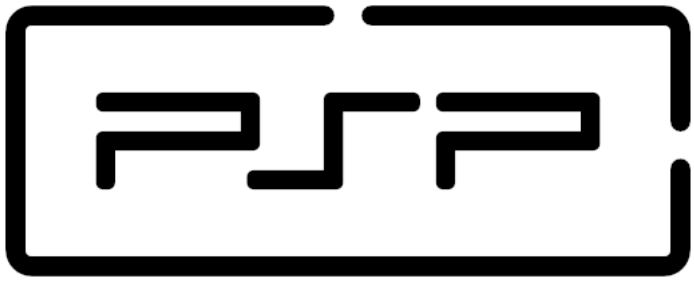


Process and Service Programming

5.2 FTP Client

I.E.S.
Doctor Balmis

5.2 FTP Client

5.2.1 Apache Common Net FTP

Apache Commons Net™ library implements the client side of many basic Internet protocols.

The purpose of the library is to provide fundamental protocol access, not higher-level abstractions. Therefore, some of the design violates object-oriented design principles.

Our philosophy is to make the global functionality of a protocol accessible (e.g., TFTP send file and receive file) when possible, but also provide access to the fundamental protocols where applicable so that the programmer may construct his own custom implementations (e.g, the TFTP packet classes and the TFTP packet send and receive methods are exposed).

Apache Commons NET™ library

Supported protocols include:

- FTP/FTPS
- FTP over HTTP (experimental)
- NNTP
- SMTP(S)
- POP3(S)
- IMAP(S)
- Telnet
- TFTP
- Finger
- Whois
- rexec/rcmd/rlogin
- Time (rdate) and Daytime
- Echo
- Discard
- NTP/SNTP

In Java, natively, it is possible to carry out file transfers using this protocol, but it is extremely hard to do so. The Apache Commons Net library provides classes and utilities to perform any operation on an FTP or FTPS server from a Java client.

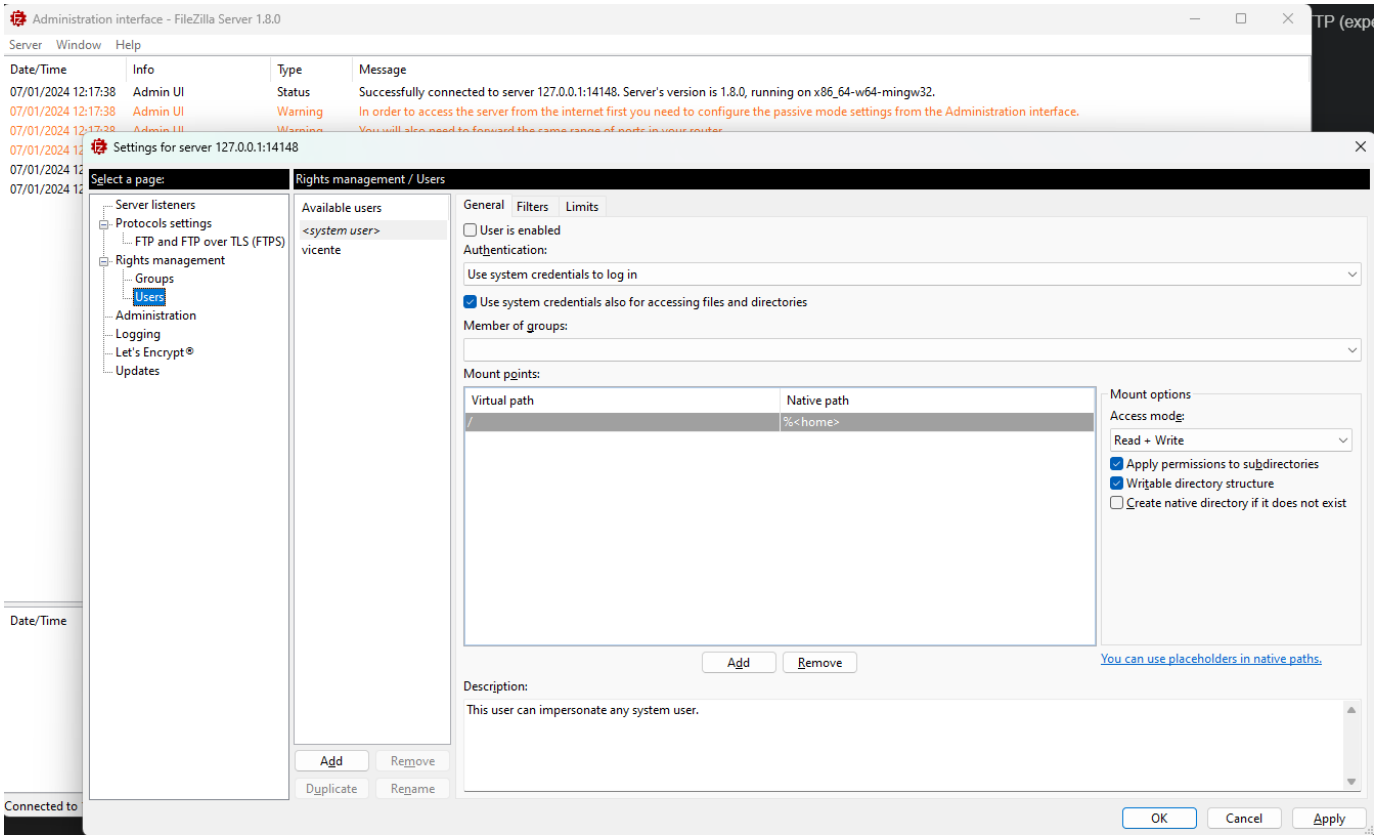
This library can be downloaded from the apache.org website through the following link: https://commo ns.apache.org/proper/commons-net/

