JerkBot documentation

User and Developer Guide.

Yves Zoundi

JerkBot documentation: User and Developer Guide.

Yves Zoundi

Publication date 21/08/2009

Table of Contents

List of Figures

Part I. User Guide

Table of Contents

Introduction

This part will hopefully help you understand JerkBot, its features and how to get it running, and maybe how to develop with it.

Note

I apologize in advance for grammar and spelling mistakes, I am aware of it and I am fixing them time to time. The initial draft of the manual was written at late night. I only hope that most sentences make sense! :-)

1. Summary

JerkBot is an IRC Bot written in Java and built on top of Jerklib IRC library. Jerklib provides high level data structures, making it easy to use the library. That's why PircBot wasn't chosen.

2. JerkBot, from the user perspective

As other IRC bots, JerkBot has plugins and commands.

From a usage point of view, JerkBot has a *relatively advanced* session tracking mechanism and the registration is similar to online registrations on website(email, confirmation, etc.).

The security implementation, while minimal at the moment, has the potential to be a whole lot more sophisticated, to support multiple authentication modules at the same time(LDAP, Database, text files, etc.).

Note

There's a non working implementation for channel logging. It's difficult to write a sophisticated logging mechanism without taking into account most features that people would expect. People might want to distribute logs(ftp, sftp, nfs, etc.), saving logs(database, text files, search engine, etc.), generate HTML logs. People might also want to change scheduler settings for logs of a particular channel, while keeping default settings for other channels, etc.

Such a task would require in my opinion, an entire project for flexibility.

3. JerkBot, from a technical perspective

The major difference between JerkBot and other Java based bots, from a technical perspective, is that JerkBot is based on OSGI and it uses JMX for administration. JerkBot is not the first OSGI based bot, but looks simpler than others.

After releasing the bot source code, I noticed that the ECF project from Eclipse, has an IRC bot implementation. It also provides many things that I was planning to add, such as *messaging protocol abstraction*, but it looks limited from a “strict IRC usage/usability/features perspective”.

4. Managing multiple bots on multiple servers

This feature has not been added because it would add extra issues : usability, code complexity. Managing settings accross multiple servers would also mean:, sharing some settings, having some specific settings per connection/channel if possible, “muting” the bot on specific servers.

I. Features

JerkBot comes with many plugins. JerkBot without plugins has the following commands available :

- *help*: to list the commands available
- *version*: to display the version of the bot
- *describe*: to display the description of a command

In a channel, a command is prefixed by the *tilde* character to prevent the bot from interpreting every message.

```
~help
```

You can configure the default characters to use to replace the default *tilde* setting.

In a private message, the command prefix(tilde or any other character) is not interpreted . You would just type directly the command name in a private message.

```
help
```

1. Funny Commands Plugin

The Funny Commands Plugin provides the following commands :

- *google* : Plugins that uses Let me google that for you
- *chuck* : Funny quotes about Chuck Norris.
- *homer* : Random quotes from Homer Simpson.
- *brb* : Tells stupid reasons about why you're away and why.
- *ridicule* : Makes the bot laugh at someone.

2. Factoid Plugin

This plugin helps the bot learns stuff. It also makes available the following commands:

- *teach* : Teaches the bot something
- *forget* : Makes the bot forget an existing factoid
- *literal* : Provides some metadata about an existing factoid
- *info* : Gives the definition of a known factoid

3. Karma Plugin

Raise or lower the karma(i.e. popularity) of a subject. No restrictions to the subject(could be channel users, tools, etc.). It uses a simple filter for karma abuse.

The Karma Plugin provides the *karma* command.

