Chaosnet FILE Protocol

September 1981

Table of Contents

1. Chaosnet FILE Protocol

1.1 Introduction

The FILE protocol allows one Chaosnet host (the *user*) to make use of a file system on another host (the *server*). The user can read, write, rename, and delete files on the server's file system. The main purpose of the protocol is to support personal workstation computers, such as Lisp Machines, that want to share a file system. It can also be used more generally for file transfer between any two Chaosnet hosts.

This document describes the protocol by which a user host and a server host communicate. It describes the requirements for both users and servers. Ordinary users need not know anything about this protocol. A *user end* program in the user host takes commands from users or programs in whatever format is convenient, and translates those commands into the form of this internal protocol. A *server end* program in the server host operates automatically (usually as a "background" or "daemon" process) and performs the commands given by the user end. This document is directed towards systems programmers writing and maintaining user end and server end software. User and server ends for this protocol already exist for several computers and operating systems; in particular, both exist for the Lisp Machine.

It is assumed that the reader of this document is familiar with the basic Chaosnet protocol, and understands the meanings of "connections", "packets", "packet opcodes", "connect names", "opening" and "closing" connections, and so on. These concepts are explained in the *Chaosnet* document.

1.2 Concepts

The FILE protocol uses Chaosnet connections in two different ways: as *control* connections and as *data* connections. The user sends commands to the server over one direction of the control connection; the server sends replies back to the user over the other direction. Data connections are used for transmitting the contents of files between the two hosts, in either direction.

To start using the FILE protocol, the user host opens a connection to the server host, using connect name "FILE". This connection is the control connection. Data connections are opened later, in response to commands transacted over the control connection.

Each data connection, like any Chaosnet connection, is bi-directional. The two directions are treated as completely independent uni-directional channels by the FILE protocol. So, two unrelated file transfers can be happening on the same data connection at the same time (in opposite directions). Data connections are re-usable; they can be used for many file transfers, one after the other.

The number of data connections that can be associated with a single control connection is limited by the server end program (often because of operating system restrictions). The limit is 3 for ITS servers and 8 for TOPS-20 servers. If you want more data connections than that, you can open a new control connection, which will create a new server end program, and use this new control connection to open and control new data connections.

1.3 The Control Connection

The user sends a sequence of *commands* to the server over the control connection. Each command is contained in a single packet, with opcode 200 (octal). For each command that it receives from the user end, the server end sends back one *response* over the control connection. Each response is contained in a single packet with opcode 200 (octal).

The only other kind of packet that the user should send on the control connection is an EOF packet (opcode 014, octal), when it wants to disconnect the control connection. When the server receives an EOF packet on the control connection, it will abort any data transfers in progress, close all data connections associated with this control connection, and close the control connection.

The only other kinds of packet that the server should send on the control connection are *asynchronous mark* packets (described in section 1.7, page 13) and *notification packets* (described in section 1.8, page 15).

The contents of a command or a response packet is interpreted as a character string, in the Lisp Machine character set (see *The Lisp Machine Manual*). Fields are generally separated either by space characters (040 octal) which we will call <sp>, or newline (return) characters (215 octal), which we will call <nl>. The formats of commands and responses is as follows:

```
command:    tid <sp> [ fh ] <sp> cmd arg...
response:   tid <sp> [ fh ] <sp> cmd result...
```

Square brackets ("[]") indicate that what they enclose is optional. Ellipses ("...") indicate that the field that preceeds them may be repeated any number of times, possibly no times at all.

`tid` Each command and each response has a *transaction identifier*, or *tid* for short. A transaction identifier is a string of between one and five printing characters made up arbitrarily by the user end. When the server end sends a response to a command, the response will have the same transaction identifier as the matching command. The purpose of the transaction identifier is to allow the user end to match up responses with commands, because the user end is allowed to send many commands before receiving the responses for any of them.

`fh` Each direction of each data connection is identified by a *file handle*, or *fh* for short. (Every data connection has two file handles associated with it, one for the user-to-server direction and one for the server-to-user direction.) Like transaction identifiers, file handles are strings of between one and five printing characters made up arbitrarily by the user end. In some commands, the file handle is required, and in others it may be omitted. If the file handle is omitted, note that both the space that would preceed it and the space that would follow it remain.

