

# JAFS events and EAI

by SuperBonBon

## 1. Introduction

Definitions :

- EIS : Enterprise Information System
- EAI : Enterprise Application Integration, the process to integrate an application into an EIS.

JAFS contains an events dispatching (push) mechanism who can be used to integrate the server into your software infrastructure quite easily.

Currently the events are defined into 3 distinct categories :

- [Service events](#) : All the events fired by the JAFS services.
- [Server events](#) : The events fired by the deployed FTP server instances.
- [Session events](#) : FTP client session related events.

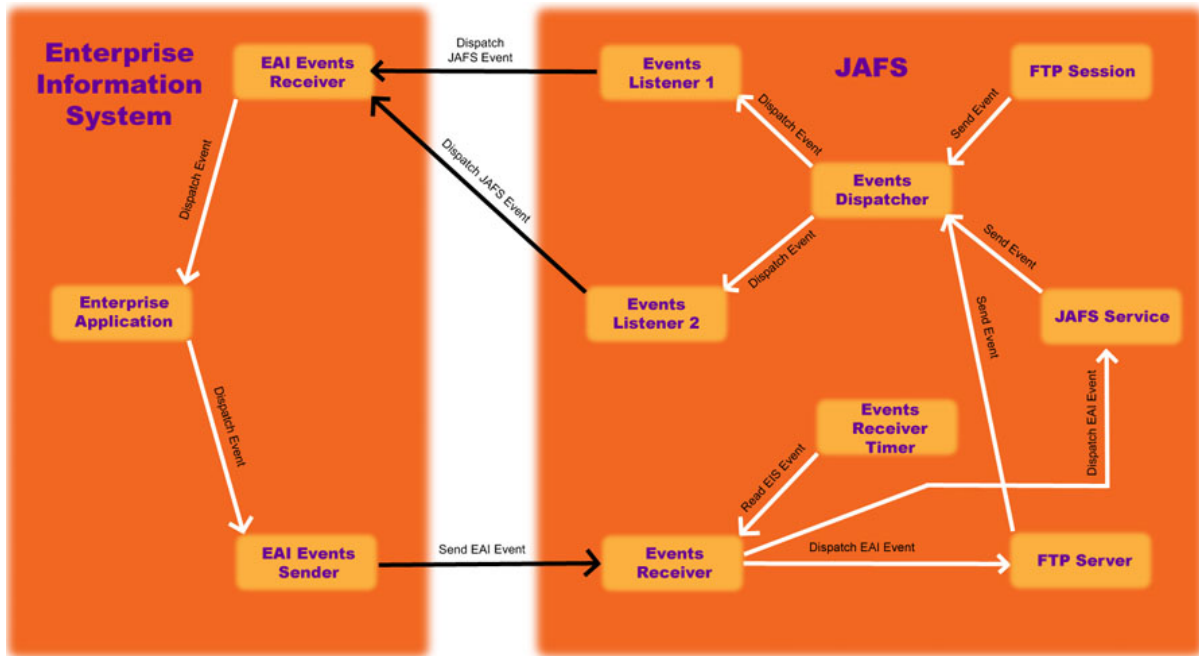
As you can see, all events names are self explanatory, so no need to document them one by one.

All events when fired have an attached events Object who contains informations related to the events execution context ( I.E : a file path for an file related event ).

Once fired by JAFS components all events are dispatched to registered [events listeners](#) who can then handle the event and interact with your EIS. All registered events listeners can select the type of events set they want to receive. The listeners can also filter the event they have received and decide (based on the data contained into the event object) if the event needs to be handled or not.

If you need to retrieve (pull) data from your EIS, JAFS provides an mechanism to plug receivers who can be used to read the data from the EIS and dispatch it within JAFS.