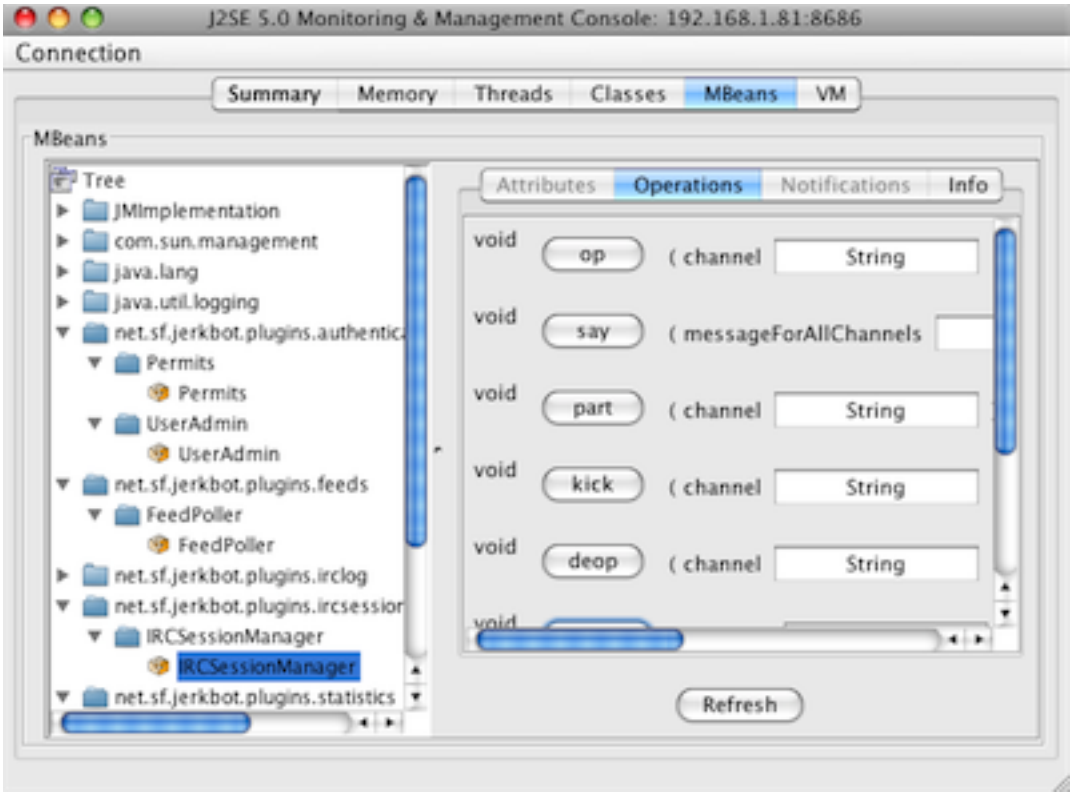
4. JMX Plugin

“To put it in a simple way, the plugin provides command wrappers that can only be used by authenticated users, through aliases, in private messages”.

The JMX plugin adds JMX support to bot. It registers JMX MBeans that are accessible to the bot using aliases. All MBeans exposed to JerkBot can be managed using JConsole.

The JMX plugin provides the *jmx* command.

Figure I.1. JMX connection with JConsole



All plugins that need to be configured by an administrative user are provided through the JMX command. **The JMX command requires authentication and can only be used in private messages**.

4.1. Listing available managed objects

Understanding the jmx command.

```
describe jmx
JMX management for JerkBot(Admin commands)

help jmx
jmx objects| jmx describe object| jmx methods object|
jmx attrs object| jmx object.attribute |
jmx object.attribute=value| jmx object.method(param1,...)
```

The JMX command uses a simple, programming language like, syntax. You'll notice it below.

To see which objects are available use:

```
jmx objects
```

With the default setup, the bot should answer:

```
Available managed objects are :
'bot', 'stats', 'log', 'svn', 'feeds', 'users', 'permits'
```

4.2. See the attributes(properties) of a managed object

Let's see the attributes of the *feeds* object :

```
jmx attrs feeds
```

The bot will answer

```
'PollingInterval' [The Feeds polling interval in seconds]
'FeedURL' [The feed URL]
```

4.3. Read an attribute value

The following command reads the PollingInterval value of the *feeds* object :

```
jmx feeds.PollingInterval
```

