PEtALS Standalone Distribution

This document is the user guide of the PEtALS standalone distribution. This release targets PEtALS users who want to build their services architecture on a centralized infrastructure

PEtALS Team
Roland NAUDIN <roland.naudin@ebmwebsourcing.com>
Christophe HAMERLING <christophe.hamerling@ebmwebsourcing.com>

- March 2009 -

Table of Contents

List of Figures

PEtALS

PEtALS is the OW2 Java Business Integration (JBI) bus (See JSR-208 specifications for further details on JBI).

Along the time, PEtALS has increased its coverage amongst high-value domains like clustering, robustness, availibility, performance... Moreover, PEtALS relies entirely on its OW2 partner technology, the Fractal component model, which brings to its architecture a strong modularity. Please visit the Fractal web site for further details at http://fractal.objectweb.org.

Since the version 2.1, the PEtALS team have decided to exploit Fractal leverage by delivering various PEtALS distributions. Each distribution is packaged and customized to be addressed to a specific audience.

This distribution targets the usage of PEtALS in a centralized infrastructure. It provides security and transaction support. It permits various optimizations on data transfer and tuning on the administration and exploitation of your JBI environment.

Chapter 1. Presentation

1.1. What is PEtALS?

PEtALS helps you to integrate your Enterprise Business Units in order to provide a coherent and rational global solution. With PEtALS, you can design your new applications by building a composition of existing ones and new ones.

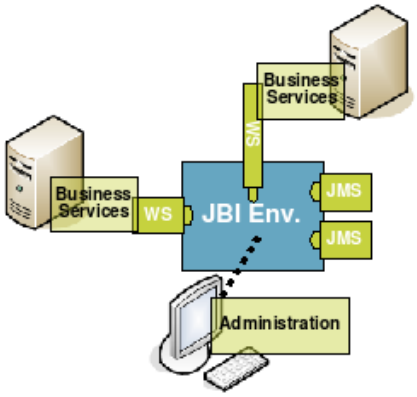
All your applications expose their business logic by exposing services. Thus, each new composed application becomes an add-on to the pool of reusable applications.

This concept is well known as **Service Oriented Architecture - SOA**.

PEtALS offers a solid backbone for your enterprise Information System and acts as a Bus, a place where all your data are exchanged. PEtALS connects services to each others. PEtALS is an Enteprise Service Bus (ESB).

Please visit our Web site for further details at http://petals.ow2.org.

Figure 1.1. PEtALS is an ESB fully JBI compliant



1.2. Acronyms

Along the document, we are using common JBI acronyms :

- **SL** : a Shared Libraries artifact.
- **BC** : a Binding Component artifact.
- **SE** : a Service Engine component artifact.
- **SA** : a Service Assembly artifact.
- **SU** : a Service Unit artifact.
- **MEP** : Message Exchange Pattern, JBI defines 4 possible MEP to invoke a service.

1.3. JBI Components

In order to build your business solutions, PEtALS proposes a large bunch of components. Each component is downloadable independently on the PEtALS website.

1.3.1. Binding Components

The BCs are bridges to interact with external services, commonly via a communication protocol.

They provide transfers of incoming requests from external service consumers to the JBI environment, as well as the transfer of outgoing requests to external service providers.

The current proposed BCs are :

- **Filetransfer** : transfers files from/to local directories.
- **FTP** : transfers files from/to remote directories.
- **TCP/IP** : transfers files from/to TCP/IP servers.
- **HTTP** : transfers files from/to HTTP servers.
- **Mail** : interacts with e-mail servers to receive/send emails.
- **XMPP** : transfers files from/to XMPP servers.
- **JMS** : interacts as JMS client with any JMS provider to receive/send JMS messages.
- **Soap** : invokes external Web Services and expose JBI services as Web Services.
- **XQuare** : interacts with databases to store and retrieve data.
- **Ejb** : exposes remote EJBs as JBI service.

1.3.2. Service Engines

The SEs are components providing services to the JBI environment. A large domain of services can be designed: routing, orchestration, transformation... You can provide your own business services too.

