jBASE jRFS

Remote file support in jBASE

Contents

Documentation Conventions

This manual uses the following conventions:

Convention	Usage
BOLD	In syntax, bold indicates commands, function names, and options. In text, bold indicates keys to press, function names, menu selections, and MS-DOS commands.
UPPERCASE	In syntax, uppercase indicates JBASE commands, keywords, and options; BASIC statements and functions; and SQL statements and keywords. In text, uppercase also indicates JBASE identifiers such as filenames, account names, schema names, and Windows NT filenames and pathnames.
UPPERCASE *Italic*	In syntax, italic indicates information that you supply. In text, italic also indicates UNIX commands and options, filenames, and pathnames.
Courier	Courier indicates examples of source code and system output.
Courier Bold	**Courier Bold** In examples, courier bold indicates characters that the user types or keys (for example, <Return>).
[]	Brackets enclose optional items. Do not type the brackets unless indicated.
{}	Braces enclose nonoptional items from which you must select at least one. Do not type the braces.
ItemA | itemB	A vertical bar separating items indicates that you can choose only one item. Do not type the vertical bar.
. . .	Three periods indicate that more of the same type of item can optionally follow.
⇒	A right arrow between menu options indicates you should choose each option in sequence. For example, “Choose **File ⇒Exit**” means you should choose **File** from the menu bar, and then choose **Exit** from the File pull-down menu.

Syntax definitions and examples are indented for ease in reading.

All punctuation marks included in the syntax—for example, commas, parentheses, or quotation marks—are required unless otherwise indicated.

Syntax lines that do not fit on one line in this manual are continued on subsequent lines. The continuation lines are indented. When entering syntax, type the entire syntax entry, including the continuation lines, on the same input line.

INTRODUCTION

The main purpose of jRFS is to allow files to be shared between two machines (A Client and a Server). It consists of a Server program which runs on the machine that hosts the files, and a remote JEDI which will allow the Client machine to access the files stored on the remote server.

jRFS also allows jQL Statements to be run on the Host machine via the client. The benefit of this is just the results of the select will be returned over the network to the client hence reducing the network traffic between the host and the client (If using standard file sharing between host and client, the client would need to read all the contents of a file over the network to process the jQL Statement).

In addition jRFS allows the execution of programs via the client on the host machine returning any output to the client.

Configuration files

The jRFS Server and client both use configuration files stored by default in the `$JBCRELEASEDIR/config` directory (`%JBCRELEASEDIR%\config` on the Windows platform) called:

```
jnet_config
jnet_map
jrfs_config
```

If you would like to store these files in a different location, you can do this but you need to make sure you set the following environment variable:

JBCNETDIR=<path to directory containing configuration files>

With this set, it will override the default location it will look for those files.

jnet_config

This file contains the configuration of how the client and server "Connect" to each other. It can contain:

`accesschk` If set to off, then it will disable client access check. (Server)

`usejnetok` If set to on, then it will use `jnet_access` file else use `ruserok` function.

`trace` If set to on, then it will trace `jnetconnect` functions as per `tracemask`.

`tracemask` If set to something, then it will use as AND mask for `jnetconnect` functions.

`Display` If set to on, then it will display trace and error messages to stderr.

`log` If set to on, then it will log trace and error message on to file specified in logfile.

`Logfile` If log is set to on, then put filename here to store trace and error messages. (This is a plain text file)

`logrequests` If set to yes, then it will log connect requests in the log file.

`mapclient` If set to off, then it will skip the search of the `jnet_map` file.

`clienthost` If set to on, then it will use the client provided hostname for access.

`encode` If set to on, then the connection string is encoded.

`dataencode` If set to on, then all the data packets are encoded.

Example `jnet_config` for Windows

```
trace=on
debug=on
display=on
log=on
logfile=C:\jbasehome\jnet.log
```

This will cause tracing messages to go to the screen as well as store them in the `c:\jbasehome\jnet.log` text file.

Example `jnet_config` for Unix

```
trace=on
debug=on
display=on
log=on
logfile=/home/jbase/5.0/tmp/jnet.log
accesschk=off
```

This will cause tracing messages to go to the screen and to the `/home/jbase/5.0/tmp/jnet.log` text file and also disable client access checking on the server end (It must be in the `jnet_config` for the server)

jnet_map

This file is used to map your client user information to the server user information. For each map required you need 2 lines, first your local client information and second the remote server information. As you can tell, this mapping is only required on the client side. The format is:

LOCAL:host servername server_u_name client_u_name
REMOTE:newhost {newservername {newserver_u_name}}

For example (see example setup later):

```
LOCAL:10.44.5.37 jRFS jdean jdean
REMOTE:10.44.5.37 jRFS jason
```

jrfs_config

This file contains configuration for the jRFS process.

