REPORT

Computer Science Technical Report: **95-21** **February 1995**

File Repository Transfer Protocol Version 1

Tage Stabell-Kulø

INSTITUTE OF MATHEMATICAL AND PHYSICAL SCIENCES

Department of Computer Science

University of Tromsø, N-9037 TROMSØ, Norway, Telephone +47 77 64 40 41, Telefax +47 77 64 45 80

Abstract

This document presents and specifies the protocol that interfaces clients and servers in the File Repository (FR). The FR is a software system that supports sharing of files. The protocol is modelled after SMTP and NNTP and is encoded in ASCII. No details of server implementation is visible in the protocol description, but we state our intentions at several occasions.

Contents

1 Introduction

This document describes a protocol, the File Repository Transfer Protocol (FRTP), used between clients and servers in a distributed system. The *File Repository* (FR) is part of the PASTA project.[1] The server implements a file repository, in which clients can store files. The protocol specifies how the client and server shall interact in order to satisfy the clients needs. The fact that servers are servers leads to a protocol in which the client *requests* and the server *provides* — if it has enough available resources and protection is not violated.

Sessions are always initiated by a client that has one or more requests for the server. The client and server exchange messages, possibly followed by data, until all requests has been fulfilled. At this point the protocol terminates. The result is that a session has three distinct phases:

1. Mutual identification, negotiation on data encoding, and decision on security. This phase is called "initialization phase";
2. The client presents its requests, one by one. This phase is called "work phase";
3. Termination of the protocol and release of resources on both sides. This phase is called "termination phase".

Alternatively, the protocol can be viewed as having two modes: command-mode and work-mode. Initially the protocol is in command-mode, and the only legal command is to initialize. During initialization, the protocol then enters the work phase, until initialization is done, when command-mode is again entered. The protocol can now enter work-mode in order to execute a request, or to terminate. When a command has been serviced, command-mode is again entered. All errors will take the state back to command-mode.

Internal problems in the server, such as lack of resources, shall lead to denial of service. It must not be presented as an error. Errors indicate to the client that *it* has behaved incorrectly.

The server stores files. How this is done is of no concern to the client as long as they are durable. Furthermore, the server stores the files without any interpretation. It does not filter characters, fold or limit lines, or otherwise process text that it handles. It is our intent that the server just pass the incoming data without interpretation onto the storage-engine that supports it.

2 The Frtp specification

The server specified in this document uses a stream connection (such as TCP) and SNMP-like (and NNTP-like) commands and responses. It is designed to accept connections from hosts, and to provide a simple interface to the File Repository.

The server that implements the protocol is only an interface between programs and the File Repository. It does not perform any user interaction or presentation-level functions. These "user-friendly" functions are better left at the client programs, which have a better understanding of the environment they operate in.

[1] PASTA is a joint effort between researchers at the Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Italy and Seksjon for Informatikk, Universitetet i Tromsø, Norway.

2.1 Character Codes

Commands and replies are composed of characters from the ASCII character set. When the transport service provides an 8-bit byte (octet) transmission channel, each 7-bit character is transmitted right justified in an octet with the high order bit cleared to zero. Even though the commands and responses are restricted to the ASCII character set, data can contain arbritrary bit patterns.

2.2 Security

The current version specifies no security features, but the two commands HELO and SECURE is reserved for this purpose. Security in the File Repository, and in this protocol, is discussed elsewhere.

2.3 Commands

Commands consist of a command word, which in some cases may be followed by one or more parameters. Commands with parameters must separate the parameters from each other, and from the command, by one or more space or tab characters. Command lines must be complete with all required parameters, and may not contain more that one command.

Commands are not case sensitive. That is, a command or parameter word may be upper case, lower case, or any combination of upper or lower case.

Each command line must be terminated by a `CR-LF` (Carriage Return–Line Feed) pair.

Command lines shall not exceed 1024 characters in length, counting all characters including spaces, separators, punctation, and the trailing `CR-LF` (thus there are 1022 characters maximum allowed for the command and its parameters). There is no provision for continuation command lines.

2.4 Responses

Responses are of two kinds, textual and status. In addition, data is normally send across the network on dedicated connections.