The current proposed SEs are :

- **EIP** : provides routing and chaining of services based on Enterprise Integration Patterns.
- **Orchestra** : provides orchestration of services based on BPEL language.
- **Drools** : process rules on different language to provide routing services.
- **Bonita** : provides services to interacts with the Bonita Workflows.
- **XSLT** : transforms XML documents to XML or another forms, based on XSL sheets.
- **Validation** : validates XML documents against XML schemas.
- **CSV** : transforms CSV documents to XML documents.
- **Transcoder** : transcodes XML fields against database fields.
- **Quartz** : provides scheduling services.
- **Script** : supports scripting to provide your own business logic.
- **Pojo** : exposes Java classes as services to provide your own business logic.
- **JSR-181** : exposes Java classes as services to provide your own business logic.
- **RMI** : brings the access of the JBI component context to a remote RMI client, in order to provide your remote business logic.

1.3.3. Samples

Several sample components are delivered with PEtALS to ease the tests when building your solutions:

- **SampleClient** : a simple graphical client to invoke the services exposed in the bus.
- **HelloWorld** : the famous HelloWold paradigm exposed as a service...
- **Clock** : provides a clock service.
- **PerfConsumer/PerfProvider** : two components running together to test container performances.

Chapter 2. PEtALS container

2.1. Pre-Requisites

- **Java** : to run PEtALS, you need at least a Java JRE 1.5 distribution.

  The Sun JVM can be downloaded at : http://java.sun.com/javase/downloads/index_jdk5.jsp.

- **Ant** : to run the provided use cases, you need to install an Ant distribution.

  Apache Ant can be downloaded at : http://ant.apache.org/bindownload.cgi.

2.2. Installation

PEtALS does not require any installer to be installed. Just unzip the provided archive where you want to install it.

Generally, PEtALS find by itself its directory location. On some systems, the installation path can not be automatically found, in such case you must set the *PETALS_HOME* environment variable :

- On Unix/Linux systems :

```
# export PETALS_HOME=your_installation_path
```

- On Windows system :

```
# set PETALS_HOME=your_installation_path
```

2.3. Directory structure

- ant : includes a sample file containing exemples to illustrate the usage of JBI specialized Ant tasks.
- `bin` : includes scripts to launch PEtALS on Unix (*.sh) or Windows (*.bat) systems.
- `conf` : includes PEtALS configuration files.
- `install` : auto-loader directory; put a JBI component (SE or BC) or a SA to have them automaticaly installed, deployed and started in the PEtALS container.
- `installed` : components and services assemblies are automaticaly copied into this directory after a succesful installation/deployment.
- `lib` : includes libraries required by PEtALS system.
- `licenses` : includes the licenses of all the libraries used in PEtALS container
- `logs` : includes logs generated during PEtALS execution.
- `lost+found` : contains lost JBI elements (components or SAs no more referred in the PEtALS repository).
- `repository` : includes the resources;libraries, config files... of the installed/deployed components, SLs and SAs.
- `schema` : includes useful XML schemas.
- `work` : used by PEtALS engine to store internal resources during runtime.

2.4. Starting PEtALS

1. Go to `$PETALS_HOME/bin`

2. Launch `startup.sh` or `startup.bat` :

```
# ./startup.sh
```

You can launch PEtALS in console mode, to be able to interact with it via the terminal. In this case, use the option -C in the terminal :

```
# ./startup.sh -C
```

2.5. Stopping PEtALS

There is several ways to stop PEtALS :

1. If PEtALS is launched in a simple terminal, type **<ctrl>**+ **c** to stop PEtALS.
2. If PEtALS is launched in console mode, just type **q** or **x** and **<enter>** in the terminal :

```
# q
```

By typing **q**, PEtALS is stopped; components, SAs ant SLs life-cycle states are hold.

By typing **x**, PEtALS is shut down; components, SAs and SLs are uninstalled/undeployed.

3. If PEtALS is launched in background, launch the `stop.sh` script :

```
# ./stop.sh
```

4. If PEtALS is launched in background mode, launch the `shutdown.sh` script :

```
# ./shutdown.sh
```


Note

Stopping PEtALS means to stop the running components and SAs without changing their life-cycle state. At your next start, the components and SAs would be recovered to their previous state. Shutting down PEtALS means to undeploy and uninstall all the components and SAs loaded in the container. It is useful to clean up your container.