`cmd` Each command has a *command name*, which is a string of characters. All of the command names and their meanings are listed in section 1.4, page 3. Command names must always be in upper case.

`args`
`results` The meaning of the the "args" and "results" field vary depending on the command, and will be described separately for each command.

A response will always have the same "tid" and "fh" fields as the command to which it is responding. If the response is a "success" response (the server correctly performed the requested action), the "cmd" field will be the same, as well. If the response is an "error" response, the "cmd" field will be the word "ERROR" instead. Error responses and error handling are described

further in section 1.7, page 13.

1.4 Commands

This section describes all of the commands of the FILE protocol. For each command, we give the name, the format of the "args" in the command, and the format of the "results" in the response. The following conventions are used.

<sp> means the space character (040 octal).

<nl> means the newline or return character (215 octal).

<none> means an empty field in a packet.

<num> means a sequence of characters that are all digits (codes 060 through 071 octal, i.e., "0" through "9"). Such numbers are always interpreted in base 10 (decimal). Numbers in commands are unsigned, but may have leading zeroes. Numbers in responses may sometimes have leading minus signs (e.g., the "-1" of the version field of an OPEN response).

The syntax of the arguments and results for some commands is too long to be presented on a single line in this document; note that there is no newline in the packet when we continue the syntax description on a second line. That is, all characters in every command or response are written down explicitly.

Also note that the end of a command or response is not marked by a special character, because the length field of the Chaosnet packet determines where the data ends.

DATA-CONNECTION

```
args = <sp> ifh <sp> ofh
results = <none>
```

Open a new data connection. The "ifh" and "ofh" fields are the file handles to be associated with the READ (server-to-user) and WRITE (user-to-server) directions of the connection, respectively. Each file handle should be distinct from the other and from all other file handles of other data connections associated with this control connection.

When the server receives this command, it opens a new Chaosnet connection to the user, using "ofh" as the contact name. Therefore, the user should listen for this connect name after sending the command. The server may send the response to the command before the new data connection becomes open, but, of course, the user cannot begin to use that new data connection until it is open. (The "fh" field must not be present.)

UNDATA-CONNECTION

```
args = <none>
results = <none>
```

Close the data connection whose input or output file handle appears in the "fh" field of the command. The UNDATA-CONNECTION command implies a CLOSE command on each file handle of the data connection for which there is a file transfer in progress. Having many pairs of data connections open can degrade the performance of some servers; users not keep extra connections open if they are not being used. (The "fh" field is must be present.)

OPEN

```
args = [ options ] <nl> filename <nl>
options = <sp> option1 <sp> option2 ... <sp> optionN
results = <sp> version <sp> date <sp> len <sp> qfasl
          <nl> realname <nl>
```

Open a file for reading or writing, and start a file transfer. The "fh" field specifies the data connection to use and the direction of the file transfer. If an input file handle is given in the "fh" field of the command, then the file is to be opened for reading. If an output file handle is given, then the file is to be opened for writing. A file handle must be supplied, unless a PROBE is being done (see below).

The "filename" field is the filename to use, expressed in the syntax of the server host. It may contain spaces or other punctuation characters, but may not contain an <nl>.

The "options" field of the OPEN command consists of a number of options, always in upper case, each preceded by a space (to separate them from the "OPEN" and from each other). Some options apply only to BINARY transfers, and others only to CHARACTER transfers. Here is a list of the options and their meanings.

READ — Open the file for reading. This is redundant, and is simply checked for consistency (i.e., the user must have specified an input file handle).

WRITE — Analogous to READ.

PROBE — Rather than opening for READ or WRITE, simply check for the existence of the file; do not really open the file and do not start any file transfer. This option is implied if no file handle is given in the OPEN command. The user end should not supply a file handle when doing a PROBE. The response will contain the same information as a response to OPEN'ing the file for reading. BINARY vs. CHARACTER makes a difference in a PROBE, as described in the discussion of the response to OPEN.

CHARACTER — Open the connection for the transfer of character data as opposed to binary data. This affects the mode in which the server end will open a file, as well as whether or not character set translations are performed (see section 1.5, page 8). CHARACTER is the default, so it need never be specified. The check for QFASL files is not performed (see below).

