

# JAFS integration

by SuperBonBon

## 1. JBI support

A JBI oriented distribution is available since release 0.9.10, the release has been tested with the following JBI containers :

- [ServiceMix](#)
- [Open ESB](#)

The release provides a JAFS binding component. The component allows to start the server within the JBI container and sends [events](#) messages via the JBI Normalized Message Routing (NMR) bus to target components deployed in the JBI container.

The release contains three JBI jar files :

- `sbbi-jafs-$version-installer.jar` : The JAFS JBI binding component libs.
- `sbbi-jafs-$version-assembly.jar` : The service assembly (config files). This assembly is used to deploy a JAFS service instance.
- `sbbi-jafs-$version-client.jar` : The JAFS JBI client service engine.

The binding component provides a standard JBI WSDL 2 interface for service engines in the container to register against a specified set of events. The interface grammar documentation can be found [here](#). The provided client component can be used as a WSDL 2 messaging abstraction layer to simplify events registration and reception.

An admin tool is provided to deploy JAFS within the JBI container. This tool is located at : `bin/admin.bat/sh`, you will also have to modify the `bin/conf/jbi_admin.properties` file to edit your JBI container JMX connection settings.

To deploy JAFS within your JBI container, edit the `bin/conf/jbi_admin.properties` to match your JBI container settings, start your JBI container and launch the following command : `admin.sh deploy-jafs`.

The client component can be deployed using the `admin.sh deploy-client` command, you can also launch the `admin.sh` command without args to see all the available admin commands.

## 2. JBI JAFS client component

The client service engine is used to receive events from deployed JAFS JBI binding components. Once deployed the client looks in the `jar:file:/sbbi-jafs-$version-client.jar!/META-INF/jbi.xml` file for its configuration :

```
<clients-configs xmlns="http://www.sbbi.net/jafs/1.0/jbi-client" auto-discovery
  <client-config client-id="mainClient" endpoint-name="JAFS-SU_endpoint">
    <!-- class-name must implement the
         net.sbbi.jafs.services.jbi.client.EventsReceiver interf
    <events-receiver id="systemout" class-name="net.sbbi.jafs.servi
      <setting name="foo">bar</setting>
    </events-receiver>
    <target-events events-receiver-id="systemout">
      <server-event name="*" />
      <services-event name="*" />
      <session-event name="onConnect" />
      <session-event name="onDisconnect" />
      <filter name="targetFtpServersCode">sample_server</filt
    </target-events>
    <!--target-events events-receiver-id="systemout">
      <session-event name="onConnect" />
    </target-events-->
  </client-config>
</clients-configs>
```

This configuration sample contains a *client-config* entry, the *endpoint-name* attribute defines the deployed JAFS JBI component to be used by the client for events notifications, multiple *client-config* entries can be defined.

The *endpoint-name* value matches the value defined at :

```
jar:file:/sbbi-jafs-$version-assembly.jar!/META-INF/jbi.xml/service-assembly/service-unit/identification/name
+ "_endpoint"
```

The *events-receiver* setting defines an events receiver, this class is the WSDL messaging abstraction layer and must implement the *net.sbbi.jafs.services.jbi.client.EventsReceiver* interface (look at javadocs for more info about this interface). You can define custom settings for your class via the *setting* tag.

Once the events receiver are set, you can define via the *target-events* tag what events must be received and where they have to be dispatched using the *events-receiver-id* attribute.

The provided default configuration will output to the system console all the received events and is provided for demo purpose. You will need to call the admin command *admin.sh unpack* and edit the *sbbi-jafs-\$version-client/META-INF/jbi.xml* file to modify the config and add your own *net.sbbi.jafs.services.jbi.client.EventsReceiver* class.

Once your code is packed into a jar file (I.E myreceiver.jar) you'll have to add it within the

## JAFS integration

sbbi-jafs-\$version-client.jar. To do it, edit the jbi.xml file and add your jar to the component classpath :

```
...
<component-class-path>
  <path-element>libs/sbbi-jafs-$version.jar</path-element>
  <path-element>libs/castor-0.9.9-xml.jar</path-element>
  <path-element>libs/xercesImpl.jar</path-element>
  <path-element>libs/myreceiver.jar</path-element>
</component-class-path>
...
```

Finally copy myreceiver.jar into the *sbbi-jafs-\$version-client/libs* dir and call the *admin.sh pack* script to rebuild the *sbbi-jafs-\$version-client.jar* file, you're now ready to call the *admin.sh deploy-client* command and see the magic or the stacktrace ;o)

### 3. JBI known issues

#### 3.1. ServiceMix (2.0.2)

The admin tool does not work since servicemix ant tasks are currently [bugged](#) and [not compliant](#) with the JBI ant tasks specs.