`trace`	If set to on, then trace jrfs functions as per `tracemask`
`tracemask`	If set, then use as AND mask for jrfs functions.
`display`	If set to on, then display trace and error messages to stderr.
`log`	If set to on, then log trace and error message on logfile.
`Logfile`	If log is set to on, then this contains the file where the log is to be sent.
`servermaxfiles`	Initial maximum files reserved for each server, default 100
`servermaxselect`	Maximum selects each file allowed, default 5.
`sambapath`	Set to samba configuration file path
`serverport`	Set to jRFS socket port number
`remexec`	If set to off, then no remote executions are allowed restricted remote execution can be configured by creating a `jrfs_remexec` file, with the following example entries.
`SELECT *`	All users can exec SELECT command.
`MYPROG tm jim`	Only clients tm and jim can exec MYPROG.An example of how a remote connection was set up between a Windows XP Client PC to a Linux AS4 Host Server.

jQL and Remote Files

Normally when you run a jQL statement on a remote file, it will scan the file over the network, which causes lots of network traffic and is quite slow. As a way round this, jRFS allows jQL commands to be executed on the Server and just results get sent back over the network. This is much more efficient use of the network and is much quicker. To enable this functionality, you need to set the following environment variable:

```
JRFS_REMOTE_JQL=1
```

This will tell jBASE to execute the JQL command on a remote file on the server machine. A second environment variable should be set on the SERVER if the STUB-file on the client and the remote file on the server have different filenames:

```
JRFS_LOCALPATH_JQL=1
```

This will tell jBASE to enable use of remote file path.

Starting jRFS as a Service on Windows

On windows, you can setup jRFS so that it runs as a windows service. There needs to be a few environment variable changes done to windows first to allow this to happen. As a service, the jRFS does not use the local user settings (It uses the local SYSTEM user on the windows machine), so the environment needs to be set up in the Global Environment variable area. The following are the main variables required.

```
JBCRELEASEDIR={Where jBASE is installed}
JBCGLOBALDIR={Where jBASE is installed}
JEDIFILEPATH={Path to your files}
PATH={Where the jBASE bin files are installed};%PATH%
```

Other environment variables if needed, for example, JEDI_FILENAME_MD and JEDIFILENAME_SYSTEM, will also need to be set globally.

An example setup would be:

```
JBCRELEASEDIR=c:\jbase4\4.1
JBCGLOBALDIR=c:\jbase4\4.1
JEDIFILEPATH=c:\jbase4\home\data
PATH=c:\jbase4\4.1\bin;{Remainder of path that already exists}
```

Remember to make sure that the local SYSTEM user has permissions to access the directories you set up.

Once you have made these global environment variable changes, re-boot the machine so that they are available to all users on the machine.

Next from a command prompt or jshell type:

```
jservcontrol jRFS install
jservcontrol jRFS start
```

Now have a look at the task manager and make sure the jRFS is running and you can check via the Administrative Tools menu to see that the jBASE Remote File Service is showing.

Creating stub files

In order to access a remote file, a STUB file is required on the client side. This file is a plain text file with one line specifying basic information about the server host and the remote file:

JBC__SOB JediInitREMOTE <Remote filename> <Server addr>

Remote filename does not have to specify the remote path as the remote file will be located in the directory specified by the JEDIFILEPATH environment variable on the server.

If the remote file is again a STUB file, then the following entry has to be added to the **jnet_env** file (on the server, where this stub file is located):

ENV: servername server_u_name client_u_name

e.g: ENV:`jRFS 10.44.5.70 jason jdean`

Creating Remote pointers

Instead of creating STUB-files on the client, remote pointers (R-pointers) can be used to access remote files. The idea of R-pointers is based on the same principles as the STUB-files mention above, they contain basic information about the server host and the remote file. R-pointers are defined as an entry inside the SYSTEM file, which again may be accessed with a Q-pointer from the master dictionary (MD/VOC) file.

SYSTEM file:

→ Create a record with the following information

0001 R
0002 <Account name entry in Remote SYSTEM file> *[1]
0003
0004 <Hostname of remote machine>

MD file:

→ Create a record with the following information:

0001 Q
0002 <Local system account name entry>
0003 <Remote filename>

As we may notice, an R-pointer does not point directly to a remote file but instead it points to an account entry in the remote SYSTEM file. This entry on the remote SYSTEM file must again have a local pointer to the remote file.