BINARY — Open the connection for the transfer of binary data as opposed to character data. A check is made to see whether the file is a QFASL files when opening for READ (the result of the test is supplied in the response to the OPEN; see below). The default byte size is 16 bits (see the discussion of byte sizes below).

The following options pertain to BINARY transfers only:

BYTE-SIZE <sp> <num> — Use bytes with <num> bits each. The interpretation is explained below.

The following options pertain to CHARACTER transfers only:

RAW — Suppress character set translation (see section 1.5, page 8).

SUPER-IMAGE — Suppress use of 177 for quoting (this pertains to character set translation; see section 1.5, page 8).

The following options are specific to particular foreign hosts, and they should should only be presented to servers on the specified kind of host. Servers on other hosts will return an "unknown open option" error if presented with one of these.

TEMPORARY *(TOPS-20 server only)*
: Use GJ%TMP in the GTJFN. This is useful mainly when writing files, and indicates that TOPS-20 is to treat the file as temporary. See TOPS-20 documentation for more about the implications of this option.

DELETED *(TOPS-20 server only)*
: Set GJ%DEL in the GTJFN. This is useful mainly for READ'ing or in PROBE's, and indicates that deleted files should be considered when looking for a file. See TOPS-20 documentation for information about this.

Details for ITS servers:
: A CHARACTER OPEN will be in "unit ASCII" mode, and a BINARY OPEN will be in "unit image" mode. An OPEN for writing always opens a file called "_LSPM_ OUTPUT" in the specified directory, and the server does a RENMWO just before closing the file, like all ITS programs. The server takes care of the traditional problems associated with the last word of text files, involving ASCII Control-C characters and the like; you will never see the extra characters.

Details for TOPS-20 servers:
: When OPEN'ing for READ or PROBE, GJ%OLD is always set. When OPEN'ing for WRITE, GJ%FOU and GJ%NEW will be set. These are independent of the setting of GJ%DEL and GJ%TMP (see the TEMPORARY and DELETED options, above).

  CHARACTER mode does a 7-bit OPENF, normal mode, with line number checking off when reading. BINARY mode does a 36-bit mode OPENF—the server handles the packing and unpacking of bytes within words.

The result of an OPEN command has the following format:

```
results = <sp> version <sp> date <sp> len <sp> qfasl
          <nl> truename <nl>
```

version
: A <num>, expressing the version number of the file. On ITS servers, this will be the numeric second name, or the leading numeric characters of the second name. If the name has no numerics, the field will be "-1". On TOPS-20 this will be the generation number of the file.

date
: The creation date of the file, expressed in the form "mm/dd/yy hh:mm:ss". (That is a real <sp> between the date and the time.)

len
: A <num>, expressing the length of the file. For a CHARACTER transfer, "len" is the number of characters in the file, in the server's character set. For a BINARY transfer, "len" is the number of bytes in the file. The ITS and TOPS-20 file servers may return inexact answers, due to lack of generality in their file systems. This will always be 0 when writing, since the file does not exist yet.

qfasl
: "T" if we are doing a BINARY READ and the file is in QFASL format. "NIL" otherwise. (The server generally figures this out by seeing if the first word of the file is "QFASL" in sixbit.)

truename — The full name of the file, e.g., what RFNAME (on ITS) or JFNS (on TOPS-20) will tell you, when asking for all the usual fields (device, directory, first name, second name, generation number). This can be different from the filename you specified because of version numbers, links, etc.

CLOSE

```
args = <none>
results = <sp> version <sp> date <sp> len
          <nl> truename <nl>
```

Close the file transfer specified by the file handle. A file handle must be given, and a transfer must be active on that handle. For a READ connection, the server will transmit a synchronous mark; for a WRITE connection, the user must send a synchronous mark. See section 1.6.3, page 11 for complete details.

The results are mostly the same as for OPEN, except that the "len" number gives the length of the file even if the transfer is for WRITE, since, at this point, the file exists.

FILEPOS

```
args = <sp> <num>
results = <none>
```

Reset the file access pointer to character or byte <num> of the file. The file handle must be supplied, and it must be an input file handle. This sends a synchronous mark before starting to send more data (see section 1.6.2, page 11).

DELETE

```
args = <none>  or  <sp> filename <nl>
results = <none>
```

Delete a file. If a file handle is supplied, the file being read or written on the specified DATA connection will be deleted after it is closed (regardless of direction), and there should be no argument. If no file handle is given, then the optional argument must be present, and it specifies a file to be deleted immediately.