5.2.2 FTP Server

The FTP server is a program that runs on a computer and allows other computers to connect to it and transfer files using the FTP protocol.

To make tests we are going to use the FileZilla FTP server, which can be downloaded from the Filezilla site (https://filezilla-project.org/download.php?type=server) .

The installation is very simple, just follow the steps of the installation wizard. Once installed, we will have to configure the server, for this we will have to open the FileZilla Server Interface program, which will be in the Windows start menu.



Once you have installed the server, you will have to configure it. To do this, you will have to open the FileZilla Server Interface program, which will be in the Windows start menu and open the Server menu and select the option "Configure".

Then, in the Users section, we will have to create a user, for this we will have to click on the "Add" button and fill in the fields with the data of the user that we want to create. Then change the Authentication type to "Require a password to login" and enter the password that we want to assign to the user.

Then, in the Mounting Points section, we will have to add a new mounting point, for this we will have to click on the "Add" button and fill in the fields with the data of the mounting point that we want to create. In the "Native path" field we will have to select the directory that we want to share.

> **Virtual Paths vs Native Paths**
>
> Virtual paths are used to map a directory on the server to a virtual directory. This allows you to create a virtual directory structure that is different from the actual directory structure on the server. For example, you can map the directory "C:\My Documents" to the virtual directory "/Documents". When a client connects to the server and changes to the "/Documents" directory, the client will actually be in the "C:\My Documents" directory on the server.

Once we have created the user and the mounting point, we will have to click on the "OK" button to save the changes.

5.2.3 FTP Client

The main classes and methods in the org.apache.commons.net.ftp package are shown below.

5.2.3.1 Connect and login

```java
public class ApacheFTPClient {

    private String server;
    private int port;
    private String user;
    private String password;
    private FTPClient ftp;

    public ApacheFTPClient(String server, int port, String user, String password) {
        this.server = server;
        this.port = port;
        this.user = user;
        this.password = password;
    }

    void open() throws IOException {
        ftp = new FTPClient();

        ftp.connect(server, port);
        int reply = ftp.getReplyCode();
        if (!FTPReply.isPositiveCompletion(reply)) {
            ftp.disconnect();
            throw new IOException("Exception in connecting to FTP Server");
        }

        ftp.login(user, password);
    }

    void close() throws IOException {
        ftp.disconnect();
    }

    public static void main(String[] args) throws IOException {
        ApacheFTPClient client = new ApacheFTPClient("localhost", 21, "alumnodam", "psp");
        client.open();
        client.close();
    }
}
```

In the previous example we can see how to connect to an FTP server using the Apache Commons Net library.

1. To do this, we will have to create an instance of the **FTPClient** class and call the connect method, passing as parameters the server and the port to which we want to connect.
2. Then we will have to check that the connection has been established correctly, for this we will have to call the *getReplyCode* method and check that the value returned by this method is positive.
3. Finally, we will have to call the login method, passing as parameters the user and the password with which we want to log in.

5.2.3.2 Change directories

In the FTP protocol, the local directory is the directory on the client's computer and the remote directory is the directory on the server. Both the local and remote directories are called working directories and both can be changed with the **changeWorkingDirectory** method for the remote files and the **File** methods to set the origin for the local files.

