

# BYTEMAN / BYTEMANPKG

Byteman is a JBOSS Community project hosted at : <https://www.jboss.org/byteman>  
license LGPL

led by Andrew DINN <https://community.jboss.org/people/adinn>

BytemanPkg ( JL PASTUREL) is a Gui Front-End of Byteman , including Rules\* templates and Helpers\* to facilitate the use of Byteman.

The Gui is developped with JFX, so a JVM HotSpot 1.7 at least is needed for BytemanPkg, however the helper classes are compiled with the target 1.6 byte-code. It runs also with OpenJDK 8 + OpenJFX8.

The sources and binaries of BytemanPkg are available at :  
<https://github.com/PASTJL/bytemanPkg> license Apache 2.0

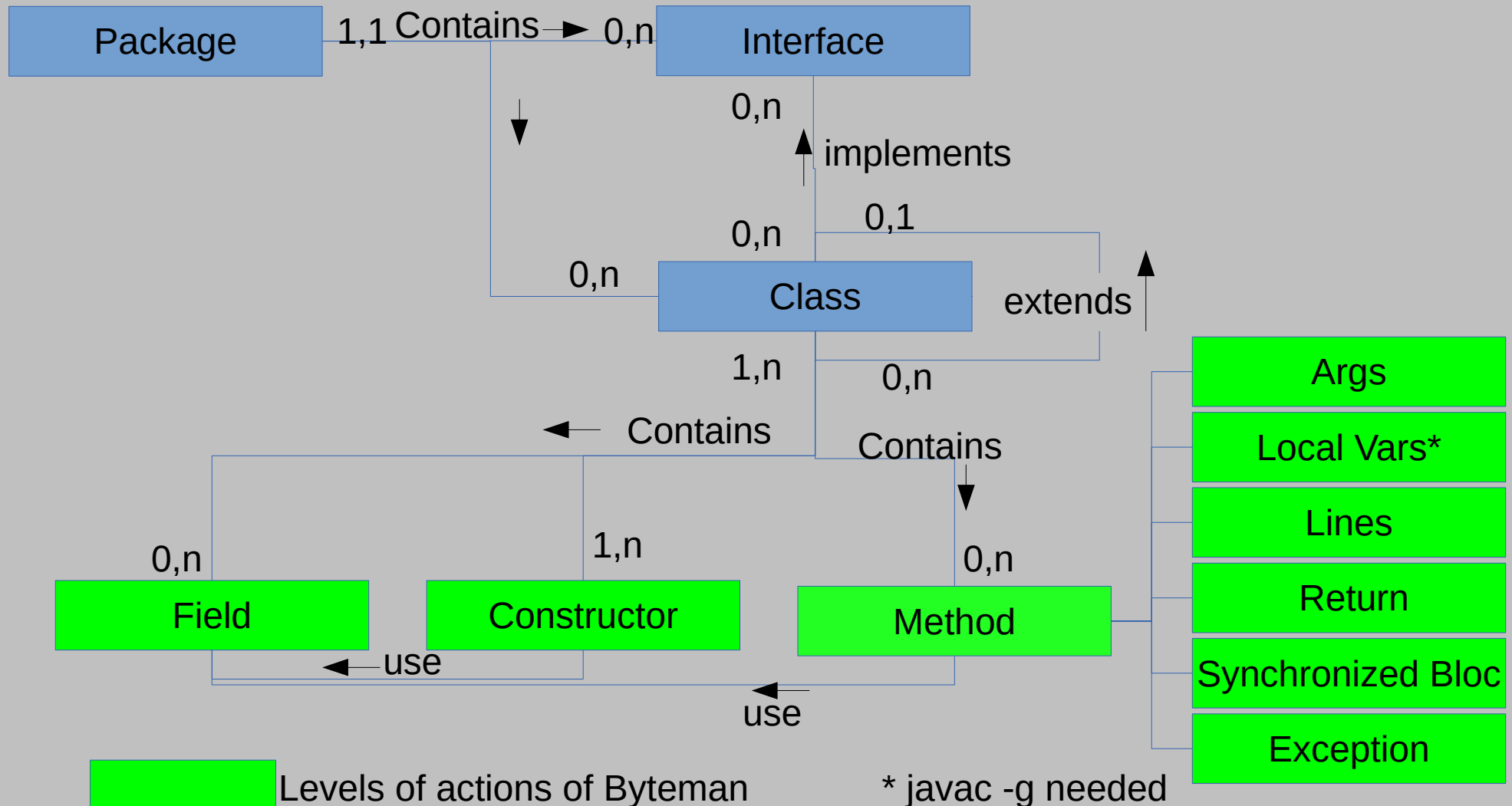
Byteman means Byte(code)man(ipulation). It is a framework with a similar approach used by [AspectJ](#) but the architecture is different.

The main feature of Byteman is to inject bytecote at runtime. Byteman is based on ASM.<http://asm.ow2.org/>

\* Rules and Helpers are explained further in this presentation.

# BYTEMAN / BYTEMANPKG

Reminder Java concepts : Simplificated relations graph



# BYTEMAN Presentation 1/9

## What can you do with Byteman ?

- **Stress tests**

- Profile / trace / logs cabled methods which values are not available with other tools (JMX for example, without modifying the original war or ear archive).  
Easier to configure than a profiler tool
- Mock Object to simulate partner remote applications.( ex Javamail)

- **Development process**

- Unit test with BMUnit / Junit / TestNG by « fault injection » . Not treated in these slides.

More information at :

<https://community.jboss.org/wiki/FaultInjectionTestingWithByteman#top>

# BYTEMAN Presentation 2/9

## Quick start configuration

Byteman can run in two modes :

- **Static submission**

- At start of the JVM, with a **-javaagent** added to the JAVA\_OPTS parameters in the start script of the WAS or the standalone application.
- Some WAS servers must also define extra parameters in the startup script and / or configuration file due to the special architecture of class loaders ( ex : JBOSS modules, OSGi bundles) See BytemanPkg document that treated the main servers WAS..
- Ex : **-javaagent:/tmp/byteman.jar=script:/tmp/bytemanrules.btm**  
The script file **bytemanrules.btm** contains the rules that will be triggered.