Environment variables JEDIFILENAME_MD and JEDIFILENAME_SYSTEM have to be set accordingly on client and server side.

*[1] If you use an account name in the SYSTEM file on the client, then you need to make sure that the name is also in the SYSTEM file on the SERVER, pointing to a valid account location.

Access security

Access security can be enabled by adding the following line to the `jnet_config` file:

```
accesschk=on
```

The following settings only apply if access security has been enabled.

jnet_access

This file has to be configured on the server and specifies which client shall be granted access. One line has to be added which varies depending on whether the server host is a Windows or a UNIX machine.

UNIX:

hostname client_u_name server_u_name servername {umask}

e.g: `10.44.5.70 jdean jason jRFS`

Windows:

hostname client_u_name server_u_name servername password

e.g: `10.44.5.70 jdean jason jRFS mypwd`

If the server is on a Windows host then the password must be specified.

By default, the `jnet_access` file is located under the `config` path in the release directory. If this file is being placed on a different location, then the following environment variable must point to this path:

`JBCNETACCESS=`<path to jnet_access file>

usejnetok

A second access security mechanism exists which will use host security instead of the previously described method.

Edit the `jnet_config` configuration file and add the following file in order to enable this type of access security (`accesschk` must still be enabled).

```
usejnetok=on
```

This option is a UNIX feature only and should not be set on Windows servers.

jRFS example with Linux Server and a Windows Client

Username on the Client PC is jdean
Username on the server running the jRFS daemon is jason

Initial server side configuration.

Firstly we need to make sure the jRFS service is in the /etc/services file. Log in as root and vi the /etc/services file, if an entry already exists for jRFS then make note of the port number used, if not then create the following jRFS entry, then write the file back:

```
jRFS      50003/tcp                # jRFS Server
```

Now we need to be able to see all the tracing information when initially setting up the server so we can track down any configuration issues.

The `$JBCRELEASEDIR/jnet_config` file is modified first to the following:

```
accesschk=off
trace=on
debug=on
display=on
#tracemask=0x000f
log=on
logfile=/home/jason/jnet.log
```

We have set `accesschk` to `off` so we do not complicate the setting up of the remote connection with more security.

The `$JBCRELEASEDIR/jrfs_config` file is modified next to be the following:

```
trace=on
debug=on
display=on
#tracemask=0x000f
log=on
logfile=/home/jason/jrfs.log
```

Now we will create a directory called `remote` and create a J4 file in that directory called `REMFILE`.

```
mkdir remote
cd remote
CREATE-FILE REMFILE 1 1
```

```
cd
```

Now we set do

```
export JEDIFILEPATH=/home/jason/remote
jRFS –ib
```

This has now started the jRFS Server program. To check it started ok we:

```
cat jnet.log
```

You should now see something like:

```
Mar  7 16:16:38:jason:13194[Accept] accept request using host , server jRFS
Mar  7 16:16:38:jason:13194[GetAddr] find addr for host , server jRFS
Mar  7 16:16:38:jason:13194[GetAddr] initial host name
Mar  7 16:16:38:jason:13194[GetAddr] host name null string
Mar  7 16:16:38:jason:13194[GetAddr] looking for server jRFS by name
Mar  7 16:16:38:jason:13194[GetAddr] server name jRFS found
Mar  7 16:16:38:jason:13194[GetAddr] server using port 50003
Mar  7 16:16:38:jason:13194[GetAddr] transport address : 0200c35300000000
Mar  7 16:16:38:jason:13194[Accept] accepting on handle 3
```

```
cat jrfs.log
```

You should now see something like:

```
Mar  7 16:16:38:13194[Init] Server: INIT request
```

Client Side Configuration

Now we have the server started, lets move onto the windows client.

Got to the `windows\system32\drivers\etc` directory in windows explorer and edit the `services` file in notepad.

Look for the jRFS entry, if it exists make sure that it matches the port number on the server. If it doesn't exist then add the following line:

```
jRFS       50003/tcp                  # jRFS Server
```

Next we need to be able to see all the tracing information when initially setting up the client so we can track down any configuration issues. You can use notepad, jed or any other text editor to modify these files.

The `%JBCRELEASEDIR%\jnet_config` file is modified first to the following:

```
trace=on
debug=on
```

```
display=on
#tracemask=0x000f
log=on
logfile=c:\jnet.log
```

The `%JBCRELEASEDIR%\jrfs_config` file is modified next to be the following:

```
trace=on
debug=on
display=on
#tracemask=0x000f
log=on
logfile=c:\jrfs.log
```

Now we will create a remote Stub File to allow us to try and access the remote file we created on the server. In your case you will need to make sure you use the machine name/IP address of your server, the ones used in this example are of machines available to the author of this document.

