# CRaSH

***CRaSH guide***

**Julien Viet**

eXo Platform

**Table of Contents**

**List of Examples**

# Preface

The Common Reusable SHell (CRaSH) deploys in a Java runtime and provides interactions with the JVM. Commands are written in Groovy and can be developped at runtime making the extension of the shell very easy with fast development cycle.

# 1

# Running CRaSH

There are several ways to run CRaSH, as a standalone application it controls its own JVM or as an embedded service in an existing runtime like a web application or a Spring application.

## 1.1. Standalone

### 1.1.1. Standalone mode

The standalone mode allows you to run CRaSH from the command line directly. It provides the same functionality as the war deployment but does not require a web container as it runs its own virtual machine. The crash directory in the application contains the standalone distribution.

The bin directory */crash/bin* can be added to the system path, it contains the *crash.sh* script that will start the standalone mode, for instance you can set it up this way:

```
> export PATH=/.../crash/bin:$PATH
> crash.sh
   ______
 .~      ~. |`````````,       .'.                    ..'''' |         |
|           |'''|'''''      .''```.               .''       |_________|
|           |    `.       .'       `.          ..'          |         |
 `.______.' |      `.    .'          `. ....''              |         | 1.0.0-c

Follow and support the project on http://vietj.github.com/crash
Welcome to jerry + !
It is Thu Apr 12 21:19:35 CEST 2012 now
```

Let's review quickly what you can find in standalone crash:

- The *bin* directory contains the *crash.sh* script and the standalone crash jar file
- The *conf* directory contains the configuration properties *crash.properties* and JVM logging configuration *logging.properties*
- The *cmd* directory contains the commands that will be available in crash by default it contains a few example commands
- The *lib* directory contains the various libraries used by crash, you should place additional jar files there

### 1.1.2. Attach mode

The attach mode allows you to attach CRaSH to a JVM located on the same host with the attach API provided by the Hotspot JVM. It is the standalone mode attached to a running JVM specified by a process id. CRaSH will hook into the targetted JVM instead of the JVM started by CRaSH. Let's see quickly an example of how to use it

```
> jps
3165 RemoteMavenServer
20650 Test
20651 Jps

> crash.sh 20650
   ______
 .~      ~. |`````````,       .'.                   ..'''' |         |
|           |'''|'''''      .''```.              .''       |_________|
|           |    `.       .'       `.         ..'          |         |
 `.______.' |      `.   .'           `. ....''             |         | 1.0.0-c

Follow and support the project on http://vietj.github.com/crash
Welcome to jerry + !
It is Thu Apr 12 22:09:23 CEST 2012 now
%
```

In this example we attached crash to the Test JVM. We obtained the Test JVM PID thanks to the *jps* command that belongs to the Java Platform. During this mode the commands are executed in the target JVM.

### 1.1.3. Configuration

The standalone mode relies on the `org.crsh.standalone.CRaSH` class main method to configure and run the shell. The startup scripts *crash.sh* and *crash.bar* configures this class. You can tweak those scripts to your environment, let's review the options and arguments of `CRaSH`:

#### 1.1.3.1. *--cmd* option

The *--cmd* option specifies a directory containing command source files. Such directory contains commands as *.groovy* files, commands can be in directories for grouping purpose. Several folders can be specified by repeating the option.

#### 1.1.3.2. *--cmd-mode* option

The standalone shell search commands in folders (specified with the *--cmd* option and in the classpath (under the */crash/commands/* folder). The *--cmd-mode* option defines how to handle the classpath commands:

- The *read* option value uses commands from directories and classpath.
- The *copy* option value scans the classpath during the startup and copies the commands in the first command folder, then commands are used from the folders. This value requires at least one command directory to be specified for extracting the commands.

#### 1.1.3.3. *--conf* option

The *--conf* option specifies a directory containing configuration files. Several folders can be specified by repeating the option.

#### 1.1.3.4. *--conf-mode* option

The standalone shell search configuration files in folders (specified with the *--conf* option and in the classpath (under the */crash/* folder). The *--conf-mode* option defines how to handle the classpath configuration:

- The *read* option value uses configuration files from directories and classpath.
- The *copy* option value scans the classpath during the startup and copies the files in the first configuration folder, then configuration are used from the folders. This value requires at least one conf directory to be specified for extracting the configuration files.

#### 1.1.3.5. *--property* option

The *--cmd* option sets and overrides a shell configuration property, the value follows the pattern *a=b*, for instance:

```
crash.sh --property crash.telnet.port=3000
```

#### 1.1.3.6. *pid* argument

The `org.crsh.standalone.CRaSH` main has a single optional argument that is a JVM *process id*. When a JVM process id is specified, CRaSH will dynamically attach to this virtual machine and will be executed in that machine. The two JVM will communicate with a socket.

### 1.1.4. Resource extraction

When the options *--cmd-mode* or *--conf-mode* are set to the *copy*, CRaSH will scan the classpath and extract the resources in the corresponding directory.

The default value of these options is *copy* however no copy happens unless at least one directory for extracting the resources is specified, therefore

- The `org.crsh.standalone.CRaSH` does nothing by default
- The *crash.sh* or *crash.bat* extracts the resources in the corresponding directory as the *cmd* and *conf* directories are specified

To prevent any resource copying the value *read* should be used/

## 1.2. Embedded mode

### 1.2.1. Embedding in a web app

CRaSH can use a standard web archive to be deployed in a web container. The war file is used for its packaging capabilities and triggering the CRaSH life cycle start/stop. In this mode CRaSH has two packaging available:

- A core war file found under *deploy/core/crash.war* provides the base CRaSH functionnalities
- A gatein war file found under *deploy/gatein/crash.war* provides additional Java Content Repository (JCR) features but deploys only in a GateIn server (Tomcat or JBoss). It extends the core packaging and adds
    - JCR browsing and interactions
    - SCP support for JCR import and export

You have to copy the *crash.war* in the appropriate server, regardless of the packaging used.

If you want you can embed CRaSH in your own *web.xml* configuration:

**Example 1.1. Embedding CRaSH in a web application**

```
<web-app>
  <listener>
    <listener-class>org.crsh.plugin.WebPluginLifeCycle</listener-class>
  </listener>
</web-app>
```

### 1.2.2. Embedding in Spring

CRaSH can be easily embedded and configured in a Spring configuration

#### 1.2.2.1. Embedding as a Spring bean

Here is an example of embedding crash:

**Example 1.2. Embedding CRaSH in Spring**

```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://ww

 <bean class="org.crsh.spring.SpringBootstrap">
    <property name="config">
      <props>
        <!-- VFS configuration -->
        <prop key="crash.vfs.refresh_period">1</prop>

        <!-- SSH configuration -->
       <prop key="crash.ssh.port">2000</prop>

        <!-- Telnet configuration -->
        <prop key="crash.telnet.port">5000</prop>

        <!-- Authentication configuration -->
        <prop key="crash.auth">simple</prop>
        <prop key="crash.auth.simple.username">admin</prop>
        <prop key="crash.auth.simple.password">admin</prop>
      </props>
    </property>
 </bean>

</beans>
```

The configuration properties are set as properties with the *config* property of the `SpringBootstrap` bean.

Any Spring managed beans that extend `org.crsh.plugin.CRaSHPlugin` will be automatically registered as plugins in addition to those declared in `META-INF/services/org.crsh.plugin.CRaSHPlugin`.