To show the current working directory, we can use the **printWorkingDirectory** method and to show the contents of the current working directory, we can use the **listFiles** method or the **listNames** method.

```java
// Get the current remote working directory
String workingDirectory = ftp.printWorkingDirectory();

// Show contents of current remote working directory
for (String name : ftp.listNames()) {
        System.out.println(name);
}

// Change the current remote working directory
ftp.changeWorkingDirectory(fileName)
```

To show the current local working directory, we have to manage it with the File class and its methods. We can also rely on the System class and its methods as we studied in the process management unit.

```java
// Get the current local working directory
String workingDirectory = System.getProperty("user.dir");

// Show contents of current local working directory
File file = new File(workingDirectory);
for (String name : file.list()) {
        System.out.println(name);
}

// Change the current local working directory
System.setProperty("user.dir", fileName);
```

5.2.3.3 Upload files

In FTP files can be upload in two different ways:

ASCII mode: The file is uploaded as text, with the line endings converted to the network standard.

```java
public boolean sendTextFile(String fileName) throws FileNotFoundException, IOException {
    ftp.setFileType(FTP.ASCII_FILE_TYPE);
    // The getLocalWorkingDirectory() it's a custom method (not from the Apache Commons library)
    // that returns the path to the local file we want to upload
    File file = new File(getLocalWorkingDirectory() + "/" + fileName);
    // The file name in the server can be different from the local file name
    String fileRemote = fileName;
    InputStream input = new FileInputStream(file);
    boolean upload = ftp.storeFile(fileRemote, input);
    input.close();
```

```
        return upload;
    }
```

Binary mode: The file is uploaded as is, without any conversion.

```java
public boolean sendBinaryFile(String fileName) throws IOException {
        ftp.setFileType(FTP.BINARY_FILE_TYPE);
        // The getLocalWorkingDirectory() it's a custom method (not from the Apache Commons library)
        // that returns the path to the local file we want to upload
        File file = new File(getLocalWorkingDirectory() + "/" + fileName);
        // The file name in the server can be different from the local file name
        String fileRemote = fileName;

        InputStream input = new FileInputStream(file);
        OutputStream output = ftp.storeFileStream(fileRemote);
        byte[] bytesIn = new byte[4096];
        int read = 0;

        while ((read = input.read(bytesIn)) != -1) {
            output.write(bytesIn, 0, read);
        }

        input.close();
        output.close();
        boolean upload = ftp.completePendingCommand();

        return upload;
    }
```

5.2.3.4 Download files

As with the upload, files can be downloaded in two different ways:

ASCII mode: The file is downloaded as text, with the line endings converted to the local standard.

```java
    public boolean getTextFile(String fileName) throws IOException {
        ftp.setFileType(FTP.ASCII_FILE_TYPE);
        // The getLocalWorkingDirectory() it's a custom method (not from the Apache Commons library)
        // that returns the path to the local file we want to download
        File file = new File(getLocalWorkingDirectory() + "/" + fileName);
        // The file name in the server can be different from the local file name
        String fileRemote = fileName;
        OutputStream output = new FileOutputStream(file);
        boolean download = ftp.retrieveFile(fileRemote, output);
        output.close();
        return download;
    }
```

Binary mode: The file is downloaded as is, without any conversion.

```java
    public boolean getBinaryFile(String fileName) throws IOException {
        ftp.setFileType(FTP.BINARY_FILE_TYPE);
        // The getLocalWorkingDirectory() it's a custom method (not from the Apache Commons library)
        // that returns the path to the local file we want to download
        File file = new File(getLocalWorkingDirectory() + "/" + fileName);
```

```
        // The file name in the server can be different from the local file name
        String fileRemote = fileName;

        OutputStream output = new FileOutputStream(file);
        InputStream input = ftp.retrieveFileStream(fileRemote);
        byte[] bytesIn = new byte[4096];
        int read = 0;

        while ((read = input.read(bytesIn)) != -1) {
            output.write(bytesIn, 0, read);
        }

        boolean download = ftp.completePendingCommand();
        input.close();
        output.close();
        return download;
    }
```

5.2.3.5 Other operations

Other operations that can be performed on the FTP server are:

Show file information and file properties

java

```
    public void showFileInfo(String fileName) throws IOException {
        FTPFile[] files = ftp.listFiles(fileName);
        for (FTPFile file : files) {
            System.out.println(file.getName());
            System.out.println(file.getTimestamp().getTime());
            System.out.println(file.getSize());
            if (f.getType() == FTPFile.FILE_TYPE) {
                tipo = "File";
            }
            if (f.getType() == FTPFile.DIRECTORY_TYPE) {
                tipo = "Folder";
            }
        }
    }
```

Add and remove remote FTP directories

java

```
    public void addRemoteDirectory(String directory) throws IOException {
        ftp.makeDirectory(directory);
    }

    public void removeRemoteDirectory(String directory) throws IOException {
        ftp.removeDirectory(directory);
    }
```

Create and remove remote FTP files

java

```
    public void addRemoteFile(String fileName) throws IOException {
        ftp.storeFile(fileName, new ByteArrayInputStream(new byte[0]));
    }

```

```java
    public void removeRemoteFile(String fileName) throws IOException {
        ftp.deleteFile(fileName);
    }
```

Rename remote FTP files and directories

```java
    public void renameRemoteFile(String oldName, String newName) throws IOException {
        ftp.rename(oldName, newName);
    }
```