The following schema summarize everything :



EAI Schema

## 2. Provided events listeners

Currently six events listener are provided :

- net.sbbi.jafs.eventslisteners.JMXNotificationEventsListener : Fires events as JMX notifications via a JMX Model bean.
- net.sbbi.jafs.eventslisteners.JabberEventsListener : Fires events on an Jabber IM Server for a given user.
- net.sbbi.jafs.eventslisteners.JMSEventsListener : Fires events on an Java Messaging Queue.
- net.sbbi.jafs.eventslisteners.MailEventsListener : Publish the events into an e-mail
- net.sbbi.jafs.eventslisteners.SoundEventsListener : Plays a sound for a given event
- net.sbbi.jafs.eventslisteners.SystemOutEventsListener : Output the event data to the System.out ( for testing purpose ).

## 3. Provided events receivers

Currently three events receivers are provided :

- net.sbbi.jafs.eventslisteners.SyslogEventsListener : Unix syslog generic receiver
- net.sbbi.jafs.eventslisteners.NetgearDG834GSyslogEventsListener : Netgear DG834

## JAFS events and EAI

DSL router modem syslog receiver to automatically shield the JAFS deployed servers against DOS and ports scan reported by the router via syslog message broadcast.

- net.sbbi.jafs.eventslisteners.SystemOutEventsListener : Output recieved syslogs to System.out ( for testing purpose ).

### 4. Events listeners setup

To plug an events listener you'll have to add an [events listener](#) configuration entry into the JAFS service config file.

Each configuration entry consists of :

- An entry name
- An listener class name ( such as described above )
- A list of settings for the listener
- A list of events names ( ServicesEvents,ServerEvents,SessionEvents class ) that the listener will receive

You can see a sample configuration entry here which will register 2 events listener :

```
<jafs-service xmlns="http://www.sbbi.net/jafs/1.0/jafs-service"
    ...>
    ...
    <events-listeners>
        <listener name="systemoutSessionEvents"
            class-name="net.sbbi.jafs.eventslisteners.SystemOutEventsListener">

            <setting name="targetFtpServersCode">sample_server</setting>
            <setting name="targetPaths">../ftp/pub/userhome,../ftp/pub/testdir</setting>
            <setting name="targetClientsIp">127.0.0.1,192.168.1.33</setting>

            <target-events>
                <server-event name="onBlackListMatch"/>
                <session-event name="*" />
            </target-events>
        </listener>

        <listener name="jabberSessionEvents"
            class-name="net.sbbi.jafs.eventslisteners.JabberEventsListener" asy

            <setting name="targetPaths">../ftp/pub/userhome,../ftp/pub/testdir</setting>
            <setting name="targetClientsIp">127.0.0.1,192.168.1.33</setting>

            <setting name="sslServer">true</setting>
            <setting name="serverName">jabber.mycomp.com</setting>
            <setting name="serverPort">1234</setting>
            <setting name="userName">username</setting>
            <setting name="userServer">jabber.mycomp.com</setting>
            <setting name="password">password</setting>
```

```

<setting name="messagesReceiver">messagesReceiver@jabber.mycomp.com</setting>
<setting name="eventsPerMessage">4</setting>
<setting name="sendTestMessage">true</setting>
<setting name="reconnectRetriesMaxTime">10</setting>

<target-events>
  <services-event name="onIPBlackListReload"/>
  <session-event name="onFileUploadStart"/>
  <session-event name="onFileUploadFinished"/>
</target-events>
</listener-->
</events-listeners>
...
</jafs-service>

```

The first listener( `systemoutSessionEvents` ) will output to the console all session events ( `<session-event name="*/>` ) and when an clients matchs an ip blacklist entry ( `<server-event name="onBlackListMatch"/>` ). The second listener ( `jabberSessionEvents` ) will send a message asynchronously to a Jabber server each time 4 events of type `onFileUploadStart`, `onFileUploadFinished` are received for client IP 127.0.0.1 or 192.168.1.33 for a file contained into the `../ftp/pub/userhome` or `../ftp/pub/testdir` directory. The event `onIPBlackListReload` will always be received since it does not contains any client IP or file path related information.