For example, the following implements a CRaSH authentication plugin that uses a JDBC DataSource managed by Spring:

**Example 1.3. Spring managed authentication plugin**

```
package example;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import javax.sql.DataSource;

import org.crsh.auth.AuthenticationPlugin;
import org.crsh.plugin.CRaSHPlugin;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("dbCrshAuth")
public class DbCrshAuthPlugin extends CRaSHPlugin<AuthenticationPlugin>
        implements AuthenticationPlugin {

    @Autowired
    private DataSource dataSource;

    @Override
    public AuthenticationPlugin getImplementation() {
        return this;
    }

    @Override
    public boolean authenticate(String username, String password)
            throws Exception {
        Connection conn = dataSource.getConnection();

        PreparedStatement statement = conn
                .prepareStatement("SELECT COUNT(*) FROM users WHERE username =
        statement.setString(1, username);
        statement.setString(2, password);

        ResultSet rs = statement.executeQuery();
        return rs.getInt(1) >= 1;
    }

    @Override
    public String getName() {
        return "dbCrshAuth";
    }

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }
}
```

The above code uses Spring annotation driven beans, but this works the same with beans configured in XML:

**Example 1.4. Custom authentication bean in spring.xml**

```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://ww

    <bean class="example.DbCrshAuthPlugin">
        <property name="dataSource" ref="dataSource"/>
    </bean>

</beans>
```

#### 1.2.2.2. Embedding in a Spring web app

In case you are embedding CRaSH in a Spring application running with a servlet container, the bean `org.crsh.spring.SpringWebBootstrap` can be used instead of `org.crsh.spring.SpringBootstrap`. The `SpringWebBootstrap` extends the `SpringBootstrap` class and adds the *WEB-INF/crash* directory to the command path.

An example packaging comes with the CRaSH distribution, a spring war file found under *deploy/spring/crash.war* provides the base CRaSH functionnalities bootstrapped by the Spring Framework. It can be used as an example for embedding CRaSH in Spring.

This example is bundled with a *spring* command that shows how the Spring factory or beans can be accessed within a CRaSH command.

# 2

# Interacting with the shell

## 2.1. Shell usage

### 2.1.1. Connection

You need to connect using telnet, SSH or *directly* to use the shell. The last method is a special mode using the JVM input and output.

#### 2.1.1.1. Telnet access

Telnet connection is done on port 5000:

```
(! 520)-> telnet localhost 5000
Trying ::1...
Connected to localhost.
Escape character is '^]'.
   ______
 .~      ~. |`````````,       .'.                     ..'''' |         |
|           |'''|'''''      .''```.                 .''      |_________|
|           |    `.       .'       `.            ..'         |         |
 `.______.' |      `.   .'           `. ....''               |         | 1.2.0-c

Follow and support the project on http://vietj.github.com/crash
Welcome to julien.local + !
It is Fri Dec 03 16:20:40 CET 2010 now
```

The bye command disconnect from the shell.

#### 2.1.1.2. SSH access

SSH connection is done on port 2000 with the password ***crash*** :

```
juliens-macbook-pro:~ julien$ ssh -p 2000 -l root localhost
root@localhost's password:
CRaSH 1.2.0-cr7 (http://vietj.github.com/crash)
Welcome to juliens-macbook-pro.local!
It is Fri Jan 08 21:12:53 CET 2010 now.
%
```

The bye command disconnect from the shell.

#### 2.1.1.3. Native access

A third mode is available for standalone CRaSH usage where it uses the JVM native input and output. When you run in standalone, CRaSh will be available just after the JVM is launched.

### 2.1.2. Features

- Line edition: the current line can be edited via left and right arrow keys
- History: the key up and key down enable history browsing
- Quoting: simple quotes or double quotes allow to insert blanks in command options and arguments, for instance *"old boy"* or *'old boy'*. One quote style can quote another, like *"hi, it's me"*.
- Completion: an advanced completion system is available

## 2.2. Command usage

### 2.2.1. Getting basic help

The `help` command will display the list of known commands by the shell.

```
[/]% help
% help
Try one of these commands with the -h or --help switch:

  cd                changes the current node
  commit            saves changes
  consume           collects a set of nodes
  cp                copy a node to another
  env               display the term env
  exportworkspace   Export a workspace on the file system (experimental)
  fail              Fails
  help              provides basic help
  importworkspace   Import a workspace from the file system (experimental)
  invoke            Invoke a static method
  log               logging commands
  ls                list the content of a node
  man               format and display the on-line manual pages
  mixin             mixin commands
  mv                move a node
  node              node commands
  produce           produce a set of nodes
  pwd               print the current node path
  rm                remove one or several node or a property
  rollback          rollback changes
  select            execute a JCR sql query
  setperm           modify the security permissions of a JCR node
  sleep             sleep for some time
  thread            vm thread commands
  version           versioning commands
  wait              Invoke a static method
  ws                workspace commands
  xpath             execute a JCR xpath query
```

### 2.2.2. Command line usage

The basic CRaSH usage is like any shell, you just type a command with its options and arguments. However it is possible to compose commands and create powerful combinations.

#### 2.2.2.1. Basic command usage

Typing the command followed by options and arguments will do the job

```
% ls /
...
```

#### 2.2.2.2. Command help display

Any command help can be displayed by using the -h argument:

```
% ls -h
usage: ls [-h | --help] [-h | --help] [-d | --depth] path

   [-h | --help]  command usage
   [-h | --help]  command usage
   [-d | --depth] Print depth
   path           the path of the node content to list
```

In addition of that, commands can have a complete manual that can be displayed thanks to the `man` command:

```
% man ls
NAME
       ls - list the content of a node

SYNOPSIS
       ls [-h | --help] [-h | --help] [-d | --depth] [-d | --depth] path

DESCRIPTION
       The ls command displays the content of a node. By default it lists the 
       accepts a path argument that can be absolute or relative.

       [/]% ls
       /
       +-properties
       | +-jcr:primaryType: nt:unstructured
       | +-jcr:mixinTypes: [exo:owneable,exo:privilegeable]
       | +-exo:owner: '__system'
       | +-exo:permissions: [any read,*:/platform/administrators read,*:/platf
       +-children
       | +-/workspace
       | +-/contents
       | +-/Users
       | +-/gadgets
       | +-/folder


PARAMETERS
       [-h | --help]
           Provides command usage

       [-h | --help]
           Provides command usage

       [-d | --depth]
           Print depth

       path
           the path of the node content to list
```

### 2.2.2.3. Advanced command usage

A CRaSH command is able to consume and produce a stream of object, allowing complex interactions between commands where they can exchange stream of compatible objets. Most of the time, JCR nodes are the objects exchanged by the commands but any command is free to produce or consume any type.

By default a command that does not support this feature does not consume or produce anything. Such commands usually inherits from the `org.crsh.command.ClassCommand` class that does not care about it. If you look at this class you will see it extends the the `org.crsh.command.BaseCommand`.

More advanced commands inherits from `org.crsh.command.BaseCommand` class that specifies two generic types <C> and <P>:

- <C> is the type of the object that the command consumes

- <P> is the type of the object that the command produces

The command composition provides two operators:

- The pipe operator **|** allows to stream a command output stream to a command input stream
- The distribution operator **+** allows to distribute an input stream to several commands and to combine the output stream of several commands into a single stream.

#### 2.2.2.4. Connecting a `<Void,Node>` command to a `<Node,Void>` command through a pipe

**Example 2.1. Remove all nt:unstructed nodes**

```
% select * from nt:unstructed | rm
```

#### 2.2.2.5. Connecting a `<Void,Node>` command to two `<Node,Void>` commands through a pipe

**Example 2.2. Update the security of all nt:unstructed nodes**

```
% select * from nt:unstructured | setperm -i any -a read + setperm -i any -a wr
```

#### 2.2.2.6. Connecting two `<Void,Node>` command to a `<Node,Void>` commands through a pipe

**Example 2.3. Add the mixin mix:referenceable to any node of type nt:file or nt:folder**

```
% select * from nt:file + select * from nt:folder | addmixin mix:referenceable
```

#### 2.2.2.7. Mixed cases

When a command does not consume a stream but is involved in a distribution it will not receive any stream but will be nevertheless invoked.

Likewise when a command does not produce a stream but is involved in a distribution, it will not produce anything but will be nevertheless invoked.