- **Dynamic submission**

- No need of javaagent, but extra parameters must be set.
- Use of attach API ( runs correctly with HotSpot JVM, IBM JVM issue). Needs to be launched with the same unix user of the JVM that runs the WAS / Application.
- 3 scripts to handle Byteman rules with dynamic submission :
  - **bminstall.sh** => install dynamically the agent ( That is allowed only one time for a running JVM)
  - **bmsubmit.sh** => dynamically submit / activate Byteman Rules
  - **bmunsubmit.sh** => dynamically unsubmit / deactivate Byteman Rules

Several cycles submit/unsubmit are allowed. These 3 features are embedded in a Gui interface of BytemanPkg.

# BYTEMAN Presentation 3/9

- **Byteman allows you to inject java code at runtime**
- Write declarative rule to match injection point with native Helper or Custom Helper
- Execute java code as consequence

Refer to the last version of the [documentation of Byteman](#)

**Important : Byteman needs at least a JVM 1.6 .**

The screenshot shows the JBoss Community website for the Byteman project. The top navigation bar includes links for GET STARTED, GET INVOLVED, PROJECTS, and PRODUCTS. A user profile for Jean-Louis Pasturel is visible, along with a search bar. The main header features the Byteman logo, a stylized character with a red head and yellow body. Below the header, a secondary navigation bar lists Overview, Downloads, Documentation, Community, Issue Tracker, Source Code, and Build. The Documentation section is active, showing a breadcrumb trail 'Byteman > Documentation'. The main content area lists documentation resources: Programmer's Guide (2.1.4 (latest) PDF and 2.1.4 (latest) HTML), Wiki, Getting Started Tutorial, Fault Injection Tutorial, and Rulecheck Maven Plugin Tutorial. A sidebar on the right titled 'Useful Links' contains links to Overview, Downloads, and Documentation.

JBoss Community

GET STARTED GET INVOLVED PROJECTS PRODUCTS

Jean-Louis Pasturel | Log Out SEARCH

Byteman

Overview Downloads Documentation Community Issue Tracker Source Code Build

Byteman > Documentation

Documentation

Programmer's Guide 2.1.4 (latest) PDF 2.1.4 (latest) HTML

Wiki Getting Started Tutorial Fault Injection Tutorial Rulecheck Maven Plugin Tutorial

Useful Links

Overview Downloads Documentation

# BYTEMAN Presentation 4/9

## Inside a Byteman Rule :

It is a script file, it can contains several rules

# Comments

**RULE** <Name of a Rule must be unique>

**CLASS** <name of Class full qualified or not>

# alternately **CLASS** can be replaced by **INTERFACE**

**METHOD** <Name of the Method>

**HELPER** <Fully Qualified Class for Helper>

# default helper is org.jboss.byteman.rule.helper.Helper

**AT** <location>

# **AT** is replaced by **AFTER** for certain locations

**BIND** <bindings of variables reusable in **IF** and **DO** expressions>

**IF** <Condition for applying or not applying the rule>

**DO** <execution statement when **IF** is true>

**ENDRULE**

# BYTEMAN Presentation 5/9

## Inside a Byteman Rule details / examples :

**RULE** : Give an unique name, space are allowed

Ex :

RULE myRule giving method duration

**CLASS** < full qualified Name or simple name without prefixed by a package>

Ex :

**CLASS** ^com.mchange.v2.c3p0.impl.AbstractPoolBackedDataSource

The ^ character means all Classes/ Interfaces extending or implementing the Class / Interface . No joker character allowed ( \*, ?, , )

**METHOD** <Name of method and optionnally return type and argument types>

Ex :

METHOD String myMethodReturningAString( Type1,Type2)

No joker character allowed

**HELPER** <full qualified name of Helper>

Ex :

HELPER jlp.byteman.helper.DbcpPoolManagement

**AT** <location>

Ex :

AT ENTRY

There many sort of locations ( see Byteman Programmer Guide for the details) , the main used locations are :

AT ENTRY ( the default), AT EXIT, AT INVOKE <method>, AFTER INVOKE <class.method> ,

AT THROW <opt EXCEPTION>, AT LINE <int>, AT READ < var / field> var, AT WRITE < var / field> ,

AFTER READ <var / field>, AFTER WRITE <var/field>, AT SYNCHRONIZE , AFTER SYNCHRONIZE.

# BYTEMAN Presentation 6/9

## Inside a Byteman Rule details / examples (cont) :

**BIND** <bindings of variables reusable in **IF** and **DO** expressions>

Ex :

```
BIND frequencyMeasure:int=Integer.parseInt(getProps().getProperty("dbcpPoolManagement_alias0.frequencyMeasure","1"));
count:int=incCountGlobal();
ds:org.apache.commons.dbcp.BasicDataSource=$0;
isCongruent:boolean=isCongruent(count,frequencyMeasure);
```

**IF** < Conditionnal expressions >

Ex :

IF getTrace().activeRules AND isCongruent

**DO** <statements>

Ex :

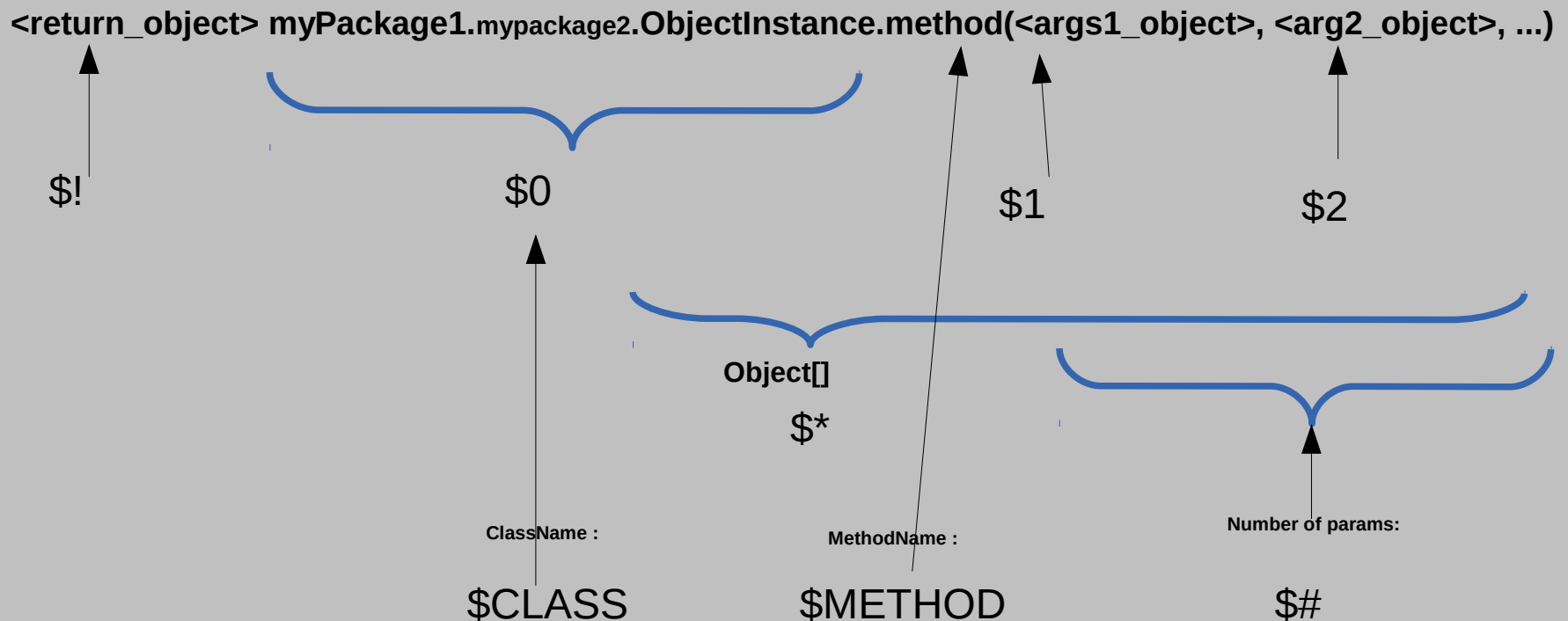
```
DO    getTrace().openTrace(getTitle(),"dbcpPoolManagement.log");
      getTrace().append(getTitle(),currentDate()dbcpPoolToTrace(ds,currentDate()));
```

