Network Working Group
Request for Comments: 520
NIC: 16819
J. Day
Center for Advanced Computation
25 June 1973

A Proposed File Access Protocol Specification

Attached is a proposal for the File Access Protocol. FAP is an extension to FTP. I believe the specification is fairly general and should provide a good jumping-off place. I hope the protocol is specified in such a way as to fit with idiosyncrasies of most systems. If the protocol would cause an inordinate amount of burden on your system for one reason or another I would like to hear about it.

At some later date when the difficulties of implementation are better known, I would like to see several levels of implementation specified and implementation be done in terms of those levels.

From rumors I have heard I believe this will also allow creation and transfer of what TENEX calls "holey" files. But, I am not sure of all of the implications of that, or what would happen (or should happen) when a "holey" file is moved to a site that doesn't really have such a thing, per se. Comments from the TENEX crowd would be appreciated.

I think some further work could be done to make FAP easier for record oriented systems. This would probably require an extra command or parameter to specify all operations are in terms of records. Comments are invited.

In the long run though, I would like to see FAP thrown away. The commands as they are described merely add a finer structure to the present RETR, STOR, and APPE without much additional overhead. The sequence:

```
OPEN R FOO.BAR CRLF
      READ ALL CRLF
      CLOS CRLF
```

is equivalent to RETR FOO.BAR CRLF. FAP could be merged with FTP to give a much richer, coherent whole.

In writing this document, I ran into the deficiency of reply codes for protocols. Three digits is no where near enough. I would like to suggest that as another interim solution we go to a five digit

reply with two for specific categories (such as Primary access, FTP results, etc.) and two for specific results. In the meantime, the NWG should begin considering a general scheme for reply codes -- one that doesn't need revising every two years.

Comments, complaints, etc. are welcomed. I may be reached through network mail at ISI (DAY) or Multics (DAY Cnet) or by phone at the University of Illinois (217) 333-6544.

A Proposed File Access Protocol Specification

John Day

6/7/73

I. INTRODUCTION

The purpose of the File Access Protocol is to provide a method for processes to access non-local files in either a sequential or non-sequential manner. Unlike the proposed Mail Protocol, FAP is an extension of FTP and not a subsystem. In general FAP is compatible with the rest of FTP. Those modifications which are necessary are specified below.

The intent of this protocol is to allow processes to specify to the remote file system where in the file they wish the next operation to start and how much data to move. Thus only the part of a file necessary for a process' computation need be transferred, rather than the entire file. Thus transmission times and storage requirements may be held down. In short, the rationale for a File Access Protocol on the network is the same as the rationale for "random-accessed" files in a standard operating system.

The file Access Protocol uses the connection model, data representations, and transmission methods of the File transfer Protocol. All data transmissions in FAP are handled according to the description in FTP Section III.C with the following modifications. In Stream mode, the minimum byte size is increased to 4 bits. Another control code (value 4) is used to indicate "end of transmission". An combination of EOT, EOR, or EOF may be indicated by the proper control code. With this method it is not necessary to close the connection after each access; a practice not highly recommended. In Block mode, bit 5 of the descriptor field of the header is set noting that this block is the end of transmission. In addition to this, FAP uses a File Pointer (FP). The file pointer

points into the file and is the point at which the next FAP read or write will commence. The file pointer is a general mechanism for addressing a file and should be flexible enough to handle both stream and record oriented systems.

II. PROBLEMS OF IMPLEMENTATION

As usual, not all systems will be able to implement this protocol in its full generality. The approach that should be taken is that no host should be required to provide for network users (in the name of complete protocol implementation) service it does not provide its local users.

Some systems allow "random" access to some kinds of files on its system and not to others. In this case, this should be their implementation, i.e., not all operations are valid for all kinds of files.

Some systems cannot move the byte pointer backwards without opening and closing the file. They should not be required to do this (although they may if they wish), but they should allow "spacing" down a file some distance before starting a transfer.

Some systems may not allow read and write access to be available without closing and reopening the file. Systems should not be required to do both.

In general, the rules of implementation are:

1) If a system normally allows that particular kind of access to that particular file then it should be allowed; if not, the system should not be forced to implement it. (In many cases, the legality cannot be known until the operation is attempted; i.e., it cannot be told of the first two cases above if they are legal when the file is opened but only on the read or write which violates the implementation restrictions).

2) A system should not try to simulate a facility if the simulation has side effects. For example, if simulating the capability of moving the byte pointer to the desired position has some side effects, then the simulation should be left to the process accessing the file.

3) All implementors should make known the capabilities of their implementations via NIC documents.

III. FILE ACCESS PROTOCOL

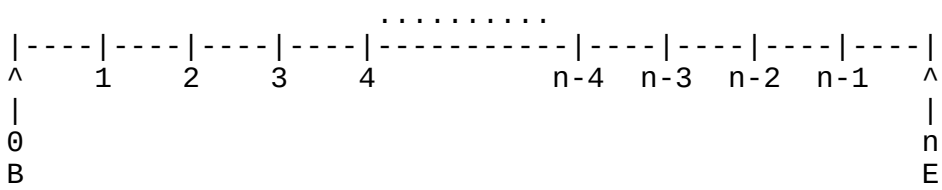
The FAP extension to FTP includes 6 new commands and the file pointer. Any implementation requires the file pointer and all six commands. But, as described above, it is not necessary to implement the commands in their full generality.

III.1 THE FILE POINTER

The file pointer represents an index or address within the file. The units by which the index is measured, is "logical byte size" and does not include any bytes related to transmission or structure. In particular, for transmission mode Stream and structure Record, the EOR and EOF markers are not counted. Local transformations on data must be taken into account. For example, Multics stores CRLF as NL. In this case, NL counts as two ASCII bytes since it was transmitted to or will be sent from Multics as CRLF. If transmission Mode is Image then the logical byte size is taken as the transmission byte size. There are two commands which operate on the file pointer: 1) SETP to move the pointer and 2) GETP to find out where it is at. These are described below in more detail.