Secured FTP with SSL or TLS does not work for an unknown reason.

#### 3.2. Open ESB (1.0.3)

You'll have to modify the application server security policy file in `$$SUNAS_ROOT/domains/domain1/config/server.policy` to make JAFS work and add :

```
grant codeBase "file:$$SUNJBIRoot/jbi/lib/-" {
    permission java.security.AllPermission;
};
grant codeBase "file:$$SUNASROOT{/}domains$${/}domain1${/}jbi${/}-" {
    permission java.security.AllPermission;
};
grant {
    permission java.io.FilePermission "$$SUNASROOT{/}domains$${/}domain1${/}j
```

### 4. J2EE Application servers integration

JAFS can be run as child service of the following application servers :

- Geronimo (net.sbbi.jafs.services.as.GeronimoFtpServerServiceGBean)
- JBoss (net.sbbi.jafs.services.as.JbossFtpServerService)
- Sun AS / Glassfish (net.sbbi.jafs.services.as.SunFtpServerService)

Sample configuration files for the Application servers can be found in the *bin/conf* distribution directory. The following doc will use the \$JAFS\_ROOT var to define the JAFS installation dir.

## 5. Geronimo integration

- Launch the \$JAFS\_ROOT/bin/SDK\_unpack.bat|sh script.
- Create the following directory : \$GERONIMO\_ROOT/repository/jafs/jars and copy all the jars located in the \$JAFS\_ROOT/libs directory ( except the \$JAFS\_ROOT/libs/opt directory ).
- Copy the \$JAFS\_ROOT/config directory where you want and modify all the paths within the config files to match the new directory path, you can also create a logs and datastore directory ( depending the modifications done to *config/service\_config.xml* and *config/logs\_config.xml* file ).
- Copy the \$JAFS\_ROOT/bin/conf/geronimo-plan.xml file in \$GERONIMO\_ROOT and edit this file to modify the *log4JConfigFile* and *serviceConfigFile* settings with the path where you have copied the config files.
- Go in the \$GERONIMO\_ROOT directory and call the following command to deploy jafs :  

```
java -jar bin/deployer.jar --user system --password manager deploy geronimo-plan.xml
```
- Go to the Geronimo admin console and manually call the Jafs service startup.
- If you want to undeploy jafs call this command :  

```
java -jar bin/deployer.jar --user system --password manager undeploy net/sbbi/jafs/JafsGBean
```

## 6. Jboss integration

- Launch the \$JAFS\_ROOT/bin/SDK\_unpack.bat|sh script.
- Create the \$JBOSS\_ROOT/server/default/deploy/jafs.sar directory.
- Copy the \$JAFS\_ROOT logs, datastore, ftp and config directories within \$JBOSS\_ROOT/server/default/deploy/jafs.sar directory.
- Copy the \$JAFS\_ROOT/libs jar files (except bsh-1.3.0.jar and the opt dir) within \$JBOSS\_ROOT/server/default/deploy/jafs.sar.
- Edit the config files located in the \$JBOSS\_ROOT/server/default/deploy/jafs.sar/config directory and correct the paths in the files with the new directories path.
- Copy the bin/conf/jboss-service.xml file into the \$JBOSS\_ROOT/server/default/deploy/jafs.sar/META-INF dir.
- You're now ready to start JBoss, The JAFS MBeans should appear in the JBoss

## JAFS integration

management console if no problems occurs.

### 7. Sun AS / Glassfish integration

- Launch the \$JAFS\_ROOT/bin/SDK\_unpack.bat|sh script.
- Copy the \$JAFS\_ROOT/config directory where you want and modify all the paths within the config files to match the new directory path, you can also create a logs and datastore directory ( depending the modifications done to *config/service\_config.xml* and *config/logs\_config.xml* file ).
- Start the server and create a new JAFS LifeCycle Module (in Applications->Lifecycle modules).
- Use class *net.sbbi.jafs.services.as.SunFtpServerService* as module class name and add 2 properties :  
serviceConfigFile : the path where you have copied the config/service\_config.xml file.  
log4JConfigFile : the path where you have copied the config/logs\_config.xml file.
- Add all the libs in the \$JAFS\_ROOT/libs dir to the life cycle classpath entry ( except the jars in the \$JAFS\_ROOT/libs/opt directory ).
- Edit the \$SUNAS\_ROOT/domains/domain1/config/server.policy file and add :

```
grant codeBase "file:$JAFS_ROOT/libs/-" {  
    permission java.security.AllPermission;  
};
```

- Restart the server and you should be done.

### 8. Known limitations

The Geronimo, SUN AS / Glassfish integration do not allow to expose the JAFS JMX interface within the AS MBeans Server. A JAFS dedicated JAFS MBeans server will then be used.