## 2.3. Base commands

### 2.3.1. *sleep* command

```
NAME
       sleep - sleep for some time

SYNOPSIS
       sleep [-h | --help] time


PARAMETERS
       [-h | --help]
           Display this help message

       time
           sleep time in seconds
```

### 2.3.2. *man* command

```
NAME
       man - format and display the on-line manual pages

SYNOPSIS
       man [-h | --help] command


PARAMETERS
       [-h | --help]
           Display this help message

       command
           the command
```

### 2.3.3. *log* command

```
NAME
       log add - create one or several loggers

SYNOPSIS
       log [-h | --help] add ... name


PARAMETERS
       [-h | --help]
           Display this help message

       ... name
           The name of the logger
```

```
NAME
       log set - configures the level of one of several loggers

SYNOPSIS
       log [-h | --help] set [-l | --level] ... name

DESCRIPTION
       The set command sets the level of a logger. One or several logger names
       and the -l option specify the level among the trace, debug, info, warn a
       specified, the level is cleared and the level will be inherited from its

       % logset -l trace foo
       % logset foo

       The logger name can be omitted and instead stream of logger can be consu
       The following set the level warn on all the available loggers:

       % log ls | log set -l warn


PARAMETERS
       [-h | --help]
           Display this help message

       [-l | --level]
           The logger level to assign among {trace, debug, info, warn, error}

       ... name
           The name of the logger
```

```
NAME
       log send - send a message to a logger

SYNOPSIS
       log [-h | --help] send [-m | --message] [-l | --level] name

DESCRIPTION
       The send command log one or several loggers with a specified message. Fo
       the javax.management.mbeanserver class and send a message on its own log

       #% log send -m hello javax.management.mbeanserver

       Send is a <Logger, Void> command, it can log messages to consumed log ol

       % log ls | log send -m hello -l warn


PARAMETERS
       [-h | --help]
           Display this help message

       [-m | --message]
           The message to log

       [-l | --level]
           The logger level to assign among {trace, debug, info, warn, error}

       name
           The name of the logger
```

```
NAME
       log ls - list the available loggers

SYNOPSIS
       log [-h | --help] ls [-f | --filter]

DESCRIPTION
       The logls command list all the available loggers., for instance:

       % logls
       org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/].[defau
       org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/eXoGadge
       org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/dashboar
       ...

       The -f switch provides filtering with a Java regular expression

       % logls -f javax.*
       javax.management.mbeanserver
       javax.management.modelmbean

       The logls command is a <Void,Logger> command, therefore any logger produ


PARAMETERS
       [-h | --help]
           Display this help message

       [-f | --filter]
           A regular expressions used to filter the loggers
```

### 2.3.4. *thread* command

```
NAME
       thread stop - stop vm threads

SYNOPSIS
       thread [-h | --help] stop ... ids

DESCRIPTION
       Stop VM threads.


PARAMETERS
       [-h | --help]
           Display this help message

       ... ids
           the thread ids to stop
```

```
NAME
       thread interrupt - interrupt vm threads

SYNOPSIS
       thread [-h | --help] interrupt ... ids

DESCRIPTION
       Interrup VM threads.


PARAMETERS
       [-h | --help]
           Display this help message

       ... ids
           the thread ids to interrupt
```

```
NAME
       thread ls - list the vm threads

SYNOPSIS
       thread [-h | --help] ls [-n | --name] [-g | --group] [-s | --state]


PARAMETERS
       [-h | --help]
           Display this help message

       [-n | --name]
           Filter the threads with a glob expression on their name

       [-g | --group]
           Filter the threads with a glob expression on their group

       [-s | --state]
           Filter the threads by their status (new,runnable,blocked,waiting,ti
NAME
       thread top - thread top

SYNOPSIS
       thread [-h | --help] top [-n | --name] [-g | --group] [-s | --state]


PARAMETERS
       [-h | --help]
           Display this help message

       [-n | --name]
           Filter the threads with a glob expression on their name

       [-g | --group]
           Filter the threads with a glob expression on their group

       [-s | --state]
           Filter the threads by their status (new,runnable,blocked,waiting,ti
```

```
NAME
       thread dump - dump vm threads

SYNOPSIS
       thread [-h | --help] dump ... ids

DESCRIPTION
       Dump VM threads.


PARAMETERS
       [-h | --help]
           Display this help message

       ... ids
           the thread ids to dump
```

### 2.3.5. *system* command

```
NAME
       system gc - call garbage collector

SYNOPSIS
       system [-h | --help] gc


PARAMETERS
       [-h | --help]
           Display this help message
```

```
NAME
       system propls - list the vm system properties

SYNOPSIS
       system [-h | --help] propls [-f | --filter]


PARAMETERS
       [-h | --help]
           Display this help message

       [-f | --filter]
           filter the property with a regular expression on their name
```

```
NAME
       system propset - set a system property

SYNOPSIS
       system [-h | --help] propset name value


PARAMETERS
       [-h | --help]
           Display this help message

       name
           The name of the property

       value
           The value of the property
```

```
NAME
       system propget - get a system property

SYNOPSIS
       system [-h | --help] propget name


PARAMETERS
       [-h | --help]
           Display this help message

       name
           The name of the property
```

```
NAME
       system proprm - remove a system property

SYNOPSIS
       system [-h | --help] proprm name


PARAMETERS
       [-h | --help]
           Display this help message

       name
           The name of the property
```

```
NAME
       system freemem - show free memory

SYNOPSIS
       system [-h | --help] freemem [-u | --unit] [-d | --decimal]


PARAMETERS
       [-h | --help]
           Display this help message

       [-u | --unit]
           The unit of the memory space size {(B)yte, (O)ctet, (M)egaOctet, (G

       [-d | --decimal]
           The number of decimal (default 0)
```

```
NAME
       system totalmem - show total memory

SYNOPSIS
       system [-h | --help] totalmem [-u | --unit] [-d | --decimal]


PARAMETERS
       [-h | --help]
           Display this help message

       [-u | --unit]
           The unit of the memory space size {(B)yte, (O)ctet, (M)egaOctet, (G

       [-d | --decimal]
           The number of decimal (default 0)
```

### 2.3.6. *jdbc* command

```
NAME
       jdbc props - show the database properties

SYNOPSIS
       jdbc [-h | --help] props


PARAMETERS
       [-h | --help]
           Display this help message
```

```
NAME
       jdbc close - close the current connection

SYNOPSIS
       jdbc [-h | --help] close


PARAMETERS
       [-h | --help]
           Display this help message
```

```
NAME
       jdbc table - describe the tables

SYNOPSIS
       jdbc [-h | --help] table ... tableNames


PARAMETERS
       [-h | --help]
           Display this help message

       ... tableNames
           the table names
```

```
NAME
       jdbc open - open a connection from JNDI bound datasource

SYNOPSIS
       jdbc [-h | --help] open globalName


PARAMETERS
       [-h | --help]
           Display this help message

       globalName
           The datasource JNDI name
```

```
NAME
       jdbc connect - connect to database with a JDBC connection string

SYNOPSIS
       jdbc [-h | --help] connect [-u | --username] [-p | --password] [--proper


PARAMETERS
       [-h | --help]
           Display this help message

       [-u | --username]
           The username

       [-p | --password]
           The password

       [--properties]
           The extra properties

       connectionString
           The connection string
```

```
NAME
       jdbc info - describe the database

SYNOPSIS
       jdbc [-h | --help] info


PARAMETERS
       [-h | --help]
           Display this help message
```

```
NAME
       jdbc execute - execute a SQL statement

SYNOPSIS
       jdbc [-h | --help] execute ... statement


PARAMETERS
       [-h | --help]
           Display this help message

       ... statement
           The statement
```

```
NAME
       jdbc select - select SQL statement

SYNOPSIS
       jdbc [-h | --help] select ... statement


PARAMETERS
       [-h | --help]
           Display this help message

       ... statement
           The statement
```

```
NAME
       jdbc tables - describe the tables

SYNOPSIS
       jdbc [-h | --help] tables


PARAMETERS
       [-h | --help]
           Display this help message
```

### 2.3.7. *java* command

```
NAME
       java type - print information about a java type

SYNOPSIS
       java [-h | --help] type name


PARAMETERS
       [-h | --help]
           Display this help message

       name
           The full qualified type name
```

