PETALS 1.0
User Guide

eBM Websourcing

Contents

Figures

1 Installation

PETALS is an open source Enterprise Service Bus.

It fully implements the Java Business Integration specification.

You can find information about JBI at http://www.jcp.org/en/jsr/detail?id=208.

1.1 *Requirement*

To run PETALS, you need a Java[1] JRE 1.5.

You can get it at http://java.sun.com/j2se/1.5.0/download.jsp

1.2 *Download the distribution*

You can retrieve the latest version of PETALS from the download section.

Download the distribution archive file (bin) at

http://forge.objectweb.org/project/showfiles.php?group_id=213

1.3 *Install the distribution*

Explode the downloaded file into the directory where you want PETALS to be installed. This directory is the PETALS root path.

PETALS computes itself its installation path. But on some systems, the root path can not be deduced correctly. In such a case, PETALS_HOME environment variable has to be set manually:

On UNIX like systems: `export PETALS_HOME=/path_to_Petals_directory`

On WINDOWS: `set PETALS_HOME=/ path_to_Petals_directory`

1.4 *PETALS directory structure*

PETALS distribution has the following directory structure:

- ***ant*** includes a sample file containing examples ant task definitions and scripts and the libraries you have to import in your project to use these ant tasks.
- ***bin*** includes scripts to launch PETALS on UNIX (`startup.sh`) or WINDOWS (`startup.bat`) systems.
- ***components*** includes all components archives bundled with PETALS.
- ***conf*** includes all PETALS configuration files. For example: the general PETALS server properties (`server.properties`), the PETALS core component architecture (`*.fractal`)...
- ***demos*** includes all demonstration components and service assemblies that illustrate application composition with PETALS (ex : Mortgage demonstrator).

[1]Java is a Tradmark of Sun Microsystems, Inc. in the United States and other countries.

- ***`extras`*** is used to bundle all complementary libraries and tools.
- ***`install`*** is the directory where you have to copy components (service engines or binding components) or service assemblies so that PETALS install (or deploy) and start them automatically.
- ***`installed`*** components and services assemblies are automatically copied into this directory after a successful installation (or deployment).
- ***`JORAM`*** includes all files generated by the JORAM transporter during PETALS execution.
- ***`lib`*** includes all libraries used by PETALS system.
- ***`logs`*** includes all logs generated during PETALS execution.
- ***`lost+found`*** contains elements previously installed that have not been correctly recovered during container restart.
- ***`monitoring`*** includes configuration files and libraries allowing to start PETALS with monitoring capabilities.
- ***`repository`***: includes all the libraries, configuration files (etc.) of the installed or deployed components, shared libraries and service assemblies.
- ***`schema`*** includes all XML schemas used by PETALS.
- ***`uninstalled`*** is the directory where components and services assemblies are automatically copied after a successful uninstallation (or undeployment).
- ***`work`*** is used by PETALS system to put components or SA archives during installation or deployment phases.

2 Configuration

Once you have downloaded and installed PETALS, you can tune the configuration of the container.

All the configuration files can be found in the **`conf`** directory.

2.1 Server configuration

The server configuration file is `server.properties`, divided into 4 parts:

- Container configuration (the inner properties),
- JMX configuration (management from external console),
- JNDI configuration (distributed directory),
- JORAM configuration (JMS server, for message transport).

PETALS assigns itself the configuration properties, even if participating in network of several PETALS servers. You don't have to change them unless you need to set specific ports, for instance if PETALS runs behind a proxy or a firewall.

2.1.1 Container configuration

- ***`exchange.validation`***: this property is used to set the behavior of the container regarding the validation of an exchange of message between JBI components (see the method `isExchangeOkayWithProvider` or `isExchangeOkayWithConsumer` in the JBI specification).
- ***`tribe.multicast`***: this property is a multicast IP address used in the JNDI server which uses Tribe.
- ***`JORAM.domain`***: this property is the name of the domain that the JNDI server and the JORAM server will use.

The `tribe.multicast` and the `JORAM.domain` define the network of PETALS containers. Thus if you want to have two containers in the same network, you have to set the same value to this two properties in their respective configuration files.

2.1.2 JMX configuration

- ***`html.port`***: HTML port adaptor of the JMX server. You can point your preferred browser to http://ip_of_the_host:html.port to visualize the content of the JMX server.
- ***`jmx.port`***: this port is the port of the JMX server. You can connect to the JMX server using the following address:

  `service:jmx:rmi:///jndi/rmi://if_of_the_host:jmx.port/management/rmi-jmx-connector.`
- ***`jmx.user`***: user name to connect to the JMX server. Right now there is no security on the JMX server, thus there is no authentication.

- ***jmx.pwd***: password to connect to the JMX server.

2.1.3 JNDI configuration

- ***java.naming.factory.port*** is the port where used by the JNDI server on stertup.
- ***java.naming.factory.initial***: this property contains the name of the JNDI factory that needs to be used to connect to the JNDI server.

2.1.4 JORAM configuration

- ***JORAM.id*** is the id of the JORAM server. If not specified, this property is automatically set during container initialisation.
- ***JORAM.domainport***: this property is the port used by this JORAM server to communicate within the domain.
- ***JORAM.tcpport***: this port is the port used to connect to the *AdminModule* of the JORAM server.
- ***JORAM.user***: this property is the username to connect to the *AdminModule*.
- ***JORAM.pwd***: this property is the password of the user to connect to the *AdminModule*.

2.2 Loggers configuration

The logging configuration is defined in `%PETALS_HOME%/conf/loggers.properties`.

There are 3 logging levels:

- ***logger.PETALS.level***, set to `INFO by default`,
- ***logger.fr.dyade.aaa.level***, set to `FATAL by default`,
- ***logger.org.objectweb.JORAM.level***, set to `FATAL by default`.

The first one is for the PETALS logs, the others are used by JORAM.

Each PETALS internal component have a specific logging LEVEL which is commented by default. If you need to change the LEVEL for a specific component, uncomment the related line.

The LEVEL used to log the JBI components messages is

logger.PETALS.components.level

By default, messages are logged to:

- the standard console,
- the `%PETALS_HOME%/logs/PETALS.log`, for PETALS messages,
- the `%PETALS_HOME%/logs/JORAM.log`, for JORAM messages.

3 Start PETALS

To start PETALS, just follow these following steps:

Go to `%PETALS_HOME%/bin`.

Use `startup.sh` (for UNIX systems) or `startup.bat` (for Windows systems).

You can also use the `java -jar server.jar` command line.

```
[petals.jndi-server]  SocketServerThread.run() Start JNDI server on
192.168.1.149:7720

[petals.joramagent]   JoramAgentImpl.startServer() Joram server started at
[host:port 192.168.1.149:7740,tcp port 7760, jndi port , domain name
Petals, server name 0, server id 0

HttpAdaptor version 3.0.1 started on port 7080

[petals.rmi]          RMIConnector.startFc() JMX RMI server started at :
/jndi/rmi://192.168.1.149:7700/management/rmi-jmx-connector

##### Petals container correctly started #####
```

`startupWithCommandLine.sh/bat` allows you to interact with the server with commands. Use '`help`' at prompt to see the commands list.

```
##### Petals container correctly started #####

Petals prompt. Tape 'help' for help.

 - hotdeploy [ZIP filePath] (or hd): install and start a component, a
service assembly or a share library

 - hotundeploy [ZIP fileName] (or hu): shutdown and uninstall a component,
a service assembly or a share library

 - path (or p)      : display current file system path

 - setpath [Path] (or sp): change current file system path

 - jndi (or d)      : display the naming directory

 - help (or h)      : this help

 - stop (or q)      : stop Petals

 - shutdown (or x) : shutdown Petals (remove from the Petals network
```

When you stop PETALS, the server configuration is stored. If the server takes part of a PETALS network, all the Services registered on this server are always visible and messages can be sent to these services.