## 5. Generic settings

All events listener can filter an event based on the event object content. Currently 3 settings to filter the events are available :

- **targetFtpServersCode** : For server and session events. A comma delimited list of [ftp server codes](#) to determine for which FTP server instance the event will be received by the listener.
- **targetPaths** : For session events. A comma delimited list of file path. The value will instruct the listener to only accept file transferts related events occuring into a certain set of directories. This gives you the possibility to monitor file related session events for certain directories.
- **targetClientsIp** : For session events. A comma delimited list of IP, the setting allows the listener to only accept session events for a certain set of target IP.

If you're not happy with the provided filters you can write some code to create your own filters quite easily.

## 6. Listeners settings

You can see here all the available settings for the provided events listeners:

## 6.1. net.sbbi.jafs.eventslisteners.JMXNotificationEventsListener settings

All events received by this listener will be send to notifications listener via the JMX notifications mechanism. This listener is itself a ModelMbean, broadcasting notifications messages based on the received events. Once the listener is started, you can register your own JMX events listeners to receive the JMX notifications.

The filter accepts the following settings:

- **mbeanServerDomain** : Optional, The MBean server where this listener Model Mbean must be registred, The JAFS Mbeans server will be used when no setting is provided.
- **objectName** : Optional, The Model Mbean JMX object name used for registration within the MBeans server, A default name will be used when no setting is provided.

## 6.2. net.sbbi.jafs.eventslisteners.JMSEventsListener settings

This filter can be used to fire received events on an JMS message queue and/or topic. The fired message is of type javax.jmx.MapMessage. The message will contain all the event object fields name as the map keys and the fields values as the map values. The Object primitive types ( String, long, int... ) will be mapped with the corresponding MapMessage ( setString(), setLong(), setInt()... ) methods. Non primitive Object fields will be mapped with the MapMessage setObject() method . Null fields will be mapped with a null value.

The listener can fire each events either to a JMS message queue or topic or to both. The is the reason why the queueConnectionFactoryJNDIName and topicConnectionFactoryJNDIName settings are optional, but at least one must be provided.

The filter accepts the following settings:

- **JNDIInitialContext** : Mandatory, The JNDI Initial context.
- **JNDIProviderUrl** : Mandatory, The JNDI Provider URL.
- **JNDISecurityProtocol** : Optional, The JNDI security protocol, *ssl* for SSL encryption.
- **JNDISecurityAuthentication** : Optional, JNDI authentication method, *simple* for username password based auth, *none* by default.
- **JNDISecurityPrincipal** : Optional, The username to access the JNDI directory.
- **JNDISecurityCredentials** : Optional, The password to access the JNDI directory.
- **queueConnectionFactoryJNDIName** : Optional, The JMS queue connection factory JNDI object name.
- **topicConnectionFactoryJNDIName** : Optional, The JMS topic connection factory JNDI object name.
- **queueJNDIName** : Optional, The JMS queue JNDI object name.
- **topicJNDIName** : Optional, The JMS topic JNDI object name.

### 6.3. net.sbbi.jafs.eventslisteners.JabberEventsListener settings

This filter can be used to fire received events to a Jabber IM server. You can define after how many received events the instant message will be sent to the jabber server and who will send/receive the events.

The filter accepts the following settings:

- **sslServer** : Optional, boolean to set the connection with an SSL enabled jabber server, false by default.
- **serverName** : Required, the jabber server name.
- **serverPort** : Optional, the jabber server port, jabber default port or SSL port (if sslServer set to true) by default.
- **userName** : Required, the jabber user name used to send the messages.
- **userServer** : Optional, the jabber user name server, default set with the serverName setting value.
- **password** : Required, the jabber user password.
- **messagesReceiver** : Optional, the instant messages receiver, default set with the userName@userServer settings values.
- **eventsPerMessage** : Optional, defines after how many received events an message will be sent to the messages receiver, default set to 1.
- **sendTestMessage** : Optional, boolean to send a test message when the listener is started, usefull to test if the setup is ok, false by default.
- **reconnectRetriesMaxTime** : Optional, the reconnection timeout in minutes to try to reconnect to the Jabber server when the connection is lost, 5 minutes by default.

