceforge.net/samplesite/forbidden/resource.html
  STATUS :  FETCH_FORBIDDEN

  SPIDER DECISION :
  [DecisionStep] GENERAL rule
     net.javacoding.jspider.mod.rule.OnlyHttpProtocolRule   -
     ACCEPT  - (no comment given)
  [DecisionStep] GENERAL rule Ruleset  –
     ACCEPT  - ruleset final decision
  [DecisionStep] SITE rule
     net.javacoding.jspider.mod.rule.InternallyReferencedOnlyRule   –
     ACCEPT  - url is within same site  - accepted
  [DecisionStep] S ITE rule
     net.javacoding.jspider.mod.rule.ForbiddenPathRule   –
     DON'T CARE  - (no comment given)
  [DecisionStep] SITE rule
     net.javacoding.jspider.core.rule.impl.  RobotsTXTRule  –
     FORBIDDEN  - access forbidden
  [DecisionStep] SITE rule Rule set –
     FORBIDDEN  - ruleset final decision

  PARSE DECISION :
  [Not yet taken]
```

There is also a resource that is not there (a dead link, 404 error):

```
  http://j-spider.sourceforge.net/samplesite/unexisting.html
  STATUS :  FETCH_ERROR

  SPIDER DE CISION :
  [DecisionStep] GENERAL rule
     net.javacoding.jspider.mod.rule.OnlyHttpProtocolRule   –
     ACCEPT  - (no comment given)
  [DecisionStep] GENERAL rule Ruleset  –
     ACCEPT  - ruleset final decision
  [DecisionStep] SITE rule
     net.javacodi ng.jspider.mod.rule.InternallyReferencedOnlyRule   –
     ACCEPT  - url is within same site  - accepted
  [DecisionStep] SITE rule
```

```
    net.javacoding.jspider.mod.rule.ForbiddenPathRule   –
    DON'T CARE  - (no comment given)
  [DecisionStep] SITE rule
    net.javacoding.jspider.core.rule.impl.RobotsTXTRule   –
    DON'T CARE  - (no comment given)
  [DecisionStep] SITE rule Ruleset  –
    ACCEPT  - ruleset final decision

  PARSE DECISION :
  [Not yet taken]

  HTTP Status:  404
  REFERERS: 2
    http://j-spider.sourceforge.net/samplesite/index.html
    http://j-spider.sourceforge.net/samplesite/
```

You can also see that this resource is linked from the home page of our little testing site.

And this part describes the robots.txt file we fetched:

```
  http://j-spider.sourceforge.net/robots.txt
  STATUS :  PARSE_IGNORED

  SPIDER DECISION :
  [Not yet taken]

  PARSE DECISION :
  [Not yet taken]
```

You see this type of file fetched, but never parsed.

B. Forgetting about robots.txt

Now, since this test site belongs to use, we feel we can do what we want (and allow you to do so also).
Since there was one resource forbidden by the robots.txt file, we ll be not obeying the robots.txt file in order to get this resource spidered as well.

Change in the conf/default/sites/default.properties the following line:

```
site.robotstxt.obey=true
```

to

```
site.robotstxt.obey=false
```

It s as simple as that! This will still fetch the robots.txt file, but not obey it anymore.
Restart JSpider the same way as you did before and look at the results in velocity-dump.out:

The forbidden resource seems to have a link to a resource we didn t know yet, since there shows up a new one in the list:

```
  [Site: http://j-spider.sourceforge.net  - ROBOTSTXT_HANDLED *]
    http://j-spider.sf.net/samplesite/forbidden/resour ce.html
```

```
  http:// j -spider.sf.net /samplesite/unexisting.html
  http:// j -spider.sf.net /samplesite/index.html
  http:// j -spider.sf.net /samplesite/another/index.html
  http:// j -spider.sf.net /robots.txt
  http:// j -spider.sf.net /samplesite/so me/folder/test.html
  http:// j -spider.sf.net /samplesite/
  http:// j -spider.sf.net /samplesite
```

And if we look at the 'forbidden' resource in the detailed section now, we find:

```
http://j -spider.sourceforge.net/samplesite/forbidden/resource.html
STATUS : PARSED

SPIDER DECISION :
[DecisionStep] GENERAL rule
  net.javacoding.jspider.mod.rule.OnlyHttpProtocolRule   –
  ACCEPT  - (no comment given)
[DecisionStep] GENERAL rule Ruleset  –
  ACCEPT  - ruleset final decision
[DecisionStep] S ITE rule
  net.javacoding.jspider.mod.rule.InternallyReferencedOnlyRule   –
  ACCEPT  - url is within same site  - accepted
[DecisionStep] SITE rule
  net.javacoding.jspider.mod.rule.ForbiddenPathRule   –
  DON'T CARE  - (no comment given)
[DecisionS tep] SITE rule
  net.javacoding.jspider.core.rule.impl.RobotsTXTRule   –
  DON'T CARE  - (no comment given)
[DecisionStep] SITE rule Ruleset  –
  ACCEPT  - ruleset final decision

PARSE DECISION :
[DecisionStep] GENERAL rule
  net.javacoding. jspider.mod.rule.TextHtmlMimeTypeOnlyRule  –
  ACCEPT  - mimetype is 'text/html'  - resource accepted
[DecisionStep] GENERAL rule Ruleset  –
  ACCEPT  - ruleset final decision
[DecisionStep] SITE rule
  net.javacoding.jspider.mod.rule.BaseSiteOnlyR  ule –
  ACCEPT  - url accepted
[DecisionStep] SITE rule Ruleset  –
  ACCEPT  - ruleset final decision
HTTP Status: 200, Content size: 422,
Mime Type: text/html, Fetch time: 20

REFERERS: 1
  http://j -spider.sourceforge.net/samplesite/some/fol  der/test.html
REFERENCES: 1
  http://j -spider.sourceforge.net/samplesite/another/index.html
E-MAIL ADDRESSES: 0
```

Here, out expectations are confirmed: this resource has a link to a previous unknown resource!

C. Going not too deep

Now, suppose we're not interested in resources that are deeper then 2 levels in the site.
Let's configure a rule for that!
We're going to do this one the site level, although we also could have configured it on the global level (jspider.properties), then it would be applied on all sites.
Since we only spider one site, this doesn't matter for now.

Open the conf/default/sites/default.properties, and add a rule to the end:

```
site.rules.spider.count= 3

site.rules.spider.1.class=
    net.javacoding.jspider.mod.rule.InternallyReferenc  edOnlyRule

site.rules.spider.2.class=
    net.javacoding.jspider.mod.rule.ForbiddenPathRule
site.rules.spider.2.config.path=/content/javadoc

site.rules.spider.3.class=
    net.javacoding.jspider.mod.rule.BoundedDepthRule
site.rules.spider.3.config.depth. min=0
site.rules.spider.3.config.depth.max=2
```

This will cause any resource in a third-level folder or deeper to be ignored!

Now, spider again, and see that these resources are found:

```
  [Site: http://j -spider.sourceforge.net  - ROBOTSTXT_HANDLED *]
    http://j-spider.sf.net/samplesite/unexisting.html
    http://j-spider.sf.net/robots.txt
    http://j-spider.sf.net/samplesite/some/folder/test.html
    http://j-spider.sf.net/samplesite/
    http://j-spider.sf.net/samplesite
```

Now, let's look at the statuses:

- http://j-spider.sf.net/samplesite/unexisting.html
  - FETCH_ERROR, same as before
- http://j-spider.sf.net/robots.txt
  - PARSE_IGNORED, same as before
- http://j-spider.sf.net/samplesite/some/folder/test.html
  - FETCH_IGNORED, because of our rule:

```
  http://j -spider.sour ceforge.net/samplesite/some/folder/test.html
  STATUS :  FETCH_IGNORED

  SPIDER DECISION :
  [DecisionStep] GENERAL rule
     net.javacoding.jspider.mod.rule.OnlyHttpProtocolRule   –
     ACCEPT  - (no comment given)
  [DecisionStep] GENERAL rule Ruleset  –
    ACCEPT  - ruleset final decision
  [DecisionStep] SITE rule
```

```
    net.javacoding.jspider.mod.rule.InternallyReferencedOnlyRule   –
    ACCEPT  - url is within same site  - accepted
  [DecisionStep] SITE rule
    net.javacoding.jspider.mod.rule.ForbiddenPa  thRule –
    DON'T CARE  - (no comment given)
  [DecisionStep] SITE rule
    net.javacoding.jspider.mod.rule.BoundedDepthRule   –
    IGNORE  - depth is 3, higher than maximum 2
  [DecisionStep] SITE rule Ruleset  –
    IGNORE - ruleset final decision