When you restart this PETALS server, all the elements installed on the server are restarted and pending messages are delivered.

4 Administration

PETALS provides management tasks accessible through JMX. These tasks are described in the JBI specification. JBI implementation provides management beans (MBeans) that expose management functions.

Please read fully the Java Business Integration specification to know how to manage the JBI container.

The JBI management MBeans are:

AdminService MBean

This component allows retrieving a single component or list of components, to find the nature (binding or engine) of a component, to retrieve system services and system information. For more information see AdminService MBean section in the JBI specification.

InstallationService MBean

This component allows to install/uninstall shared libraries and components and to manage (create, register, retrieve, destroy) InstallerMBean. Installers are used to perform real installation/uninstallation tasks on components. For more information see InstallationServiceMBean section in the JBI specification.

DeploymentService MBean

This component provides capabilities to deploy or undeploy service assemblies and to create/manage their life cycle (start, stop, and shutdown). It allows listing Service Assemblies and Service Units that match particular criteria (for instance: service units deployed to a particular component). For more information see DeploymentService MBean section in the JBI specification.

Installer MBean (one for each component)

It allows processing the installation or uninstallation of a component. During installation it creates a `ComponentLifeCycle`, which is the implementation of the `ComponentLifeCycleMBean` and `LifeCycleMBean`. For more information see InstallerMBean section in the JBI specification.

ComponentLifeCycle MBean (one for each component)

It allows to init, start, stop and shutdown a component. For more information see ComponentLifeCycleMBean section in the JBI specification.

4.1 JMX administration via HTTP

An HTTP JMX connector is launched with the server. This connector allows you to manage JBI elements, through the PETALS tab.

This connector can be accessed on your machine at http://<server>:<port>/. The default port is 7080.

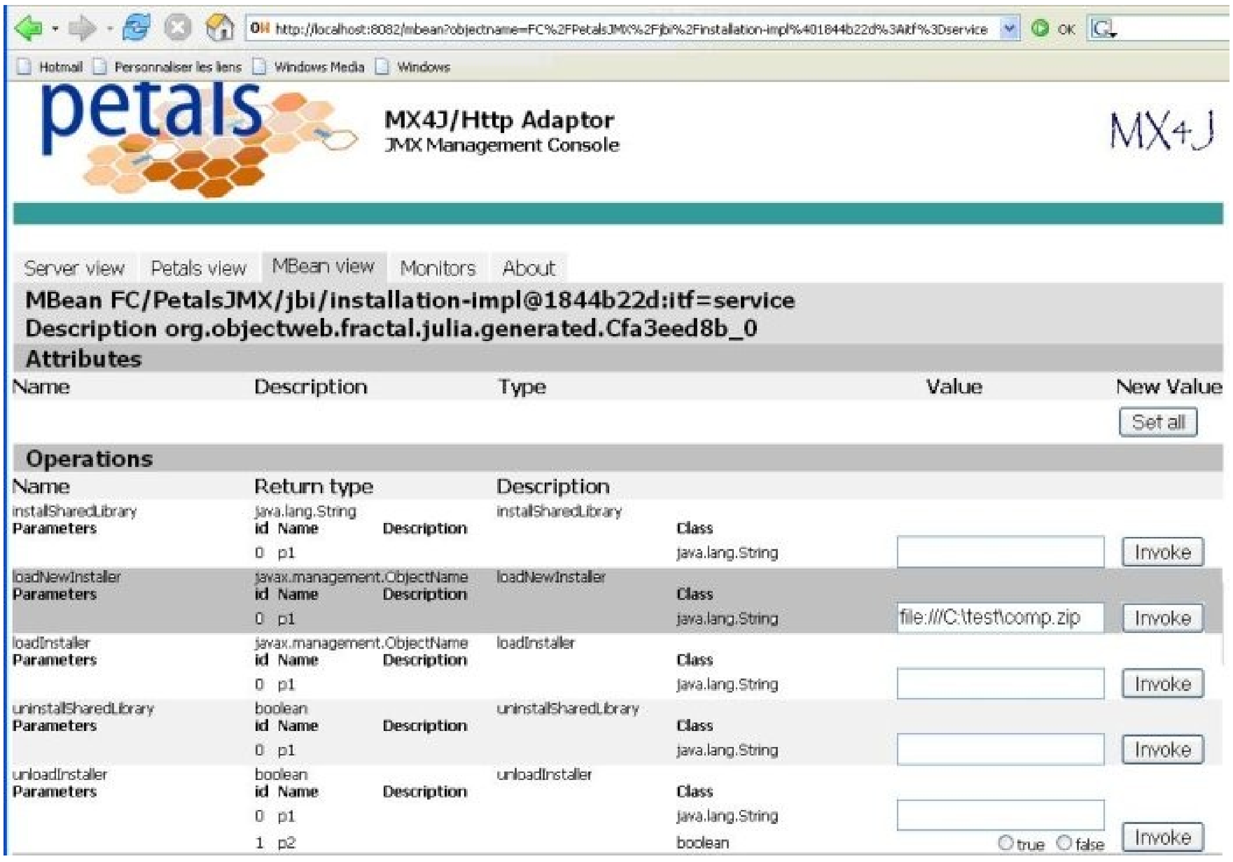
Use explicitly your machine name or IP, not `'localhost'` or `'127.0.0.1'`.

If multiple containers are started on the same machine, ports are `'7081'`, `'7082'`, and so on. This port number is displayed at PETALS launch.

4.1.1 Install a Component

The MBean that manages component installation is the `InstallationServiceMBean`. The `loadNewInstaller()` method explodes and analyses the archive corresponding to the given URL and returns the name of a new MBean object, an `InstallerMBean`.

Figure below shows how to load an Installer with the JMX API, through the PETALS HTML console.



MBean FC/PetalsJMX/jbi/installation-impl@1844b22d:itf=service
Description org.objectweb.fractal.julia.generated.Cfa3eed8b_0

Attributes

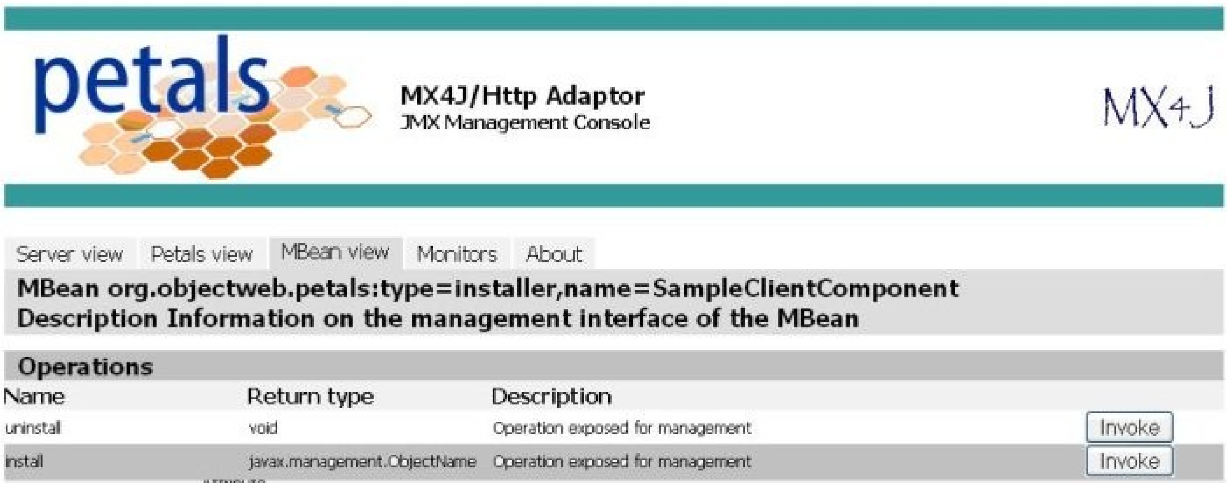
Name	Description	Type	Value	New Value
				Set all