### 6.4. net.sbbi.jafs.eventslisteners.MailEventsListener settings

This filter will send an email with all the received events during a defined time lapse.

The filter accepts the following settings:

- **smtpServerId** : Required, the SMTP service file configuration [entry id](#) that will be used to send the mails.
- **executeTime** : Optional, the time interval in minutes to send the mails, 10 by default.

### 6.5. net.sbbi.jafs.eventslisteners.SoundEventsListener settings

This filter can launch a sound playback each time an event is received.

The filter accepts the following settings:

Set of eventsId : Required, A Wav or AIFF file path for each events id of type [Service](#)

## JAFS events and EAI

, [Server events](#) or [Session events](#) that you have configured to be received. For example if the listener is configured to receive events of type *onConnect*, add a setting named *onConnect* with the corresponding sound file path to play when the event is received :

```
<listener name="soundsListener" class-name="net.sbbsi.jafs.eventslisteners.Sound
  <setting name="targetFtpServersCode">sample_server,my_other_server_code</sett

  <setting name="onConnect">/foo/bar/onConnectSoundFile.wav</setting>
  <setting name="onDisconnect">/foo/bar/onDisconnectSoundFile.wav</setting>
  <target-events>
    <session-event name="onConnect"/>
    <session-event name="onDisconnect"/>
  </target-events>
</listener>
```

If you forget an event id into the settings, no sounds will be played when the event is received.

### 7. Events receivers setup

Each events receivers are configured into an [events receiver](#) entry. Each entry consists of an [name](#), a receiver [class name](#), an [execution time](#) interval in ms and finally a set of custom setting use to configure the receiver.

```
<jafs-service xmlns="http://www.sbbsi.net/jafs/1.0/jafs-service"
  ...>
  ...
  <events-receivers>
    <receiver name="myEventsReceiver" class-name="com.mycompany.TestPullEventsLis
      <setting name="mySampleSetting">testValue</setting>
    </receiver>
  </events-receivers>
  ...
</jafs-service>
```

### 8. Receivers settings

You can see here all the available settings for the provided events receivers:

#### 8.1. net.sbbsi.jafs.eventslisteners.SyslogEventsListener

This events receiver can be used to receive syslog messages from the network. This class cannot be used directly but provides all the mechanics for extending classes ( such as the `net.sbbsi.jafs.eventslisteners.SystemOutEventsListener` and

net.sbbi.jafs.eventslisteners.NetgearDG834GSyslogEventsListener classes ) to handle syslog messages.

The receiver requires the following settings:

- **port** : Optional, The syslog listening port, 514 by default.
- **destIp** : Mandatory, The syslog destination IP.
- **srcIp** : Optional, A comma delimited list of syslog messages senders IP.
- **bufferLength** : Optional, The messages max length, 2048 by default.
- **readTimeout** : Optional, The syslog messages listening timeout in ms, 250 by default.
- **syslogFacility** : Optional, A comma delimited list of desired syslog messages facilities (KERNEL,AUTH,LOCAL0...), when not provided the receiver will accept all syslog facility.
- **syslogPriority** : Optional, A comma delimited list of desired syslog messages priorities (INFO,WARNING,CRITICAL..), when not provided the receiver will accept all syslog facility.

## 9. Create your own events listeners

To create your own listener you'll have to write a class that implements the *net.sbbi.jafs.events.PushEventsListener* interface and implement the following methods :

- `filterEvent()` : filters the event, used to accept (true returned) or reject an event based on the event object event content.
- `getName()` : retrieve the name of the events listener.
- `setName()` : sets the name of the events listener, called after the listener object creation.
- `handleEvent()` : handle the event.
- `enableLogging()` : for logging purpose called before the parameterize method.
- `service()` : used to lookup services objects.
- `start()` : called when the listener is started.
- `stop()` : called when the listener is stopped.
- `parameterize()` : used to retrieve the settings of the listener, called before the start method.