RENAME

```
args = <nl> filename1 [ <nl> filename2 ] <nl>
results = <none>
```

If a file handle is given, only the first filename should be present, and the file will be renamed to filename1 sometime. On ITS, a RENMWO call is done immediately if the file being read; if the file is being written, the RENMWO done before the file is closed will rename to the new name. On TOPS-20, a GTJFN is done to filename1, and when the file is closed, the CLOSF is done with CO%NRJ set, and then an RNAMF is done to the previously obtained jfn.

If no file handle is supplied, then both filenames must be present, and the effect is to rename filename1 to be filename2. On ITS renaming can only be done within a directory. The ITS server simply ignores the device and directory parts of filename2, so be careful. On TOPS-20, renaming from one structure to another will fail.

CONTINUE

```
args = <none>
results = <none>
```

The file handle must be supplied, and the indicated channel must be in the asynchronously marked state (that is, an error has occurred). If the error might be recoverable, then CONTINUE will try to resume from it. If the error is non-recoverable, the server will return an error response. See the discussion of errors in section 1.7, page 13.

SET-BYTE-SIZE

```
args = <sp> nbs <sp> [ npos ]
results = <none>
```

Change the byte size of the connection to nbs. This is a special command that only works for PDP-10 (ITS and TOPS-20) file servers. The file handle must be supplied, and it must refer to a file open in BINARY READ mode. Both nbs and npos are <num>'s. "nbs" is the new byte size, and must be in the range 1 to 16. "npos" is the new position in the file in terms of the *old* byte size. The npos part may be omitted for output files, and is ignored if present (but the second <sp> *must* be there). If npos is omitted for input files, it defaults to 0. The server sends a synchronous mark; see section 1.6.2, page 11.

LOGIN

```
args = <nl> [ userid [ <nl> password [ <nl> account ]]]
results = <sp> uname <sp> hsname <nl> persname <nl>
```

Log into the foreign host. You must log in before any OPEN commands will be accepted. Logging in to ITS is simple: you just give the desired UNAME as the userid argument. On TOPS-20, the userid must correspond to a real directory (RCUSR is used), and the password and account are passed to a LOGIN JSYS. If the first character in the password is an asterisk, capabilities will be enabled.

Uname is the user's login ID (the same as userid unless an abbreviation has been expanded). Hsname is the user's "home" directory, where his personal files are kept. Persname is the user's personal name, e.g. "Leslie Q. Hacker".

If userid is omitted, i.e. args consists of just a <nl>, this command means to log out (the same as sending an EOF packet on the control connection).

DIRECTORY

```
args = option... <nl> filename
results = <none>
```

The DIRECTORY command is like the OPEN command, except that instead of opening an actual file, when you read from the data connection you get information about the contents of a directory in the file system. The file handle must be supplied, and must be an input channel on which no file transfer is active.

The data that the user is given consists of a sequence of records. The first record gives properties of the directory as a whole. Each following record describes one file, and gives its properties. There is one record for each file in the directory that matches the "filename" argument, which may contain wildcard specifications in whatever syntax the server's file system expects them.

Each record consists of a filename, a <nl>, a sequence of property lines separated by <nl>'s, and an extra <nl> to mark the end of the record. The filename for the first record is empty. Each property line consists of the name of the property, optionally followed by a <sp> and the value of the property. If no value is present, the property is a yes/no property and the value is yes.

The properties are not documented here; consult *The Lisp Machine Manual* for a list of all the properties.

The only option allowed is DELETED, which says to include deleted files (TOPS20) or "*" files (ITS) in the listing.

COMPLETE

```
args = options <nl> default-filename <nl> string
results = status <nl> new-string <nl>
```

This does filename completion. "String" is a filename typed in by the user and "default-filename" is the default name against which it is being typed. The filename is completed according to the files present in the file system and the possibly-expanded string is returned.

The allowed options are DELETED, READ, WRITE, OLD, NEW-OK. DELETED means not to ignore deleted files; this applies to Tenex and TOPS-20 only. READ means the file is going to be read (this is the default). WRITE means the file is going to be written, and affects version number defaulting. OLD means an existent file is required (this is the default). NEW-OK means that it is permissible for the string to complete to the name of a file that does not exist.