With the default settings the bot will answer

```
300
```

4.4. Change an attribute value

The following command changes the PollingInterval attribute of the *feeds* object :

```
jmx feeds.PollingInterval=800
```

The bot will answer

```
Operation Successfull!
```

4.5. See the methods of a managed object

To list all the methods available on the *feeds* object, do :

```
jmx methods feeds
```

The bot will answer

```
start()
stop()
status()
```

4.6. Call a method of a managed object

```
jmx feeds.start()
```

The bot will answer

```
Operation Successfull!
```

5. Authentication Plugin

5.1. Signin, Signout

To create a new session, run the following command

```
signin username password
```

If you don't use the JMX command for 30 minutes, your session will expire and you'll have to login.

To signout use

```
signout
```

.

The above commands can only be used in private messages.

Note

The default administrative user is **admin** with the password **admin**.

5.2. Registration process

- An administrator grants a permit to a user that allows him to register/confirm a pending registration. The *permit* is only valid for some minutes.

  ```
  jmx permits.createPermit(john, 192.168.1.30, 10)
  ```

  . The above command will grant a permit for registration to the user *john* whos hostname is *192.168.1.30*. The permit will be valid for *10* minutes.

  If you don't know the hostname, make a */whois* request.

- The user provides, in a private message, a username, password and email

  ```
  register john password john.doe@gmail.com
  ```

- After few minutes, he'll receive an email with detailed instructions to confirm his pending registration.

- The user will then confirm the pending registration, in a private message, by calling the *confirm* command:

  ```
  confirm john tokenid-xxxxx-xxxxxxx
  ```

- Upon successful registration the bot will send a private notice to the user.

5.3. Managing users and roles

In JerkBot, administrative users are part of groups. There are two base groups : *SuperAdmin* and *Admin*. A user in the *SuperAdmin* group can perform additional operations such as managing the users, terminating the bot, etc.

You can add and remove users, add users to groups, etc.

The users administration is provided through JMX by the *users* object

```
jmx users.listUsers()
```

Use the *jmx* command parameters to get more information about the *users* object.

6. IRC administration plugin

It provides some, but not all, IRC administrative operations : kick, ban, op, deop, join, part, quit, etc.

This plugin exposes an MBean called *bot* which is accessible through the *jmx* command.

7. RSS Plugin

The plugin fetches RSS Feeds at defined intervals and display the last news feed. It supports various feeds formats such as ATOM, RSS, RDF. The RSS plugin used a modified version of Commons FeedParser to avoid classLoader issues. Other Libraries such as Rome or Informa could have been used.

The RSS plug-in registers the *feeds* object through JMX. You can configure the polling interval, the URL and other settings.

8. SVN Plugin

The plugin fetches SVN revisions at defined intervals and display the last log message and revision. The SVN plugin uses the SVNKit library no system calls or no SVN installation is required.

The plug-in is provided through the *jmx* command with the alias *svn*. You can configure the polling interval, the URL, credentials and other settings.

9. Statistics Plugin

This plug-in displays OS Runtime statistics. It is provided through the *jmx* command with the alias *stats*.

```
jmx describe stats
...
jmx methods stats
...
jmx attrs stats
...
jmx stats.retrieveStats()
(FreeBSD 8.0-BETA3) (JDK 1.6.0_07) (Processors:1) (RAM:99.0%)
```

10. Javadoc Plugin

The plugin provides Javadoc search for the JDK API 5.0. It uses a pre-built Lucene Index of the API. It's probably faster to use a search engine than a database. Some people might have preferred to use Hibernate Search or Compas+JPA .

The Javadoc plugin provides the *javadoc* command.

```
~javadoc ArrayList
http://is.gd/2COV4
```

11. Weather Plugin

The Weather plugin calls a Web Service to get the weather for a given city in the USA.

This plugin provides the *weather* command.

```
~weather Atlanta
Weather for 'Atlanta': Sunny
```

II. Configuration

> **Note**
>
> In the source distribution, configuration settings are located in the `runtime` sub-directory!