You can see a small example class here :

```
package com.mycompany;

import net.sbbi.jafs.services.ServersDeploymentService;

import org.apache.avalon.framework.logger.Logger;
import org.apache.avalon.framework.parameters.ParameterException;
import org.apache.avalon.framework.parameters.Parameters;
import org.apache.avalon.framework.service.ServiceException;
import org.apache.avalon.framework.service.ServiceManager;

public class TestEventsListener extends GenericEventsListener
```

## JAFS events and EAI

```
implements PushEventsListener {

    private String myNeededParam;
    private String myOptionalParam;

    public void handleEvent( int topic, String event,
                            Event eventData, long eventTimestamp ) {
        System.out.println( "Received event " + event +
                            " with context object " + eventData );
    }

    public boolean filterEvent( Event eventData ) {
        // no special filter needed
        return true;
    }

    public void service( ServiceManager manager ) throws ServiceException {
        ServersDeploymentService deploySrv =
            (ServersDeploymentService)manager.lookup( ServersDeploymentService.ROLE );
        System.out.println( "Deployed server codes are : " +
                            deploySrv.getDeployedServersCodes() );
    }

    public void start() throws Exception {
    }

    public void stop() throws Exception {
    }

    public void parameterize( Parameters params ) throws ParameterException {
        myNeededParam = params.getParameter( "myNeededParam" );
        myOptionalParam = params.getParameter( "myOptionalParam", null );
    }
}
```

You can extend the *net.sbbi.jafs.events.GenericEventsListener* interface if you want a part of the job to be done automatically and the *net.sbbi.jafs.events.GenericServerEventsListener*, *net.sbbi.jafs.events.GenericServicesEventsListener* or *net.sbbi.jafs.events.GenericSessionEventsListener* if you want certain generic filtering capabilities.

Once compiled, put the class into a jar file and drop it into the JAFS *libs* directory, configure the service config file to register your new listener and start JAFS, everything should be alright :o)

### 10. Create your own events receivers

This time you'll need to implement the *net.sbbi.jafs.events.PullEventsListener* interface which is quite similar to the *net.sbbi.jafs.events.PushEventsListener* interface. You'll have to

implement the following methods :

- `isEventReady()` : reads from the EIS if an event is ready to be handled, this method will be called each [listening time](#) ms. When the method returns true, the `handleEvent()` method will be called.
- `getName()` : retrieve the name of the events receiver.
- `setName()` : sets the name of the events receiver, called after the receiver object creation.
- `handleEvent()` : handle the readen event.
- `enableLogging()` : for logging purpose called before the parameterize method.
- `service()` : used to lookup services objects.
- `start()` : called when the receiver is started.
- `stop()` : called when the receiver is stopped.
- `parameterize()` : used to retrieve the settings of the receiver, called before the start method.

You can see a small example class here :

```
package com.mycompany;

import java.io.IOException;

import org.apache.avalon.framework.logger.Logger;
import org.apache.avalon.framework.parameters.ParameterException;
import org.apache.avalon.framework.parameters.Parameters;
import org.apache.avalon.framework.service.ServiceException;
import org.apache.avalon.framework.service.ServiceManager;

public class TestEventsListener implements PullEventsListener {

    private String name;
    private Logger log;

    public String getName() {
        return name;
    }

    public void setName( String listenerName ) {
        name = listenerName;
    }

    public void enableLogging( Logger logger ) {
        this.log = logger;
    }

    public void handleEvent() throws Exception {
        // handle the event stored into memory
    }

    public boolean isEventReady() throws IOException {
        // read the event into memory
    }
}
```

## JAFS events and EAI

```
        return true;
    }

    public void parameterize( Parameters params ) throws ParameterException {
    }

    public void service( ServiceManager manager ) throws ServiceException {
    }

    public void start() throws Exception {
    }

    public void stop() throws Exception {
    }
}
```

Once compiled, put the class into a jar file and drop it into the JAFS *libs* directory, configure the service config file to register your new receiver and start JAFS.