The returned status is NIL, OLD, or NEW. NIL means that an error occured, OLD means that the string completed to the name of an existent file, and NEW means that the string completed to a legal filename that is not the name of an existent file.

CHANGE-PROPERTIES

```
args = filename <nl> property-name <sp> new-value <nl> ...
results = <none>
```

This allows you to change one or more properties of a file. The properties that can be changed are those in the SETTABLE-PROPERTIES property of the header record returned by the DIRECTORY command. The legal values for yes/no properties are T and NIL.

1.5 Character Set Translations

The Lisp Machine character set uses 8-bit bytes, and has 256 characters. However, most existing computers use the ASCII character set, which uses 7-bit bytes and has only 128 characters. In order to be able to represent Lisp Machine characters on ASCII file computers, a translation must be performed. This translation is only done for CHARACTER file transfers, not BINARY file transfers.

Figure 1 shows how the translation works. Some Lisp Machine characters are represented by two ASCII characters. The notation "x in (c1, c2)" means "for all character codes x such that c1 <= x <= c2".

Lisp Machine character	ASCII character(s)
x in (000, 007)	x
x in (010, 012)	177 x
013	013
x in (014, 015)	177 x
x in (016, 176)	x
177	177 177
x in (200, 207)	177 (x - 200)
x in (210, 212)	(x - 200)
213	177 013
214	014
215	015 012
x in (216, 376)	177 (x - 200)
377	*no corresponding code*

Figure 1. Translation table. *(Note: All numbers are in octal.)*

The translation scheme is designed so that if you only use characters that are contained in the ASCII character set, your file will make sense in ASCII on the server file system. Note that, to this end, the Lisp Machine "newline" character, 215, translates into the ASCII "carriage return, line feed" sequence: 015 followed by 012. The ASCII character code 177 is used as a quoting character.

The translation described above is performed for all CHARACTER transfers unless either the RAW or the SUPER-IMAGE option is specified in the OPEN command. The meanings of these options are:

RAW — Don't translate characters; ignore the eighth (high-order) bit. This is useful for transferring files between two ASCII hosts.

SUPER-IMAGE — Suppress the use of the 177 character as an escape character. That is, do the usual translation, but in the two-character sequences that would normally begin with a 177, don't output the 177 itself. Figure 2 shows the translation table for SUPER-IMAGE transfers. This ia a many-to-one mapping, and so when a file is read in SUPER-IMAGE mode, the translation has to make a decision about which Lisp Machine character to produce for each ASCII character. This reverse translation is given in Figure 3.

Lisp Machine character	ASCII character(s)
x in (000, 177)	x
x in (200, 214)	(x - 200)
215	015 012
x in (216, 376)	(x - 200)
377	*no corresponding code*

Figure 2. SUPER-IMAGE Translation. *(Note: All numbers are in octal.)*

ASCII character	Lisp Machine character
x in (000, 007)	x
x in (010, 012)	x + 200
013	013
x in (014, 015)	x + 200
x in (016, 176)	x
177	177

Figure 3. SUPER-IMAGE Reverse Translation. *(Note: All numbers are in octal.)*

1.6 Data Transfer

Once a data connection has been established and an OPEN command has succeeded, transfer of a file proceeds on the data connection. If the OPEN was in the READ direction, the server is the sender and the user is the receiver; if it was in the WRITE direction, the server is the receiver and the user is the sender. In either case, the sender proceeds to send data packets on the data connection, containing successive characters or bytes of the file. CHARACTER data is always sent in packets with opcode 200 (octal), and binary data is always sent in packets with opcode 300 (octal).

The remainder of this section discussed how data are packed into packets, how synchronous marks are used in the data connection, and how file transfers are finished.

1.6.1 Packing of Data

The characters or bytes of a file are placed into packets by the sender, in order. There are no particular restrictions on the minimum or maximum sizes of data packets, other than hardware or operating system limits. Usually it is most efficient to send the largest packets possible.

CHARACTER data is sent in packets of 8-bit bytes, with one character stored in each byte. The packets must have opcode 200 (octal), which tells the Chaosnet that the packet is considered to be made up of 8-bit bytes.