2.4.1 Text Responses

Text is sent only after a numeric status response line has been sent that indicates that data will follow. Text is sent as a series of successive lines of textual matter, each terminated with a `CR-LF` pair. A single line containing a single period (".") is sent to indicate the end of the text (i.e., the server will send a `CR-LF` pair at the end of the last line of data, a period, and another `CR-LF` pair).

If the text contained a period as the first character of the text line in the original, that first period is doubled. Therefore, the receiver must examine the first character of each line received, and for those beginning with a period, determine either that this is the end of the data or whether to collapse the double period to a single one. In particular, the LIST command (see below) will always return a line containing the filename "..", which is then presented as "...".

The intention is that text is interpreted by the client program and selectively presented to the user while status codes is used only by the client program.

2.4.2 Status Responses

These are status reports from the server and indicates the response to the last command received from the client.

Status response lines always begin with a 3 digit numeric code. This code is sufficient to distinguish all responses. Some of these may herald the subsequent transmission of data.

The first digit of the response broadly indicates the success, failure, or progress of the previous command.

1xx Information message;

2xx Command ok;

3xx Command ok so far, send the rest of it;

4xx Command was syntactically correct, but the server was unable to perform the command for some semantic or internal reason;

5xx Command not implemented, or incorrect.

The next digit indicates the function response category

x0x Connection, setup and miscellaneous messages;

x1x Naming files;

x2x Reading data from, and writing data to files;

x3x Locking;

x4x Status;

x8x Nonstandard (private implementation) extensions;

x9x Debugging output.

The exact response codes that should be expected from each command are detailed in the description of that command. In addition, below is listed a general set of response codes that may be received at any time.

Certain status responses contain parameters such as numbers and names, The number and type of such parameters is fixed for each response code to simplify interpretation of the response.

Parameters are separated from the numeric response code and from each-other by a single space. All numeric parameters are decimal and might have leading zeroes.

Time is given in seconds since 00:00:00 January 1st 1970 UTC. On initialization of a session the server will print its notion of the time. The client must examine the time to ensure that it agrees with the server on what the time is. The time is used to manage locks and is not meant for system clock synchronization.

All string parameters begin after the first separating space and end before the following terminating space or the `CR-LF` pair at the end of the line. (String parameters may not, therefore, contain spaces; this also applies to names of files). All text, if any, in the response which is not a parameter of the response must follow, and be separated from, the last parameter by a space. Also, note that the text following a response number may vary in different implementations of the server. The 3-digit numeric code should be used to determine what response was sent.

Response codes not specified in this standard may be used for any installation-specific additional commands also not specified. These should be chosen to fit the pattern of `x8x` specified above. (Note that debugging is provided for explicitly in the `x9x` response codes.) The use of unspecified response codes for standard commands is prohibited.

We have provided a response pattern `x9x` for debugging. Since much debugging output may be classed as "informative messages", we would expect, therefore, that responses `190` through `199` would be used for various debugging outputs. There is no requirement in this specification for debugging output, but if such is provided over the connected stream, it must use these response codes. If appropriate to a specific implementation, other `x9x` codes may be used for debugging. (An example might be to use for example `290` to acknowledge a remote debugging request.)

2.4.3 General responses

The following is a list of general response codes that may be sent by the FRTP server. These are not specific to any one command, but may be returned as the result of a connection, a failure, or some unusual condition.

In general, `1xx` codes may be ignored or displayed as desired; code `200` or `201` is sent upon initial connection to the FRTP server depending upon permission; code `4xx` will be sent when the server discontinues service (by operator request, for example); and `5xx` codes indicate that the command could not be performed for some unusual reason. It is our goal that internal problems in the server is signalled by means of `5xx` codes, and incorrect behavior of the client, as understood by the server, is responded upon by `4xx` codes.

100 help text

19x debug output

200 time - server ready, writing allowed

201 time - server ready, no writing allowed

400 service discontinued

401 current location removed

402 protection violation

403 program fault

500 command not recognized

501 command syntax error

502 access restriction or permission denied

The responses `200` and `201` has a required parameter `time`. It is the servers notion of what time it is. The code `502` is send when the client tries an operation it should have known it would not be allowed to perform, while `402` is send when the request is denied for some internal reason.