Chapter 3. Configuration

3.1. Introduction

All the PEtALs configuration resources are located in the `conf` directory.

- `topology.xml` : defines the configuration and resources involved in the PEtALS topology.
- `server.properties` : defines the local configuration, customization and resources of your JBI container.
- `logger.properties` : defines the logging configuration of your container.
- `log4j.properties` : defines the logging configuration of third part libraries using log4j logger.
- router.cfg : configures the modules used by the PEtALS container router.
- `julia.cfg` : contains the FRACTAL configuration, reserved to PEtALS specialists.
- `login.properties` : security file. See Security section for further details.
- `user-passwords.properties` : security file. See Security section for further details.
- `groups.properties` : security file. See Security section for further details.
- `authorization.cfg` : security file. See Security section for further details.

3.2. PEtALS topology - topology.xml

In this distribution, you can define only one container. Here is the default configuration:

```
<tns:topology xmlns:tns="http://petals.ow2.org/topology"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://petals.ow2.org/topology petalsTopology.xsd">
  <tns:domain mode="standalone" name="PEtALS">
    <tns:description>standalone topology</tns:description>

    <!-- the internal jndi server is used by default
      <tns:jndi>

 <tns:factory>org.ow2.petals.communication.jndi.client.naming.NamingContextFactory</tns:factory>
        <tns:host>host</tns:host>
        <tns:port>0</tns:port>
      </tns:jndi>
    -->

    <tns:sub-domain name="subdomain1">
      <tns:container name="0">
        <tns:host>localhost</tns:host>
        <tns:description>description of the features of the Standalone container</tns:description>
        <tns:user>petals</tns:user>
        <tns:password>petals</tns:password>
        <tns:jmx-service>
          <tns:rmi-port>7700</tns:rmi-port>
        </tns:jmx-service>
      </tns:container>
    </tns:sub-domain>
  </tns:domain>
</tns:topology>
```

The meaning of the domain and subdomain is not used in this distribution but only in platform distribution.

PEtALS uses an internal JNDI directory. You can however configure PEtALS to connect to an external JNDI directory, to integrate it in your enterprise Information System. To do that, just uncomment the JNDI part and fill the parameters.

`jndi` Optional.

- `factory` The JNDI factory class name.
- `host` The host name or IP of the JNDI server.
- `port` The port to connect to the JNDI server.

In the container section, the standalone needs only resources for its JMXService.

`container`

- `host` The host name or ip address of the container.
- `description` Optional. The description of the container.
- `user` The login name, used for JMX authentication.
- `password` The password, used for JMX authentication.
- `jmx-service` The JMX service, used to manage MBean objects.
  - `rmi-port` The port used to by the JMX RMI connector. The JMX Server would be reachable on this port at the URI `service:jmx:rmi:///jndi/rmi://host:rmiport/jmxRmiConnector`.

3.3. Server properties - server.properties

`petals.repository.path` Optional. The URL path to the PEtALS repository. PEtALS holds its JBI configuration in this repository and can recover its configuration from it.

`petals.exchange.validation` Optional. This property is used to activate the control of exchange acceptance by target component when the NMR routes messages (see `isExchangeWithConsumerOkay()` and `isExchangeWithProviderOkay()` methods in JBI Component interface for further details).

`petals.task.timeout` Optional. Maximum duration of the processing of a life-cycle operation on a JBI components and SAs (start, stop, ...). After this duration, if the processing of the operation is not finished, it is interrupted. It prevents from hanging threads.

`petals.classloaders.isolated` Optional. Activate/Unactivate the isolation of the ClassLoaders created for Shared Libraries and Components from the PEtALS container one. It can be useful to avoid concurrent libraries loading issues.

`petals.autoloader` Optional. Activate/Unactivate the autoloader service.It can be useful in production environment to unactivate it.

3.4. Loggers configuration - loggers.properties

PEtALS is relying on FRACTAL component framework which uses Monolog.

In this file, we configure monolog to wrap the JavaLog loggers.