BINARY data is sent in packets of 16-bit bytes, with each byte of the file stored in one 16-bit byte of the packet, regardless of the byte-size of the connection. The packets must have opcode 300 (octal), which tells the Chaosnet that the packet is considered to be made up of 16-bit bytes. If the byte-size of the transfer is less than 16, then the bytes of the file will be stored right-justified within the 16-bit bytes of the packets; the remaining high-order bits are unused and undefined. Each packet should contain an integral number of 16-bit bytes, i.e., each should have an even byte-count field.

For more information about Chaosnet data packet formats in general, see section 3.6 ("Data Formats") of the *Chaosnet* document.

1.6.2 Synchronous Marks

Information in a data transfer is exchanged over two connections: the control connection and the data connection. Sometimes a command on the control connection makes some reference to a point in the stream of data in the data connection. However, these two connections are not synchronized with respect to each other. When a command or response appears on the control connection, there is no way that the other end can know whether the referenced point in the data connection has been transmitted yet or not; this is because the two connections are independent streams.

In order to let the control connection refer to points in the data connection, special markers, called *synchronous marks*, can appear in the data connection. A synchronous mark is a packet with opcode 201 (octal). The FILEPOS, SET-BYTE-SIZE, and CLOSE commands need to refer to points in the data stream, and they cause synchronous marks to appear on the data connection to mark these points.

When the user sends a FILEPOS command, the server will put a synchronous mark into the data stream at the point at which the file access pointer was changed. So, after the user receives the response from the FILEPOS, if it continues to get normal data packets before seeing a synchronous mark, it means that those data packets were transmitted by the server before the server received the FILEPOS command on the control connection. Once the user sees the synchronous mark, then the packets that follow it are the portion of the file beginning at the specified position. Similarly, when the server sees a SET-BYTE-SIZE command, it puts a synchronous mark into the data stream, so that the user can see the point at which the byte size was changed.

Synchronous marks are also used whenever a data transfer stops; this is explained below.

The server should always send synchronous marks *before* sending the response to the command that elicited the mark. This can prevent a certain subtle bug, which is too complex to go into here.

1.6.3 Finishing a File Transfer

The transfer of a file can end either after all of the file has been transferred, or in the middle of the file transfer if the user end aborts the transfer. Chaosnet EOF packets (opcode 014, octal) are used to mark the point in the data stream at which a file ends; when the sender gets to the end of a file, it must send an EOF packet after the last data packet of the file. EOF packets are always empty; they contain no data.

A file transfer ends when the user sends a CLOSE command. The sender sends a synchronous mark on the data connection to mark the point at which it stops sending the file. The receiver uses the synchronous mark to tell where the transfer ends; any packets before the synchronous mark are part of this file transfer, and any packets after the synchronous mark are part of the next file transfer done on this data connection. When the user closes a READ transfer, the server puts a synchronous mark into the data connection; when the user closes a WRITE transfer, it must put a synchronous mark into the data connection itself.

The difference in purpose between the EOF packet and the synchronous mark is that the former marks the end of the file itself, whereas the latter marks the end of the transfer. In a READ connection, the user will usually see an EOF followed by a synchronous mark at the end of a file transfer; however, if the user sends a CLOSE command before the server has sent all of the

file, then the user will see a synchronous mark but will not see any EOF. In a WRITE connection, there is always an EOF followed by a synchronous mark, since the server can't abort transfers.

Here are the steps for a user to read a file, assuming that the control connection is established, the user is logged in, and there is an idle data connection in the input direction. First, the user sends an OPEN command, and awaits the response. The user then reads packets from the data connection, awaiting successive packets until it encounters an EOF packet. At this point, nothing else is coming on the data connection, and so the user should not await more packets on it. The user then sends a CLOSE command, and awaits the response. Then the user awaits a synchronous mark on the data connection, indicating that further packets on the data connection will refer to future file transfers.

If the user wants to abort in the middle of reading a file, it should send a CLOSE command, and await its reponse on the command connection. Then the user should read packets from the data connection until it encounters a synchronous mark, in order to flush out of the data connection any packets that the server got into the network before it saw the CLOSE command. (This assumes that there would be no other synchronous marks in transit because of, say, a previous FILEPOS command. If there were any of those, the user would have to wait through the appropriate number of synchronous marks.)