The method getTrace() and child methods come from the helper Class declared in **HELPER** paragraph.



# BYTEMAN Presentation 7/9

## Main Byteman variables :



# BYTEMAN Presentation 8/9

## Custom Helper :

The simplest way to implement a Custom Helper is to extend :

`org.jboss.byteman.rule.helper.Helper`

Ex : **MyHelper** used in **BytemanPkg** :

```
imports ...
import org.jboss.byteman.rule.Rule;
import org.jboss.byteman.rule.helper.Helper;
public class MyHelper extends Helper {
    public static Trace trace = null;
    ... fields
    protected MyHelper(Rule rule) {
        super(rule);
        synchronized (MyHelper.class) {
            if (null == trace) {
                trace = Trace.getInstance();
                ... some stuff
            }
        }
    }
    public Trace getTrace(){
        return trace ;
    }
}
```

# BYTEMAN Presentation 9/9

## A Simple Mock Rule for Javamail.

The goal is to trigger the method call below :

**void javax.mail.Transport.send(..)**

with any number of parameters for method send.

Assuming that this method is called in a Class like :

```
package mypackage ;
imports ... ;
public class MyClass {
    //fields
    //methods
    public void sendAMessage(Message msg,...) {
        // some stuff , get a Transport Object => transport.
        transport.send(msg, ...) ;
        // continuing other stuff
    }
}
```

## The simplest Rule :

RULE mock javax.mail.Transport.send

CLASS **javax.mail.Transport**

METHOD **send**

# All send methods are triggered

AT **ENTRY**

IF callerEquals("**mypackage.MyClass.sendAMessage**", true,true)

# Controlling the Tree call if needed.

DO return

ENDRULE

In the DO statement, before the return statement, we can add Rule expressions like Random timer, trace to file log, using the default build-in Helper or with a custom Helper.

# BYTEMANPKG Presentation 1/8

## Quick View :

BytemanPkg is a front-end GUI for Byteman, that constructs a Byteman Rules script file, with the help of template Rules and custom Helpers. All is packaged in a unique jar => **mybyteman.jar** ( byteman agent, properties, script rule, and mandatory helpers), and uploaded to the target servers.

The running JVM target needs to be instrumented with a javaagent, or with a dynamic install/submit . For some WAS servers, we need to set variables or nwe need to modify configuration files.

The tested WAS servers are ( needs at least a JVM 1.6) :

- JBOSS 7.2 / JBOSS -EAP-6.2 (\*1)
- JOnAS 5.2.3
- TOMCAT 7.0.47 / 6.0.37
- WebSphere 8.5.5 (\*2)
- WebLogic 12c (\*3)
- GlassFish 4.0 ( JEE7) (\*4) needs JVM 1.7.
- Eclipse / Jetty 9.1