For each PEtALS Fractal component loggers, 2 handlers are used:

- a dedicated console handler to provide logs in the standard output of your console.
- a shared file handler to provide logs in a common file stored in the `logs` directory.

The global default log level is set to `INFO`.

The shared file handler is filtering `DEBUG` logs, whereas the console handlers are not.

If you want to see the DEBUG logs on your console:

- you can activate the DEBUG level for PEtALS by changing the PEtALS global log level

```
logger.Petals.level DEBUG
```

- you can activate the DEBUG level for some PEtALS components by changing their log level (uncomment pertinent lines in the file)

```
#logger.Petals.JBI-Management.SystemRecoveryServiceImpl.level DEBUG
logger.Petals.JBI-Management.InstallationServiceImpl.level DEBUG
logger.Petals.JBI-Management.DeploymentServiceImpl.level DEBUG
#logger.Petals.JBI-Management.AutoLoaderServiceImpl.level DEBUG
```

At the end of the file, the configuration for the DREAM and JORAM transporters is defined.

As for PEtALS Fractal component, 2 handlers are defines for both DREAM and JORAM.

For these loggers, the log level is set to `ERROR`.

You can activate the `DEBUG` level alike the PEtALS loggers.

3.5. Router configuration - router.cfg

The PEtALS router is used to elect potential endpoints matching a service invocation.

The router architecture has been structured in modules. Modules to used are defined in the `router.cfg` file like this :

```
# Define the modules to be loaded in the router. They are processed in the following defined order
# Lines starting with '#' characters are comments ones
#org.ow2.petals.jbi.messaging.routing.module.AuthorizationModule
org.ow2.petals.jbi.messaging.routing.module.AddressResolverModule
#org.ow2.petals.jbi.messaging.routing.module.TraceRouterModule
org.ow2.petals.jbi.messaging.routing.module.RouterSenderModule
```

Each line is a module which will be loaded at router initialization. When a message need to be routed, it goes through all the router modules in the order they are defined in the configuration file.

Chapter 4. Management

4.1. Install/Uninstall components

The installation and uninstallation of components is fully described in the JBI specification at the sections 6.4 and 6.5.

4.1.1. Install/Uninstall with repertories

PEtALS embeds an autoloader service which takes care of all the default installation steps until the start of the component.

Components can be installed directly by copying the zip archive in the `install` directory of PEtALS. PEtALS will detect the new file and proceed to the installation and the start of the component.

To uninstall a component, remove the zip archive of the `installed` directory of PEtALS. PEtALS will detect the removal and proceed to the stop/shutdown/uninstallation of the component.

4.1.2. Install/Uninstall with the web administration console

The PEtALS web application proposes an administration console to install/uninstall your components. To use it, you need a servlet container like Tomcat.

Check the web console documentation for further details.

4.2. Install/Uninstall SLs

The installation and uninstallation of SLs is fully described in the JBI specification at the sections 6.4.

in the same manner of for the component, you can use repertories or the web console to do it.

4.3. Deploy/Undeploy SAs

The deployment and undeployment of SAs is fully described in the JBI specification at the sections 6.6.

in the same manner of for the component, you can use repertories or the web console to do it.

4.4. Monitoring PEtALS

The PEtALS web application proposes a monitoring console. To use it, you need a servlet container like Tomcat.

With the monitoring console, you can :

- visualize PEtALS nodes information of the domain
- control and set monitoring parameters for each PEtALS node
- visualize informations and statistics about the exchanges in the domain
- visualize your PEtALS container information
- control and set the monitoring parameters
- visualize informations and statistics about the exchanges

check the web console documentation for more information.

4.5. Advanced Management

JBI has designed the management if JBI engine on the JMX technology. Fur further details on JMX, please refers to official documentation.

PEtALS provides its management MBeans under the `PETALS` JMX domain.

There is 6 'top level' JBI management services :

1. **Administration Service**

   This service allows to retrieve information about JMX ObjectName, life-cycle state... on loaded JBI artifacts. For further information, please check the JavaDoc of the `AdminServiceMBean` interface.

2. **Deployment Service**

   This service allows to deploy/undeploy SAs and to manage their life-cycle (start, stop, shutdown). It allows to list SUs hold by a SA or SA deployed in a given component. For further information, please check the JavaDoc of the `DeploymentServiceMBean` interface. .