2.5 Transport of Data

The File Repository stores files, and there is no interpretation of their context. Therefore, data being read from, or written to files are not interpreted by the FRTP server in any way.

Furthermore, since files can contain any data, their contents is sent on dedicated connections, that is, not across the same connection as the FRTP protocol is spoken. The commands in question, READ and WRITE, has as one of their outcomes that a new connection between the server and the client is opened, and the data transferred across it. The connection is freed when data has been successfully transferred.

The XINLINE command changes the protocol so that data is encoded and transported on the same channel; see section 3.6.1 for additional details. Note, however, that XINLINE (and any other extension for that matter) is an extensions to the protocol and no server is required to support it.

2.6 Extensions

Response codes not specified in this standard may be used for any installation-specific additional commands also not specified. These should be chosen to fit the pattern of `x8x` specified above. (Note that debugging is provided for explicitly in the `x9x` response codes.) The use of un-specified response codes for standard commands is prohibited. See section 3.6 for additional details.

3 Command and Response Details

On the following pages are descriptions of each command recognized by the FRTP server and the responses which will be returned by those commands.

Each command is shown in upper case for clarity, although case is ignored in the interpretation of commands by the FRTP server. Any parameters are shown in lower case. A parameter shown in [square brackets] is optional. For example, `[GMT]` indicates that the triglyph `GMT` may present or omitted.

Every command described in this section must be implemented by all FRTP servers. There is no prohibition against additional commands being added; however, it is recommended that any such unspecified command begin with the letter `X` to avoid conflict with later revisions of this specification. Implementors are reminded that such additional commands may not redefine specified status response codes. Using additional unspecified responses for standard commands is also prohibited.

In general, the commands can be divided in five categories. The first is used to establish and terminate the connection, QUIT is the only command in this category. The second is related to the name space in which files reside. The WALK, CREATE, LIST and DELETE commands are in this category. The third category is used to manage locks. The commands in this category are LOCK, REFRESH, and RELEASE. The fourth set of commands are those to read and write data to (and from) files. To accomplish this, use the commands READ and WRITE. The last is STAT for obtaining information about files.

3.1 Termination

In order to terminate a session in an orderly fashion, theQUIT command should be used. If usage of resources are accounted for, simply closing the connection does not guarantee that the server will shut down properly. By terminating properly, the client ensures that resources are freed in the server; the server will close the connection after acknowledging the command.

3.1.1 The Quit command

QUIT

The QUIT command will terminate the connection.

3.1.2 Responses

202 connection closed — Goodbye

3.2 The name space and the Create, List, Walk and Delete commands

With these commands the client will walk through the name space and, presumably, ultimately select a file for usage. The usage of directories are for the convenience of the user, and only regular register semantics are guaranteed for operations on the name space. That is, there is no locking mechanism that can be applied on directories. The server shall maintain a notion of *current location*. This is the file which the client will operate on. Initially this is set to the top (root) of the name space.

The word "file" usually means any name in the name space, regardless of what this name represents. When it is necessary to be specific about the type, the terms "ordinary file" and "directory" will be used.

The operations on the name space has been modelled after the ones found in `9P` in Plan 9.

3.2.1 Details of the Name Space

Every directory will upon creation contain exactly one file (that is, all directories have at least one file). This file is named "`..`" (dot-dot), and is the name of the parent directory when seen from this directory. No file can be named "`.`" (dot). Relative file names are not supported, the name space will be a directed acyclic graph. [2]

Filenames are restricted to a length of 255 characters, where the characters must be taken from the ASCII character set. The allowed characters are the ASCII code from 33 to 126, except 47; that is, from '!' to '~' (i.e. all printable characters excluding SP (space) and '/').

3.2.2 The Walk command

WALK [name]

The optional parameter is the name of a file. This file will now become the new current location. If the name is a regular file, the client can use the new current location to read and write data and meta data. If the name is a directory, this directory becomes the new current location. If no name was supplied, the current location is set (back) to the root of the tree.

Walking to the file "`..`" from the root of the tree is not possible; an attempt returns an error.

3.2.3 The List command

LIST

Lists out the names in the current directory; the current location must therefore be a directory. The directory is listed, each file on a separate line, unless protection violation inhibits the server from releasing the name of some (or all) files to this particular user.