1. Setting up the database

There's no setup needed for the current distribution. The default settings are populated on boot. The default administrative user is *admin* with the password *admin*.

The previous version of JerkBot used PostgreSQL but it might have been slightly more complicated to setup. That's why the distribution is shipping with an embedded database(HSQLDB). JerkBot has been tested on MySQL, PostgreSQL, Apache Derby and HSQLDB.

2. IRC settings

In the `etc` directory, edit the file `jerkbot.properties`. Change the default settings : IRC server, username, password, etc.

```
jerkbot.server=anthony.freenode.net
#comma separated channels to join upon successful connection
jerkbot.channels=##swing
jerkbot.name=mybot
jerkbot.nick=mybot
jerkbot.realname=mybot
jerkbot.greeting=Hello World, I'm here
jerkbot.password=mybot_optional_registered_password
#comma separated characters that will make the bot react
jerkbot.triggers=~
```

3. Email settings

In the `etc` directory, edit the file `mail.properties`. Fill the smtp properties for the server to use. The default settings are for gmail.

```
mail.user=MY_GMAIL_ACCOUNT@gmail.com
mail.from=MY_GMAIL_ACCOUNT@gmail.com
mail.password=MY_GMAIL_PASSWORD
mail.debug=true
mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
mail.smtp.socketFactory.fallback=false
mail.smtp.socketFactory.port=465
mail.smtp.port=465
mail.smtp.auth=true
mail.smtp.host=smtp.gmail.com
mail.smtp.starttls.enable=true
```

> **Note**
>
> You don't need to restart the application if you modify the email settings

III. Using the bot

1. Start the bot

> **Note**
>
> As a reminder, the default administrative user is **admin** with the password **admin**. You can remove the default user later using administrative commands.

1.1. Binary distribution

Go to the bin folder for jerkbot-bin distribution and run the command *./launcher.sh* or *launcher.bat* if you use Windows.

After connecting to IRC, the bot will join the channel(s) that you specified in the file `jerkbot.properties` previously.

> **Note**
>
> The binary distribution uses Apache Felix as OSGI container.

1.2. Source distribution

If you are using the source distribution, there's no launcher, use

```
mvn install pax:provision
```

Maven will compile and package the project artifacts and run the bot in Equinox OSGI container.

2. Getting help

When confused, use the *help* command

```
~help
```

If you're using the default distribution, you should see something similar to this

```
Available commands are: 'brb', 'chuck',
'confirm', 'describe', 'forget', 'google', 'help', 'homer',
'info', 'javadoc', 'jmx', 'karma', 'literal', 'register',
'ridicule', 'signin', 'signout', 'slap', 'teach', 'version',
'weather'. You can also ask me about ~help | ~help <command>
```

3. Information about a command

When you're curious about a command, use the *describe* command to get more information.

```
~help describe
~describe <command>
~describe info
Provides the definition of a factoid
~help info
~info <factoid>
~info Linux
I don't know anything about 'Linux'
```

4. Addressing a message to a user

```
~~john info Java
```

The bot knows that the tilde characters means that it needs to target the message. In this case, it will use the *info* command and attempt to resolve the *Java* factoid.

It will then answer to *john*.

```
john : Java is a programming language.
```

Thanks to *command resolvers*, explained in the developer guide, you can also type:

```
~Linux
```

Part II. Developer Guide

Table of Contents

Introduction

1. Target Audience

I assume the following things :

- You know how to use an IDE or write Java code in a text editor.
- You have used Maven as a developer or as a user to compile a library.
- You've used OSGI or read some articles about it, as I won't be explaining it.
- You're not a Java beginner as I won't be giving much definitions about Java technologies in General.