Operations

Name	Return type	Description			
installSharedLibrary	java.lang.String	installSharedLibrary			
Parameters	**id Name Description**		**Class**		
	0 p1		java.lang.String		Invoke
loadNewInstaller	javax.management.ObjectName	loadNewInstaller			
Parameters	**id Name Description**		**Class**		
	0 p1		java.lang.String	file:///C:\test\comp.zip	Invoke
loadInstaller	javax.management.ObjectName	loadInstaller			
Parameters	**id Name Description**		**Class**		
	0 p1		java.lang.String		Invoke
uninstallSharedLibrary	boolean	uninstallSharedLibrary			
Parameters	**id Name Description**		**Class**		
	0 p1		java.lang.String		Invoke
unloadInstaller	boolean	unloadInstaller			
Parameters	**id Name Description**		**Class**		
	0 p1		java.lang.String		
	1 p2		boolean	○ true ○ false	Invoke

Figure 1 - JMX HTML Installation Service MBean

The `InstallerMBean` installs and uninstalls the component with the `install()` and `uninstall()` methods.



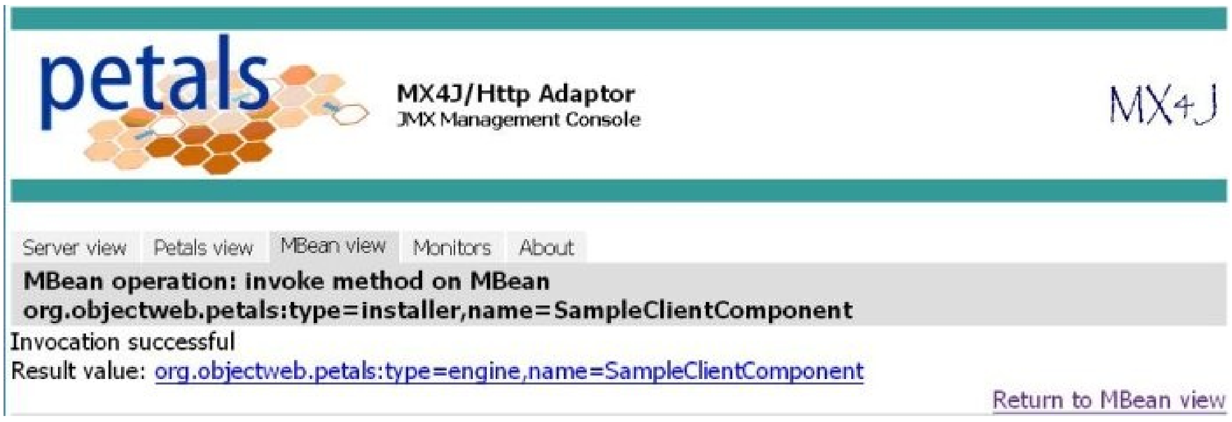
MBean org.objectweb.petals:type=installer,name=SampleClientComponent
Description Information on the management interface of the MBean

Operations

Name	Return type	Description
uninstall	void	Operation exposed for management
install	javax.management.ObjectName	Operation exposed for management

Figure 2- JMX HTML the component's Installer MBean

The `install()` method returns the name of the component's life cycle MBean.

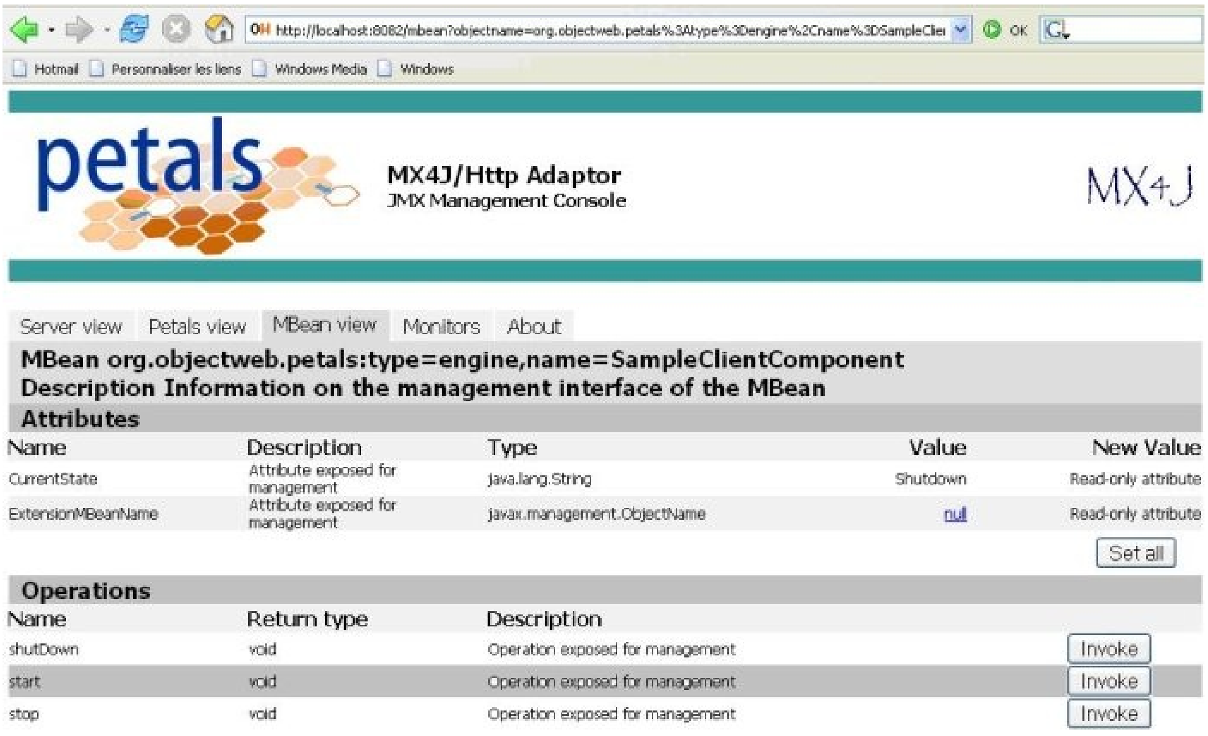


MBean operation: invoke method on MBean
org.objectweb.petals:type=installer,name=SampleClientComponent
Invocation successful
Result value: org.objectweb.petals:type=engine,name=SampleClientComponent

Figure 3 - JMX HTML install() invocation return the life cycle MBean name.

4.1.2 Start a Component

The `ComponentLifeCycleMBean` manages the component's life cycle through `start()`, `stop()`, and `shutdown()` methods.