3. **Installation Service**

   This service allows to install/uninstall SLs and components and to manage (create, register, retrieve, destroy) `InstallerMBean` instances. Installers are used to perform the installation/uninstallation tasks on components. For more information check the javaDoc of the `InstallationServiceMBean` interface.

4. **Logger Service**

   This service allows to manage the loggers and their log handlers.

5. **Monitoring Service**

   This service allows to activate or deactivate the monitoring mode in PEtALS.

6. **PEtALS Administration Service**

   This service allows to execute Petals specific administration tasks like shutting down the PEtALS container or retrieve the domain topology.

4.5.1. JMX Management

An RMI JMX connector is launched within PEtALS container. The MBean methods can be accessed by RMI client, such as the JDK JConsole tool. The JConsole is commonly located in the `$JAVA_HOME/bin` directory of your environment.

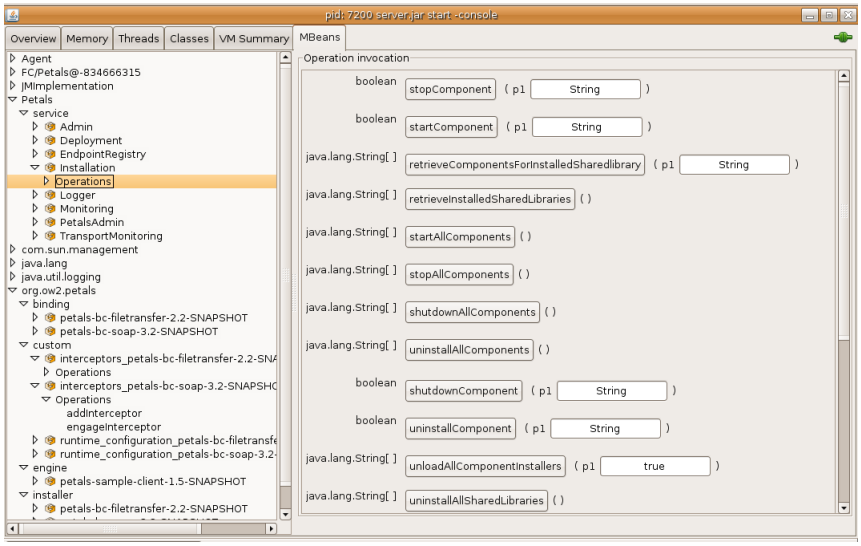
At startup, JConsole prompts for a server to connect to. Go to the `Advanced` tab and connect to the URI `service:jmx:rmi://jndi/rmi://myhost:myport/jmxRmiConnector`.

Note

The URI pattern is always the same, but `myhost` and `myhost` must be the ones defined in your topology file.

Once connected, you can find the MBeans exposed on the JMX server, including the PEtALS management services under `Petals` domain :

Figure 4.1. The JConsole management view



4.5.2. Ant Management

JBI specifies a set of dedicated ant task to be able to manage JBI container with batch/script programs. The full list of Ant tasks and their relatives parameters is described at section 6.10 of the JBI specification.

Moreover you can find a sample of an Ant build file in `$PETALS_HOME/ant` that illustrate the possibilities of management.

4.5.2.1. Install a component

To install a component just run this ant task :

```
<jbi-install-component file="file:///mydirectory/myfile.zip" port="myport" host="myhost"
 user="myuser" password="mypassworld" />
```

This will load a new `InstallerMBean` and execute the `install` method on this one. Then the component is completly installed.

4.5.2.2. Start a component

To start a component just run this ant task :

```
<jbi-start-component name="mycomponentname" port="myport" host="myhost" user="myuser"
 password="mypassworld" />
```

4.5.3. Console mode Management

The user can interact with PEtALS on a command-line console by starting PEtALS with the argument `-console` (`-c`).