2. General note

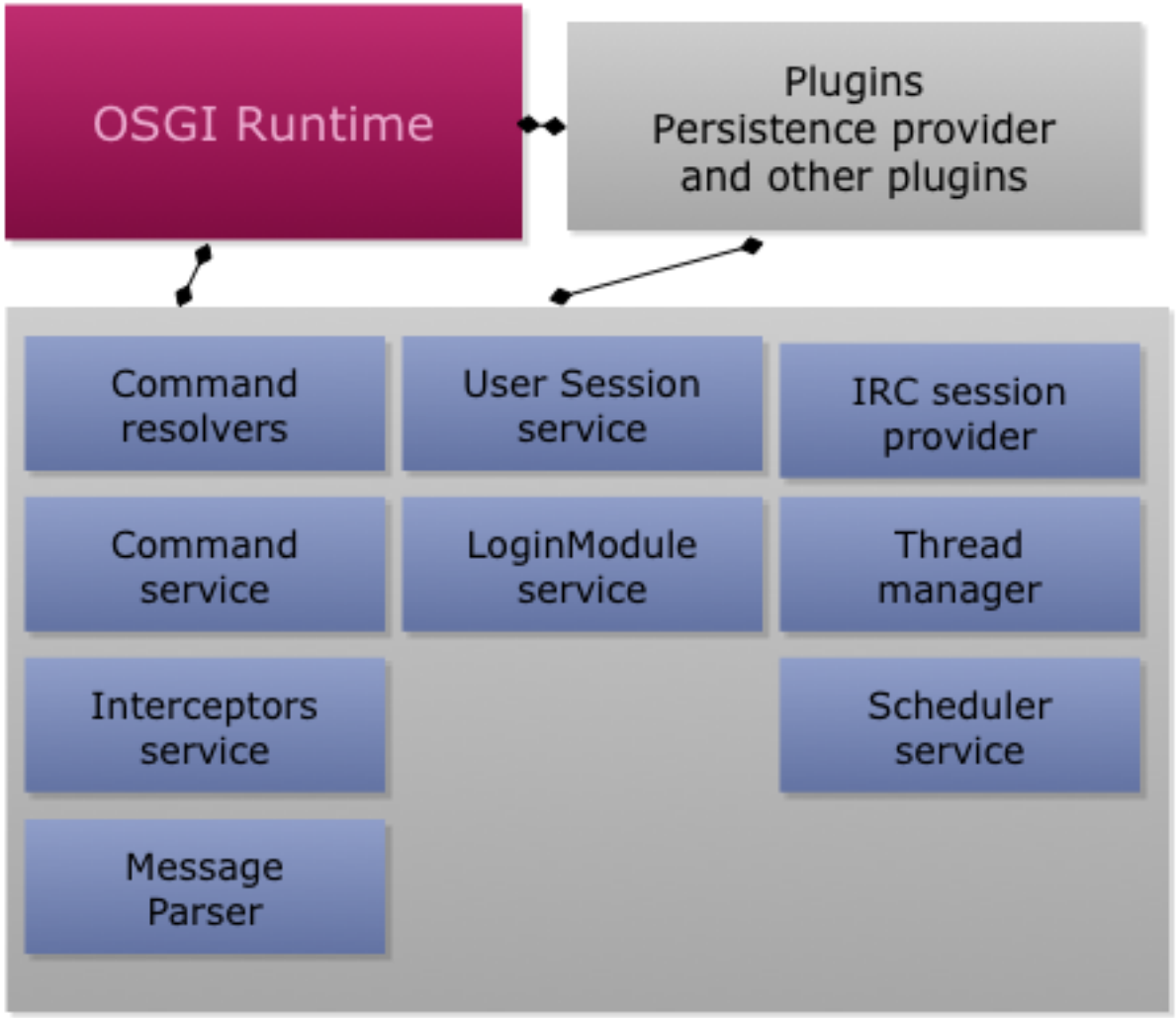
The bot has only been tested on Mac OS X 10.5.x, Linux(Slackware, Debian), FreeBSD. It should work on Windows too :-).