MBean org.objectweb.petals:type=engine,name=SampleClientComponent
Description Information on the management interface of the MBean

Attributes

Name	Description	Type	Value	New Value
CurrentState	Attribute exposed for management	java.lang.String	Shutdown	Read-only attribute
ExtensionMBeanName	Attribute exposed for management	javax.management.ObjectName	null	Read-only attribute

Operations

Name	Return type	Description	
shutDown	void	Operation exposed for management	Invoke
start	void	Operation exposed for management	Invoke
stop	void	Operation exposed for management	Invoke

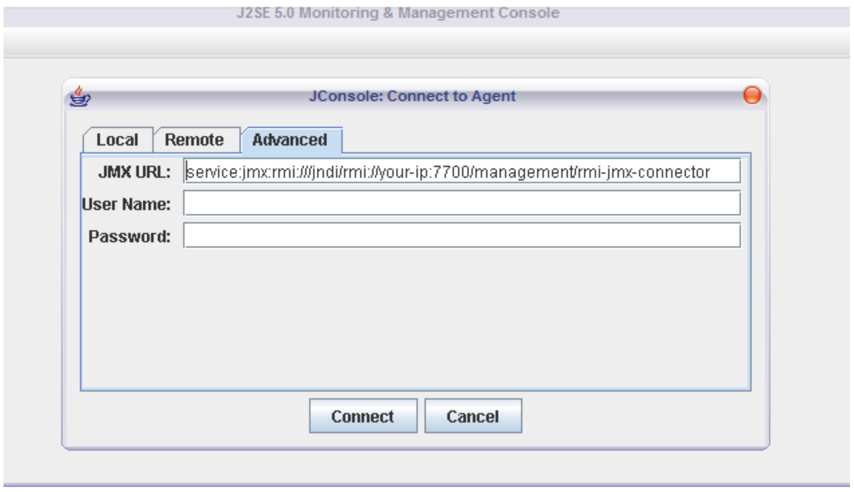
Figure 4 - JMX HTML Component LifeCycle Mbean

4.2 JMX administration via RMI (Java Console)

A RMI JMX connector is launched with the server.

Then, you can administrate PETALS from any RMI client, and specially by using the graphical Java JMX console provided with the JDK: `%JAVA_HOME%/bin/jconsole`.

Use the advanced connection mode to connect to a PETALS server.


J2SE 5.0 Monitoring & Management Console
JConsole: Connect to Agent
Local
Remote
Advanced
Connect
Cancel

JMX URL: service:jmx:rmi:///jndi/rmi://your-ip:7700/management/rmi-jmx-connector
User Name:
Password:

Figure 5 - JMX RMI Connection dialog box

<u>JMX URL:</u>

```
service:jmx:rmi:///jndi/rmi://[server-ip]:[server-rmi-
port]/management/rmi-jmx-connector
```

Use explicitly your machine name or IP, not `'localhost'` or `'127.0.0.1'`.

If multiple containers are started on the same machine, ports are `'7701'`, `'7702'`, and so on. This port number is displayed at launch.

Just click on the "`Connect`" button and you will be connected to the PETALS JMX server through the RMI connector. A user interface like below will appear,

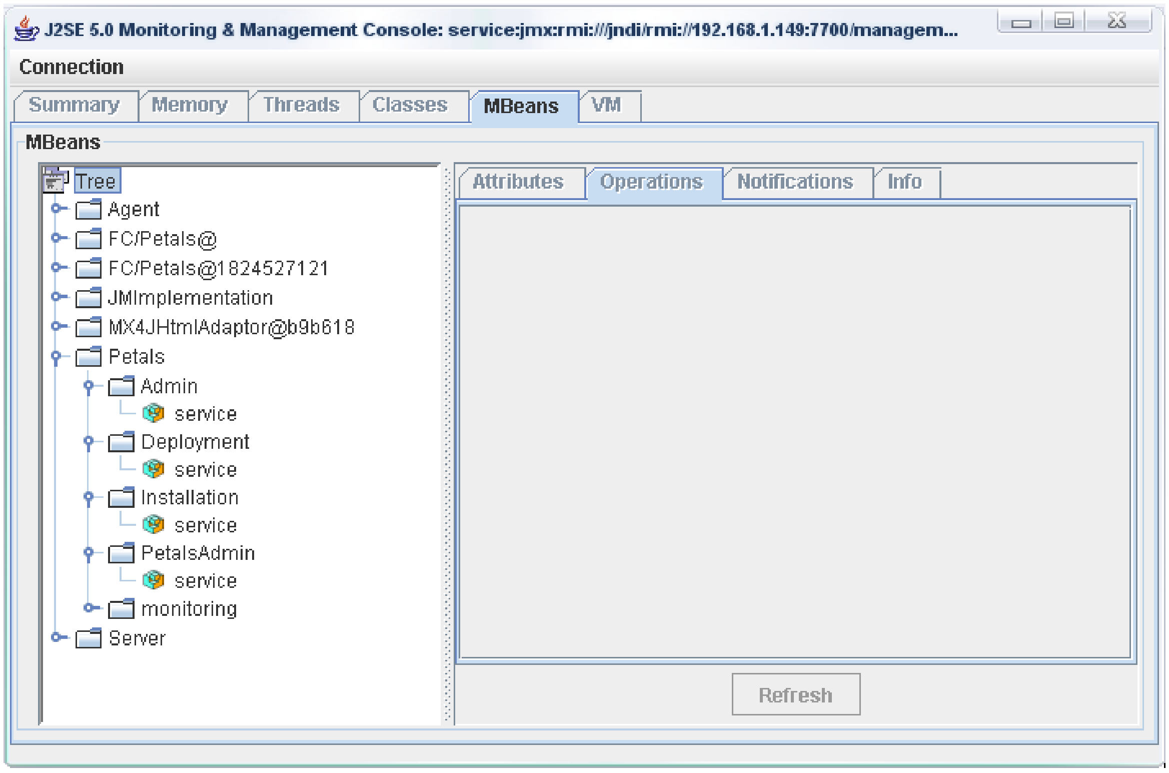


Figure 6: JMX RMI Main frame

On the left hand of the RMI Console, you will retrieve all the MBeans that allow administrating the PETALS Platform. On the right hand, you can find the list of operations available for the selected MBean (`PETALS/Installation`, `PETALS/Deployment`, `PETALS/Admin`, `PETALS/PetalsAdmin`).

4.2.1 Install a Component

To install a component just click on the `InstallationServiceMBean` (`PETALS/Installation`) on the left part of the RMI Console and go to `Operation` tab on the right part. Fill the `String` parameter of the `loadNewInstaller` method with the URL of the component and click on the `loadNewInstaller` button.