The file pointer may take on three classes of values. All may be mapped to some decimal number. The value B represents the beginning of the file (Byte 0). The value E represents the end of the file (or Byte n for a file n bytes long). The byte pointer may also take on any value between 0 and n.

A file of n bytes



If a file is stored under set of parameters (TYPE, etc.) and operations are attempted on it under different parameters, the server does not guarantee that the information will be valid.

III.2 COMMANDS

III.2.1 OPEN <direction> <pathname>

This command instructs the server to "open" the file <pathname> for access in the direction specified. The directions are read, R write, W; or both, B. A read direction implies that the data connection is

from server to user; write, from user to server; and both implies connections each ways. Functionally, this command corresponds to RETR or STOR. Therefore, all the FTP parameter commands (TYPE, MODE, etc.) must be sent before the file is opened. If the direction is write (W) and the file specified by the pathname does not exist, there is an implied create with the open. The success of this create, is, of course, dependent on local access privileges and possibly whether or not an ALL command was sent. If applicable, the file created should be of the most general kind of file on which "random" access is allowed. (This is to allow the largest degree of compatibility with operations that may follow). This should be ignored if some site specific command has already specified the kind of file. This command identifies the file on which subsequent operations are to be performed. After the file is opened, the file pointer is at B and any of the other five FAP commands may be sent. It is acknowledged that some systems cannot open a file for access in both directions; an error reply 402 should be sent for this response.

Replies

258	451	500	504	550
402	454	501	505	
434	455	502	506	
4550	457	503	507	

III.2.2 SETP <argument>

This command causes the file pointer to be set to the number specified in the argument. This value will be the ordinal number of the starting position of the next operation. (Byte 0 is the first byte in the file). The argument may take on two other values besides <decimal number> : B, for BEGIN, which sets the file pointer at the beginning of a file (i.e. 0) and E, for END, which sets the file pointer to the last byte in the file. Two error conditions are possible. If the argument specifies an illegal change of file pointer (such as moving it backwards on some systems), then the error reply 402 should be sent. If the argument attempts to move the file pointer off the end of the file, then the EOF: <byte number> reply should be sent with the address of the end of the file (E), and the file pointer left at E.

Replies

258
402
480

III.2.3 GETP

This command requests the server to return the value of the file pointer as a decimal number.

Reply

483
504

III.2.4 READ <arg>

This command instructs the server to move as many bytes as specified (of size logical byte size) from the server to the user. The values the argument may take on are <decimal number> and ALL. ALL is interpreted as all data from the present position of the file pointer to the end-of-file. If a read requests more bytes than in the file, the number of bytes from the present position to the end of file should be transferred and an EOF: <byte number> response returned noting the position of the end of file. If the file is Record structured and a READ requests more bytes than in the record, then the number of bytes in the record from the file pointer are moved and the EOR: <byte number> reply is sent noting the end of record. The action of a READ leaves the file pointer at the position before the read plus the number of bytes moved, (i.e., updated). The EOF condition leaves it at E.

Replies
----
258 480
402 481
450 482
452 500-507
455

III.2.5 WRITe <arg>

This command instructs the server to accept as many bytes as specified from the user. The result updates the value of the file pointer. The values the argument may take on are <decimal number> or ALL. ALL is interpreted as all data from the present position of the byte pointer to the end-of-file (or beyond). Associated with the write is an implied "append", if necessary previous information has been sent (such as allocation) and if the file's access privilege allow the append. If a write specifies more bytes than there are between the file pointer and the end-of-file, and expansion is not allowed, no data is sent and the file pointer is not moved. An error is returned specifying the byte position of the EOF. If the file is

Record structured and a WRIT attempts to move more bytes than there are in the record, the file pointer is not moved and the EOR: <byte number> reply is sent noting the end of record.

Replies
----
258 480
402 481
450 482
452 500-507
453

III.2.6 CLOS

This command instructs the server to "close" the presently open file, if any. The receipt of a CLOS without an open file is not an error. The effect is to notify the server that further operations are not directed at the file which is presently open. If an open is received by the server and it has a file open, it should close the open file and open the new one.

Reply

258

IV. SUMMARY

IV.1 SYNTAX

OPEN <direction> <pathname> CRLF
CLOS CRLF
SETP <byte pointer arg> CRLF
GETP CRLF
READ <transfer argument> CRLF
WRIT <transfer argument> CRLF

<direction>::= R|W|B

<byte pointer argument>::= B|E|<decimal number>

<transfer argument>::=ALL|<decimal number>

<byte number>::= <decimal number>

IV.2 REPLIES USED BY FAP

258	Operation successful
402	Command not implemented for requested value or action
433	Cannot transfer files w/o valid account. Enter account & resend command.
450	FTP: file not found
451	FTP: file access denied
452	FTP: file transfer incomplete, data connection closed.
453	FTP: file transfer incomplete, insufficient storage space.
454	FTP: cannot connect to your data socket
455	FTP: file system error not covered by other reply codes.
457	FTP: transfer parameters in error.
480	EOR: <byte number>
481	EOF: <byte number>
482	File not open for operation
483	FP: <byte pointer>
500	Last command line completely unrecognized.
501	Syntax of last command is incorrect.
502	Last command invalid (ignored), illegal parameter combination.
504	Last command invalid, action not possible at this time.
505	Last command conflicts illegally with previous command(s).
506	Last command not implemented by the server.
507	Catchall error reply.
550	Bad pathname specification (e.g., syntax error).