Note

In the binary distribution, the bot launchers are located in the `bin` directory of the distribution. In the source distribution, you would use Maven, by executing the *pax:provision* goal : *mvn install pax:provision*

IV. Technical information

1. OverView



2. The build system : Apache Maven

Apache Maven is JerkBot's build system. I believe that it's easier to manage OSGI projects and project dependencies in general with Maven. For sure, using Maven for very small projects is overkill.

Maven plays 2 goals here :

- Manage dependencies
- Build the project and package the project artifacts into OSGI bundles

Plugins are in the plugins pom project.

3. Main components and code structure

3.1. Logging

JerkBot uses SLF4J. The default download uses Log4J binding, but you could use any other binding available for SLF4J such as JDK *java.util.logging*.

The logging parameters for log files are configured in the launcher using system properties.

3.2. Exceptions

At the root of the exceptions hierarchy is `net.sf.jerkbot.exceptions.JerkBotException`. That exception extends `java.lang.RuntimeException` which means it's unchecked and won't prevent an operation to continue if possible. If you don't catch the exception, the bot will trap it and attempt to display an appropriate message.

There are 4 base exceptions that derive from `JerkBotException`, all located in the package net.sf.jerkbot.exceptions:

- `AccessDeniedException` : This exception is thrown to notify a user that he doesn't have access to a command. For example, a command might only be executed given one or all the following conditions : the message is private, the user is authenticated, the user is a member of a specific group.
- `CommandExistsException` : Exception thrown by the command service if a command attempts to register a prefix that is already *mapped*.
- `CommandSyntaxException` : Exception thrown by a `Command` when the syntax is invalid.
- `ConfigurationException` : Exception thrown by a component when a configuration parameter, setting or file is missing. That exception should provide useful information about how to fix the problem.

3.3. OSGI

JerkBot is built on top of OSGI to provide a modular architecture among other things.

The previous builds of the Bot used different approaches

- The first build of the bot was written using “traditional OSGI” with BundleActivator, etc. It was painful to write code, too verbose, but not so complicated.
- SCR was better, but most examples use XML component declarations.
- SCR also supports annotations and a maven plugin can generate the XML descriptors.
- IPojo. IPojo was ideal for JerkBot because of all its features(JMX support, etc.). However IPojo has some blocking limitations(abstract classes for services, etc.).

The choice was made to go with SCR annotations because it was easy, stable, mature and convenient. A partial rewrite of a previous build helped removing unecessary abstractions and simplifying the code.

3.4. Job Scheduling

The bot is bundled with Quartz Job Scheduler. The Job scheduler is used by :

- the session service to track user session expiration
- the *register* command to register new users. The pending registrations are placed in the job queue and deleted as soon as the registration is confirmed, or after a delay if the user doesn't confirm his registration.

There are some parts of the application that only use a ScheduledExecutorService.

3.5. Threading

The bot has a custom thread pool to manage requests and avoid consuming too many resources. When the bot is busy, awaiting requests are placed in a queue.

3.6. The message parser service

The message parser service creates an enhanced IRC message from the raw IRC message. It parses the raw IRC message and resolves the command to use, the person to answer to, etc.

Consider the following original irc message

```
sender=xxx, message=~~john info swing
```

It will be enhanced to this (omitting additional info)

```
sender=xxx,
operation=info,
target=john,
rawText=~~john info swing
message=swing
```

3.7. The command service

The command service reads the enhanced IRC message to extract the name of the command to call. It performs a lookup of registered commands by their name and executes the command that matches a prefix. There's no iteration or loop, as the commands are hold by a *map*.

3.8. The command resolver service

The command resolver is called whenever there's no command found after parsing the IRC message. As JerkBot is strictly command based, i.e. it doesn't iterate through available commands and for example match messages against regular expressions.

However, instead of typing *~info swing*, the user might prefer a shortcut such as *~swing*. That's what command resolvers provide in the command resolver service.

Warning

Creating a command resolver for the *jmx* command would introduce security concerns.

3.9. The session service

The session service tracks user sessions. It relies on the `LoginModuleService`(provided by the authentication plugin). You can think about it as an account session in a webmail account or a website. The session expires if you don't use administrative commands for a while.