### 2.3.8. *jmx* command

```
NAME
       jmx get - Get managed bean attribute

SYNOPSIS
       jmx [-h | --help] get [-a | --attributes]


PARAMETERS
       [-h | --help]
           Display this help message

       [-a | --attributes]
```

```
NAME
       jmx find - Find managed bean

SYNOPSIS
       jmx [-h | --help] find [-p | --pattern]


PARAMETERS
       [-h | --help]
           Display this help message

       [-p | --pattern]
           The object name pattern
```

### 2.3.9. *thread* command

```
NAME
       thread stop - stop vm threads

SYNOPSIS
       thread [-h | --help] stop ... ids

DESCRIPTION
       Stop VM threads.


PARAMETERS
       [-h | --help]
           Display this help message

       ... ids
           the thread ids to stop
```

```
NAME
       thread interrupt - interrupt vm threads

SYNOPSIS
       thread [-h | --help] interrupt ... ids

DESCRIPTION
       Interrup VM threads.


PARAMETERS
       [-h | --help]
           Display this help message

       ... ids
           the thread ids to interrupt
```

```
NAME
       thread ls - list the vm threads

SYNOPSIS
       thread [-h | --help] ls [-n | --name] [-g | --group] [-s | --state]


PARAMETERS
       [-h | --help]
           Display this help message

       [-n | --name]
           Filter the threads with a glob expression on their name

       [-g | --group]
           Filter the threads with a glob expression on their group

       [-s | --state]
           Filter the threads by their status (new,runnable,blocked,waiting,ti
NAME
       thread top - thread top

SYNOPSIS
       thread [-h | --help] top [-n | --name] [-g | --group] [-s | --state]


PARAMETERS
       [-h | --help]
           Display this help message

       [-n | --name]
           Filter the threads with a glob expression on their name

       [-g | --group]
           Filter the threads with a glob expression on their group

       [-s | --state]
           Filter the threads by their status (new,runnable,blocked,waiting,ti
```

```
NAME
       thread dump - dump vm threads

SYNOPSIS
       thread [-h | --help] dump ... ids

DESCRIPTION
       Dump VM threads.


PARAMETERS
       [-h | --help]
           Display this help message

       ... ids
           the thread ids to dump
```

### 2.3.10. *sleep* command

```
NAME
       sleep - sleep for some time

SYNOPSIS
       sleep [-h | --help] time


PARAMETERS
       [-h | --help]
           Display this help message

       time
           sleep time in seconds
```

### 2.3.11. *jpa* command

```
NAME
       jpa close - Close the current JPA session

SYNOPSIS
       jpa [-h | --help] close


PARAMETERS
       [-h | --help]
           Display this help message
```

```
NAME
       jpa open - Open a JPA session

SYNOPSIS
       jpa [-h | --help] open jndiName


PARAMETERS
       [-h | --help]
           Display this help message

       jndiName
```

```
NAME
       jpa select - Execute select JPA query

SYNOPSIS
       jpa [-h | --help] select ... statements


PARAMETERS
       [-h | --help]
           Display this help message

       ... statements
```

```
NAME
       jpa entity - Display JPA entity

SYNOPSIS
       jpa [-h | --help] entity name


PARAMETERS
       [-h | --help]
           Display this help message

       name
```

```
NAME
       jpa entities - List JPA entities

SYNOPSIS
       jpa [-h | --help] entities


PARAMETERS
       [-h | --help]
           Display this help message
```

### 2.3.12. *env* command

```
NAME
       env - display the term env

SYNOPSIS
       env [-h | --help]


PARAMETERS
       [-h | --help]
           Display this help message
```

# 3 Configuration

## 3.1. Configuration properties

CRaSH is configured by a set of properties, these properties are defined in a configuration file. In the war file packaging, the configuration file can be found under */WEB-INF/crash/crash.properties* file of the archive. Configuration can be overriden by Java Virtual Machine system properties by using the same property name.

CRaSH properties are always prefixed by the *crash.* value

## 3.2. Changing SSH server key

The key can be changed by replacing the file *WEB-INF/sshd/hostkey.pem*. Alternatively you can configure the server to use an external file by using the *crash.ssh.keypath* parameter in the *crash.properties*. Uncomment the corresponding property and change the path to the key file.

```
#crash.ssh.keypath=/path/to/the/key/file
```

## 3.3. Changing telnet or SSH server ports

The ports of the server are parameterized by the *crash.ssh.port* and *crash.telnet.port* parameters in the *crash.properties* file

```
# SSH configuration
crash.ssh.port=2000
```

```
# Telnet configuration
crash.telnet.port=5000
```

## 3.4. Removing telnet or SSH access

- to remove the telnet access, remove the jar file in the *WEB-INF/lib/crsh.shell.telnet-1.2.0-cr7.jar* .
- to remove the SSH access, remove the jar file in the *WEB-INF/lib/crsh.shell.ssh-1.2.0-cr7.jar* .

## 3.5. Configuring shell default message

The */WEB-INF/crash/commands/base/login.groovy* file contains two closures that are evaluated each time a message is required

- The `prompt` closure returns the prompt message
- The `welcome` closure returns the welcome message

Those closure can be customized to return different messages.

## 3.6. Configuring authentication

Authentication is used by the SSH server when a user authenticates. Authentication interface is pluggable and has default implementations. The Section 5.2, “Pluggable authentication ” explains how to write a custom authentication plugin, in this section we cover the configuation of the authentication.

The configuration of the authentication plugin is done via property, this is necessary because several plugins can be detected by CRaSH, and the plugin is selected via the property *crash.auth* that must match the authentication plugin name:

```
crash.auth=simple
```

CRaSH comes out of the box with two authentication plugins.

### 3.6.1. Simple authentication

Simple authentication provides a simple username/password authentication configured with the *crash.auth.simple.username* and *crash.auth.simple.password* properties:

```
# Authentication configuration
crash.auth=simple
crash.auth.simple.username=admin
crash.auth.simple.password=admin
```

### 3.6.2. Jaas authentation

Jaas authentication uses jaas to perform authentication configured with the *crash.auth.jaas.domain* property to define the jaas domain to use when performing authentication:

```
# Authentication configuration
crash.auth=jaas
crash.auth.jaas.domain=gatein-domain
```

# 4

# Developers

## 4.1. Developping commands

A CRaSH command is written in the Groovy language. The Groovy language provides several signifiant advantages:

- Commands can be bare scripts or can be a class
- Java developers can write Groovy commands without learning much of it
- Groovy is a dynamic language and can manipulate unknown types

Each command has a corresponding Groovy file that contains a command class that will be invoked by the shell. The files are located in

- *cmd* directory for the standalone distribution
- */WEB-INF/crash/commands* directory for the web archive deployment

New commands can directly be placed in the commands directory; however they can also be placed in a sub directory of the command directory, which is useful to group commands of the same kind.

In addition of that there are two special files called *login.groovy* and *logout.groovy* that are executed upon login and logout of a user. They are useful to setup and cleanup things related to the current user session.

### 4.1.1. Commands as a script

The simplest command can be a simple script that returns a string

```
return "Hello World";
```

The `out` implicit variable can be used to send a message to the console:

```
out.println("Hello World");
```

It can be even Groovier:

```
out << "Hello World"
```

### 4.1.2. Commands as a class

Class can also be used for defining a command, it provides significant advantages over scripts:

- A command can declare options and arguments for the command
- Sub command style (git style) can be expressed easily

When the user types a command in the sell, the command line is parsed by the *cmdline* framework and injected in the command class.