Use notepad or jed to create a file called `REM` in the directory where you want to do the test. In this case it is `c:\home\REM`. Put the information relevant to your machine in place of the addresses used here.

```
JBC__SOB JediInitREMOTE REMFILE 10.44.5.37
```

Once this has been filed, any attempt to access with any jBASE command will try to access the file on the remote machine. (So using jed again will trigger the remote jEDI, if you think you have made a mistake use `jed . REM`)

Next we will use the same method to create the dictionary. Use jed or notepad to create c:\home\REM]D:

```
JBC__SOB JediInitREMOTE REMFILE]D 10.44.5.37
```

Now comes the first test of the remote file. In this case I know it will fail as the jnet_map file has yet to be configured. But doing this initial test will aid you in working out what is required in the jnet_map file.

```
LIST REM
```

You should get something similar appearing on your screen:

```
6632[Open] Client: Shared object detected
6632[Open] Client: Optional arguments REMFILE 10.44.5.37
6632[Open] Client: CONNECT request, RemoteAccount -, RemoteFileName REMFILE,
Host 10.44.5.37, Server jRFS
6632[Open] Client: allocate file handles
6632[Open] Client: Check if Host 10.44.5.37 already connected
6632[Open] Client: Testing table pointer 0
```

```
6632[Open] Client: Testing table pointer 1
6632[Open] Client: Testing table pointer 2
6632[Open] Client: Testing table pointer 3
6632[Open] Client: Testing table pointer 4
6632[Open] Client: Testing table pointer 5
6632[Open] Client: Testing table pointer 6
6632[Open] Client: Testing table pointer 7
6632[Open] Client: Testing table pointer 8
6632[Open] Client: Testing table pointer 9
6632[Open] Client: No Match find jfree host entry
6632[Open] Client: CONNECT entry 1 allocated to Host 10.44.5.37
6632[Connect] connect request for host 10.44.5.37, server jRFS
6632[Connect] HostId not null 10.44.5.37
6632[GetUser] GetUserName succesful, username jdean, size 6
6632[Connect] RemoteId null defaulted to User jdean
6632[NameMap] mapping host 10.44.5.37, server jRFS, user jdean, client jdean
6632[NameMap]   searching   C:\jbase4.1\config\jnet_map   for   map   string
'LOCAL:10.44.5.37 jRFS jdean jdean'
6632[NameMap] checking entry
6632[NameMap] rechecking entry
6632[NameMap] no entry found for map string
6632[GetAddr] find addr for host 10.44.5.37, server jRFS
6632[GetAddr] initial host name 10.44.5.37
6632[GetAddr] Initial HostName 10.44.5.37
6632[GetAddr] using dot host name 10.44.5.37
6632[GetAddr] looking for server jRFS by name
6632[GetAddr] server name jRFS found
6632[GetAddr] server using port 50003
6632[GetAddr] transport address : 0200c3530a2c0525
6632[Connect] initial socket connect successful, fd 1668
6632[ChkConnect] creating connect request info
6632[ChkConnect]  connect  info  Host  10.44.5.37,  User  jdean,  Remote  jdean,
Server jRFS
6632[ChkConnect] ConnectStr:10.44.5.37jdeanjdeanjRFS
6632[ChkConnect] EncryptedStr:10.44.5.37jdeanjdeanjRFS
6632[ChkConnect] sending connect request
6632[ChkConnect] connect request hdr sent, Tx'ed 40
6632[ChkConnect] sending str string, Tx'ed 24
6632[ChkConnect] waiting for connect confirmation
6632[ChkConnect] recieving confirmation response, Rx'ed 40
6632[ChkConnect] check for disconnect magic
6632[ChkConnect] disconnect detected , connection refused
6632[DisconSeen] confirming disconnect, fd 1668
6632[DisconSeen] sent disconnect confirmation, Tx'ed 40
6632[Connect] connect check failed
6632[Disconnect] disconnect request, fd -1
6632[Disconnect] disconnected already, handle invalid
6632[Open] Client: Host 10.44.5.37 connect request failed , errno -1
6632[Open] Client: Shared object detected
6632[Open] Client: Optional arguments REMFILE]D 10.44.5.37
6632[Open]  Client:  CONNECT  request,  RemoteAccount  -,  RemoteFileName
REMFILE]D, Host 10.44.5.37, Server jRFS
6632[Open] Client: Check if Host 10.44.5.37 already connected
6632[Open] Client: Testing table pointer 0
6632[Open] Client: Testing table pointer 1
6632[Open] Client: Testing table pointer 2
6632[Open] Client: Testing table pointer 3
6632[Open] Client: Testing table pointer 4
6632[Open] Client: Testing table pointer 5
6632[Open] Client: Testing table pointer 6
6632[Open] Client: Testing table pointer 7
6632[Open] Client: Testing table pointer 8
6632[Open] Client: Testing table pointer 9
6632[Open] Client: No Match find jfree host entry
6632[Open] Client: CONNECT entry 1 allocated to Host 10.44.5.37
6632[Connect] connect request for host 10.44.5.37, server jRFS
6632[Connect] HostId not null 10.44.5.37
6632[GetUser] GetUserName succesful, username jdean, size 6
6632[Connect] RemoteId null defaulted to User jdean
6632[NameMap] mapping host 10.44.5.37, server jRFS, user jdean, client jdean
6632[NameMap]   searching   C:\jbase4.1\config\jnet_map   for   map   string
'LOCAL:10.44.5.37 jRFS jdean jdean'
6632[NameMap] checking entry
```

```
6632[NameMap] rechecking entry
6632[NameMap] no entry found for map string
6632[GetAddr] find addr for host 10.44.5.37, server jRFS
6632[GetAddr] initial host name 10.44.5.37
6632[GetAddr] Initial HostName 10.44.5.37
6632[GetAddr] using dot host name 10.44.5.37
6632[GetAddr] looking for server jRFS by name
6632[GetAddr] server name jRFS found
6632[GetAddr] server using port 50003
6632[GetAddr] transport address : 0200c3530a2c0525
6632[Connect] initial socket connect successful, fd 1668
6632[ChkConnect] creating connect request info
6632[ChkConnect]  connect  info  Host  10.44.5.37,  User  jdean,  Remote  jdean,
Server jRFS
6632[ChkConnect] ConnectStr:10.44.5.37jdeanjdeanjRFS
6632[ChkConnect] EncryptedStr:10.44.5.37jdeanjdeanjRFS
6632[ChkConnect] sending connect request
6632[ChkConnect] connect request hdr sent, Tx'ed 40
6632[ChkConnect] sending str string, Tx'ed 24
6632[ChkConnect] waiting for connect confirmation
6632[ChkConnect] recieving confirmation response, Rx'ed 40
6632[ChkConnect] check for disconnect magic
6632[ChkConnect] disconnect detected , connection refused
6632[DisconSeen] confirming disconnect, fd 1668
6632[DisconSeen] sent disconnect confirmation, Tx'ed 40
6632[Connect] connect check failed
6632[Disconnect] disconnect request, fd -1
6632[Disconnect] disconnected already, handle invalid
6632[Open] Client: Host 10.44.5.37 connect request failed , errno -1
No file name could be found for your query
```