(\*1) JBOSS and JBOSS-EAP are RedHat trademarks

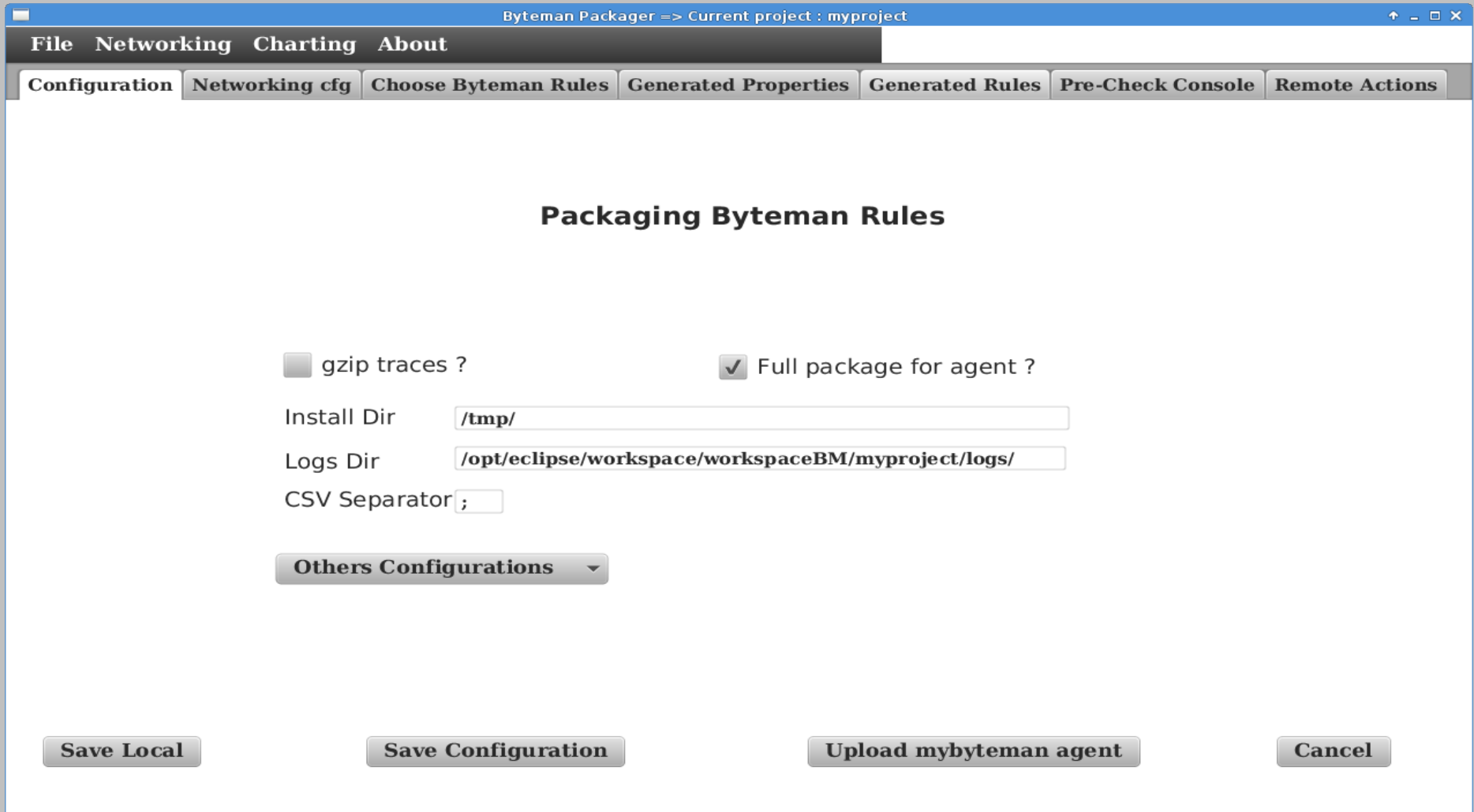
(\*2) IBM® and WebSphere® are IBM trademarks

(\*3) Oracle® WebLogic Server is an Oracle trademark

(\*4) Glassfish has a dual license <https://glassfish.java.net/public/CDDL+GPL.htm>

# BYTEMANPKG Screen Shots 2/8

## General configuration



The screenshot shows the 'Byteman Packager' application window with the title bar 'Byteman Packager => Current project : myproject'. The menu bar includes 'File', 'Networking', 'Charting', and 'About'. The main window has a tabbed interface with the following tabs: 'Configuration' (selected), 'Networking cfg', 'Choose Byteman Rules', 'Generated Properties', 'Generated Rules', 'Pre-Check Console', and 'Remote Actions'.

The 'Configuration' tab displays the 'Packaging Byteman Rules' section. It contains the following options:

- ☐ gzip traces ?
- ☒ Full package for agent ?
- Install Dir:
- Logs Dir:
- CSV Separator:
- Others Configurations** (dropdown menu)

At the bottom of the window, there are four buttons: 'Save Local', 'Save Configuration', 'Upload mybyteman agent', and 'Cancel'.

# BYTEMANPKG Screen Shots 3/8

Configuring remote SSH accesses

Byteman Packager => Current project : myproject

File Networking Charting About

Configuration Networking cfg Choose Byteman Rules Generated Properties Generated Rules Pre-Check Console Remote Actions

Connections Uploads Downloads

ID Server	Address Server	Port	Login	Password	RootPassword
myserver	localhost	22	JLP	.....	.....

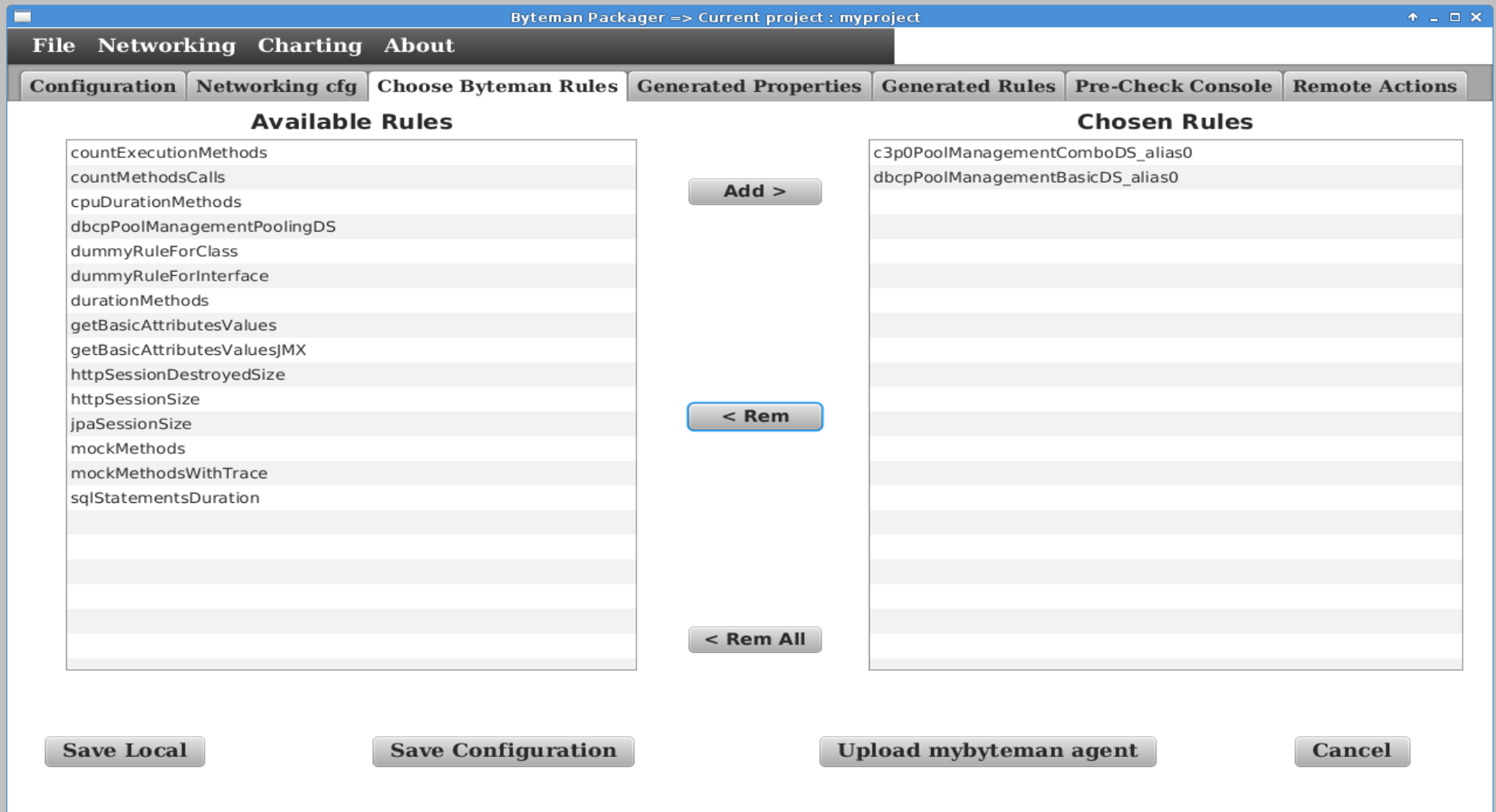
ID Server	Address Server	Port	Login	Password	RootPassword
Id Server	Address Server	22	User Login		