Let's study a simple class command example:

```
import org.crsh.cli.Command;
import org.crsh.cli.Usage;
import org.crsh.cli.Option;

class date {
  @Usage("show the current time")
  @Command
  Object main(
     @Usage("the time format")
     @Option(names=["f","format"])
     String format) {
    if (format == null)
      format = "EEE MMM d HH:mm:ss z yyyy";
    def date = new Date();
    return date.format(format);
  }
}
```

The command is pretty straightforward to grasp:

- The `@Command` annotation declares the `main` method as a command
- The command takes one optional `format` option declared by the `@Option` annotation
- The `@Usage` annotation describes the usage of the command and its parameters

```
% date
Thu Apr 19 15:44:05 CEST 2012
```

The `@Usage` annotation is important because it will give a decent human description of the command

```
% date -h
usage: date [-h | --help] [-f | --format]

   [-h | --help]   command usage
   [-f | --format] the time format
```

### 4.1.3. Sub commands

A class can hold several commands allowing a single file to group several commands, let's study the JDBC command structure:

```
@Usage("JDBC connection")
class jdbc {

  @Usage("connect to database with a JDBC connection string")
  @Command
  public String connect(
          @Usage("The username")
          @Option(names=["u","username"])
          String user,
          @Usage("The password")
          @Option(names=["p","password"])
          String password,
          @Usage("The extra properties")
          @Option(names=["properties"])
          Properties properties,
          @Usage("The connection string")
          @Argument
          String connectionString) {
     ...
  }

  @Usage("close the current connection")
  @Command
  public String close() {
     ...
  }
}
```

We can see that the class declares two commands `connect` and `close`, they are invoked this way:

```
% jdbc connect jdbc:derby:memory:EmbeddedDB;create=true
Connected to data base : jdbc:derby:memory:EmbeddedDB;create=true
% jdbc close
Connection closed
```

### 4.1.4. Command line annotations

Let's review the various annotations for declaring a command.

#### 4.1.4.1. `@org.crsh.cli.Command`

Defines a command method, when using a mono command the method should be named `main`:

```
public class sample {

  @Command
  public void main() {
    ...
  }
}
```

Using this annotation automatically turns a class into a class command.

> Previous versions of CRaSH required command classes to extend the `org.crsh.command.CRaSHCommand` class, this is not necessary anymore as the `@Command` annotation is enough.

Sub commands will simply declares several methods:

```
public class sample {

  @Command
  public void sub1() {
    ...
  }

  @Command
  public void sub2() {
    ...
  }
}
```

#### 4.1.4.2. `@org.crsh.cli.Option`

Declares an option, the *names* member must be specified: single letter name are turned into posix style option (single hyphen) other names are turned into GNU style option (double hyphen). Several names can specified as aliases of the same option. Option can be declared as method parameters or a class fields.

```
public class sample {

  @Option(names = ["o", "opt1"])
  private String opt1;

  @Command
  public void sub1(@Option(names = ["opt2"]) String opt2) {
    ...
  }
}
```

```
> sample foo
> sample -o foo
> sample --opt1 foo sub1
> sample sub1 --opt2 bar
> sample --opt1 foo foo sub1 --opt2 bar
```

#### 4.1.4.3. `@org.crsh.cli.Argument`

Declares an argument, this annotation should be declares as method parameters.

```
public class sample {

  @Command
  public void sub1(@Argument String arg) {
    ...
  }
}
```

```
> sample sub1
> sample sub1 foo
```

#### 4.1.4.4. `@org.crsh.cli.Required`

By default a parameter is optional, the `@Required` annotation can be used to force the user to specify a parameter:

```
public class sample {

  @Command
  public void sub1(@Required @Argument String arg) {
    ...
  }
}
```

#### 4.1.4.5. `@org.crsh.cli.Usage` and `@org.crsh.cli.Man`

Those annotations are useful for documenting commands help and manual:

```
@Usage("sample commands")
public class sample {

  @Command
  @Usage("command description, begins with lower case")
  @Man("Verbose descrition of the argument, it should begin with an upper case'
  public void sub1(
    @Usage("argument description, begins with a lower case")
    @Man("Verbose description of the argument, it should begin with an upper ca
    @Argument String arg) {
    ...
  }
}
```

- `@Usage` specifies the usage, rather a short description
- `@Man` provides the manuel, rather a verbose description

### 4.1.5. Parameter types

Option and argument parameters are represented by *simple* types. The string type is universal and will work with any value provided by the user, other types will require parsing.

#### 4.1.5.1. Builtin types

CRaSH provides supports a few builtin simple types other than string:

- *Integer* type
- *Boolean* type
- `java.util.Properties` type
- `javax.management.ObjectName` type
- *Enum* types

Boolean type is special because it does not need a value when combined with options. The option declaration is enough to set the value to true:

```
public class sample {

  @Command
  public void sub1(@Option(names = ["o"]) Boolean opt) {
    ...
  }
}
```

The option will be true with:

```
> sample sub1 -o
```

#### 4.1.5.2. Providing your own type

Providing a custom type is possible, CRaSH uses the `ServiceLoader` discovery mechanism to discover custom types. Custom types are implemented by a `org.crsh.cli.type.ValueType` subclass and implement its `parse` method:

**Example 4.1. Our custom value type**

```
package my;

public class CustomValueType extends ValueType<Custom> {

  public CustomValueType() {
    super(Custom.class); ❶
  }

  @Override
  public <S extends Custom> S parse(Class<S> type, String s) throws Exception
    return type.cast(new Custom(s)); ❷
  }
}
```

❶ The custom type is passed to the super class

❷ The parse method should reutrn an instance of the type

The `parse` method uses the <S> generic type because the implementation of enum types has an effective type which is a subclass of the base enum type.

In order to make the custom type discovered by CRaSH, a file named *org.crsh.cli.type.ValueType* should be placed in the */META-INF/services/* directory of the jar containing the custom value type:

**Example 4.2. The custom value type declared in META-INF/services/org.crsh.cli.type.ValueTy**

```
my.CustomValueTpye
```

### 4.1.6. Parameter multiplicity

The multiplicity is the number of values expected by a parameter, the multiplicity with simple types is always 1. The arity can also be *several* when the `java.util.List` type is used.

```
public class sample {

  @Command
  public void sub1(@Option(names = ["o"]) List<String> opts) {
    ...
  }
}
```

The option can now accept several values:

```
> sample sub1 -o foo -o bar
```

## 4.2. Parameter annotations: Don't Repeat Yourself

When one or several commands uses the same parameter (option or argument), there is the opportunity to avoid repetition and define a custom annotation that can be used for declaring the parameter:

```
@Retention(RetentionPolicy.RUNTIME)
@Usage("A color")
@Option(names = "c")
public @interface PathOption {
}
```

The annotation can then be used instead for declaring an option:

```
public class mycommand {
  @Command
  public void foo(@ColorOption String color) {
    ...
  }
  @Command
  public void bar(@ColorOption String color) {
    ...
  }
}
```

## 4.3. Command context

During the execution of a command, CRaSH provides a *context* for interacting with it : the property *context* is resolved to an instance of `org.crsh.command.InvocationContext`, the invocation context class extends the `org.crsh.command.CommandContext`. Let's have a look at those types:

**Example 4.3. The command context**

```
/**
 * The minimum set of services that a command needs for executing properly:
 * <ul>
 *   <li>a session map</li>
 *   <li>an attribute map</li>
 * </ul>
 */
public interface CommandContext {

  /**
   * Returns the current shell session.
   *
   * @return the session map
   */
  Map<String, Object> getSession();

  /**
   * Returns the current shell attributes.
   *
   * @return the attributes map
   */
  Map<String, Object> getAttributes();

}
```

The `CommandContext` provides access to the shell session as a `Map<String, Object>`. Session attributes can be accessed using this map, but they are also accessible as Groovy script properties. It means that writing such code will be equivalent:

**Example 4.4. Using shell session**

```
context.session["foo"] = "bar"; ❶
out.println(bar); ❷
```

❶ Bind the session attribute foo with the value bar

❷ The bar is resolved as an session attribute by Groovy

The `CommandContext` provides also access to the shell attributes as a `Map<String, Object>`. Context attributes are useful to interact with object shared globally by the CRaSH environment:

- When embedded in a web application context, attributes resolves to servlet context attributes.
- When embedded in Spring context, attributes resolve to Spring objects:
  - *attributes.factory* returns the Spring factory
  - *attributes.beans* returns Spring beans, for example *attribute.beans.telnet* returns the

*telnet* bean

- When attached to a virtual machine, the context attributes has only a single `instrumentation` entry that is the `java.lang.instrument.Instrumentation` instance obtained when attaching to a virtual machine.