[2] In UNIX terminology: there are no links, neither symbolic nor hard.

3.2.4 The Create command

CREATE name type

The required parameters are the name of the new file and the type the file will have. If type is set to '0' (zero), 'F' or 'f' the new file will be an ordinary file. If the type is set to '1' (one), 'D' or 'd', the new file will be a directory in which new names can be stored.

Creating a file does not make it the current location.

3.2.5 The Delete command

DELETE file

The required parameter file is the name of a file. If file is a directory, it must be empty (i.e. contain no files except the file ".." which can-not be removed). A list of files in a directory can be obtained by the LIST command.

3.2.6 Responses

The commands related to maintenance of the name space can receive the following responses:

210 file selected

211 file created

212 file deleted

213 changed directory

214 directory read - text follows

215 directory created

216 directory deleted

410 no such file or directory

411 file is a directory

412 file is not a directory

413 file exists

414 directory not empty

415 walking up from root

510 invalid filename

Note that response 214 will be ensued by at least one line of text since all directories contain the directory ".." (dot-dot).

3.3 The Read and Write commands

Data transferred as a result of a read or write command is not send over the same connection as the command was issued. Instead, the response from the server includes a TCP/IP port number on which data can be read (or written). The responses `320` and `321` are sent to indicate to the client that the server has allocated, and will be listening, on a new port. The responses `220` and `221` are send by the server to inform the client that data was received, or

send, on the port. Consult the example in section 4.6 to see how the responses are combined with data although they are presented on different connection.

If a client reads or writes to a file without locking it, unpredicted results can occur due to concurrency.

3.3.1 The Read command

READ size offset

The required parameters are the offset at which reading is to start and how much data, measured in octets, is to be read. If size is set to 0 (zero) data will be read until the end of the file. No connection will be established, and no data will be send, if the file is empty.

3.3.2 The Write command

WRITE offset size

The required parameters are the offset at which the new data is to be stored, and the amount of data, measured in octets, that will follow. The server will not accept to write 0 (zero) octets to a file. Writing beyond the current end of the file (not appending but leaving a "hole" in the file) might, or might not, be supported by the server.

3.3.3 Responses

220 data send ok

221 data received ok

320 size port read ok. *The server will write* `size` *octets of data to port number* `port`

320 port writing ok. `port` *is the port number onto which the data can be written by the client*

420 can-not write a directory

421 beyond EOF

422 file to big

423 out of resources

520 will not do anything with 0 bytes

The response `422` is to be returned as a response to a WRITE command if the file is to big for the server to handle. The response `423` shall be returned if the server runs out of resources while the client is writing data, i.e. after response `320` but instead of response `221`.

3.4 The Lock, Refresh, and Release commands

These commands are used to obtain locks to enforce concurrency-control on files. If a client deem that several files belong together, it must lock them one by one; there is no provision for locking more than one file (in each operation). The exact semantics of issues related to concurrency are beyond the scope of the protocol.

3.4.1 The Lock command

LOCK time [string]

The required parameter is the time for which the lock is wanted. If the time is 0 (zero) the lock is requested for the maximum duration possible; the maximum time is server dependent. The optional parameter string is an informal description of possible value to other *users* wanting to access the file.

3.4.2 The Refresh command

REFRESH time [string]

The required parameter is the time for which the lock is wanted. If the time is 0 (zero) the lock is requested for the maximum duration possible; the maximum time is server dependent. The optional parameter string is an informal description of possible value to other *users* wanting to access the file. If no string is provided, the current string, if any, is kept.

3.4.3 The Release command

RELEASE

The command will release a lock previously obtained with LOCK (and possibly prolonged with REFRESH). The lock in question is the one (if any) placed on the current location.

3.4.4 Responses

The LOCK command, and its semantics, is expected to change in future releases. Backwards compatibility with this description will not be enforced.

230 lock ok

231 refresh granted

232 release ok

430 file already locked

432 no lock on file

433 can-not set lock in past

434 can-not set lock so far in future

530 can-not lock directory

In order to know how long the lock actually was granted the client must use the STAT command.