  PARSE DECISION :
  [Not yet taken]
```

- http://j-spider.sf.net/samplesite/
  - FETCHED, same as before
- http://j-spider.sf.net/samplesite
  - FETCHED, same as before

These resources are missing this time, because they were only referenced from resources that are now ignored because of our new rule, or by other resources that are not found anymore:

- http://j-spider.sf.net/samplesite/forbidden/resource.html
- http:// j-spider.sf.net /samplesite/index.html
- http:// j-spider.sf.net /samplesite/another/index.html

This should have given you a good idea on how to tune your JSpider configurations for a particular use.

XI. Using JSpider-tool

JSpider-tool is a set of utilities **built on top of the JSpider** application. It has it's own set of configuration files, found under the folder 'conf/tool'.

JSpider-tool is meant to be a small and simple-to-use test- and **diagnostic tool**.
While JSpider on its own is used to spider complete sites, the usage of jspider-tool limits itself to **one web resource** only.

The things you can do with JSpider-tool are:

- Print out the **headers** sent by a web server
- Display **information** about a web resource
- Show the **content** of a web resource
- **Download** a certain file from a web server to a local file
- Find all **links** to other resources in a certain page
- Find all **e-mail addresses** mentioned in a web page

We'll cover these usages one by one

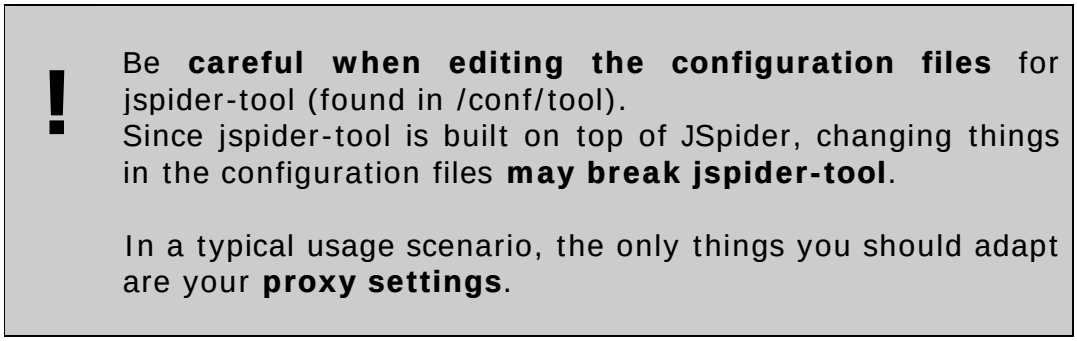
A. Usage

jspider-tool can be started from the same location as the JSpider application (/bin), and also from anywhere if JSPIDER_HOME is set:

windows:
```
jspider-tool (toolName) http://localhost
```

unix:
```
./jspider-tool.sh (toolName) http://localhost
```

Please remark that since jspider-tool is built on top of JSpider, you must adapt the jspider.properties file in 'conf/tool' to include your proxy settings.



B. Tools

There are several tools implemented in JSpider-tool.

1. headers

This utility prints out the headers sent by a web server, which can be very useful in order to understand what's being sent to web clients.

Some examples:

```
jspider-tool headers http://localhost
```

Results in something like this:

```
null:HTTP/1.1 200 OK
Date:Thu, 24 Apr 2003 13:34:11 GMT
Server:Apache/1.3.27 (Win32) PHP/4.3.2 -RC1
Last-Modified:Thu, 10 Apr 2003 13:15:45 GMT
ETag:"0-165-3e956e81"
Accept-Ranges:bytes
Content-Length:357
Keep-Alive:timeout=15, max=100
Connection:Keep-Alive
Content-Type:text/html
```

If there is a redirect on the resource, you'll see something like this:

```
null:HTTP/1.1 302 Found
Date:Thu, 24 Apr 2003 13:37:35 GMT
Server:Apache/1.3.27 (Win32) PHP/4.3.2 -RC1
X-Powered-By:PHP/4.3.2-RC1
Location:http://localhost/target.html
Keep-Alive:timeout=15, max=100
Connection:Keep-Alive
Transfer-Encoding:chunked
Content-Type:text/html
```

And if the server gave us a cookie to set, you'll also find that information in the output:

```
null:HTTP/1.1 200 OK
Date:Thu, 24 Apr 2003 13:42:01 GMT
Server:Apache/1.3.27 (Win32) PHP/4.3.2 -RC1
X-Powered-By:PHP/4.3.2-RC1
Set-Cookie:testCookie=someValue
Keep-Alive:timeout=15, max=100
Connection:Keep-Alive
Transfer-Encoding:chunked
Content-Type:text/html
```

As you see, inspecting the headers sent back by the web server can be quite informative!

2. info

The info utility is very similar to the headers utility, but gives a bit more information:

```
jspider -tool info http://localhost
```

```
URL          : http://localhost
HTTP Headers :
  null:HTTP/1.1 200 OK
  Date:Thu, 24 Apr 2003 13:44:13 GMT
  Server:Apache/1.3.27 (Win32) PHP/4.3.2 -RC1
  Last -Modified:Thu, 10 Apr 2003 13:15:45 GMT
  ETag:"0 -165 -3e956e81"
  Accept -Ranges:bytes
  Content -Length:357
  Keep -Alive:timeout=15, m ax=100
  Connection:Keep -Alive
  Content -Type:text/html
Mime Type    : text/html
Size         : 357
Time (ms)    : 60
```

It returns the URL fetched, the mime type, size of the content and the time it took to fetch the resource.

3. fetch

The 'fetch' utility does a real simple job: it fetches the requested resource and displays it's content.

```
jspider -tool fetch http://j -spider.sourceforge.net/robots.txt
```

results in the robots.txt file being printed out:

```
User -agent: JSpiderUnitTest
Disallow: /testcases/specific/ robotstxt/disallowedFolder2
Disallow: /testcases/specific/robotstxt/disallowedResource2.html

User -agent: JSpider
Disallow: /testcases/specific/robotstxt/disallowedFolder1
Disallow: /testcases/specific/robotstxt/disallowedResource1.html
```

4. download

The download utility downloads the requested resource and saves it in a file on the local filesystem.

```
jspider -tool download http://j -spider.sourceforge.net/robots.txt
  local_robots.txt
```

result:

```
Downloaded resource to 'local_robots.txt'  (304 bytes)
```

You ll find a file called local_robots.txt 'on your filesystem now.

5. findlinks

The findlinks utility fetches, parses and lists all links found in a certain resource.

```
jspider -tool findlinks http://j -spider.sourceforge.net
```

results in a list of all resources linked to by our project s main page:

```
http://j -spider.sourceforge.net/style/ns4_only.css
http://j -spider.sourceforge.net/style/maven_ns4_only.css
http://j -spider.sourceforge.net/style/print.css
http://www.javacoding.net
http://j -spider.sourceforge.net/images/ja karta -logo -blue.gif
http://j -spider.sourceforge.net
http://j -spider.sourceforge.net/images/blue -logo.gif
http://www.javacoding.net
http://www.sourceforge.net
http://j -spider.sourceforge.net/index.html
http://j -spider.sourceforge.net/docs/manual/index.html
http://j -spider.sourceforge.net/docs/manual/install/index.html
http://j -spider.sourceforge.net/docs/manual/install/binaries.html
http://j -spider.sourceforge.net/docs/manual/install/cvs.html
http://j -spider.sourceforge.net/docs/manual/install/ant.html
http://j -spider.sourceforge.net/docs/manual/install/testing.html
http://j -spider.sourceforge.net/docs/manual/usage/index.html
http://j -spider.sourceforge.net/docs/manual/config/index.html
http://j -spider.sourceforge.net/docs/manual/config/proxy.html
http://j -spider.sourceforge.net/docs/manual/config/threading.html
http://j -spider.sourceforge.net/docs/manual/config/sites.html
http://j -spider.sourceforge.net/docs/manual/config/site -props.html
http://j -spider.sourceforge.net/index.html
http://j -spider.sourceforge.n et/project -info.html
http://j -spider.sourceforge.net/maven -reports.html
http://j -spider.sourceforge.net/apidocs/index.html
http://j -spider.sourceforge.net/xref/index.html
http://jakarta.apache.org/turbine/maven/development -process.html
```

Resources that are linked several times are printed out each time.

6. email

The email tool works the same way as the findlinks tool, but reports all e-mail addresses found in the web resource:

```
jspider email http://j -spider.sourceforge.net
```

Issuing this statement will print out all e-mail addresses found in this web page.

Part

4

Configuration

XII. Environment

This section explains some **environmental configurations** that can/should be made in order to use JSpider.

You can skip this chapter at first, and come back later when you are more familiar with JSpider.

The only thing you ll need to do in order to be able to use JSpider, is **to configure an XML parser if you re using JDK1.3**.
Otherwise, no configuration issues explained here are absolutely necessary to do.

A. Java 1.3: XML parser Configuration

JSpider needs an XML Parser in order to read some configuration files that are specified in XML.
As of Java 1.4 (J2SE1.4), the needed XML API's and an implementation are already delivered by the platform.
For Java 1.3, you'll need to add the XML parser to the classpath in order to use JSpider.

If you don't do this, you'll see an exception like this when starting JSpider or JSpider-tool:

```
Exception in thread "main" java.lang.NoClassDefFoundError:
javax/xml/parsers/FactoryConfigurationError
(...StackTrace..)
```

You can download a free, open source XML parser from Apache, Xerces. You can find it at:

```
http://xml.apache.org/dist/xerces-j/
```

The CLASSPATH is a Java related environment variable that tells where the system can find needed classes.

The easiest thing to do is to add the xmlApis.jar and parserImpl.jar files to the {JSPIDER_INSTALLATION_DIR}/lib folder.
After that, add these libraries to the classpath:

Windows:
```
set CLASSPATH=%CLASSPATH%;../lib/xmlApis.jar
set CLASSPATH=%CLASSPATH%;../lib/parserImpl.jar
```

Unix:
```
export CLASSPATH=$CLASSPATH:../lib/xmlApis.jar
export CLASSPATH=$CLASSPATH:../lib/parserImpl.jar
```
When this is done, you can simply start JSpider from the 'bin' folder and the xml libraries should be found.

B. JSPIDER_HOME env. variable

In the section about the usage of JSpider, you've learned that you can start it by launching the jspider.bat or jspider.sh startup script. Without any extra configuration, this script assumes that your current working directory is {JSPIDER_INSTALL_DIR}/bin.

This can be problematic if you want to launch JSpider from another location.

When JSpider is started, it will look for an environment variable called JSPIDER_HOME. If this is present, that folder is taken as reference for the JSpider installation.
If this environment variable is not found, the value '..' is assumed, changing the current directory from the /bin folder to the {JSPIDER_INSTALLATION_DIR} if started from /bin.

Setting JSPIDER_HOME is very easy:

On unix:
```
export JSPIDER_HOME=/opt/jspider
```

On windows:
```
set JSPIDER_HOME=c: \jspider
```

make sure you use the path to which you installed JSpider.

XIII. Configuration overview

This section will explain how you can configure JSpider to adapt it to your needs. Most of the explanation will be related to the configuration of add-on components (plugins and other). While many standard components come with JSpider, it is possible to create your own, as described in the developer's guide.

JSpider configuration consists out of **different levels**:

- Common Configuration
- General configuration
- Site-specific configuration

A. Common configuration

The **common configuration** (which is currently limited to logging configuration), is the place for all system-wide configurations.

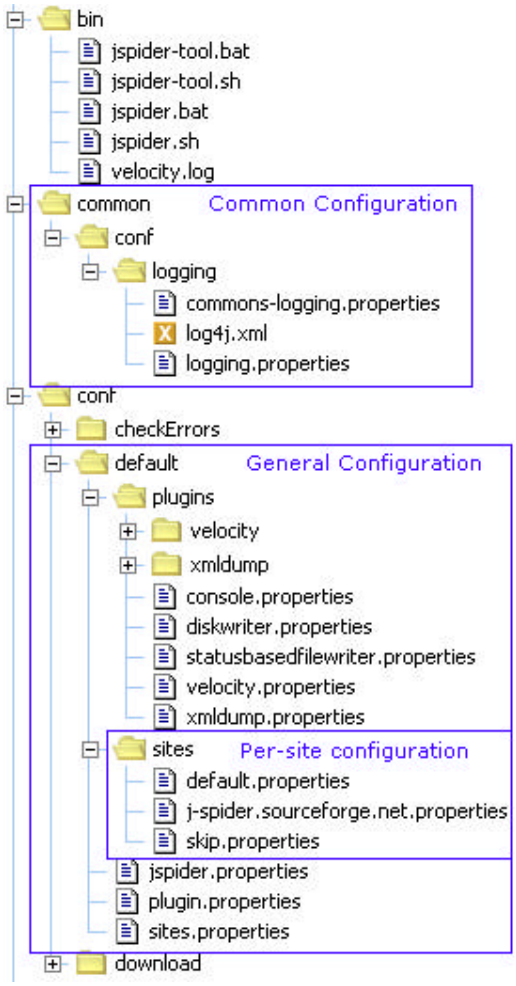
B. General configuration

The **general configuration** exists in several versions. Under the /conf folder in your JSpider distribution, you'll find several configurations. This way, you can create different configurations from which you can pick one when starting JSpider. This way, it is easy to keep different configurations for different purposes or environments.

C. Per-site configuration

Site-specific configurations are part of the general configuration, but are duplicated for one or more sites.
This way, you can use different settings of certain aspects of JSpider for different websites.
An example of this would be to use the proxy server to connect to a website on the internet, while using a direct connection for a site on the local LAN.

The locations where you can find these are shown in the picture below (the 'default' configuration is assumed – a similar file structure exists under 'download' and 'checkErrors' as well):

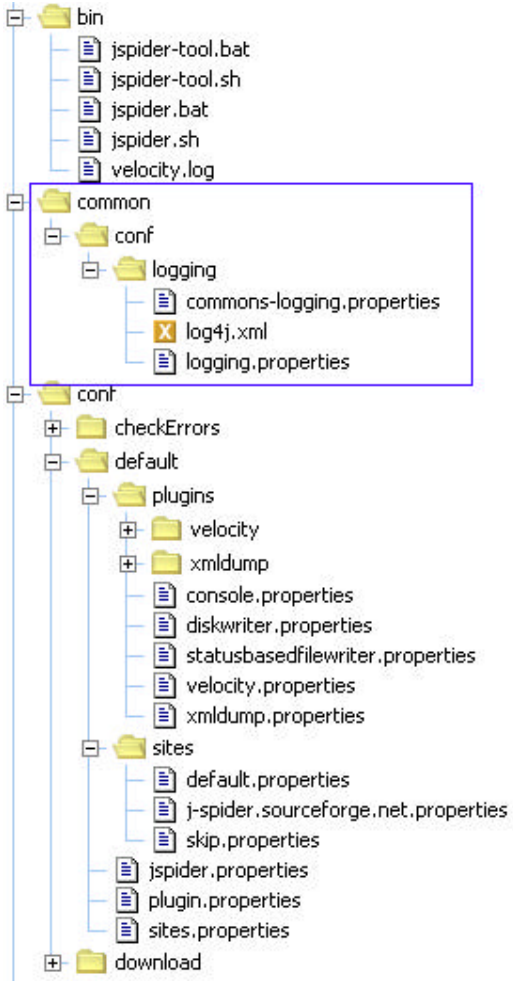


This information may still seem a bit abstract for now, but you ll understand better as we discuss each configuration topic in-depth.

XIV. Common configuration

The **common configuration** defines all JSpider behaviour that is common to all specifically created configuration sets.
For the moment, the only thing defined here is the logging system.

Under your JSpider distribution directory, you'll find a 'common/conf' folder, which is the folder in which all common configurations reside:



A. Logging subsystem

Logging in JSpider is done via Jakarta Commons-Logging, an open source wrapper implementation for logging systems. It implements a very basic logging service itself, but has an interface that can front any Logging framework.

When JSpider is installed, you're using the open source Jakarta **Log4j** logging framework (http://jakarta.apache.org/log4j) as the default logging system.

Another logging system that's supported out-of-the box by JSpider is **JDK1.4 logging** (javax.util.logging)

1. Logged items

Be aware that logging is only used for information about JSpider. It's **startup** procedure, eventual configuration or environmental **errors**, etc... are the things that are logged.

Spidering events and progression is **not logged**. These type of things are dispatched via the event system.
Plugins can then choose to write these things events down in a file or on the console.

It's very important to keep the distinction between what output is produced by plugins (although plugins can also log via the logging system), and what output is produced by the JSpider logging subsystem.

Try this by disabling the logging system. Change this line in your jspider.properties file:

```
jspider.log.provider=
  net.javacoding.jspider.core.logging.impl.CommonsLoggingLogProvider
```

To this one:

```
jspider.log.provider=
  net.javacoding.jspider.core.logging.impl.DevNullLogProvider
```

And you'll see what rests when the logging is turned off completely!

2. Configuration

The class that handles the logging is specified in the jspider.properties file. Normally, this is left on the default setting

(using commons-logging, which will decide on log4j or jdk1.4 logging):

```
jspider.log.provider=
  net.javacoding.jspider.core.logging.impl.CommonsLoggingLogProvider
```

You can disable the logging by changing this to:

```
jspider.log.provider=
  net.javacoding.jspider.core.logging.impl.DevNullLogProvider
```

Or simply log to the console straight away with:

```
jspider.log.provider=
  net.javacoding.jspider.core.logging.impl.SystemOutLogProvider
```

While the possibility to change this is there, normally commons-logging would be preferable.

3. Using Log4j

Since Log4J is the default logging system used be JSpider, you can just adapt the 'log4j.xml' file that configures it.

By default, we have configured a:

- *Console appender*
  Which writes logging info to your console (level INFO)
- *File appender*
  Which writes logging information to a 'log4j.out' file in your output folder (also level INFO)

These should give you a good basis to adapt the logging infrastructure to your needs.

Adapting the log4j configuration

How to do this is beyond the scope of this user manual (although we'll give some examples of configuration changes in a moment). Please refer to the log4j information for this. (See the project website for this information: http://jakarta.apache.org/log4j).

Configuration change example

When you use JSpider to spider a site with it's default configuration, you'll end up with output on the console that starts like this:

```
INFO  [core.impl.PluginFactory] Loading 4 plugins.
```

```
INFO  [core.impl.PluginFactory] Loading plugin configuration
'console'...
INFO  [mod.plugin.console.ConsolePlugin] Prefix set to '[Plugin] '
INFO  [core.impl.PluginFactory] Plugin not configured for local event
filtering
INFO  [core.impl.PluginFactory] Plugin Name    :   Console writer
JSpider module
INFO  [core.impl.PluginFactory] Plugin Version : v1.0
INFO  [core.impl.PluginFactory] Plugin Vendor  :
http://www.javacoding.net
INFO  [core.impl.PluginFactory] Loading plugin configuration
'velocity'...
INFO  [core.impl.PluginFactory] Plugin uses local event filtering
  --- continued ---
```

Now, open the log4j.xml configuration file, and search for this piece:

```
  <appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
    <param name="Threshold" value="INFO"/>
    <param name="Target" value="System.out"/>

    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%  -5p [%c] %m%n"/>
    </layout>
  </appender>
```

Change it to (bold shows the changes):

```
  <appender name="CONSOLE" class ="org.apache.log4j.ConsoleAppender">
    <param name="Threshold" value=" DEBUG"/>
    <param name="Target" value="System.out"/>

    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%  -5p {%c{1}} %m%n"/>
    </layout>
  </appender>
```

If you now start JSpider again, it will be quite more verbose, and reports in a slightly different way:

```
INFO  {PluginFactory} Loading 4 plugins.
INFO  {PluginFactory} Loading plugin configuration 'console'...
DEBUG {PluginInstantiator}  first trying to instantiate via ctr with
(name, config) params
DEBUG {ConsolePlugin} plugin 'console' prefix is '[Plugin]'
DEBUG {ConsolePlugin} adding space after prefix
INFO  {ConsolePlugin} Prefix set to '[Plugin] '
DEBUG {PluginInstantiator} plugin ins tantiated.
INFO  {PluginFactory} Plugin not configured for local event filtering
INFO  {PluginFactory} Plugin Name    : Console writer JSpider module
INFO  {PluginFactory} Plugin Version : v1.0
INFO  {PluginFactory} Plugin Vendor  : http://www.javacoding.n  et
INFO  {PluginFactory} Loading plugin configuration 'velocity'...
DEBUG {PluginInstantiator} first trying to instantiate via ctr with
(name, config) params
DEBUG {PluginInstantiator} plugin instantiated.
INFO  {PluginFactory} Plugin uses local event filt  ering
```

```
DEBUG {EventDispatcher} EventDispatcher for Plugin 'Velocity Template
JSpider module' configuring...
DEBUG {EventDispatcher} EventFilter for engine events =
net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
DEBUG {EventDispatcher} EventFilte r for monitor events =
net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter
DEBUG {EventDispatcher} EventFilter for spider events =
net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
DEBUG {EventDispatcher} EventDispatcher EventDispatcher fo  r Plugin
'Velocity Template JSpider module' configured.
  -- continued --
```

4. Using JDK 1.4 logging

You can enable JDK1.4 logging by simply removing the log4j.jar library from the JSPIDER/lib folder. This way, the commons-logging will not find log4j anymore, and fall back to JDK 1.4 logging.
This is of course only possible when you're using a VM of version 1.4 or higher.
You can customize the logging behaviour by adapting the confuration file 'logging.properties'.
Specifics on what you can do in this file can be found on the java.sun.com site.

XV. General configuration

General configuration each have a separate folder under the JSPIDER/conf folder.
You can pick the wanted one to use when starting JSpider by typing:

```
jspider http://localhost myconfig
```

or (Unix)

```
./jspider.sh http://localhost myconfig
```

Which will then search for a 'myconfig' folder under the /conf folder and use that one for the spider session.

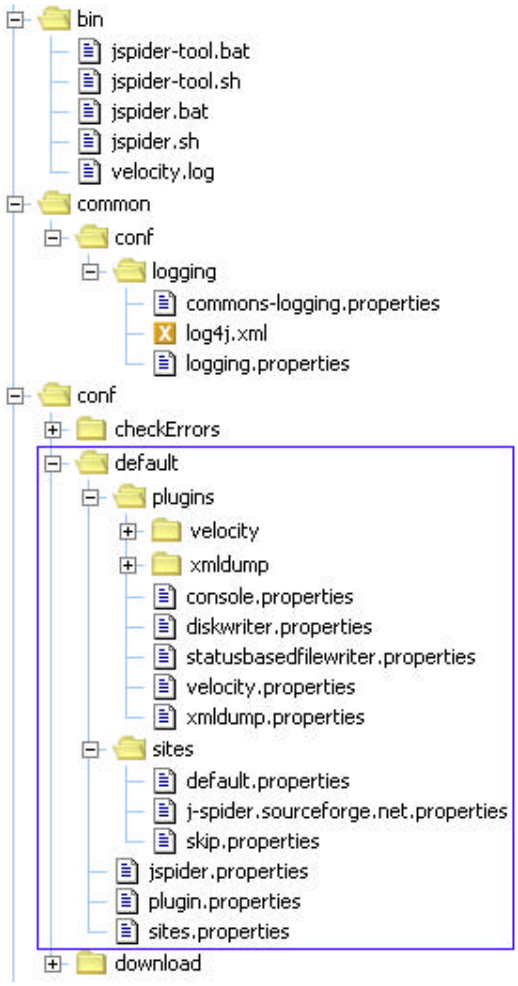
A. The 'default' configuration.

The default configuration (found under the directory /conf/default) is the one that is selected when you start JSpider without any configuration specified, for example:

```
jspider http://localhost
```

It is also a good example of a configuration to copy, and to use as a base for your own custom-made configurations.

The layout of the configuration folder for the 'default' configuration is shown below:



B. Other configurations

Next to the 'default' one, some other out-of-the box configurations come with JSpider (for instance, 'download' and 'checkErrors').

You can start these with:

```
jspider http://localhost download
```

And

```
jspider http://localhost  checkErrors
```

Respectively.

If you create your own folder (for instance: 'myconf') under JSPIDER_HOME/conf, you can start JSpider using this configuration by using:

```
jspider http://localhost myconf
```

It's always a good idea to copy an out-of-the box configuration (like 'default' or 'download' to another folder, and to start from there

instead of creating your won from scratch – many settings will be the same anyway!

C. Configuration Files

There several important files for the general configuration:

- jspider.properties
- sites.properties
- plugin.properties

jspider.properties contains the largest number of settings, and specifies every configuration this is applicable to all sites.

sites.properties links certain sites to certain site-specific configurations, which are put under the '**sites**' subfolder, and are explained further in this document.

plugin.properties defines which plugins will be used be JSpider to dispatch events to, and any filtering upon these events.
The actual per-plugin configuration is described in files underneath the '**plugins**' subfolder.

You'll find these inside each configuration folder (conf/default, conf/download, conf/checkErrors, ...).

Jspider.properties will be explained in this section, the other two are explained in the chapter about per-site configurations and plugin configuration respectively.

D. jspider.properties

The 'jspider.properties' file is the main configuration file for JSpider, and is located directly under the configuration folder of the specific configuration. (For instance, ${JSPIDER_HOME}/conf/default' for the default configuration.

This section will explain the customisations that can be done in this file.

For each section in the configuration file, we'll explain the purpose, and show the actual default configuration as found in a freshly installed JSpider instance.

1. Proxy settings

When spidering sites with JSpider, it is possible that you'll have to pass through a **proxy server** in order to reach sites on the internet.
It is even possible that you'll have to **provide a username and a password to authenticate** on the proxy server.
This can be configured in jspider.properties by changing these properties:

```
jspider.proxy.use=false
jspider.proxy.host=
jspider.proxy.port=
jspider.proxy.authenticate=false
jspider.proxy.user=
jspider.proxy.password=
```

The property **jspider.proxy.use** (which defaults to 'false') determines whether a proxy server should be used when doing http requests.
If this is set to true, you must also provide the **jspider.proxy.host** and **jspider.proxy.port** properties!

In case your proxy server needs authentication, you must also set the **jspider.proxy.authenticate** to 'true', and fill in the correct values for your **jspider.proxy.user** and **jspider.proxy.password**

A fictive example of such a configuration could be:

```
jspider.proxy.use=true
jspider.proxy.host=proxy.myisp.com
jspider.proxy.port=8080
jspider.proxy.authenticate=true
jspider.proxy.user=myaccount
jspider.proxy.passw ord=mypassword
```

When you're not sure about these settings, you can check your browser (internet explorer, netscape, mozilla, …) to see the correct settings for your proxy server.

Setting the wrong properties might result JSpider to seem to "hang" upon the first request, with a timeout after a long period.
If the proxy server and port is correct, but the authentication information is not, you would get an HTTP Status 407 (FORBIDDEN) upon each request.

In the per-site configurations (see later), you'll be able to define that the use of the proxy is not needed for certain sites (who reside on your local network and can be accessed in direct)

2. Threading

JSpider is designed to be **multi-threaded**. Multiple web requests are done in parallel, while previously fetched resources are inspected and interpreted.
There are **two groups of threads** (pools): **Spiders** and **Thinkers**.

Spider threads are the ones that will execute fetch commands (go out on the network and fetch data).

Thinker threads are used to interprete gathered data, apply rules, etc…

One thinker thread can keep several spider threads busy in a typical situation. By default the amount is 1 Thinker and 5 Spiders.
You can change the threading behaviour and it's monitoring by changing these properties:

```
jspider.threads.spiders.count=5
jspider.threads.spiders.monitoring.enabled=true
jspider.threads.spiders.monitoring.interval=1000
jspider.threads.thinkers.count=1
jspider.threads.thinkers.monitoring.enabled=true
jspider.threads.thinkers.monitoring.interval=1000
```

If enabled, the monitoring generates an overview of the thread pool occupation every x milliseconds, determined by the interval.
This event will then be received by plugins, who can show a progress bar, calculate the time elapsed and estimated spidering finish time, etc…

3. User Agent

The User-Agent sent with each HTTP request by JSpider can be changed.
The settings for this are commented by default, but can be uncommented for setting the User Agent to another value than the default one.
By default, the User Agent string looks like this:

(JSpider 0.4.0 DEV (http://j-spider.sourceforge.net)

You can change this by setting this property:

```
jspider.userAgent=JSpider ( http://j-spider.sourceforge.net )
```

Please note that there is also a setting site.userAgent on the site-level (see further) that can override this one in it's turn on a per-site basis.

You might want to change the JSpider user agent in the following cases:

- Test your dynamic site that generates different output for different user agents.
- To pass a very restrictive proxy server that only allows browsing sessions with a certain browser.

> **!** Please note that the User Agent is also used to determine which rules from a site's robots.txt apply. Changing the user agent can also change the obeyed rules!

Anyway, the user agent property is commented out by default, so the string compiled into JSpider, containing it's version, is used by default.

4. Rules

Resources are processed in different steps:

- Discovered when referred by another resource
- Spidered when fetched
- Parsed when interpreted and searched for new links

If this process went on forever, JSpider would ultimately index the whole internet. Because of this, you have to scope the spidering process.

This can be done by creating **rules** that determine which resources are eligible for spidering and/or parsing.

In the jspider.properties file, there are rules for both steps:

For the spidering (fetching) of resources:

```
jspider.rules.spider.count=1
jspider.rules.spider.1.class=
      net.javacoding.jspider.mod.rule.OnlyHttpProtocolRule
```

This makes sure that we only spider resources with the http:// protocol in front of the URL.

And the parsing rules:

```
jspider.rules.parser .count=1
jspider.rules.parser.1.class=
      net.javacoding.jspider.mod.rule.TextHtmlMimeTypeOnlyRule
```

Which makes sure that only fetched resources with a mime type containing "text/html" are parsed.
This way, no URLs are sought in images (gif,jpg,…), text, PDF, and other file types.

These rules are only on the **global level**, you can add additional rules on a per-site level basis.

Before a resource is fetched, it must first pass all **spider rules** on the global level, then all spider rules on the site level (taken from the per-site configuration assigned to the site related to the URL).

Before a resource is parsed, the same procedure is followed with the global and per-site **parsing rules**.

There are a number of Rule implementations that come with JSpider. It is also possible to implement your own. This is explained in greater detail in the developer manual.

An overview of some rules that come with JSpider:
(all in the package net.javacoding.jspider.mod.rule)

AcceptAllRule
: Lets all URLs pass

BaseSiteOnlyRule
: Only if the URL is part of the base site (site used to start the spidering process)

BaseURLOnlyRule
: Only if the URL is the same as the base URL (url used to start the spidering process)

BoundedDepthRule

Rule that accepts an extra min and max depth parameter. Only if the resource is at a minimum depth of <min> and a maximum folder depth of <max> in the site, it is accepted.
Parameters:

- depth.min
- depth.max

ExternallyReferencedOnlyRule

Only accepts an URL if it is referred to by a resource from another site

ForbiddenPathRule

Accepts a path (folder or folders) from which no resources may be retrieved – can be seen as an addition to robots.txt disallows
Parameters:

- path

InternallyReferencedOnlyRule

Only accepts URLs if they are references by resources from the same site

MaxNumberOfURLParamsRule

Only accepts URLs that have a maximum number of parameters (configurable) in their URL query string
Parameters:

- max

MaxResourcesPerSiteRule

Only accepts a limited number of resources (configurable via a parameter) from the same site
Parameters:

- max

NoURLParamsRule

Only accepts an URL if it has no HTTP GET parameters (Query String in the form of "?param=value")

OnlyDeeperInSiteRule

Only accepts a URL if it is in the same site but 'deeper' in the folder structure than the base URL, the URL used to start the spidering process

OnlyHttpProtocolRule

Only accepts a URL if it starts with "http://"

RejectAllRule
Rejects all resources

TextHtmlMimeTypeOnlyRule
Only accepts a resource if it's content type is text/html

Parameters can be passed to the rule like this:

```
jspider.rule.spider.<rule_nr>.config.<param_name>=<param_value>
```

An example:

```
jspider.rules.spider.count=2

jspider.rules.spider.1.class=
      net.javacoding.jspider.mod.rule.OnlyHttpProtocolRule

jspider.rule.spid er.2.class=
      net.javacoding.jspider.mod.rule.BoundedDepthRule
jspider.rule.spider.2.config.depth.min=3
jspider.rule.spider.2.config.depth.max=4
```

Which will cause any resource that is not on the third or fourth level in the site's folder hierarchy to be skipped for fetching.

URLs accepted would be:
http://somesite/one/two/three/file.html
http://somesite/first/second/third/fourth/index.html

URLs rejected could be:
http://somesite
http://somesite/index.html
http://somesite/first/second/third/fourth/fifth/index.html

By combining the rule sets on the global and per-site level, it is possible to scope the spidering process very good.

By making the distinction between rules for spidering and parsing, it is possible to fetch certain resources (to see whether they exist), without parsing them to look for new URLs.
A good example of this would be to check your site for any outbound links, check these to find any 404 errors (dead links), but not parse the external site pages, as this might lead us too far.
This way, you can simply check your site for dead links to external sites.

Please not that also the site.handle property in the per-site configuration can cause any URL from a site assigned that configuration to be skipped for spidering and parsing.

5. Storage

The object model for JSpider is backed by a Storage implementation. By default, JSpider uses an in-memory data store.

It is, however, possible to use a JDBC (database) store for this.

> **!** As of this writing, the **JDBC storage option is still experimental**, it has only been tested on MySQL 3, with the Connector/J MySQL JDBC Driver version 2.14.
>
> If you encounter any problems with the JDBC storage, other databases or driver versions, please file a bug report on sourceforge (see appendices at the end of this document for the bug tracker URL)

In order to enabled the JDBC storage option, comment out this line in your jspider.properties file (put a "# " in front of it):

```
jspider.storage.provider=
 net.javacoding.jspider.core.storage.memory .InMemoryStorageProvider
```

And uncomment the following:

```
jspider.storage.provider=
 net.javacoding.jspider.core.storage.jdbc.JdbcStorageProvider
jspider.storage.config.driver=com.mysql.jdbc.Driver
jspider.storage.config.url=jdbc:mysql://localhost/jspider
jspider.storage.config.user=
jspider.storage.config.password=
```

(Don't forget to adapt the settings to your needs)

This will give JSpider the instructions to use the JDBC storage option, with the connection information as in the properties.

First, you'll have to **prepare your database with the tables**. There's a script to do this in the JSpider CVS repository (res/jdbc). Any time you start JSpider, it will clean up all data in the tables.

You'll also need to have the **appropriate JDBC driver** in your classpath when you start JSpider.

You can **verify that the JDBC storage option was used** by examining the logs (In this example, the Log4J logging treshhold was set to TRACE):

```
INFO  [core.storage.StorageFactory] Storage provider class is 'class
net.javacoding.jspide r.core.storage.jdbc.JdbcStorageProvider'
DEBUG [core.storage.jdbc.DBUtil] jdbc driver = com.mysql.jdbc.Driver
DEBUG [core.storage.jdbc.DBUtil] jdbc user =
```

Of course, you ll also find you database filled with data after the spidering process.

Using the JDBC storage has some advantages:

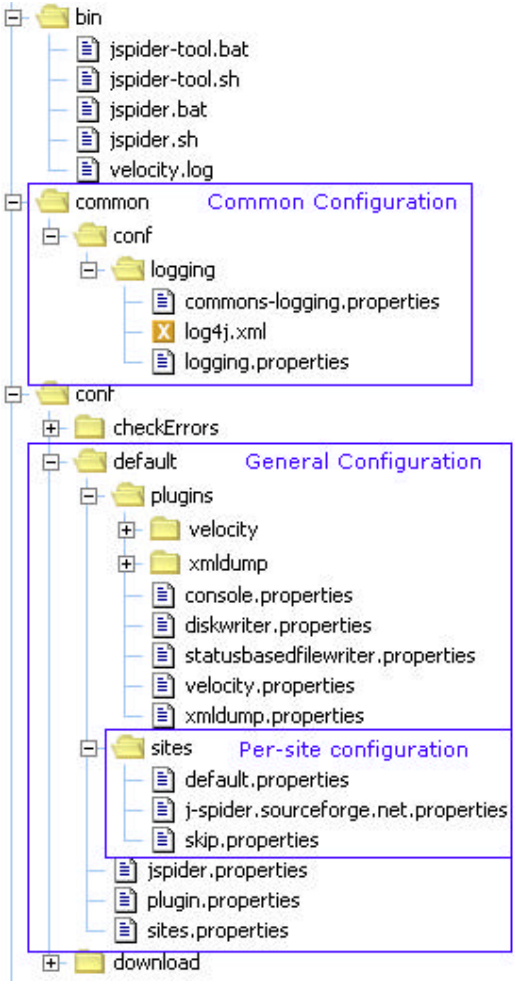
- ?? uses less memory, can spider larger scopes
- ?? you can query the database afterwards

There is, however, also the drawback of a performance hit in comparison to in-memory storage.

XVI. Per-site configurations

In 'sites.properties', you assign a per-site configuration based on the host and/or port of each site.
For each configuration specified in this file, you'll have to add a per-site configuration file used for handling all sites that are assigned this specific configuration.

In the picture below, you'll find three per-site configurations:



The three site configurations found here are:

- *j-spider.sourceforge.net.properties*
  This configuration will be used when spider our own site
- *skip.properties*
  This configuration will be used for sites that are out of scope for processing
- *default.properties*
  This one will be used for any other site

A. sites.properties

You can create as much different per-site configurations as you want. The only thing that you have to do, is to assign these configuration files to specific sites. This is done in **sites.properties**.

A snippet from the sites.properties file in the 'default' configuration:

```
jspider.site.config.base=default
jspider.site.config.default=skip

j-spider.sourceforge.net=j-spider.sourceforge.net
```

This means that the **base site**, the site from which JSpider starts (the one typed at the console when starting JSpider) will be handled according to the settings in default.properties.

The **explicitly mentioned** site j-spider.sourceforge.net gets it's very own configuration file.

Any site that doesn't get a separate config file assigned, will **default** to the config file 'skip' to be used.

If you assign a certain configuration in sites.properties to a site, you'll have to make sure that there's a matching properties file in the sites subfolder for that configuration.

For example, if you add a line:

```
www.google.com=anotherconf
```

Then you'll have to create a file named 'anotherconf.properties' in the 'sites' folder. All spidering actions regarding www.google.com will then be done according to the settings found in that file.

For the rest of this section, we'll explain the settings that are in a site-specific configuration file (the files under the 'sites' folder).

B. Site-specific configuration files

The names of the site-specific configurations files is your own choice, as you reference them from sites.properties.
You must put them under the 'sites' subfolder.

We'll now cover the different settings that can be configured in these type of files:

1. Site handling

You can configure JSpider to ignore certain sites at all. This is actually what the 'skip' configuration in the default folder does:

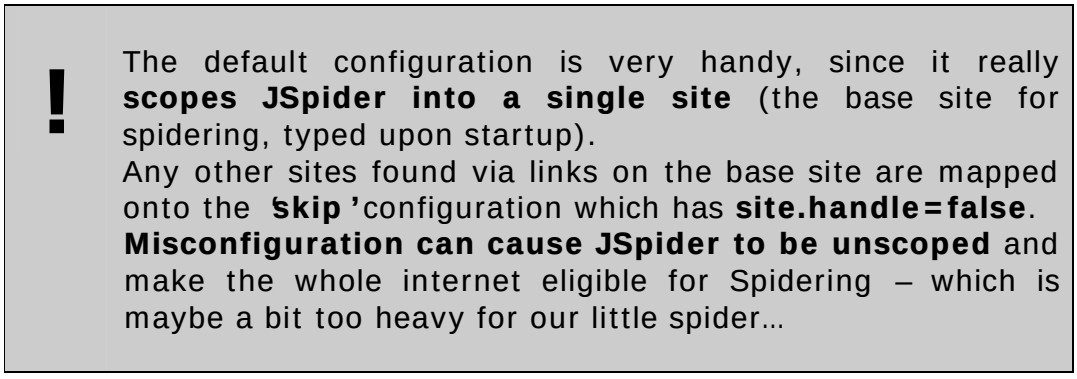
```
site.handle=false
```

this line makes sure that JSpider doesn't fetch any resource from the site (not even robots.txt), but ignores it right away.

The normal setting is:

```
site.handle=true
```

If you want to limit your spidering process to one or a few sites only, you can assigned these sites a certain configuration, while putting the default configuration to one that has it's site.handle set to false. (like the default configuration does):



The default configuration is very handy, since it really **scopes JSpider into a single site** (the base site for spidering, typed upon startup).
Any other sites found via links on the base site are mapped onto the 'skip' configuration which has **site.handle=false**.
Misconfiguration can cause JSpider to be unscoped and make the whole internet eligible for Spidering – which is maybe a bit too heavy for our little spider…

2. Robots.txt

As a well-behaving web robot, JSpider is configured to **obey any robots.txt file** present on a webserver. If there is no robots.txt file present, any resource is assumed to be accessible.
When a problem occurs fetching the robots.txt file other than the file being not present, any resource from the site is ignored for the rest of the spidering session.

The **matching between User-Agents specified in the robots.txt file** is very simple: if the User-Agent specification is a substring of the current spidering user agent, a match is found.

An example: When using the JSpider default user-agent:

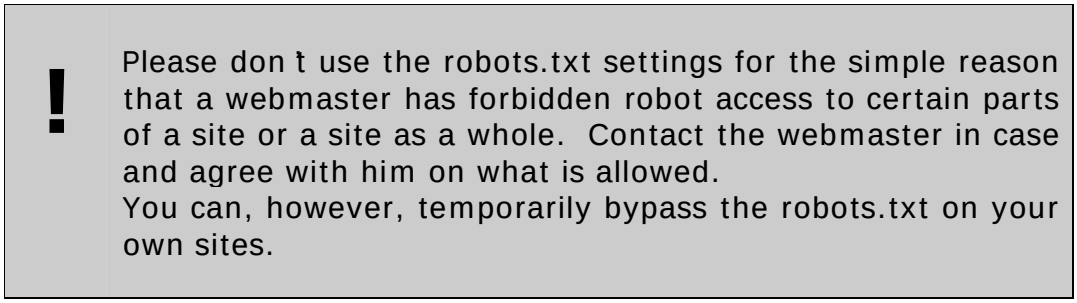
```
JSpider v0.5.0-DEV (http://j-spider.sourceforge.net)
```

The spider would obey the same string, "JSpider", "spider", "jspider v0.5", etc...
The matching is done in a case-insensitive way.

You can change the behaviour of JSpider towards robots.txt file handling in such a way that is never retrieved, or never obeyed.
Please use these modified settings only when spidering your own sites, as it disables the robots.txt support built into JSpider.

These properties control the behaviour:

```
site.robotstxt.fetch=true
site.robotstxt.obey=true
```



3. Throttling

JSpider can make many requests at the same time. Webservers, however, have a limited capacity in serving user requests. Also, when serving more requests at the same time, response times degrade.

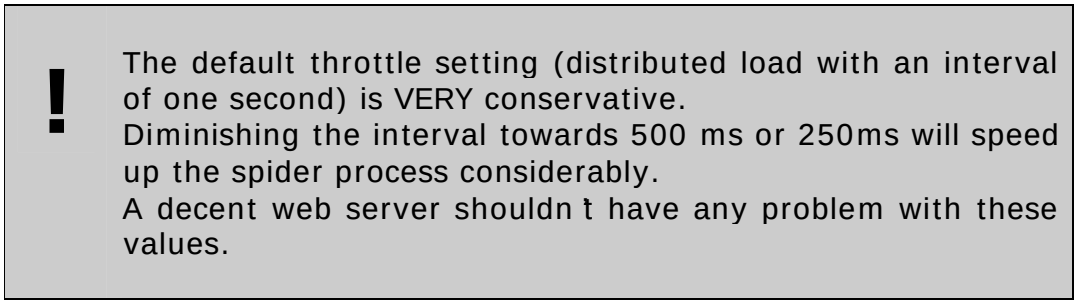
You should have control over **when and how often JSpider makes requests** to a web server.

This is exactly what the throttle component does. On a per-site basis, the throttle controls the spider threads, and blocks them if necessary until they're allowed to do a next request.

The default configuration in 'sites/default/properties' is:

```
site.throttle.provider=net.javacoding.jspider.core.throttle.impl.Dist
ributedLoadThrottleProvider
site.throttle.config.interval=1000
```

This will use the DistributedLoad throttle implementation that comes with JSpider, with an interval of 1000 milliseconds. The result of this configuration will be that there will be at **maximum 1 request per second** towards sites assigned this configuration.



In fact, if you leave this value as-is, your CPU usage will be almost zero when running JSpider, as most of the time will be spent waiting for a next timeslot to fetch.
There is, however, a hardcoded minimum of 250 milliseconds to avoid JSpider to do an accidental DOS attack on a webserver.

An alternative configuration would be:

```
site.throttle.provider=net.javacoding.jspi der.core.throttle.impl.Simu
ltaneousUsersThrottleProvider
site.throttle.config.thinktime.min=2000
site.throttle.config.thinktime.max=5000
```

This throttle implementation **simulates real users on the web site**. Each Spider thread assigned (see threading configuration) will become a virtual web surfer, which will be 'thinking' and reading between 2 and 5 seconds before doing another web requests towards the web server.

Please note that although **multiple sites can use the same configuration file** (defined in sites.properties), the throttling (and all other configured objects) are assigned on a per-site basis.
This means that two sites both using the 'default' configuration will both be throttled independently.

JSpider will not fetch more resources in a certain timeframe than allowed per site.

4. Proxy

The configuration of your proxy server is done in the general jspider.properties configuration file.
You can, however specify on a per-site basis whether this proxy should be used for a certain site. If you are spidering a site that is internal to your network, you could disable the use of the proxy server for this site:

```
site.proxy.use=true
```

enables the use of the proxy server for sites that are assigned this configuration, while

```
site.proxy.use=false
```

disables the use of the proxy server.

5. User Agent

The User-Agent sent with each HTTP request by JSpider can be changed.
The settings for this are commented by default, but can be uncommented for setting the User Agent to another value than the default one.
By default, the User Agent string looks like this:

(JSpider 0.4.0 DEV (http://j-spider.sourceforge.net)

You can change this by setting this property:

```
site.userAgent=JSpider ( http://j-spider.sourceforge.net )
```

Please note that there is also a setting jspider.userAgent on the global level (see above), but the setting on site-level can override that one.
See the jspider.userAgent configuration discussion for some important remarks regarding the changing of the user agent.

6. Cookies

The handling of cookies is configured on a per-site bases. This means that you ll have to decide whether cookies given by the server will be sent back with later requests.

The configuration is quite easy:

```
site.cookies.use=true
```

The default value is true, set to 'false' to disable cookie support for all sites assigned the specific configuration.

The biggest reason for JSpider to support cookies, is to allow all requests towards a certain site during a spidering session to be handled in one session.

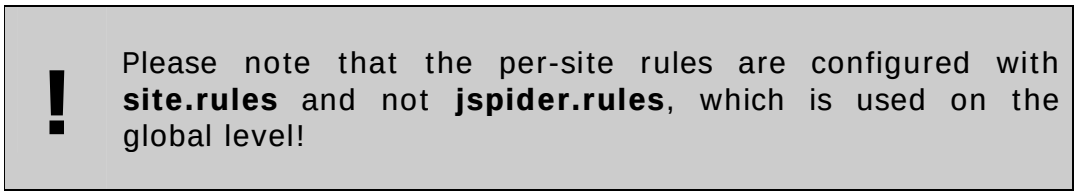
7. Rules

Just like on the global level, you can assign Rules to URLs to be spidered or parsed on a per-site level. These rules will then be tested after all general rules have passed.

For more information about how rules work and how they can be configured, please refer to the general configuration section.

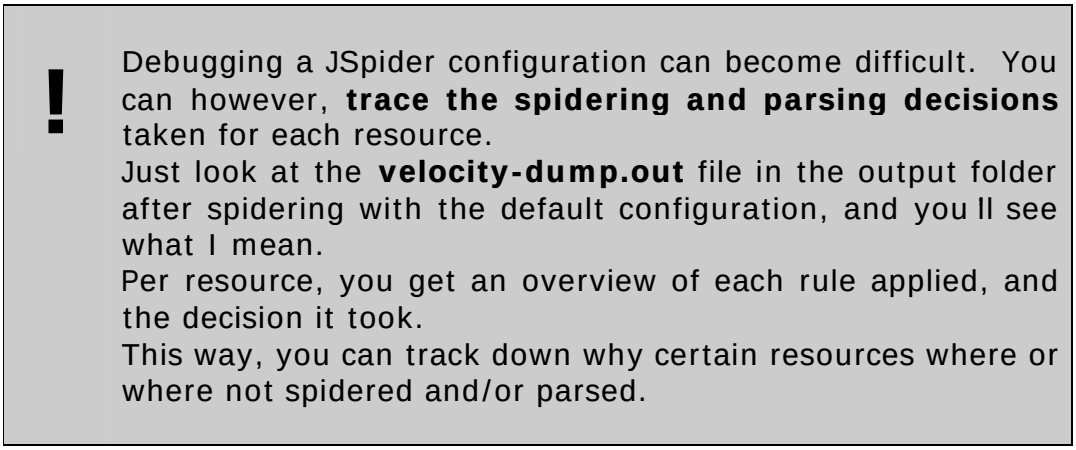
A snippet from sites/default.properties should get you on the right track:

```
site.rules.spider.count=2
site.rules.spider.1.class=
      net.javacoding.jsp ider.mod.rule.InternallyReferencedOnlyRule
site.rules.spider.2.class=
      net.javacoding.jspider.mod.rule.ForbiddenPathRule
site.rules.spider.2.config.path=/content/javadoc

site.rules.parser.count=1
site.rules.parser.1.class=
      net.javacoding.jspider.mod.rule .BaseSiteOnlyRule
```



This tells JSpider that resources should only be spidered (fetched) if they were referred by a resource on the same site, and that we're going to ignore all resources from the /content/javadoc directory.

Only resources from the base site will be interpreted (parsed), all other resource will not be inspected to find references to other sites and resources.



Debugging a JSpider configuration can become difficult. You can however, **trace the spidering and parsing decisions** taken for each resource.
Just look at the **velocity-dump.out** file in the output folder after spidering with the default configuration, and you ll see what I mean.
Per resource, you get an overview of each rule applied, and the decision it took.
This way, you can track down why certain resources where or where not spidered and/or parsed.

XVII. Plugin configuration

The **plugin** is the type of component that will be of the most interest to you.
It will receive **event notifications** during the spidering process, and has access to the **object model** of the spidered sites and resources.

A. Plugin.properties

The plugin.properties file is used to list all plugins that should be used in the configuration.

It also describes the filtering that should be applied on events before they are dispatched to any plugin.

These are the settings configured in the plugin.properties file for the default configuration:

```
jspider.filter.enabled=false
jspider.filter.engine=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
jspider.filter.monitoring=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
jspider.filter.spider=
      net.javacoding.jspider.mod.even tfilter.AllowAllEventFilter

jspider.plugin.count=4
jspider.plugin.1.config=console
jspider.plugin.2.config=velocity
jspider.plugin.3.config=statusbasedfilewriter
jspider.plugin.4.config=xmldump
```

1. Global event filtering

The first part concerns the **global event filtering**:
jspider.filter.enabled tells whether global event filtering should be applied. If put to false, the filters are not used.

The settings for **jspider.filter.engine**, **jspider.filter.monitoring** and **jspider.filter.spider** are classes that will be used for filtering the events of the corresponding type. You can use other implementations to filter the events that you don't want to reach any plugin.

The default event filters that come with JSpider are:

- net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
  Which lets all events pass
- net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
  Which blocks all events
- net.javacoding.jspider.mod.eventfilter.ErrorsOnlyEventFilter
  Which blocks all events except for those expressing an error

Event filtering can also be done on a per-plugin basis (see further)

So, if you want to suppress all monitoring messages, you can simply change the line:

```
jspider.filter.monitoring=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
```

to

```
jspider.filter.monit oring=
      net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter
```

and no more monitoring events will reach any plugin.

It is also very easy to implement your own event filters, which is explained in the developer manual.

2. Plugin definition

The second part of the file defines the plugins that should be used while spidering with the current configuration.
It is simply a list of names given to plugins. Each name must have a corresponding '<name>.properties' file inside the 'plugins' folder.
So, for the line:

```
jspider.plugin.2.config=velocity
```

There must be a file named 'velocity.properties' inside the 'plugins' folder.

B. Plugin configuration files

This type of configuration file is put in the 'plugins' folder, and can have any name. Since you define a plugin in plugin.properties, you should have a file with the name <plugin-name>.properties in the 'plugins' folder.

The content of a plugin configuration file is partly dependent on the plugin.
What you always need is this basis:

```
plugin.class=net.javacoding.j spider.mod.plugin.console.ConsolePlugin

plugin.filter.enabled=false
plugin.filter.engine=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
plugin.filter.monitoring=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
plugin.filter.spide r=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
```

1. Plugin implementation class

The **plugin.class** property defines which plugin class will be used for this plugin. In the example, this is a plugin implementation class named 'ConsolePlugin', which will print information in the console.

2. Local event filtering

The **plugin.filter** properties work the same way as the jspider.filter properties (see above), but filter events for this plugin only. Changes made here will not influence other plugins.

> **!** The filter properties on the plugin level are called **plugin.filter**, while on the global level (jspider.properties), the filter properties are called **jspider.filter**

3. Plugin parameters

The rest of a plugin configuration file is composed out of parameters needed by that particular plugin implementation.
A plugin writing a report to a file might need a filename, etc...
We'll explain the configurations of the default plugins in the next section.

C. Default plugins

Some default plugins come with JSpider. These are:

- *Console*
  A simple plugin that prints information in the console where JSpider is started

- *Velocity*
  A powerful plugin that writes report files according to customisable templates, based on the Jakarta velocity template engine.

- *FileWriter*
  A simple plugin (like the console plugin) that writes spidering information to a file

- *StatusBasedFileWriter*
  A plugin that organizes web resources according to the HTTP status (so you ll end up with a file called 200.out with all good resource, a file called 404.out, with all 404 error resources, a file named 301.out 'with all redirected resources, etc…

- *DiskWriter*
  A plugin that creates a file on the file system per fetched resource, and writes the resource content in it. This can be used to download web resources or event complete sites to your local disk.

Of course, you can also implement your own plugin classes. This is explained in detail in the developer guide.

1. Console Plugin

The console plugin is the simplest plugin, and is configured by putting this line as the setting for plugin.class in your plugin configuration file:

```
plugin.class=net.javacoding.jspider.mod.plugin.console.ConsolePlugin
```

Configuration

This is enough to configure the console plugins, but some customisations can be done. An example from the default configuration:

```
plugin.config.prefix=[Plugin]
plugin.config.addspace=true
```

The **plugin.config.prefix** defines the prefix that is put before each line printed out by the console plugin.
Whether or not a space should be added after the prefix is determined via the property **plugin.config.addspace**

Using different prefixes can be helpful when you configure to plugins to be a ConsolePlugin, to difference the output between the two.

Example

Let's say you want three different Console plugins:

- One for monitoring events
- One for the engine and spidering events
- One for all errors

You can create these by configuring your **plugin.properties** like this:

```
jspider.filter.enabled=false

jspider.plugin.count=3
jspider.plugin.1.config=monitoring
jspider.plugin.2.con fig=other
jspider.plugin.3.config=errors
```

This way, we don't filter any events on the global level, and we define three plugins.

Now we're going to create the needed file in the 'plugins' folder for these three plugins, namely:

- monitoring.properties
- other.properties

?? errors.properties

In **monitoring.properties** we ll put:

```
plugin.class=net.javacoding.jspider.mod.plugin.console.ConsolePlugin

plugin.filter.enabled=true

plugin.filter.engine=
      net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter
plugin. filter.monitoring=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
plugin.filter.spider=
      net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter

plugin.config.prefix=[Monitoring]
plugin.config.addspace=true
```

By doing so, all events will be filtered out (for this plugin), except for the monitoring events.
The output will be given fronted by the prefix "[Monitoring] ".

In **other.properties** we ll put:

```
plugin.class=net.javacoding.jspider.mod.plugin.console.ConsolePlugin

plugin.filter.enabled =true

plugin.filter.engine=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
plugin.filter.monitoring=
      net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter
plugin.filter.spider=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilte  r

plugin.config.prefix=[Other]
plugin.config.addspace=true
```

This way, all non-monitoring related events will be printed out with a prefix "[Other] ".

And finally, **errors.properties** will look like:

```
plugin.class=net.javacoding.jspider.mod.plugin.console.Co  nsolePlugin

plugin.filter.enabled=true

plugin.filter.engine=
      net.javacoding.jspider.mod.eventfilter.ErrorsOnlyEventFilter
plugin.filter.monitoring=
      net.javacoding.jspider.mod.eventfilter.ErrorsOnlyEventFilter
plugin.filter.spider=
      net.javacoding.jspid er.mod.eventfilter.ErrorsOnlyEventFilter

plugin.config.prefix=[Error]
plugin.config.addspace=true
```

Which will print any type of event that tells about an error situation fronted by the "[Error] " prefix.

Sample output

If we then spider a host where no web server is running, we get the following output (snippets – commented in between):
(Not the different prefixes that allow the distinction of the three plugin instances)

```
[Monitoring] Module : Console writer JSpider module
[Monitoring] Version: v1.0
[Monitoring] Vendor : http://www.javacoding.net
[Monitoring] Spidering Started, baseURL = http://localhost
```

This is the monitoring plugin presenting itself …

```
[Other] Module : Console writer JSpider module
[Other] Version: v1.0
[Other] Vendor : http://www.javac oding.net
[Other] Spidering Started, baseURL =  http://localhost
```

Then the "other" plugin is started.

```
[Error] Module : Console writer JSpider module
[Error] Version: v1.0
[Error] Vendor : http://www.javacoding.net
[Error] Spidering Started, baseURL =  http://localhost
```

Finally, also the "Error" plugin is started!

```
[Other] site discovered :  http://localhost
[Other] resource discovered: http://localhost
```

A new site (localhost) and new resource being discovered are spidering events, which are only accepted by the "other" plugin, due to the filtering for "error" and "monitoring".

```
[Monitoring] Job monitor: 0% (0/1)  [S:0% (0/1) | T:0% (0/0)]
      [blocked:1] [assig ned:1]
[Monitoring] ThreadPool Thinkers occupation:0%
      [idle: 100%, blocked: 0%, busy: 0%], size: 1
[Monitoring] ThreadPool Spiders occupation:20%
      [idle: 80%, blocked: 0%, busy: 20%], size: 5
```

These monitoring events are only received by the monitoring plugin, again due to event filtering…

```
[Other]
net.javacoding.jspider.api.event.site.RobotsTXTFetchErrorEvent
robots.txt was unreachable on site '[Site: http://localhost   -
ROBOTSTXT_ERROR *]'
[Error]
net.javacoding.jspider.api.event.site.RobotsTXTFetchErr  orEvent
robots.txt was unreachable on site '[Site: http://localhost   -
ROBOTSTXT_ERROR *]'
```

The failure to connect to the site (since I didn't start the webserver) is a spidering event, thus allowed to pass to the "other" plugin.

It is also an error event, so it also passes the filtering for the "errors" plugin.
It doesn't reach the monitoring plugin, however, since it is no monitoring event.

```
[Other] http://localhost  - Ignored for fetching
```

The notification that a certain resource will not be fetched is only received by the "other" plugin.

```
[Other]
SPIDERING SUMMARY :
known urls ............. : 2

  visited urls ........... : 0
    parsed urls ........... : 0
    parse ignored urls ..... : 0
    parse error urls ....... : 0

  not visited urls ....... : 2
    fetching ignored urls .. : 1
    forbidden urls ......... : 0
    fetch error urls ....... : 1

  not yet  visited urls .. : 0
```

Same goes for the spidering summary event, which tells us two URLs are known during the spidering process:

- http://localhost
- http://localhost/robots.txt

Of which the robots.txt caused a fetch error (the web server wasn't running in this example)
And the other (the original baseURL – http://localhost) is ignored for fetching because JSpider wasn't able to determine whether a robots.txt file is present

```
[Monitoring] Spidering Stopped
[Other] Spidering Stopped
[Error] Spidering Stopped
```

The event that the spidering has stopped is not filterable, so reaches every plugin.

This example showed how multiple instances of the same plugin class (Console Plugin in this case) can be combined.

2. Velocity Plugin

The velocity plugin is without doubt the most powerful that comes with JSpider.
Based on the open source template engine 'velocity' (http://jakarta.apache.org/velocity), it produces output according to a template on a per-event basis, as well as an overview report.

Each event that enters a velocity plugin instance, will trigger the rendering of output according to a template assigned for that event.

When the spidering is done, there is also a possibility to write a report from the (by then) **finished object model**.

Example

The velocity plugin configured in the 'default' configuration is a good example to see how it works.

In the plugin.properties file, you'll find a plugin definition:

```
jspider.plugin.2.config=velocity
```

Which will cause a plugin called 'velocity' to be loaded, according to the settings in 'plugins/velocity.properties':

```
plugin.class=
      net.javacoding.jspider.mod.plugin.velo city.VelocityPlugin
```

Should look familiar, this is simply the implementation class of the Velocity plugin.