**Example 4.5. Obtaining a Spring bean**

```
def bean = context.attributes.beans["TheBean"];
```

Now let's examine the `InvocationContext` that extends the `CommandContext`:

**Example 4.6. The invocation context**

```

public interface InvocationContext<P> extends CommandContext, InteractionConte

  /**
   * Returns the writer for the output.
   *
   * @return the writer
   */
  RenderPrintWriter getWriter();

  /**
   * Resolve a command invoker for the specified command line.
   *
   * @param s the command line
   * @return the command invoker
   * @throws ScriptException any script exception
   * @throws IOException any io exception
   */
  CommandInvoker<?, ?> resolve(String s) throws ScriptException, IOException;

}
```

The `PrintWriter` object is the command output, it can be used also via the *out* property in Groovy scripts:

**Example 4.7. Printing on the shell**

```
context.writer.print("Hello"); ❶
out.print("hello"); ❷
```

❶ Printing using the context writer

❷ Printing using the *out*

The `readLine` method can be used to get interactive information from the user during the execution of a command.

**Example 4.8. Reading on the console**

```
def age = context.readLine("How old are you?", false);
```

Finally the `isPiped`, `consume` and `produce` methods are used when writing commands that exchange objects via the pipe mechanism.

## 4.4. Adding style

CRaSH adds (since version 1.1) the support for colored text and text decoration. Each portion of text printed has three style attributes:

- *Decoration* : bold, underline or blink, as the `org.crsh.text.Decoration` enum.
- *Foreground* color.
- *Background* color.

Available colors are grouped as the `org.crsh.text.Color` enum: black, red, green, yellow, blue, magenta, cyan, white.

Decoration and colors can be applied with overloaded `print` and `println` methods provided by the `ShellPrinterWriter`. This printer is available as the implicit *out* attribute or thanks to the `context.getWriter()` method.

**Example 4.9. Decorating and coloring text**

```
out.println("hello", red); ❶
out.println("hello", red, blue); ❷
out.println("hello", underline, red, blue); ❸
```

❶ Print hello in red color

❷ Print hello in red with a red blue

❸ Print hello in red underlined with a red blue

The combination of the decoration, background and foreground colors is a *style* represented by the `org.crsh.text.Style` object. Styles can be used like decoration and colors:

**Example 4.10. Printing styled text**

```
out.println("hello", style(red)); ❶
out.println("hello", style(red, blue)); ❷
out.println("hello", style(underline, red, blue)); ❸
```

❶ Print hello in red color

❷ Print hello in red with a red blue

❸ Print hello in red underlined with a red blue

When using the print methods, the style will be used for the currently printed object. It is possible to change the style permanently (until it is reset) using Groovy *leftshift* operator : <<

By default the << operator prints output on the console. The `ShellPrintWriter` overrides the operator to work with color, decoration and styles:

**Example 4.11. Styling with the leftshift operator**

```
out << red ❶
out << underline ❷
out << "hello" ❸
out << reset; ❹
```

❶ Set red foreground color

❷ Set underline

❸ Print hello in underlined red

❹ Reset style

Operators can also be combined on the same line providing a more compact syntax:

```
out << red << underline << "hello" << reset
```

```
out << style(underline, red, blue) << "hello" << reset
```

Throughout the examples we have used decoration, color and styles. CRaSH automatically imports those classes so they can be used out of the box in any CRaSH command without requiring prior import.

## 4.5. Inter command API

In this section we study how a command can reuse existing commands. Here is an example

**Example 4.12. dbscript.groovy**

```
jdbc.connect username:root, password:crash, "jdbc:derby:memory:EmbeddedDB;creat
jdbc.execute "create table derbyDB(num int, addr varchar(40))"
jdbc.execute "insert into derbyDB values (1956,'Webster St.')"
jdbc.execute "insert into derbyDB values (1910,'Union St.')"
jdbc.execute "select * from derbyDb"
jdbc.close
```

This script is written in Groovy and use Groovy DSL capabilities, let's study the first statement:

- the `jdbc.connect` statement can be decomposed into two steps
  - the `jdbc` is resolved as the command itself
  - the `connect` invokes the connect command
- the `username` and `password` are considered as command options
- the `SQL` statement `"jdbc:derby:memory:EmbeddedDB;create=true"` is the main argument of the command

It is equivalent to the shell command:

```
% jdbc connect --username root --password crash jdbc:derby:memory:EmbeddedDB;cr
```

The rest of the script is fairly easy to understand, here is the output of the script execution:

```
% dbscript
Connected to data base : jdbc:derby:memory:EmbeddedDB;create=true
Query executed successfully
Query executed successfully
Query executed successfully
NUM                  ADDR
1956                 Webster St.
1910                 Union St.
Connection closed
```

# 5

# Extending CRaSH

## 5.1. Embedding CRaSH

The Chapter 1, *Running CRaSH* explains how to run CRaSH as a standalone or an embedded service. We will study in this section the technical aspect of running application and show how CRaSH can be embedded in specific environments.

The root class for reusing CRaSH is the `org.crsh.plugin.PluginLifeCycle` class. This class is abstract and it cannot be used directly, instead it should be subclasses for providing specific behavior for running CRaSH. There are several subclasses using it:

- The standalone bootstrap with the `org.crsh.standalone.Bootstrap` class : designed for using CRaSH with a real file system (i.e `java.io.File`). It defines a specific layout for locating resources (libraries, configuration and commands).
- The embedded approaches
  - `org.crsh.plugin.WebPluginLifeCycle` : uses a `javax.servlet.ServletContext`
  - `org.crsh.spring.SpringBootstrap` : embeds CRaSH as a Spring bean
  - `org.crsh.spring.SpringWebBootstrap` : extends the `SpringBootstrap` and uses the existing `ServletContext`

### 5.1.1. Standalone bootstrap

The `org.crsh.standalone.Bootstrap` class is a generic class that can be used to embed the shell in your Java programs Its usage is quite straighforward and configurable. The bootstrap is a coarse grained approach and it needs a bit of configuration for running:

- The `baseLoader` properties is the `java.lang.ClassLoader` used by CRaSH for loading plugins, resources or command sources (under the */crash/commands/* path. This property is not modifiable and must be provided when the bootstrap is instantiated.
- The `config` properties provides the contextual properties used by CRaSH configuration such as *crash.vfs.refresh_period*

- The `attributes` property provides the contextual attributes used by CRaSH available at runtime via the `org.crsh.command.CommandContext`, it is useful for providing objects to commands in a similar fashion to servlet context attributes
- The `cmdPath` property is a list of `java.io.File` scanned by CRaSH for loading additional commands
- The `confPath` property is a list of `java.io.File` scanned by CRaSH for loading configuration files

Let's see an example on how to use it

### 5.1.2. Standalone CRaSH

The standalone shell is a Java class configurable and runnable from the command line that is used by the standalone distribution. It is built upon the Section 5.1.1, “Standalone bootstrap ” class.

## 5.2. Pluggable authentication

Creating a custom authentication mechanism is done by implementing a CRaSH plugin that provides an implementation of the `AuthenticationPlugin` interface. Let's study the *simple* authentication plugin implementation.