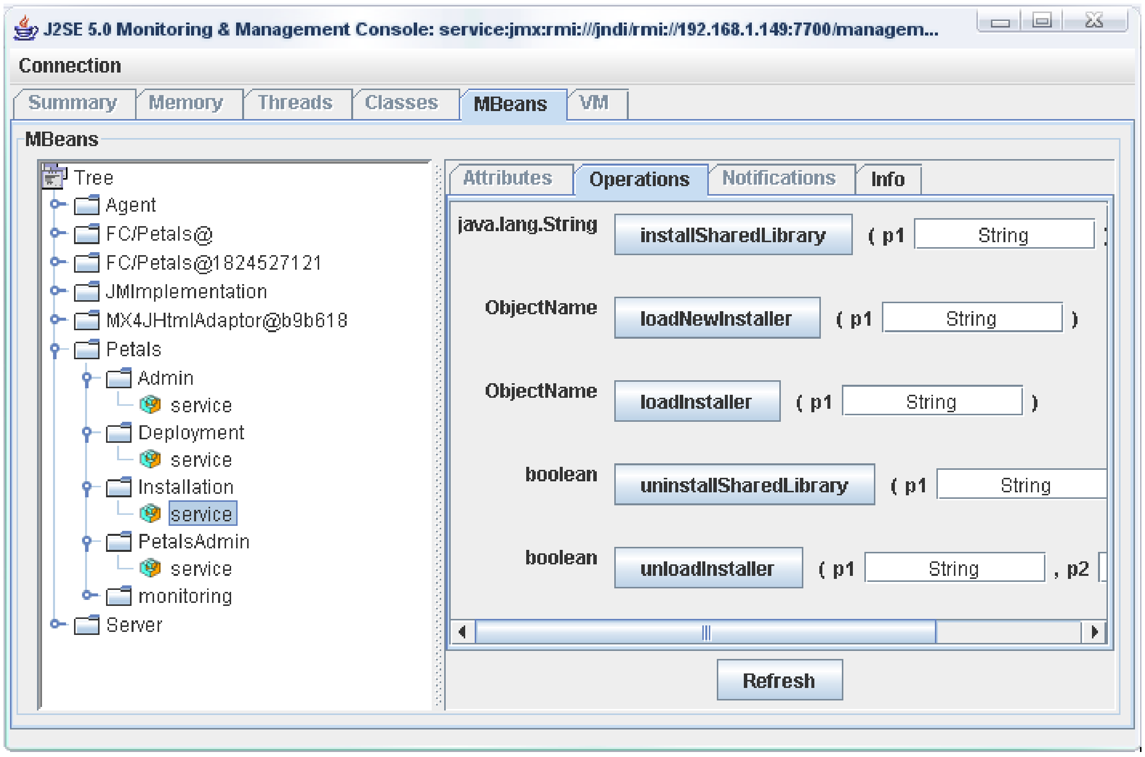

J2SE 5.0 Monitoring & Management Console: service:jmx:rmi:///jndi/rmi://192.168.1.149:7700/managem...
Connection
Summary
Memory
Threads
Classes
MBeans
VM
MBeans
Tree
Agent
FC/Petals@
FC/Petals@1824527121
JMImplementation
MX4JHtmlAdaptor@b9b618
Petals
Admin
service
Deployment
service
Installation
service
PetalsAdmin
service
monitoring
Server
Attributes
Operations
Notifications
Info
java.lang.String installSharedLibrary (p1 String
ObjectName loadNewInstaller (p1 String)
ObjectName loadInstaller (p1 String)
boolean uninstallSharedLibrary (p1 String
boolean unloadInstaller (p1 String , p2
Refresh

Figure 7: JMX RMI Installation Service MBean

To complete the installation of the component just click on the new `InstallerMbean` on the left of the RMI Console (into `org.objectweb.PETALS/installer/[component name]`) and go to `Operation` tab on the right. Now click on the `install` button.

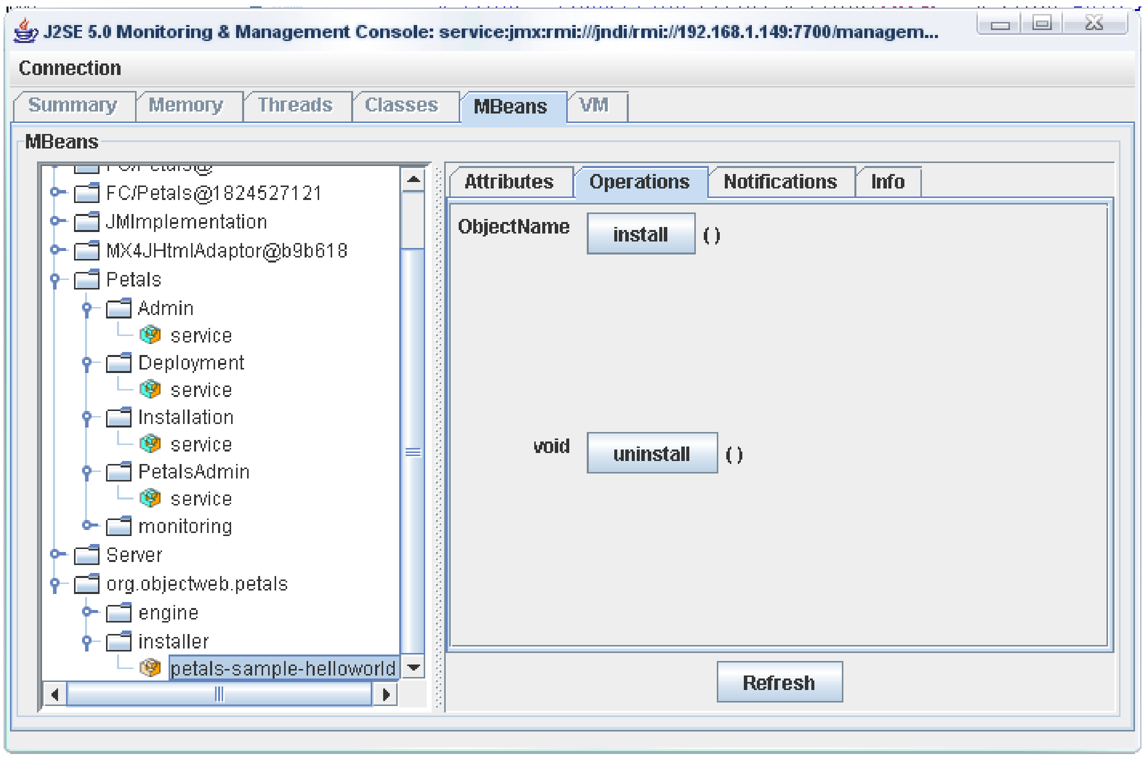


Figure 8: JMX RMI Installer Mbean

4.2.2 Start a Component

Once the component is installed a new MBean appears which allows you to start the component. To do it simply click on this Mbean.

(`org.objectweb.PETALS/[engine or binding]/[component-name]` and go to `Operation` tab. Now click on the `start` button.