Once PETALS is started, commands can be wrote to the console:

- hotdeploy -ZIP filePath- (or hd): install and start a component, a service assembly or a shared library
- hotundeploy -ZIP fileName- (or hu): shutdown and uninstall a component, a service assembly or a shared library
- path (or p) : display current file system path
- components (or c): display the installed components
- info (or i): display the container version

- setpath -Path- (or sp): change current file system path
- jndi (or d) : display the JNDI directory
- help (or h) : display this help
- stop (or q) : stop Petals
- shutdown (or x) : shut down Petals

Chapter 5. Optimizations

5.1. JBI Exchange Optimizations

5.1.1. Optimization setup

The following optimization can be exploited by JBI components by setting the related MessageExchange property to the MessageExchange transfered.

5.1.2. Acknowledgement bypassing

Each JBI MessageExchange pattern ends with a "DONE" or "ERROR" message. The sender has to accept such messages, otherwise they are accumulated in the NMR (**N**ormalized **M**essage **R**outer). This generate extra trafic, too, even in an In-Only exchange.

Setting the `org.ow2.petals.messaging.noack` property to `true` on the MessageExchange indicates that the `DONE` or `ERROR` message will not be sent through the NMR.

This feature is unactivated when using synchronous sends.

Chapter 6. Security

Since the version 2.2, PEtALS is providing security features, based on JAAS. Check http://java.sun.com/javase/technologies/security for further details on JAAS.

6.1. Configuration

6.1.1. Login Module

PEtALS uses the Login Module mechanism to handle authentification. The default login modules are provided by the petals-jaas library. The login module can be loaded by specifiing a configuration file to be used at the container startup. The `login.properties` file define the options for the provided PropertiesLoginModule :

```
petals-domain {
    org.ow2.petals.jaas.PropertiesLoginModule
        sufficient
        org.ow2.petals.security.properties.user="users-passwords.properties"
        org.ow2.petals.security.properties.group="groups.properties";
};
```

- `org.ow2.petals.jaas.PropertiesLoginModule` is the class of the JAAS module to be used. This class is provided by the petals-jaas library and needs additional parameters for its configuration which are explained in the next lines.
- `org.ow2.petals.security.properties.user` value is the file where the users and passwords are defined. In the previous example, the file is `user-passwords.properties`. This file must be located at the same level than the `login.properties` one in the conf folder of your PEtALS distribution. Its content is defined like :

  ```
  petals=petals
  chamerling=password
  ```

  Left argument is the user login, right one is the password of the user.
- `org.ow2.petals.security.properties.group` value is the file where the groups are defined. In the previous example, the file is `groups.properties`. This file must be located at the same level than the `login.properties` one in the `conf` folder of your PEtALS distribution. Its content is defined like :

  ```
  #role=users as CSV
  admin=petals
  users=petals,chamerling
  ```

  Left argument is the group/role, right one are the users which are in the group as CSV.

6.1.2. Define Service - Endpoint authorizations

Once the login module has been configured, you can define the rules to access the services and endpoints in the `authorization.cfg` file under the `conf` folder of your PEtALS distribution.

```
# Authorization configuration, lines starting with '#' are ignored
#service={http://petals.ow2.org/
helloworld}HelloworldService;endpoint=HelloworldEndpoint;operation=sayHello;roles=*
service={http://petals.ow2.org/
helloworld}HelloworldService;endpoint=HelloworldEndpoint;operation=*;roles=users
```

An entry (line) is defined like this :

- service : The service qualified name
- endpoint : The endpoint name
- operation : The operation name

- roles : The roles which can access to the previous defined values

The operation and roles entries can have wildcard values which means 'all'.

With the previous sample, it is defined that users from the role 'users' can access to any operation of HelloworldService service / HelloworldEndpoint endpoint.

This authorizations are handled at the routing level. In order to be used, you must configure the router and add the AuthorizationModule router module to the router configuration file. More details are available on the router configuration section.

When the authorization router module process the JBI message, it will check if the message security subject is conform to the rules defined in the authorization configuration file. If not, an error will be returned to the service consumer.

6.2. Startup

You must start PEtALS with the `java.security.auth.login.config` system parameter to define the root path of the security module configuration files.

```
./startup.sh -Djava.security.auth.login.config=<PETALS_HOME>/conf/login.properties
```

Chapter 7. Links

- The PEtALS Website : http://petals.ow2.org
- The JBI Specification : http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html