The `AuthenticationPlugin` is the interface to implement in order to integrate CRaSH with an authentication mechanism:

```
public interface AuthenticationPlugin {

  /**
   * Returns the authentication plugin name.
   *
   * @return the plugin name
   */
  String getName();

  /**
   * Returns true if the user is authentified by its username and password.
   *
   * @param username the username
   * @param password the password
   * @return true if authentication succeeded
   * @throws Exception any exception that would prevent authentication to happe
   */
  boolean authenticate(String username, String password) throws Exception;
}
```

The integration as a CRaSH plugin mandates to extend the class `CRaSHPlugin` with the generic type `AuthenticationPlugin`:

```
public class SimpleAuthenticationPlugin extends
    CRaSHPlugin<AuthenticationPlugin> implements
    AuthenticationPlugin {

  public String getName() {
    return "simple";
  }

  @Override
  public AuthenticationPlugin getImplementation() {
    return this;
  }

  ...

}
```

- The `getName()` method returns the *simple* value that matchs the *crash.auth* configuration property
- The `getImplementation()` method returns the object that implements the `AuthenticationPlugin` class, this method is implemented from the `CRaSHPlugin` abstract class, but in our case it simply returns `this` because `SimpleAuthenticationPlugin` is directly the implementation class.

Now let's study how the plugin retrieves the configuration properties `crash.auth.simple.username` and `crash.auth.simple.password`:

```
public class SimpleAuthenticationPlugin extends
    CRaSHPlugin<AuthenticationPlugin> implements
    AuthenticationPlugin {

  public static final PropertyDescriptor<String> SIMPLE_USERNAME =
    PropertyDescriptor.create(
      "auth.simple.username",
      "admin",
      "The username");

  public static final PropertyDescriptor<String> SIMPLE_PASSWORD =
    PropertyDescriptor.create(
      "auth.simple.password",
      "admin",
      "The password");

  @Override
  protected Iterable<PropertyDescriptor<?>> createConfigurationCapabilities()
    return Arrays.<PropertyDescriptor<?>>asList(
      SIMPLE_USERNAME,
      SIMPLE_PASSWORD);
  }

  private String username;

  private String password;

  @Override
  public void init() {
    PluginContext context = getContext();
    this.username = context.getProperty(SIMPLE_USERNAME);
    this.password = context.getProperty(SIMPLE_PASSWORD);
  }

  ...

}
```

- The `createConfigurationCapabilities()` method returns the constants `SIMPLE_USERNAME` and `SIMPLE_PASSWORD` that defines the configuration properties that the plugin uses

- The `init()` method is invoked by CRaSH before the plugin will be used, at this moment, the configuration properties are retrieved from the plugin context with the method `getContext()` available in the `CRaSHPlugin` base class

Finally the plugin needs to provide the `authenticate()` method that implement the authentication logic:

```
public boolean authenticate(String username, String password)
  throws Exception {
  return this.username != null &&
    this.password != null &&
    this.username.equals(username) &&
    this.password.equals(password);
}
```

The logic is straightforward with an equality check of the username and password.

Last but not least we must declare our plugin to make it recognized by CRaSH, this is achieved thanks to the `java.util.ServiceLoader` class. CRaSH uses the `ServiceLoader` for loading plugins and the loader needs a file to be present in the jar file containing the class under the name `META-INF/services/org.crsh.plugin.CRaSHPlugin` containing the class name of the plugin:

```
org.crsh.auth.SimpleAuthenticationPlugin
```

When all of this is done, the plugin and its service loader descriptor must be packaged in a jar file and available on the classpath of CRaSH.

You can learn more about the `java.util.ServiceLoader` by looking at the online javadoc

# 6 JCR extension

The CRaSH JCR extension allow to connect and interract with Java Content Repository implementations.

## 6.1. JCR implementations

### 6.1.1. eXo JCR

todo

### 6.1.2. Apache Jackrabbit

CRaSH has been tested with Jackrabbit in the following mode : deployment as a resource accessible via JNDI on JBoss 6.1.0.

## 6.2. JCR commands

### 6.2.1. *repo* command

```
NAME
       repo info - show info about the current repository

SYNOPSIS
       repo [-h | --help] info

DESCRIPTION
       The info command print the descriptor of the current repository.


PARAMETERS
       [-h | --help]
           Display this help message
```

```
NAME
       repo ls - list the available repository plugins

SYNOPSIS
       repo [-h | --help] ls

DESCRIPTION
       The ls command print the available repository plugins.


PARAMETERS
       [-h | --help]
           Display this help message
```

```
NAME
       repo use - changes the current repository

SYNOPSIS
       repo [-h | --help] use parameters

DESCRIPTION
       The use command changes the current repository used by for JCR commands
       as main command argument that will be used to select a repository:

       % repo use parameterName=parameterValue;nextParameterName=nextParameterV

       The parameters is specific to JCR plugin implementations, more details c



PARAMETERS
       [-h | --help]
           Display this help message

       parameters
           The parameters used to instantiate the repository to be used in this
```

### 6.2.2. *ws* command

```
NAME
       ws login - login to a workspace

SYNOPSIS
       ws [-h | --help] login [-u | --username] [-p | --password] [-c | --conta

DESCRIPTION

       This command login to a JCR workspace and establish a session with the 
       When you are connected the shell maintain a JCR session and allows you 
       oriented fashion. The repository name must be specified and optionally 
       have more privileges.

       Before performing a login operation, a repository must be first selecte

       % repo use container=portal

       Once a repository is obtained the login operation can be done:

       % ws login portal-system
       Connected to workspace portal-system

       % ws login -u root -p gtn portal-system
       Connected to workspace portal-system




PARAMETERS
       [-h | --help]
           Display this help message

       [-u | --username]
           The user name

       [-p | --password]
           The user password

       [-c | --container]
           The portal container name (eXo JCR specific)

       workspaceName
           The name of the workspace to connect to
```

```
NAME
       ws logout - logout from a workspace

SYNOPSIS
       ws [-h | --help] logout

DESCRIPTION
       This command logout from the currently connected JCR workspace


PARAMETERS
       [-h | --help]
           Display this help message
```

### 6.2.3. *cd* command

```
NAME
       cd - changes the current node

SYNOPSIS
       cd [-h | --help] path

DESCRIPTION
       The cd command changes the current node path. The command used with no a
       node. A relative or absolute path argument can be provided to specify a

       [/]% cd /gadgets
       [/gadgets]% cd /gadgets
       [/gadgets]% cd
       [/]%


PARAMETERS
       [-h | --help]
           Display this help message

       path
           The new path that will change the current node navigation
```

### 6.2.4. *pwd* command

```
NAME
       pwd - print the current node path

SYNOPSIS
       pwd [-h | --help]

DESCRIPTION
       The pwd command prints the current node path, the current node is produc

       [/gadgets]% pwd
       /gadgets


PARAMETERS
       [-h | --help]
           Display this help message
```

### 6.2.5. *ls* command

```
NAME
       ls - list the content of a node

SYNOPSIS
       ls [-h | --help] [-d | --depth] path

DESCRIPTION
       The ls command displays the content of a node. By default it lists the 
       accepts a path argument that can be absolute or relative.

       [/]% ls
       /
       +-properties
       | +-jcr:primaryType: nt:unstructured
       | +-jcr:mixinTypes: [exo:owneable,exo:privilegeable]
       | +-exo:owner: '__system'
       | +-exo:permissions: [any read,*:/platform/administrators read,*:/platf
       +-children
       | +-/workspace
       | +-/contents
       | +-/Users
       | +-/gadgets
       | +-/folder


PARAMETERS
       [-h | --help]
           Display this help message

       [-d | --depth]
           The depth of the printed tree

       path
           The path of the node content to list
```

### 6.2.6. *cp* command

```
NAME
       cp - copy a node to another

SYNOPSIS
       cp [-h | --help] source target

DESCRIPTION
       The cp command copies a node to a target location in the JCR tree.

       [/registry]% cp foo bar


PARAMETERS
       [-h | --help]
           Display this help message

       source
           The path of the source node to copy

       target
           The path of the target node to be copied
```

### 6.2.7. *mv* command

```
NAME
       mv - move a node

SYNOPSIS
       mv [-h | --help] source target

DESCRIPTION
       The mv command can move a node to a target location in the JCR tree. It
       command is a <Node,Node> command consuming a stream of node to move ther

       [/registry]% mv Registry Registry2


PARAMETERS
       [-h | --help]
           Display this help message

       source
           The path of the source node to move, absolute or relative

       target
           The destination path absolute or relative
```