3.5 Attributes and the Stat command

In order to read meta-data the client must issue a STAT command. All servers must maintain the standard attributes as described below. Any server might, in addition, maintain non-standard attributes. Non-standard attributes may not have the same name as, or be postfix of, any standard attribute, and no non-standard attribute can change the semantic of any standard attribute.

For example, a non-standard attribute could be that the encoding of 8-bit characters are in accordance with ISO-8859.1. Any client can read the file without this knowledge, but the front-end showing the text to a user would use it to ensure that the national characters were presented correctly.

The standard attributes are:

1. `size`
2. `owner`
3. `locked`
4. `time`
5. `string`

The `size` is given in octets, and therefore not necessarily characters; for example, if an encoding is used that utilizes more than one octet per character these numbers will differ. The `owner` is the *name* of the owner, as known to the server. The `locked` attribute is encoded with 0 (zero) to represent false (not locked) and 1 (one) to represent true (locked). If the file is locked, the lock will expire at time `time`. The `string` is the string supplied with LOCK or REFRESH command, if given. If not present this attribute shall be represented by an empty line (two CR-LF pairs). The standard attributes shall be presented in the order shown above.

If a server supports non-standard attributes it shall present them *after* it has presented the standard ones. It is our intention that a client that want to obtain maximum portability (that is, be able to use any server) shall read the first five lines of text following a successful STAT command, and then discard any data up to the end of the response.

3.5.1 The Stat command

STAT [attribute]

The command will return information about the file. The optional parameter attribute is an specific attribute. If no parameter is given, all attributes are shown.

3.5.2 Return values

240 stat ok - data follows

440 no such attribute

Note that, since an implementation of the FRTP server can have more attributes the ones required by this specification, asking for a non-existing attribute is not an error, and shall not result in a `54x` response. Furthermore, note that there is no need for a command to set the standard attributes as they are changed as the side effect of other commands. Non-standard attributed must be set by additional commands.

3.6 Extensions

In addition to the commands and responses described, extensions may be provided by any server. They can be new commands, or additional response codes to existing commands. Implementors are asked to note the restrictions described in section 2.6 (on page 5) and as part of the STAT command in section 3.5.1.

The status of extension, enjoyed by the command XINLINE in this version of FRTP, should be regarded as temporary. If the command is widely implemented and found useful, it will be incorporated as a required command in some future description of the protocol. If not, description of its usage will be removed.

We regard this as the preferred method of introducing new functionality into the protocol.

3.6.1 The Xinline command

In those cases where the client can-not establish a new connection to the server in order to read or write data—as described in the sections 3.3.1 and 3.3.2 respectively—it can be embedded on the command channel. The data must then be encoded as follows.

The encoded data consists of a number of lines, each at most 64 characters long (including the trailing `CR-LF` pair). These lines contains a character count, followed by encoded characters, followed by a `CR-LF` pair. The character count is a single printable character, represents as an integer, the number of bytes the rest of the line represents. Such integers will always be in the range from 0 to 63. The value can be determined by subtracting the character space (value 32) from the character. If, for example, the character count is represented with 'B', the number of the character is 65. Since $65 - 32 = 33$, the line represent 33 bytes.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a space (value 32) to make the character printable. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Garbage will be included to make the character count a multiple of 4. The encoded lines are terminated by a line with a count of zero. This line thus consists of one ASCII space; the ASCII space yields the character count 0 (zero). As usual, any lines are followed by a line with "." (dot) followed by a `CR-LF` pair.

3.6.2 Return codes

280 xinline turned on

281 xinline turned off

481 short line

4 Sample Conversations

These are samples of the conversations that might be expected with the server in hypothetical sessions. The notation `C:` indicates commands sent to the server from the client program, and `S:` indicate responses received from the server by the client. All commands are shown capitalized for clarity.

4.1 Example 1: Access denied

The means and policies on which the initial decision on access is based is server dependent and not part of the protocol. Typical restrictions will be that only local machines can access the server or that only a subset of machines.

```
S:      (listens at TCP port 24163)
C:      (requests connection on TCP port 24163)
```

```
S:      502 Access to the Frtp server denied - goodbye.
(Server closes connection)
```

4.2 Example 2: Read-Only access

The client opens the connection. Since no writing was allowed, the client closes the connection.

```
S:      (listens at TCP port 24163)
C:      (requests connection on TCP port 24163)
S:      201 769259492 Welcome to server foo - no writing allowed
C:      QUIT
S:      202 Goodbye.
(Server closes connection)
```

4.3 Example 3: Normal access

```
S:      (listens at TCP port 24163)
C:      (requests connection on TCP port 24163)
S:      200 769259492 Welcome to server foo - writing allowed
```

Exchange of commands and responses can now take place.