The lines we are interested at this time are highlighted above. In particular it's the map string: `'LOCAL:10.44.5.37 jRFS jdean jdean'`

Using this information (Remember your address and usernames will be different), we will now edit the `%JBCRELEASEDIR%\jnet_map` file and add the following to the end. Note: Make sure there is a blank line at the end of the document. This is required to make the file a valid text file. Without this, you may notice that the string is still not found:

```
LOCAL:10.44.5.37 jRFS jdean jdean
REMOTE:10.44.5.37 jRFS jason
```

Now we will try the `LIST REM` again.

This time you should get lots of trace messages and end up with:

```
PAGE    1

REM..........


 No Records Listed
```

If you get the same than congratulations, you've set up a remote file. If not then look at the log files on the server and the client to track down what went wrong.

Now we have setup and tested our jRFS connection, there is a much easier way to create files from the client end without the need for creating the files and stub files manually. This is via the `CREATE-FILE` command. Let's do it now (Remember to change the locations and addresses to match your machine setup and although the command seems to be on multiple lines below, it does, in fact, need to be typed in as one line):

```
CREATE-FILE NEWREM TYPE=REMOTE HOSTNAME=10.44.5.37
FILEREF=NEWREM FILENAME="/home/jason/remote/NEWREM"
DICTMOD=1 DATAMOD=1
```

If you check on the server, you should see in the /home/jason/remote directory (Or your equivalent directory) hash files called `NEWREM` and `NEWREM]D`

And in your local directory, you will find stub files `NEWREM` and `NEWREM]D`.