Save Connections Add New Cnx

Save Local Save Configuration Upload mybyteman agent Cancel

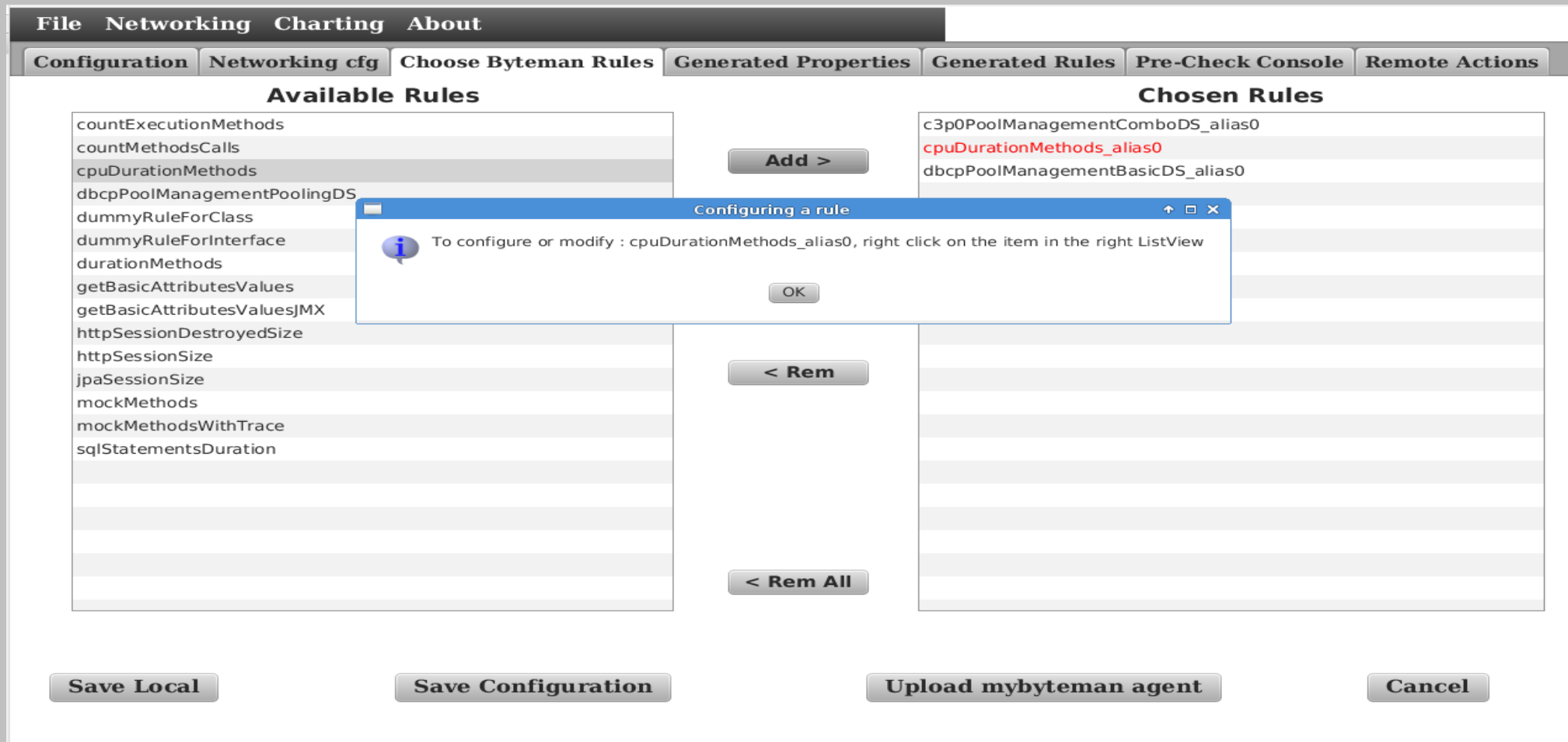
# BYTEMANPKG Screen Shots 4/8

A library of pre-configured rules. Some rules can be used more than one time, others only one time ...



# BYTEMANPKG Screen Shots 5/8

Picking template Rules on left list and ...





# BYTEMANPKG Screen Shots 6/8

... and configure custom parameters

The screenshot shows a configuration window titled "Byteman Packager => setting rule cpuDurationMethods\_alias0". It contains three main sections: "Comments", "listMethods", and "minDuration".

**Comments:** Computes the CPU duration of methods  
listMethods => list of semi-colon (;) separated (full qualified or not) names of method :  
<optional Return Type> <optional packageFullName.>className.methodName< optional list Parameter types => (Type1,Type2)>  
Pattern examples :  
- String package1.package2.MyClass.method(String,Integer)  
- Class.myMethod  
- package1.packag2.MyClass.myMethod  
- MyClass.myMethod(String, MyObjectType,Integer)  
are correct syntaxes