### 6.2.8. *rm* command

```
NAME
       rm - remove one or several node or a property

SYNOPSIS
       rm [-h | --help] ... paths

DESCRIPTION
       The rm command removes a node or property specified by its path either a
       is executed against the JCR session, meaning that it will not be effecti

       [/]% rm foo
       Node /foo removed

       It is possible to specify several nodes.

       [/]% rm foo bar
       Node /foo /bar removed

       rm is a <Node,Void> command removing all the consumed nodes.


PARAMETERS
       [-h | --help]
           Display this help message

       ... paths
           The paths of the node to remove
```

### 6.2.9. *node* command

```
NAME
       node add - creates one or several nodes

SYNOPSIS
       node [-h | --help] add [-t | --type] ... paths

DESCRIPTION
       The addnode command creates one or several nodes. The command takes at l
       take more. Each path can be either absolute or relative, relative path c
       By default the node type is the default repository node type, but the op

       [/registry]% addnode foo
       Node /foo created

       [/registry]% addnode -t nt:file bar juu
       Node /bar /juu created

       The addnode command is a <Void,Node> command that produces all the nodes


PARAMETERS
       [-h | --help]
           Display this help message

       [-t | --type]
           The name of the primary node type to create.

       ... paths
           The paths of the new node to be created, the paths can either be abs
```

```
NAME
       node set - set a property on the current node

SYNOPSIS
       node [-h | --help] set [-t | --type] propertyName propertyValue

DESCRIPTION
       The set command updates the property of a node.

       Create or destroy property foo with the value bar on the root node:

       [/]% set foo bar
       Property created

       Update the existing foo property:

       [/]% set foo juu

       When a property is created and does not have a property descriptor that
       with the -t option

       [/]% set -t LONG long_property 3

       Remove a property

       [/]% set foo

       set is a <Node,Void> command updating the property of the consumed node


PARAMETERS
       [-h | --help]
           Display this help message

       [-t | --type]
           The property type to use when it cannot be inferred

       propertyName
           The name of the property to alter

       propertyValue
           The new value of the property
```

```
NAME
       node import - imports a node from an nt file

SYNOPSIS
       node [-h | --help] import source target

DESCRIPTION
       Imports a node from an nt:file node located in the workspace:

       [/]% importnode /gadgets.xml /
       Node imported



PARAMETERS
       [-h | --help]
           Display this help message

       source
           The path of the imported nt:file node

       target
           The path of the parent imported node
```

```
NAME
       node export - export a node to an nt file

SYNOPSIS
       node [-h | --help] export source target

DESCRIPTION
       Exports a node as an nt file in the same workspace:

       [/]% node export gadgets /gadgets.xml
       The node has been exported



PARAMETERS
       [-h | --help]
           Display this help message

       source
           The path of the exported node

       target
           The path of the exported nt:file node
```

### 6.2.10. *mixin* command

```
NAME
       mixin add - add a mixin to one or several nodes

SYNOPSIS
       mixin [-h | --help] add mixin ... paths

DESCRIPTION
       The add command addds a mixin to one or several nodes, this command is a
       add a mixin from an incoming node stream, for instance:

       [/]% select * from mynode | mixin add mix:versionable



PARAMETERS
       [-h | --help]
           Display this help message

       mixin
           the mixin name to add

       ... paths
           the paths of the node receiving the mixin
```

```
NAME
       mixin remove - removes a mixin from one or several nodes

SYNOPSIS
       mixin [-h | --help] remove mixin ... paths

DESCRIPTION
       The remove command removes a mixin from one or several nodes, this comma
       remove a mixin from an incoming node stream, for instance:

       [/]% select * from mynode | mixin remove mix:versionable



PARAMETERS
       [-h | --help]
           Display this help message

       mixin
           the mixin name to remove

       ... paths
           the paths of the node receiving the mixin
```

### 6.2.11. *select* command

```
NAME
       select - execute a JCR sql query

SYNOPSIS
       select [-h | --help] [-o | --offset] [-l | --limit] [-a | --all] ... que

DESCRIPTION
       Queries in SQL format are possible via the ##select## command. You can w
       by the specification and add options to control the number of results re
       to 5 results:

       [/]% select * from nt:base
       The query matched 1114 nodes
       +-/
       | +-properties
       | | +-jcr:primaryType: nt:unstructured
       | | +-jcr:mixinTypes: [exo:owneable,exo:privilegeable]
       | | +-exo:owner: '__system'
       | | +-exo:permissions: [any read,*:/platform/administrators read,*:/plat
       +-/workspace
       | +-properties
       | | +-jcr:primaryType: mop:workspace
       | | +-jcr:uuid: 'a69f226ec0a80002007ca83e5845cdac'
       ...

       Display 20 nodes from the offset 10:

       [/]% select * from nt:base -o 10 -l 20
       The query matched 1114 nodes
       ...

       It is possible also to remove the limit of displayed nodes with the -a o

       [/]% select * from nt:base -a
       The query matched 1114 nodes
       ...

       select is a <Void,Node> command producing all the matched nodes.


PARAMETERS
       [-h | --help]
           Display this help message

       [-o | --offset]
           The offset of the first node to display

       [-l | --limit]
           The number of nodes displayed, by default this value is equals to 5

       [-a | --all]
           Display all the results by ignoring the limit argument, this should

       ... query
           The query, as is
```

### 6.2.12. *xpath* command

```
NAME
       xpath - execute a JCR xpath query

SYNOPSIS
       xpath [-h | --help] [-o | --offset] [-l | --limit] [-a | --all] query

DESCRIPTION
       Executes a JCR query with the xpath dialect, by default results are lim


PARAMETERS
       [-h | --help]
           Display this help message

       [-o | --offset]
           The offset of the first node to display

       [-l | --limit]
           The number of nodes displayed, by default this value is equals to 5

       [-a | --all]
           Display all the results by ignoring the limit argument, this should

       query
           The query
```

### 6.2.13. *commit* command

```
NAME
       commit - saves changes

SYNOPSIS
       commit [-h | --help] path

DESCRIPTION
       Saves the changes done to the current session. A node can be provided t
       this nodes and its descendants only.


PARAMETERS
       [-h | --help]
           Display this help message

       path
           The path of the node to commit
```

### 6.2.14. *rollback* command

```
NAME
       rollback - rollback changes

SYNOPSIS
       rollback [-h | --help] path

DESCRIPTION
       Rollbacks the changes of the current session. A node can be provided to
       this nodes and its descendants only.


PARAMETERS
       [-h | --help]
           Display this help message

       path
           the path to rollback
```

### 6.2.15. *version* command

```
NAME
       version checkin - checkin a node

SYNOPSIS
       version [-h | --help] checkin path

DESCRIPTION
       Perform a node checkin


PARAMETERS
       [-h | --help]
           Display this help message

       path
           The node path to checkin
```

```
NAME
       version checkout - checkout a node

SYNOPSIS
       version [-h | --help] checkout path

DESCRIPTION
       Perform a node checkout


PARAMETERS
       [-h | --help]
           Display this help message

       path
           The node path to checkout
```

## 6.3. SCP usage

Secure copy can be used to import or export content. The username/password prompted by the SSH server will be used for authentication against the repository when the import or the export is performed.

### 6.3.1. Export a JCR node

The following command will export the node */gadgets* in the repository *portal-system* of the portal container *portal*:

```
scp -P 2000 root@localhost:portal:portal-system:/production/app:gadgets gadget
```

The node will be exported as *app_gadgets.xml*.

Note that the portal container name is used for GateIn. If you do omit it, then the root container will be used.

### 6.3.2. Import a JCR node

The following command will reimport the node:

```
scp -P 2000 gadgets.xml root@localhost:portal:portal-system:/production/
```

The exported file format use the JCR system view. You can get more information about that in the JCR specification.

> The SCP feature is experimental

# 7

# Hey, I want to contribute!

Drop me an email (see my @ on www.julienviet.com), any kind of help is welcome.