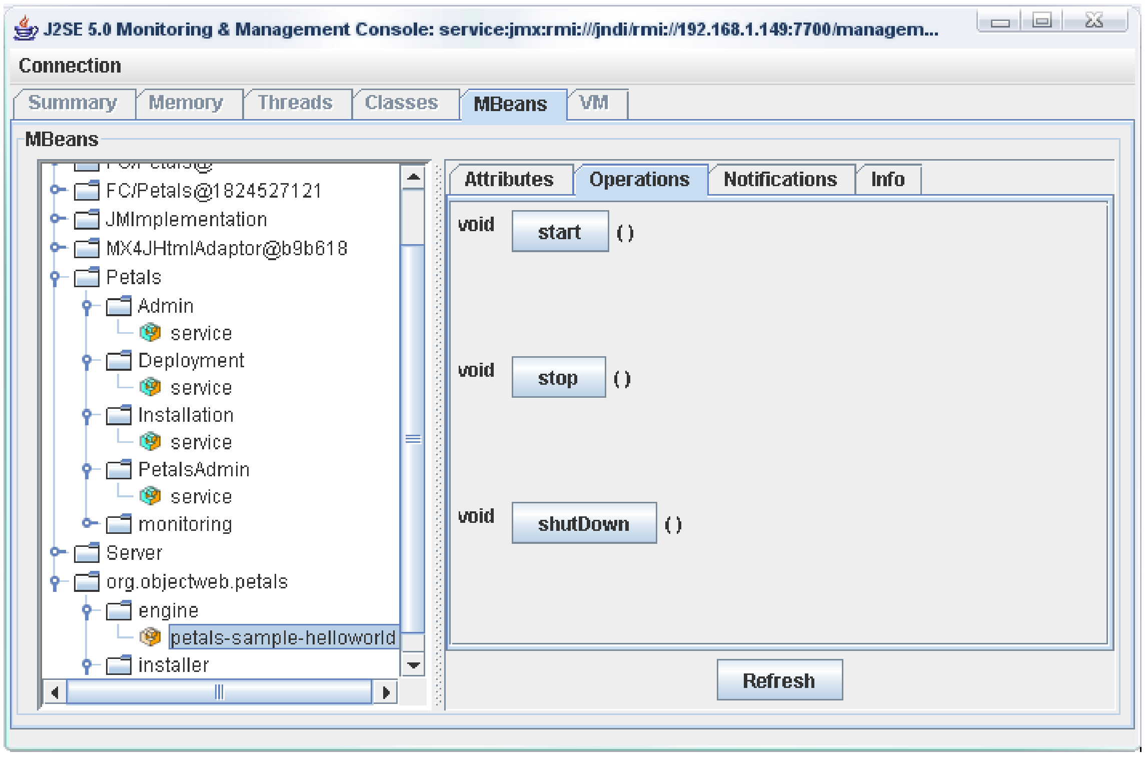


Figure 9: JMX RMI Component LifeCycle MBean

4.3 Ant tasks

You can have access to management function through Ant Tasks. These Ant tasks call the corresponding JMX tasks.

See section 6.10.6 of the JBI Specification.

4.3.1 Install a Component

To install a component, just run this ant task:

```
<path id="tests.classpath">
     <pathelement location="lib/PETALS-ant-1.0.jar" />
</path>
<taskdef name="jbi-install-compo"
classname="org.objectweb.PETALS.tools.ant.InstallComponentTask">
     <classpath refid="tests.classpath" />
</taskdef>
<target name="install-component" description="Install a component">
     <jbi-install-compo file="file:///c:/Mycomp.zip" port="7700"
host="localhost" />
</target>
```

This will load a new `InstallerMBean` and execute the `install()` method. The component is therefore completely installed.

4.3.2 Start a Component

To start a component, just run this ant task:

```
<path id="tests.classpath">
     <pathelement location="lib/PETALS-ant-1.0-SNAPSHOT.jar" />
</path>
<taskdef name="jbi-install-compo"
classname="org.objectweb.PETALS.tools.ant.InstallComponentTask">
     <classpath refid="tests.classpath" />
</taskdef>
<target name="start-compo" description="Start a JBI component">
     <jbi-start-compo name="MyComp" port="7700" host="localhost"/>
</target>
```

You can find a sample ant build file in `%PETALS_HOME%/ant`. This ant file contains examples for each ant task that allow managing PETALS.

4.4 Hot deploy

PETALS allows automatic installation of JBI components or shared libraries, and the automatic deployment of artefacts for installed components.

You can find two specific folders in PETALS home:

- ***`install`***: The Components, Service Assemblies and Shared libraries (packaged as Zip archive) put into this folder are automatically installed into the PETALS environment and started.
- ***`installed`***: The Zip archives of elements that have been correctly installed into the environment are moved from `install` to `installed`. When you remove a zip file from this directory, the corresponding entity (Component, Service Assembly or Shared Lib) will be automatically uninstalled or undeployed.

After installation/deployment, directory structure of installed element can be found in the `%PETALS_HOME%/repository` directory.

Finally, the `%PETALS_HOME%/work` directory contains the status of the auto installation/deployment process:

- ***`myEntity.succeed`***: the process succeed, the element is started,
- ***`myEntity.failed`***: the process failed, and this file contains the stack trace of the exception.

5 PETALS monitoring

5.1 Launch PETALS with monitoring

To activate the monitoring, launch PETALS with the

`%PETALS_HOME%/monitoring/startupMonitoring.[bat/sh]` script.

The monitoring tool reports information about a local PETALS server, or about the global PETALS servers' network.

If you want to monitor the complete PETALS servers' network, be sure that all PETALS servers are launched with the monitoring option.

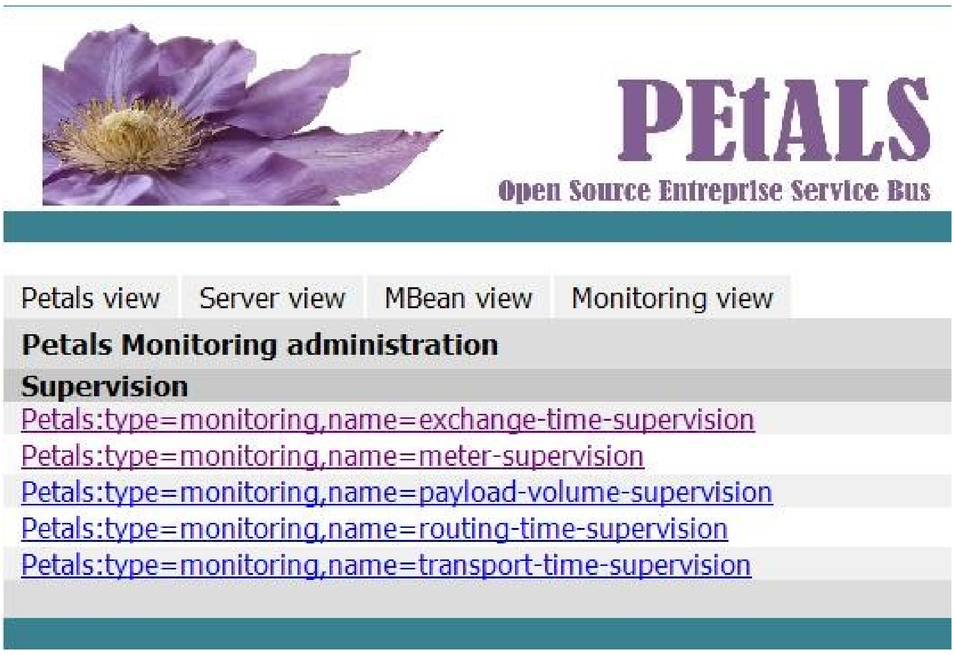
Don't forget that the activation of the monitoring might decrease PETALS performance.

5.2 Monitoring with JMX

PETALS monitoring tasks are accessible through JMX. Each monitoring function is represented by a Management Bean.

In the HTML console of PETALS, use the `Monitoring View` tab.

Monitoring tools are also accessible through the Java console. Use the Petals/monitoring/... monitoring-MBeans in the console to use them.



Petals Monitoring administration

Supervision

Petals:type=monitoring,name=exchange-time-supervision
Petals:type=monitoring,name=meter-supervision
Petals:type=monitoring,name=payload-volume-supervision
Petals:type=monitoring,name=routing-time-supervision
Petals:type=monitoring,name=transport-time-supervision

Figure 10 - JMX HTML Main Monitoring view

PETALS is provided with a set of basic monitoring functions. These monitoring functions reports local and distributed information:

- information about the server on which the user is connected through JMX,
- information about the global PETALS servers' network.

They have also a `clear()` method to clean information.

Please take care about the difference between what we call a `Message` (information sent from a JBI component to another one, with a `send()` method), and a global `MessageExchange` (a message exchange is a set of multiple messages sent between JBI components, for instance, request, response and acknowledgement).

- ***Payload volume monitoring***: return the size, in bytes, of each `MessageExchange` sent.