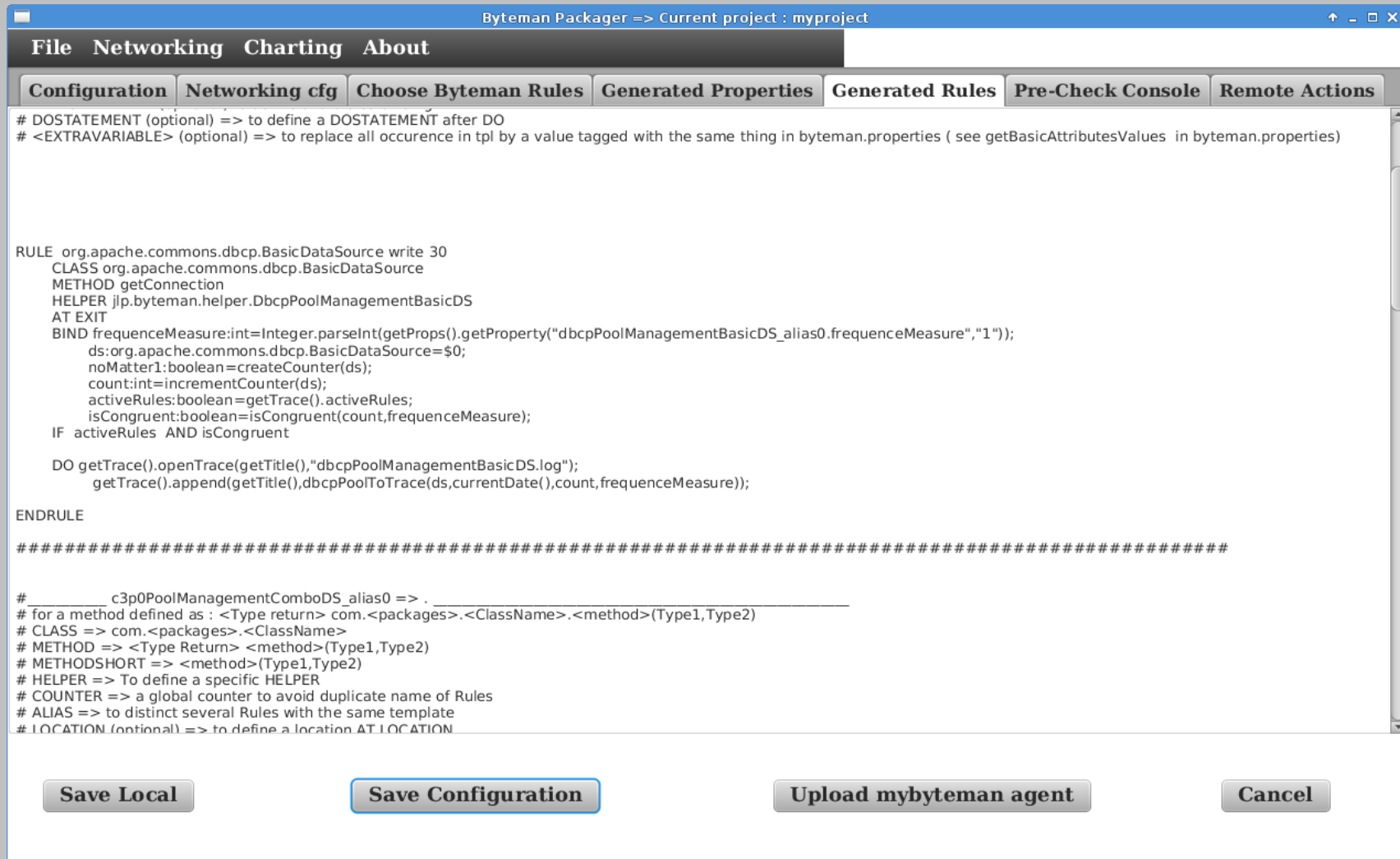
**listMethods:** Collection jdbc\_pool\_basic.data.jdbc.SimpleDAO.getData();

**minDuration:** 0

At the bottom, there are three buttons: "Save", "View Template", and "Cancel".