4.4 Example 4: Walk to a directory

The client wishes to list a directory, but has a spelling error in the first try. Notice that names of files, as opposed to commands, are case sensitive.

```
S:      (listens at TCP port 24163)
C:      (requests connection on TCP port 24163)
S:      200 769259492 Welcome to server foo - writing allowed
C:      WALK bar
S:      410 no such file or directory
C:      WALK Bar
S:      213 changed directory
C:      LIST
S:      214 directory read - names follows
S:      protocol.tex
(more lines of text here)
S:      .
```

4.5 Example 5: Obtaining information about a file

In this example we ignore the initial connection and assume that a file has already been chosen as current location.

```
C:      STAT
S:      240 STAT ok --- data follows
S:      29786
S:      Tage Stabell-Kulø
```

```
S:      1
S:      1
S:      769259492
S:      Jeg er på hytta frem til mandag
S:      server dslab3
S:      .
```

Notice that this server maintains one non-standard attribute (server). The semantic of this attribute must be known to any client inclined to use (read or write) it.

4.6 Example 6: Read and write data from (and to) a file

In this example we ignore the initial connection and assume that a directory has already been chosen as current location. The client first tries to read a non-existing file before reading the correct one. It then writes new data on the file. The write fails, presumably because the server run out of resources.

```
C:      WALK foo
S:      410 no such file or directory
C:      WALK Foo
S:      210 file selected
C:      READ 0 0
S:      320 2374 11948 read ok
(The client can now connect to port 11948 and read 2.374
bytes)
S:      220 data send ok
C:      WRITE 0 18366
S:      321 27578 write ok.
(The client can connect to port 27578.  The server will
expect 18.366 octets of data)
S:      423 out of resources
```

The response `423` was send instead of the expected `221`. The client should in this situation check whether the server was able to store any of the data; it will exhibit best effort.

4.7 Summary of Commands and Responses

This section shows all commands all servers must recognize and the minimum set of responses. Implementors are reminded that the set of commands and responses might be bigger, as described in section 3.

4.7.1 Commands

The basic commands, that must be implemented by all servers, are:

CREATE

DELETE

LIST

LOCK

QUIT

READ

REFRESH

RELEASE

STAT

WALK

WRITE

In addition, the two commands

HELO

SECURE

is reserved for future use.

4.7.2 Responses

The following responses are specified by this document. No server must use these codes for any other purpose, and no server must use other codes to signal some event or situation covered by these codes.

100 help text

190–199 debug output

200 server ready, writing allowed

201 server ready, no writing allowed

202 closing connection - goodbye

210 file selected

211 file created

212 file deleted

213 changed directory

214 directory read - text follows

215 directory created

216 directory deleted

220 data send ok

221 data received ok

230 lock ok

231 refresh ok

232 release ok

240 stat ok - text follows

320 size port read ok

321 port write ok

400 service discontinued

401 protection violation

402 current location removed

403 program fault

410 no such file or directory

411 file is a directory

412 file is not a directory

413 file exists

414 directory not empty

415 walking up from root

420 can-not write to a directory

421 beyond EOF

422 file to big

423 out of resources

430 file already locked

432 no lock on file

433 can-not set lock in past

434 can-not set lock so far in future

440 no such attribute

500 command not recognized

501 command syntax error

502 access restriction or permission denied

510 invalid filename

520 will not do anything with 0 bytes

530 can-not lock directory

The extension XINLINE utilizes the following return codes.

280 xinline turned on

281 xinline turned off

481 short line

5 Acknowledgements

The author is supported by a scholarship from the University of Tromsø, Norway. Our partners in PASTA has provided valuable input to numerous drafts of this document.

Funding was received from "The Radical Initiative" of CaberNet, ESPRIT BRA 6361; it is gratefully acknowledged.