```
plugin.config.templatefolder=velocity
plugin.config.trace.write=true
plugin.config.trace.filename=./velocity -trace.out
plugin.config.dump.write=true
plugin.config.dump.filename=./velocity -dump.out
```

This is the extra configuration needed for the velocity plugin, which we'll cover in a moment.

```
plugin.filter.enabled=true

plugin.filter.engine=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
plugin.filter.monitoring=
      net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter
plugin.filter.spider=
      net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter
```

This should also look familiar, as it is simply the local event filtering configuration for the plugin.
Note that we filter out all monitoring events, since we are only

interested in **data** in our velocity reports, not in spidering progress and system occupation information.

Configuration

The extra configuration needed for a velocity plugin are these properties:

```
plugin.config.templatefolder=velocity
plugin.config.trace.write=true
plugin.config.trace.filename=./velocity -trace.out
plugin.config.dump.write=true
plugin.config.dump.filename=./velocity -dump.out
```

Since the velocity plugin maps each incoming event to a template file, you'll have to define the name of the **subfolder** under 'plugins', from where the templates can be loaded.
By default, the **plugin.config.templatefolder** is named 'velocity'. This is thus also the name of the folder you'll find in the default configuration.

The property **plugin.config.trace.write** determines whether the velocity plugin should write a report for all incoming events during the spidering process. If this is true, the file in which the output is rendered is determined by the following property, **plugin.config.trace.filename**.

The property **plugin.config.dump.write** determines whether a dump of the object model should be rendered according to the template 'dump.vm' after the spidering is done.
If this property is set to true, the **plugin.config.dump.filename** is the name of the file written into.

In the default configuration, this is used to write an XML report of the sites spidered at the end of the process (see the xmldump plugin, which is a customized velocity plugin specially created for XML dumping the object model).

The best thing you can do is simply spider something small with the default configuration, and see the result in the two report files you'll find in the /output folder afterwards.

Creating templates

Creating velocity templates for generating the reports is quite easy: see the velocity site for the template language syntax.

Each event is mapped upon a template, so the event *net.javacoding.jspider.api.event.site.SiteDiscoveredEvent* will be rendered by the template: */conf/default/plugins/velocity/site/SiteDiscoveredEvent.vm, etc…*

The template used for the final dump of the object model is 'dump.vm'.

You can also use the original templates as a basis to create your own.

For event templates, the velocity context will contain :

- *eventName*
  the name of the event (short notation)
- *event*
  the event object (event class instance) that can be interrogated

For the dump template, the context will contain:

- *sites*
  collection of all sites encountered during spidering
- *resources*
  collection of all resource encountered during spidering

These objects can then be further interrogated (ex: get all folders in a site, get all resources in each folder, get all data about a resource, find all references to other resources, etc…)

Template example

As an example, we'll take the template for the ResourceReferenceDiscoveredEvent.

Since the class is net.javacoding.jspider.api.event.resource.ResourceReferenceDiscoveredEvent, we'll find the template in 'plugins/velocity/resource/ResourceReferenceDiscoveredEvent.vm':

```
[${eventName}] from '${event.resource.URL}'
            to '${event.referencedResource.URL}'
```

This can generate an output like this in the trace file:

```
[resource.ResourceDiscoveredEvent] from 'http://localhost'
            to 'http://localhost/ second.html'
```

Which gives you a very flexible way of writing spider reports.

3. FileWriter Plugin

The filewriter plugin is very similar to the console plugin, but it writes is output in a file, and not on the screen.
This plugin is not used in the default configuration included with JSpider, but can be added very easily.

Configuration

The configuration to be done for the FileWriter plugin is very simple. Only the target filename must be given:

```
plugin.config.filename=filewriter.out
```

This way, a file named 'filewriter.out' will be created in the JSpider output directory.

4. Status-Based FileWriter plugin

The status-based filewriter plugin is very handy to look for problems in a website.
Since it writes the URLs of resources in a file named after the HTTP status received when fetching that resource, it is easy to see which resources where found, which were not, which URLs lead to a redirect, etc...

An example of this plugin can be found in the "checkErrors" configuration:

```
plugin.class=
net.javacoding.jspider.m od.plugin.statusbasedfilewriter.StatusBasedFi
leWriterPlugin

plugin.filter.enabled=false
```

No other configuration is needed.

When you spider a site with this plugin enabled, you'll find files like these in the output folder afterwards:

- 200.out – All perfectly fetched files
- 301.out – All temporary redirects
- 302.out – All permanent redirects
- 404.out – All resources that couldn't be found
- 407.out – All resources that were forbidden
- 500.out – All resources that lead to an internal server error
- ... (other HTTP statuses)

Remark that the files are only present if at least one resource lead to the corresponding HTTP status.

When the HTTP status is about an error (like a 404), also the referring page is given.
This way, it is possible to distinguish between missing resources (should be present on the webserver but is not there), and dead links because of an error in the link (resource is there but the reference is incorrect).

5. DiskWriter Plugin

The diskwriter plugin can be used to download web pages to the local file system. Each time a successful resource fetch is notified to this plugin, it will write the content down in a file on the filesystem.

The configuration looks like this (taken from the 'download' configuration):

```
plugin.filter.enabled=true

plugin.filter.engine=
  net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter
plugin.filter.monitoring=
  net.javacoding.jspider.mod.eventfilter.AllowNoneEventFilter
plugin.filter.spider=
  net.javacoding.jspider.mod.eventfilter.AllowAllEventFilter

plugin.class=
  net.javacoding.jspider.mod.plugin.diskwriter.DiskWriterPlugin

plugin.config.output.absolute=false
plugin.config.output.folder=.
```

As you can see, we filtered out the engine and spidering events, which are of no use to this plugin.

The plugin-specific configuration properties for the download plugin might some extra explanation:

jspider.config.output.folder is the folder to which downloaded web pages should be saved. By default, this is a path relative to the 'output' folder.

If you use a dot('.') for this property (as in the given example), web pages will be downloaded to the output folder itself.

If you want to put an absolute file system path, you must set the **jspider.config.output.absolute** to true, which causes the folder to

be interpreted as an absolute path (eg: c:\downloads\sites or /var/jspider/sites).

After spidering with the DiskWriter plugin enabled, you will find the structure of the spidered site(s) under the target folder like this:

```
{FOLDER}/<sitename>/<folder>/resource.html
```

etc...

Part

5

Appendices

XVIII. Project info

This section contains global information about the JSpider project.

Official WebSite:
http://j-spider.sourceforge.net

Sourceforge.net project website:
http://www.sourceforge.net/projects/j-spider

Download page:
http://sourceforge.net/project/showfiles.php?group_id=65617

CVS repository (Anonymous access, read only):
(set your CVSROOT to this)
:pserver:anonymous@cvs.j-spider.sourceforge.net:/cvsroot/j-spider

Online CVS browsing:
http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/j-spider/

Forums:
We currently have three forums (Developer, Help en Open Discussion). These are the ones that are created by default on sourceforge:
http://sourceforge.net/forum/?group_id=65617

Bug tracking:
http://sourceforge.net/tracker/?group_id=65617&atid=511632

Feature requests:
http://sourceforge.net/tracker/?group_id=65617&atid=511635

XIX. Versioning

We distinguish different version types:

- ?? release builds
- ?? release candidates
- ?? development builds
- ?? CVS versions (not released as a download, only in CVS)

A. Release builds

The builds are considered to be stable and released as official JSpider versions.

Their name is formed as:

jspider-<Major>-<minor>-<revision>
For example: jspider-1-0-0

B. Release candidates

Before each major release, a JSpider version will first do some time as a release candidate, to ensure that everything is working fine. No real new functionality will be added to a release candidate before the actual release. Only bugfixes can be applied. When a release candidate is considered to be production quality, it is made a release build.

The name of a release candidate is constructed as follows:

jspider-<Major>-<minor>-<revision>-rc<number>
For example: jspider-1-0-0-rc3

C. Development builds

This type of version is an intermediate step between two version, with possibly many changes, new (mostly undocumented) new features, etc…
While we offer these for download to test, they cannot be considered stable.
It is however important that these are tested!

The name for a development build is constructed as follows:

jspider-<Major>-<minor>-<revision>-dev
For example: jspider-0-5-0-dev

Where the revision will be a pair number

D. CVS Versions

Each time a new release is done (development, release candidate or major release), a new version number is created for the CVS HEAD. This way, we can always distinguish between a CVS version (between two releases) and a released development version.

The name is constructed as follows:

jspider-<Major>-<minor>-<revision>-dev
For example: jspider-0-5-0-dev

Where revision is always odd.

For example, after the jspider-0-5-0-dev release, we will create a new version to work on in CVS, jspider-0-5-1-dev.
Once this one is good enough to be released, we will create a jspider-0-5-2-dev or jspider-0-6-0-dev development release.

So a version number with an odd revision is never released as a download, but is a version that comes straight from CVS, and is in between two official versions (thus possible unstable).

XX. History

This is a list of all jspider releases (both stable and development). The names are also the actual CVS tag names that can be used to check out a certain version.
(listed in reverse order, so most recent is on top)

jspider-0-5-0-dev (2003-05-01 – DEVELOPMENT)

- First version of User Manual (this doc)
- Folder-level model implementation
- Base Href support
- XML Reporting
- Email-address handling
- New default Rule implementations
- Several bugfixes, refactorings and smaller changes

jspider-0-4-0-dev (2003-04-06 - DEVELOPMENT)

- Preliminary JDBC storage
- New logging system
- DAO-based storage approach
- Velocity plugin

jspider-0-3-0-dev (2003-02-23 - DEVELOPMENT)

- Major refactorings
- Out-of-the-box download configuration
- Decent cookie support
- Http header interpretation

jspider-0-2-0-dev (2003-01-04 - DEVELOPMENT)

- Internal refactorings
- Task Scheduler introduction
- Several bug fixes
- New event filter system
- Functional JUnit tests
- Out-of-the-box checkErrors configuration

jspider-0-1-0-dev (2002-11-20 - DEVELOPMENT)

- Initial release
- Robots.txt support
- In-Memory storage of gathered data
- Basic plugin support

- Basic Rules implementation
- Basic event Filtering