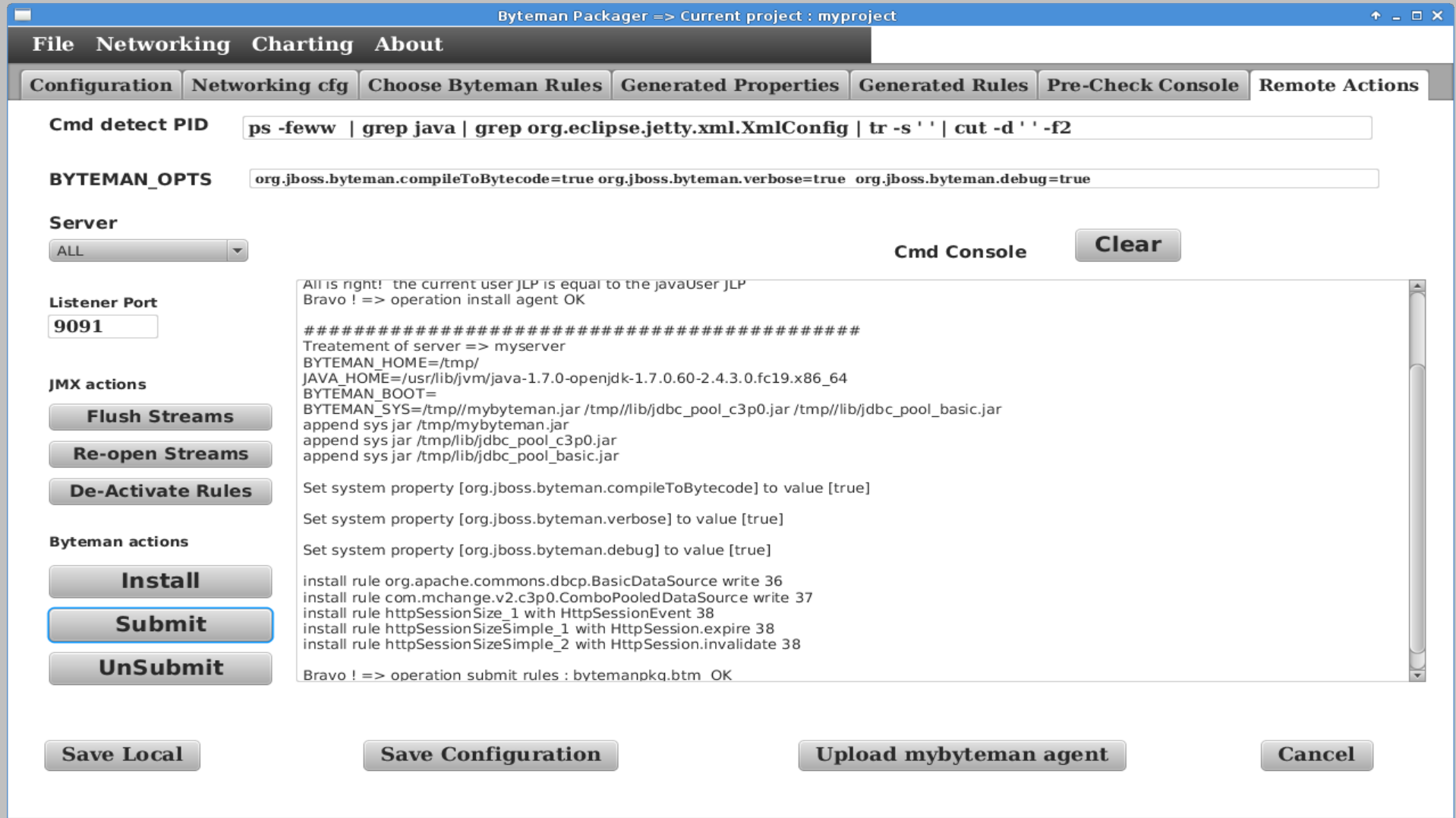
# BYTEMANPKG Screen Shots 7/8

After clicking « Save Configuration », tab « Generated Rules » gives :



# BYTEMANPKG Screen Shots 8/8

## Remote Actions and Dynamic Submission :

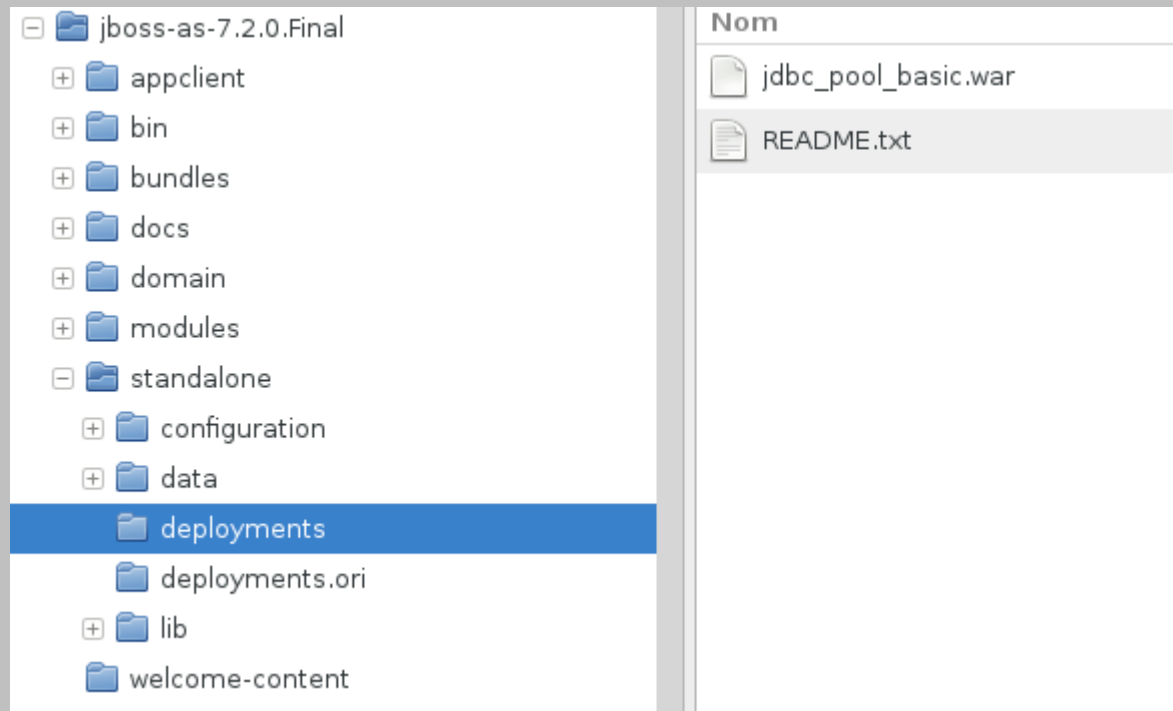


BYTEMANPKG

**DEMO  
AND  
QUESTIONS**

# BYTEMANPKG

Test with JBOSS-AS 7.2 and a little demo War application ( put/get data in a hsql database).  
This demo is done in mode « dynamic submission »



**Required configuration in file bin/standalone.conf :**

**JBOSS\_MODULES\_SYSTEM\_PKGS="org.jboss.byteman,jlp,com.ibm"**

**JAVA\_OPTS="\$JAVA\_OPTS -XX:-DisableAttachMechanism -Dcom.ibm.tools.attach.enable=yes "**

# BYTEMANPKG

After clicking on Install button ...

Byteman Packager => Current project

File Networking Charting About

Configuration Networking cfg Choose Byteman Rules Generated Properties

Cmd detect PID

BYTEMAN\_OPTS

Server

Listener Port

JMX actions

```
#####
Treatement of server => myserver
BYTEMAN_HOME=/tmp/
JAVA_HOME=/opt/jdk1.6.0_41/
PID=4928
All is right! the current user JLP is equal to the javaUser JLP
Bravo ! => operation install agent OK
```

```
Fichier Éditer Affichage Terminal Onglets Aide
15:59:01,136 INFO [org.jboss.as.server] (ServerService Thread Pool -- 26) JBAS0
18559: Deployed "jdbc_pool_basic.war" (runtime-name : "jdbc_pool_basic.war")
15:59:01,224 INFO [org.jboss.as] (Controller Boot Thread) JBAS015961: Http mana
gement interface listening on http://127.0.0.1:9990/management
15:59:01,224 INFO [org.jboss.as] (Controller Boot Thread) JBAS015951: Admin con
sole listening on http://127.0.0.1:9990
15:59:01,225 INFO [org.jboss.as] (Controller Boot Thread) JBAS015874: JBoss AS
7.2.0.Final "Janus" started in 4932ms - Started 325 of 382 services (56 services
are passive or on-demand)
16:01:16,141 INFO [stdout] (http-/127.0.0.1:8080-1) nbGet=1
16:01:16,150 INFO [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/jdbc_pool_basic]] (http-/127.0.0.1:8080-1) GetData: GetData: collection
contains 9 elements.
16:05:42,101 INFO [stdout] (Attach Listener) Setting org.jboss.byteman.allow.co
nfig.updates=true
16:05:42,101 INFO [stdout] (Attach Listener) Setting org.jboss.byteman.compileT
oBytecode=true
16:05:42,102 INFO [stdout] (Attach Listener) Setting org.jboss.byteman.verbose=
true
16:05:42,102 INFO [stdout] (Attach Listener) Setting org.jboss.byteman.debug=tr
ue
16:05:42,108 INFO [stdout] (Attach Listener) TransformListener() : accepting re
quests on localhost:9091
```

