

# Jafs custom FTP commands

by SuperBonBon

## 1. Introduction

JAFS allows you to reimplement, add or remove FTP commands for each deployed FTP server instances. This is done with the [command](#) xml tag for generic FTP protocol commands and the [feat](#) xml tag for FTP features.

For each command or feature configuration entry, you'll need to define a protocol [command name](#) ( such as RETR to retrieve a file ), a command implementation [class name](#) and optionally a class [location](#) ( such as <http://www.myhost.com/myImpl.jar> ). The commands and features config let you also provide a set of [settings](#).

### Warning:

When you add or remove a command watch out what kind of commands you modify. If you remove mandatory protocol commands, the server will not work anymore !

## 2. Commands implementation

To create your own command, you'll have to implement the *net.sbbi.jafs.commands.CommandInterface*. Here is a small class who will return an "Hello world" to the FTP client :

```
package com.myCompany;

import java.net.InetAddress;

import org.apache.avalon.framework.parameters.Parameters;

import net.sbbi.jafs.session.SessionContext;
import net.sbbi.jafs.session.SessionInstructionProcessor;

public class HelloWorld implements CommandInterface {

    public void execute( StringBuffer responseBuffer, String commandParameter,
                        SessionContext sessionContext,
                        SessionInstructionProcessor processor ) throws Exception {
```

```
// let's retrieve some info from the session context
InetAddress piAdr = (InetAddress)sessionContext.get( SessionContext.PI_INET_A
// use the StringBuffer if you have large output or multiple concats
responseBuffer.append( "Hello world to IP " ).append( piAdr.getHostAddress() )
// the commandParameter String contains all the params provided by the client
if ( commandParameter.length() > 0 ) {
    responseBuffer.append( " you param is " ).append( commandParameter );
}
// if you need to store or retrieve custom data from the context use the
// getCustomObject() and putCustomObject() methods
if ( sessionContext.getCustomObject( "myCustomKey" ) == null ) {
    sessionContext.putCustomObject( "MyCustomKey", "myUselessValue" );
}
// ok now write the response to the client
processor.getPIRw().writePI( "200", responseBuffer.toString() );
// no need to cleanup the buffer this is done automatically
}

public void parameterize( Parameters parameters ) {
    // contains the params defined in the config file for the command
}
}
```

If your command receives a file path as command parameter, you can use the *net.sbbi.jafs.commands.AbstractPathCommand* ( for a directory as param ) class or the *net.sbbi.jafs.commands.AbstractFileCommand* ( for a file as param ) class to easily access the file passed into param and make some automatic security checks.

### Warning:

The commands and features objects instances are multithreaded ! It means that the same object instance on an FTP server instance is used for every FTP client sessions. So make sure that you synchronize correctly what needs to be. Calls to the objects passed into the execute() method signature do not need synchronization.

### Note:

We recommend you to put your commands classes into the *net.sbbi.jafs.commands* package ( for commands classes auto listing via JMX interface ) however this is not mandatory.

### 3. Features implementation

The features are quite the same than the commands, they have to implement the *FeatCommandInterface* interface which consists of 2 new methods :

- *getVersion()* : returns the version of the feature.
- *getUse()* : to retrieve the usage text that will be returned when the FEAT command is called.

```
package com.myCompany;
```

## Jafs custom FTP commands

```
import java.net.InetAddress;

import org.apache.avalon.framework.parameters.ParameterException;
import org.apache.avalon.framework.parameters.Parameters;

import net.sbbi.jafs.session.SessionContext;
import net.sbbi.jafs.session.SessionInstructionProcessor;

public class HelloWorld implements FeatCommandInterface {

    private String myParam;

    public void execute( StringBuffer responseBuffer, String commandParameter,
                        SessionContext sessionContext,
                        SessionInstructionProcessor processor ) throws Exception {

        // let's retrieve some info from the session context
        InetAddress piAdr = (InetAddress)sessionContext.get( SessionContext.PI_INET_A
        // use the StringBuffer if you have large output or multiple concats
        responseBuffer.append( "HelloWorld to IP " ).append( piAdr.getHostAddress() )
        // the commandParameter String contains all the params provided by the client
        if ( commandParameter.length() > 0 ) {
            responseBuffer.append( " you param is " ).append( commandParameter );
        }
        if ( myParam != null ) {
            responseBuffer.append( " feature params is " ).append( myParam );
        }
        // if you need to store or retrieve custom data from the context use the
        // getCustomObject() and putCustomObject() methods
        if ( sessionContext.getCustomObject( "myCustomKey" ) == null ) {
            sessionContext.putCustomObject( "MyCustomKey", "myUselessValue" );
        }

        // ok now write the response to the client
        processor.getPIrw().writePI( "200", responseBuffer.toString() );
        // no need to cleanup the buffer this is done automatically
    }

    public String getUse( SessionInstructionProcessor processor ) {
        return "someHelloWorldParam";
    }

    public byte getVersion() {
        return 1;
    }

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

Once you're done simply compile the code, put the jar into the *libs* dir and add the feature the

the server configuration entry, start jafs and you're done :

```
<server xmlns="http://www.sbbi.net/jafs/1.0/jafs-server"
  ...>
  ...
  <feats>
    ...
    <feat className="com.myCompany.HelloWorld" name="HELLOWORLD">
      <setting name="myHelloWorldParam">Additional myHelloWorldParam param</setting>
    </feat>
    ...
  </feats>
  ...
</server>
```

When the FTP client will send an HELLOWORLD command the code above will be executed.

**Note:**

We recommend you to put your features commands classes into the *net.sbbi.jafs.commands.feat* package ( for features commands classes auto listing via the JMX interface ) however this is not mandatory.