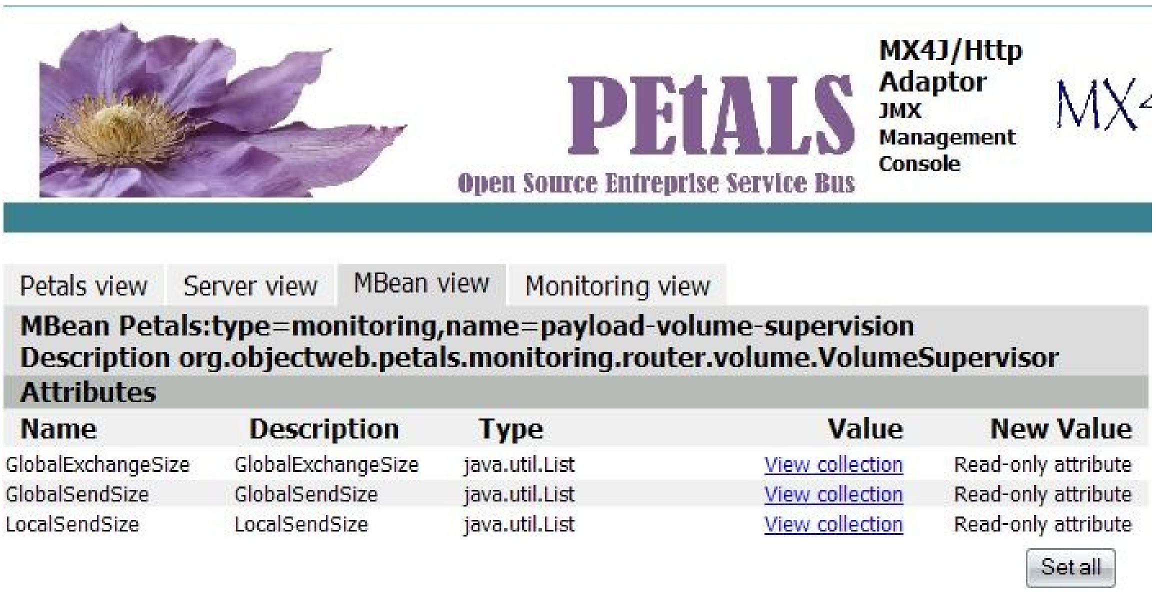
- ***Meter monitoring***: return the number of `Message` sent.

- ***MessageExchange timer monitoring***: return, for each global `MessageExchange`, the total time of its life (from the first send to the exchange to the last one).

- ***Message timer monitoring***: return, for each `Message`, the time elapsed between the JBI component's `send()` method and the delivery of this message. This time is called the `routing time`.

- ***Message transport timer monitoring***: return, for each `Message`, the time elapsed between the physical send and reception of this message. This time is called the `transport time`.

For instance, here is the Monitor dedicated to the payload volume of the `MessageExchanges` sent over PETALS:


MX4J/Http Adaptor JMX Management Console
PEtALS
Open Source Entreprise Service Bus
Petals view
Server view
MBean view
Monitoring view
Set all

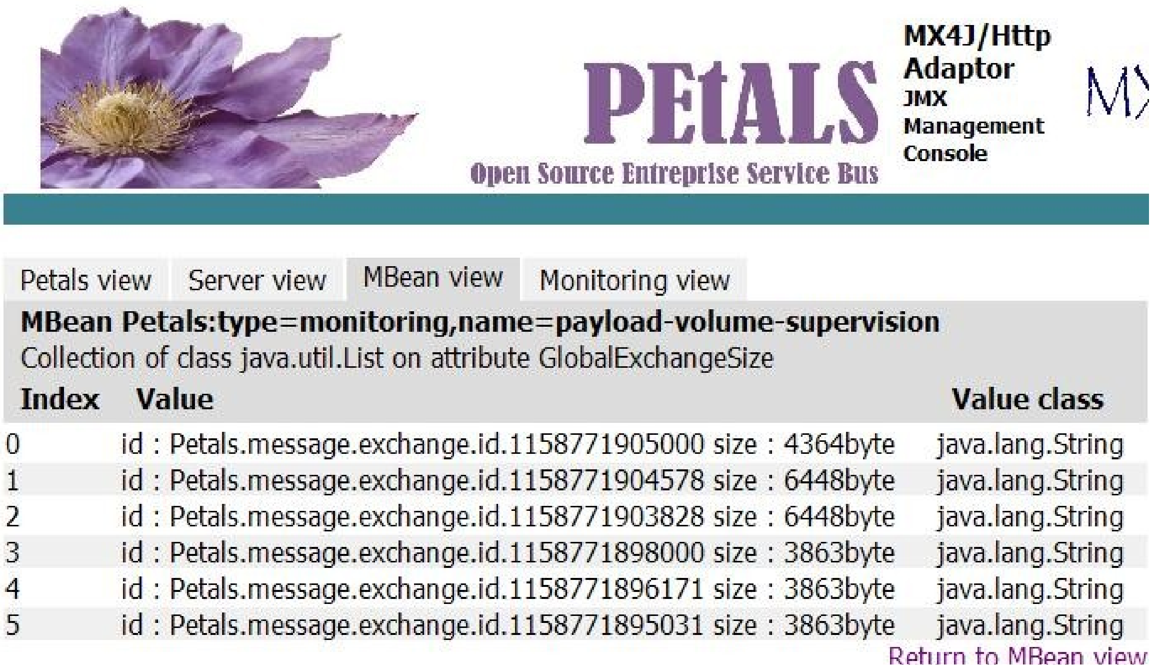
MBean Petals:type=monitoring,name=payload-volume-supervision
Description org.objectweb.petals.monitoring.router.volume.VolumeSupervisor

Attributes

Name	Description	Type	Value	New Value
GlobalExchangeSize	GlobalExchangeSize	java.util.List	View collection	Read-only attribute
GlobalSendSize	GlobalSendSize	java.util.List	View collection	Read-only attribute
LocalSendSize	LocalSendSize	java.util.List	View collection	Read-only attribute

Figure 11 - JMX HTML Payload Volume Monitor

The report for the global `MessageExchange` payload volume looks like this:


MX4J/Http Adaptor JMX Management Console
PEtALS
Open Source Entreprise Service Bus
Petals view
Server view
MBean view
Monitoring view
Return to MBean view

MBean Petals:type=monitoring,name=payload-volume-supervision
Collection of class java.util.List on attribute GlobalExchangeSize

Index	Value	Value class
0	id : Petals.message.exchange.id.1158771905000 size : 4364byte	java.lang.String
1	id : Petals.message.exchange.id.1158771904578 size : 6448byte	java.lang.String
2	id : Petals.message.exchange.id.1158771903828 size : 6448byte	java.lang.String
3	id : Petals.message.exchange.id.1158771898000 size : 3863byte	java.lang.String
4	id : Petals.message.exchange.id.1158771896171 size : 3863byte	java.lang.String
5	id : Petals.message.exchange.id.1158771895031 size : 3863byte	java.lang.String

Figure 12 - JMX HTML payload volume monitor's report

6 Distributed environment

PETALS runs natively in a distributed mode.

Communication between JBI components of all the servers is transparent for them. The JBI components and their `ServiceEndpoints` are seen as if they were hosted by a single JBI container.

Each time a PETALS server is started, it checks if others servers are running, and automatically joins their network.

Even if a server is stopped, it is always seen in the logical configuration of the network, and its `ServiceEndpoints` can always been requested.

The following figure shows a logical architecture versus a physical one :

- one logical PETALS container with 4 JBI components and their ServiceEndpoints, and the directory;
- 3 physical containers of which one is stopped, hosting each one (or two) a component(s) (and the corresponding ServiceEndpoints) and a directory.