# BYTEMANPKG

After clicking on Submit button ...

```
#####
Bravo ! => operation install agent OK

#####
Treatement of server => myserver
BYTEMAN_HOME=/tmp/
JAVA_HOME=/opt/jdk1.6.0_41/
BYTEMAN_BOOT=
BYTEMAN_SYS=/tmp//mybyteman.jar /tmp//lib/jdbc_pool_c3p0.jar /tmp//lib/jdbc_pool_basic.jar
append sys jar /tmp/mybyteman.jar
append sys jar /tmp/lib/jdbc_pool_c3p0.jar
append sys jar /tmp/lib/jdbc_pool_basic.jar

Set system property [org.jboss.byteman.compileToBytecode] to value [true]

Set system property [org.jboss.byteman.verbose] to value [true]

Set system property [org.jboss.byteman.debug] to value [true]

install rule org.apache.commons.dbcp.BasicDataSource write 32
install rule com.mchange.v2.c3p0.ComboPooledDataSource write 33
install rule cpuDurationMethods_1 jdbc_pool_basic.data.jdbc.SimpleDAO.getData() 34
install rule cpuDurationMethods_2 jdbc_pool_basic.data.jdbc.SimpleDAO.getData() 34

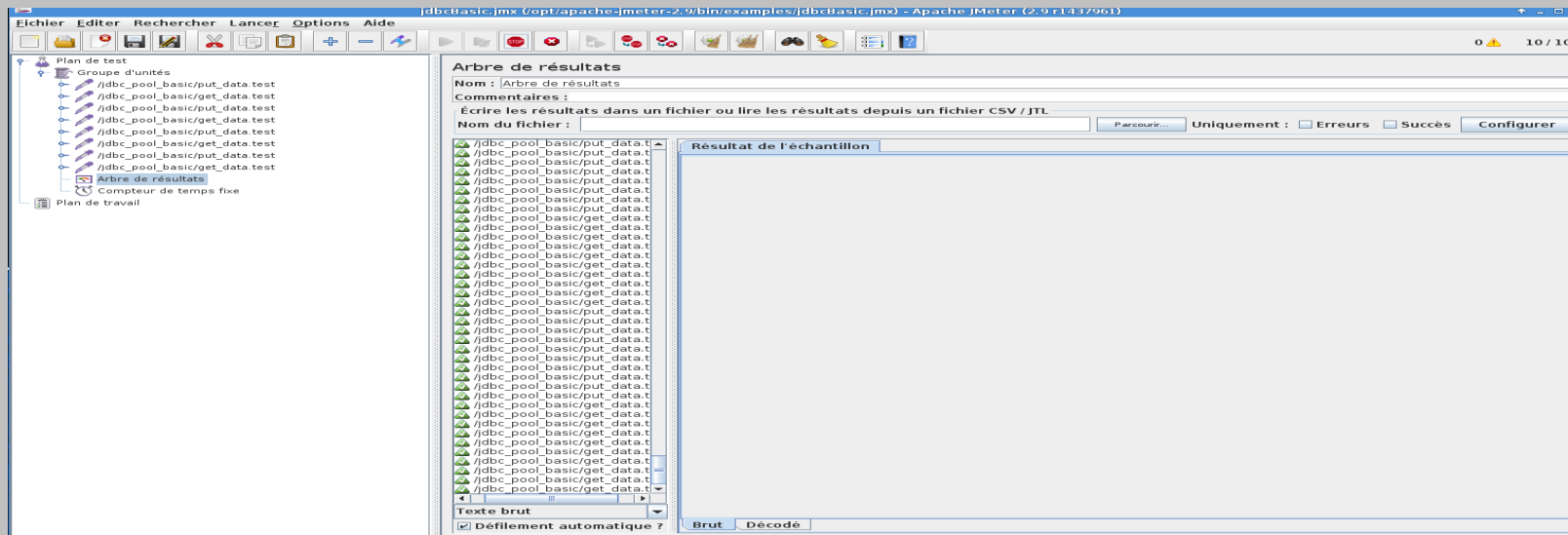
Bravo ! => operation submit rules : bytemanpkg.btm OK
```

```
requests on localhost:9091
16:10:27,749 INFO [stdout] (Thread-73) TransformListener() : handling connection on port 9091
16:10:27,846 INFO [stdout] (Thread-73) TransformListener() : handling connection on port 9091
16:10:27,945 INFO [stdout] (Thread-73) TransformListener() : handling connection on port 9091
16:10:28,040 INFO [stdout] (Thread-73) TransformListener() : handling connection on port 9091
16:10:28,135 INFO [stdout] (Thread-73) TransformListener() : handling connection on port 9091
16:10:28,181 INFO [stdout] (Thread-73) retransforming org.apache.commons.dbcp.BasicDataSource
16:10:28,221 INFO [stdout] (Thread-73) org.jboss.byteman.agent.Transformer : possible trigger for rule org.apache.commons.dbcp.BasicDataSource write 32 in class org.apache.commons.dbcp.BasicDataSource
16:10:28,232 INFO [stdout] (Thread-73) RuleTriggerMethodAdapter.injectTriggerPoint : inserting trigger into org.apache.commons.dbcp.BasicDataSource.getConnection() java.sql.Connection for rule org.apache.commons.dbcp.BasicDataSource write 32
16:10:28,254 INFO [stdout] (Thread-73) org.jboss.byteman.agent.Transformer : inserted trigger for org.apache.commons.dbcp.BasicDataSource write 32 in class org.apache.commons.dbcp.BasicDataSource
```

Rules are ready to run. Stressing the war application with Jmeter ...

# BYTEMANPKG

JMeter test running ...



Byteman Rules generate logs files ...

workspaceBM

+

jboss



-

myproject

boot

logs

...

Nom	Taille	Type	Date	Propriétaire
 dbcpPoolManagementBasicDS_20131227_184521.log	82,1 ko	journal d'application	2013-12-27 18:46:1	JLP (JLP)
 CPUdurationMethods_20131227_184523.log	32,9 ko	journal d'application	2013-12-27 18:46:1	JLP (JLP)



# BYTEMANPKG

Logs are « chartable » with the menu Charting/Viewer