No authentication plugin = No authentication(i.e no session) = No administrative tasks enabled.

3.10. The login module service

It provides authentication to the session service as well as JMX managed objects through the *jmx* command. The login module is provided by the authentication plugin. It was easier to make it a separated module instead of hardcoding many things directly.

With the *login module*, the bot knows how to authenticate users, associate them to roles, etc. Using *jmx* managed objects, you can administrate users and roles.

3.11. Persistence

JerkBot uses JPA(EclipseLink) because it's more suitable for OSGI development, easier to setup, etc. The previous versions of JerkBot used Hibernate and at the very beginning OpenJPA.

The persistence unit and the JPA entities are located in the Maven *persistence* module.

On startup the persistence 'plugin' configures the `EntityManagerFactory` in the utility class `net.sf.jerkbot.util.JPAUtil`, located in the *core* module.

If you want to add new JPA entities, you should add them in the *persistence* maven module to avoid classloading issues, unless you're introducing a new persistence unit.

The persistence module provides the following JPA entities:

- Factoid : Entity holding what the bot learned.
- Admin : Entity holding administrative users.
- Role : Entity holding administrative roles
- Karma : Entity holding popularity of subjects(channel users, tools, etc.)

There's no strict/common usage of the DAO pattern, no interface->implementation, just straightforward DAO. The bot uses few proprietary Eclipselink annotations, switching to another implementation will be trivial as there are not many DAOs.

V. Plugin example

1. Set up a maven project

I'm comfortable with the command line but it's like coding a medium/big project without an IDE. Use the command line at the beginning and switch to the IDE when you understand the concepts.

In Eclipse, I use the M2Eclipse plugin or Q4E. In IntelliJ, I just create a Maven project and same with NetBeans with the Maven plugin installed.

Note

If you prefer the command line follow this simple guide.

Warning

The maven pom configurations are not *optimal*, as not everything was cleaned up after major rewrites.

2. Setup the maven project

Create a new Maven Project. In the dependencies section of your pom.xml, add the following:

```
<dependency>
   <groupId>net.sf.jerkbot</groupId>
   <artifactId>net.sf.jerkbot.core</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   </dependency>
```

The module net.sf.jerkbot.core provides the base classes of the bot, so you need it.

Make sure that you've setup the project *packaging* type.

```
<project>
...
    <packaging>bundle</packaging>
...
</project>
```

The Maven Bundle plugin needs the *packaging* type to be set to that value to handle the OSGI manifest for you.

Setup the Maven Bundle Plugin and the Maven SCR Plugin to create the OSGI manifest for you. The Maven Bundle Plugin will add the OSGI headers to the jar archive manifest. The SCR plugin will auto-generate the XML descriptors for managed components and append some entries to the OSGI manifest.

```
<project>
...
<build>
 <plugins>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <configuration>
    <source>1.5</source>
    <target>1.5</target>
```

```
   </configuration>
  </plugin>
  <plugin>
   <groupId>org.apache.felix</groupId>
   <artifactId>maven-bundle-plugin</artifactId>
   <extensions>true</extensions>
   <configuration>
    <instructions>
     <Bundle-SymbolicName>${pom.artifactId}</Bundle-SymbolicName>
     <Bundle-Version>${pom.version}</Bundle-Version>
     <Bundle-Vendor>John Doe</Bundle-Vendor>
                    <Bundle-Name>My first plug-in</Bundle-Name>
    </instructions>
   </configuration>
  </plugin>
  <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-scr-plugin</artifactId>
            <version>1.2.0</version>
            <executions>
                <execution>
                    <id>generate-scr-scrdescriptor</id>
                    <goals>
                        <goal>scr</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
 </plugins>
</build>
...
</project>
```

If you don't manage to get the maven project setup, copy and paste the `pom.xml` from the *funny* module in the directory `plugins`, and customize it.