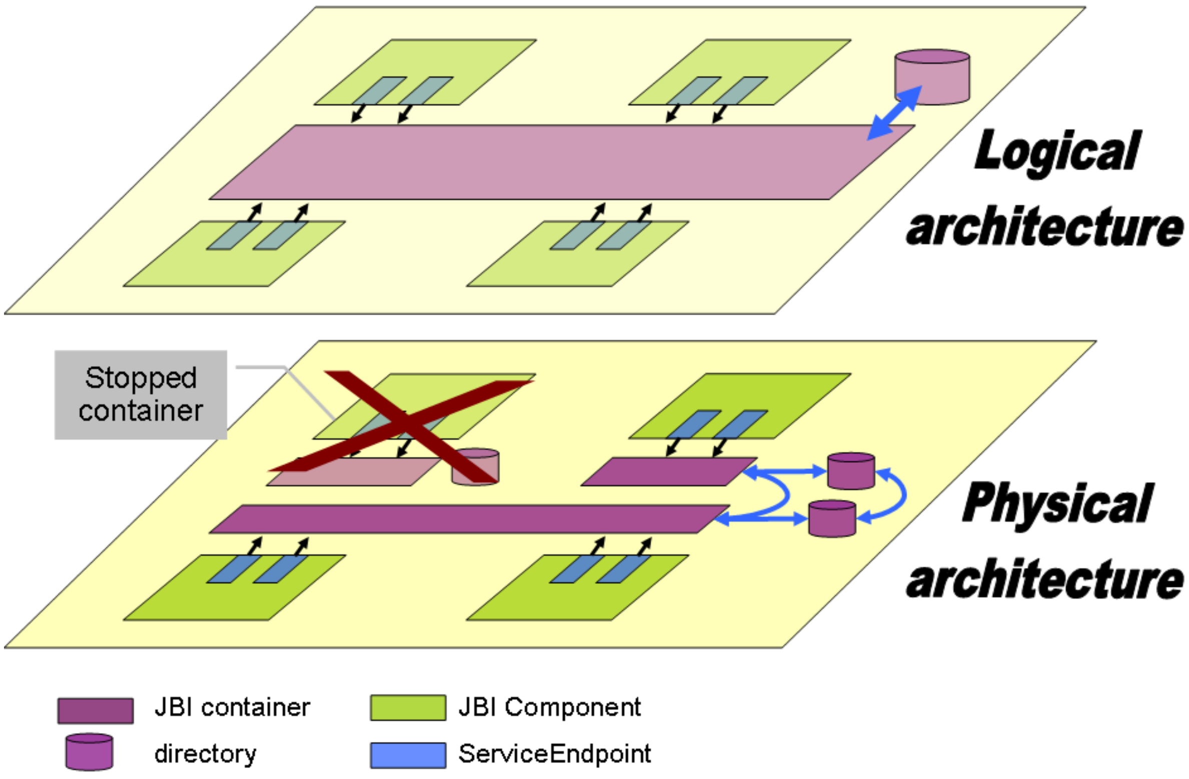


Figure 13: logical versus physical distributed architecture

6.1 *Network communication*

6.1.1 A distributed directory

PETALS uses a distributed JNDI server implemented with the *Tribe ObjectWeb* project.

Each PETALS server starts a local instance of a Tribe JNDI server. The Tribe JNDI servers use a multicast IP address to find other Tribe JNDI servers. All the Tribe JNDI servers work together and share the registered information.

That is to say, when an information is registered by a PETALS server in its JNDI server, all the other PETALS servers show this information.

When all the PETALS servers of a network are stopped, the related global JNDI directory "lives" until the last PETALS server is stopped.

6.1.2 JBI Service-Endpoints access

A `ServiceEndpoint` registered by a JBI component is stored in the global directory. Thus, all other JBI components (local to the server or hosted on another server) can see and use it.

The `ServiceEndpoint` lives in the global directory until it is unregistered by its JBI component owner. If the server is stopped but the global directory is still living (other PETALS servers are running), the `ServiceEndpoint` is always visible.

When the related PETALS server is started again, the JBI component providing this `ServiceEndpoint` is started again, and the link between them becomes active (the reference of this `ServiceEndpoint` is given to the component).

If the related PETALS server restarts but no other PETALS server lives (i.e. no global directory is already living), a new empty global directory is created, the component restarts and registers again its `ServiceEndpoint` in it.

6.1.3 JBI MessageExchange delivery

A JBI `MessageExchange` for a `ServiceEndpoint` is sent to the component that has registered this endpoint, as well as the component is collocated or not to the sender.

As a `ServiceEndpoint` is still active even if the server that hosts it is stopped, messages can be sent to a “pending” endpoint. These messages are pending in the JORAM global system, and will be delivered when the server that hosts this `ServiceEndpoint` starts again.

Even if all the PETALS servers are stopped, as the pending messages are physically stored on the server that hosts the sender, the messages will be delivered at start-up (PETALS uses the JORAM store and forward mechanism).

6.1.4 Stop, Restart and Shutdown a server

You "stop" a PETALS server when you terminate the JVM process (by "`Ctrl-C`" or "`kill`" command, or "`stop`" in the command-line mode).

You normally see the "`PETALS is stopped`" message.

Then, the server is stopped but is still referenced in the network configuration (i.e. the other living servers always know that this server exists, but that it is stopped).

When you start again this server, it retrieves its configuration and notifies the PETALS network that it is started again. It eventually sends or receives pending messages.

If you want to definitively remove this server from the PETALS network, you have to "shutdown" it.

You can shutdown the server by using the `PETALSAdmin` JMX service, or with the "`shutdown`" command in command-line mode.

6.2 Network configuration

6.2.1 Configure a network

Two parameters allow the communication between each PETALS server.

These parameters are located in the `%PETALS_HOME%/conf/server.properties` file. The default values are:

- ***`tribe.multicast`***=`224.7.65.69` (distributed JNDI server)
- ***`JORAM.domain`***=`PETALS` (JORAM domain)

At the startup, a server checks if there are any other JNDI server that are using the same multicast address and domain name.

If the starting server finds others servers, it checks if its own properties (in the `server.properties` file) are not conflicting with the other servers properties. If there is no conflict, the server starts without changing its configuration. If there are some conflicts, it updates its `JORAM ID` and its ports.

When connecting to an existing distributed configuration, a new `JORAM ID` is assigned to the starting server. If you want to define a specific `JORAM ID` for a server, set in its `%PETALS_HOME%/conf/server.properties` the `JORAM.id=XX` entry.

6.2.2 Multiple PETALS networks

If you want to use different PETALS networks, specifically set the following entries for each server:

- ***`tribe.multicast`***= [`ip address`],
- ***`JORAM.domain`***= [`domain name`]

For instance, if you have two servers working together, and three others that are working together, set in the `server.properties` of the two first servers:

- ***`tribe.multicast`***= `IP1`
- ***`JORAM.domain`***= `DOMAIN1`

And in the `server.properties` of the three second servers:

- ***`tribe.multicast`***= `IP2`
- ***`JORAM.domain`***= ***`DOMAIN2`***

When setting multiple networks, you have to change **both** `tribe.multicast` and `JORAM.domain` properties.

6.2.3 Work in a standalone way

If you want to start a PETALS server that will not participate with other servers, it must be the only one to use its multicast IP address.

6.2.4 Clean a server configuration after a blocking crash

In a distributed PETALS environment, a server may not restart correctly if it has previously crashed.

Edit the `%PETALS_HOME%/conf/server.properties`, and remove the entire following entry:

- ***`container.uid`***=`XXXXXXXX`

A new configuration is set for this container.

7 References

PETALS web site

http://petals.objectweb.org

JBI Components mechanism

http://www.javaworld.com/javaworld/jw-07-2006/jw-0717-jbi.html

Java Business Integration

http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html

JORAM web site

http://joram.objectweb.org