3. Creating your first command

This example is a *relatively simple HelloWorld* command, to greet people.

```
package com.mypackage;

import net.sf.jerkbot.commands.AbstractCommand;
import net.sf.jerkbot.commands.Command;
import net.sf.jerkbot.commands.MessageContext;
import net.sf.jerkbot.exceptions.CommandSyntaxException;
import net.sf.jerkbot.util.MessageUtil;

import org.apache.felix.scr.annotations.Activate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Service;

import org.osgi.service.component.ComponentContext;

@Component(immediate = true)
@Service(value = Command.class)
public class HelloCommandImpl extends AbstractCommand {
    private static final String COMMAND_HELP = "~hello";
```

```
    private static final String COMMAND_NAME = "HELLO";
    private static final String COMMAND_DESCRIPTION = "Say hello";

    public MyCommandImpl() {
        super(COMMAND_NAME, COMMAND_DESCRIPTION, COMMAND_HELP);
    }

    @Activate
    protected void activate(ComponentContext componentContext) {
       System.out.println("Activated HelloComponent");
    }

    public void execute(MessageContext context)
                    throws CommandSyntaxException {
        MessageUtil.say(context, "Hello");
    }
}
```

4. Create the OSGI bundle

Run *mvn install* in the project folder to generate the jar file. It will be located in your `target` folder.

Note

If you only want to setup the binary distribution, copy the resulting jar in the `libs` folder of the *jerkbot-bin* distribution.

5. Setup the bot to recognize your plugin

If you're running the source distribution, run the command below and ignore the rest of this section.

```
mvn install pax:provision
```

The rest of this section is relevant, only if you're using the binary distribution of jerkbot.

Now that you've created the jar archive, you need the bot to be aware of it, it's not a *"drop a file here and we're good"* .

Edit the etc/config.properties file from *jerkbot-bin* distribution. Add a separator at the end of the last line and register your jar.

```
....
file:../libs/net.sf.jerkbot.plugins.meteo-0.0.1-SNAPSHOT.jar \
file:../libs/MY_OSGI_JAR_FILE_GOES_HERE.jar
...
```

Go to the bin folder for jerkbot-bin distribution and run the command *./launcher.sh*

6. Test your plugin

After launching the bot, fire up your IRC client to connect to the network your bot is on and join a channel where the bot is.

Try out your plugin using the instruction:

```
~hello
```

If everything went fine, the bot should answer:

```
yournick : Hello
```

```
~~john hello
Hello john
```

The launcher is running in DEBUG mode by default to get lots of details about errors and components loaded. If something went wrong, look at the console and the logs located in the `logs` folder of the jerkbot-bin distribution(`runtime/logs` for the source distribution).

Warning

It's better to test the bot in a random channel such as *#abcde* where there's nobody. Some IRC channels don't tolerate the fact that you bring your own bot(disturbing the peace heh!). That's especially relevant when there's already a bot on the given channel.

References

IRC protocol specification (RFC1459). http://www.irchelp.org/irchelp/rfc/rfc.html .

JerkLib Website. http://jerklib.sf.net .

Some IRC bots. http://www.dmoz.org/Computers/Software/Internet/Clients/Chat/IRC/Bots .

OSGi Alliance | Specifications / HomePage. http://www.osgi.org/Specifications/HomePage .

OSGi articles. http://neilbartlett.name/blog/osgi-articles/ .

Declarative Services Dependency Injection OSGi style. http://www.slideshare.net/fmeschbe/declarative-services-dependency-injection-osgi-style .

Apache Felix Website. http://felix.apache.org/ .

IPOJO Website. http://felix.apache.org/site/ipojo-concepts-overview.html .

Pax Runner - OPS4J. http://paxrunner.ops4j.org/space/Pax+Runner .

Apache Maven. http://maven.apache.org/ .

Maven BND Plugin. http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html .

Maven SCR Plugin. http://felix.apache.org/site/apache-felix-maven-scr-plugin.html .

Maven PAX Plugin. http://www.ops4j.org/projects/pax/construct/maven-pax-plugin/ .