Here are the steps for a user to write a file, again assuming that the control connection is established,the user is logged in, and there is an idle data connection in the output direction. First, the user sends an OPEN command, and awaits the response. Then the user starts sending packets on the data connection, and when it reaches the end of the file, it sends an EOF packet. The user should now wait for the EOF packet to be acknowledged by the server, using the normal Chaosnet packet acknowledgement mechanism. Then, the user should send a synchronous mark on the data connection, and send a CLOSE command on the control connection (in either order), and await the response to the CLOSE. It is important to wait for the EOF to be acknowledged, or else the file could get closed before all of the contents of the file got through.

The steps for the server are analogous, including waiting for the EOF to be acknowledged.

1.7 Errors and Asynchronous Marks

If the server rejects a command, it responds with an error response, with the following format:

```
tid <sp> [ fh ] <sp> ERROR <sp> erc <sp> c <sp> message
```

The "tid" and "fh" fields are the transaction identifier and the file handle (if any) from the command that failed. The "erc" field is a three-character error code, specifying the nature of the problem; the possible values are listed below. The "c" field is a single character, which is always "F" ("F" stands for fatal; this field exists for consistency with the format of asynchronous marks, described below). The "message" field is a character string describing the error message, suitable for being printed to people.

The user program should not try to interpret the "message" field, since it is not specified to contain anything in particular; it should only look at the "erc" field to figure out what happened. For most errors, there isn't anything particularly useful to do other than to print the "message" field or otherwise get it to where somebody will see it. The most interesting error is the "FNF" error, which means that the file was not found. This error is *not* used for malformed filenames or non-existent directories; there are separate codes for these errors.

Asynchronous mark packets (opcode 202 octal) are sent by the server to indicate error conditions in the transfer, such as running out of disk space or allocation. Some of these may be recoverable; some are not. During a READ transfer, any asynchronous marks will appear in the data connection; during a WRITE transfer, they will appear in the control connection. (The data connection is not used in this case because it is transferring in the wrong direction.) Here is the format of an asynchronous mark packet:

```
tid <sp> fh <sp> ERROR <sp> erc <sp> c <sp> message
```

That is, it looks like a regular error response packet, except that it has a different opcode (202 instead of 200). The meaning of the "tid" field is undefined; it should be ignored. The "fh" field is the file handle of the data connection on which the error occured; this is redundant for READ transfers, since the asynchronous mark appears on that data connection anyway, but it is useful for WRITE transfers, because asynchronous marks for many data connections appear in the same control connection. The "erc" and "message" fields are the same as in error responses. The "c" field is one letter, and may be either "R", meaning that the error is recoverable, or "F", meaning that the error is fatal.

If a recoverable error occurs on a data channel, the user can attempt to continue the file transfer (after doing something that might clear up the problem, which might just mean waiting for some time to pass) by using the CONTINUE command, or it can abort the transfer by using the CLOSE command. If a fatal error occurs, the user end has no alternative but to abort the connection, and it should send a CLOSE command. (Sending a CLOSE command on such a channel, if it is a READ transfer, will cause the server to send a synchronous mark as usual; closing a WRITE channel should always be accompanied by the sending of a synchronous mark.)

The following is a list of the values of the "erc" field, that is, the error codes. There are three mutually exclusive sets of errors: those that appear in error responses (*command* errors), those that appear in asynchronous marks and are fatal (*fatal* errors), and those that appear in asynchronous marks and are recoverable (*recoverable* errors).

CCC - Channel cannot continue. *(command)*
: There is no way to continue the file transfer on this channel. This is a response to the CONTINUE command.

CNO - Channel not open. *(command)*
: The channel is not open. This is a response to the CLOSE command.

CRF - Cannot rename file. *(command)*
: It is impossible to rename the file to its actual name. This is used only on ITS, where, when a file is being written, the file is written under a temporary name and is only renamed to its real name just before closing the file. This is a response to the CLOSE command by the ITS server, and probably means that there is something wrong with the filename given for the file, possibly meaning that there is already a file by that name.

CSP - Cannot set pointer. *(fatal)*
: The server cannot set the file pointer. This error is generated by the FILEPOS or SET-BYTE-SIZE command.

FNF - File not found. *(command)*
: The specified file does not exist, although the filename is in a valid format and the directory that it specifies does exist. This is a response to the OPEN command.

IBS - Illegal byte size. *(command).*
: The specified byte size was not a number in the range 1 to 16, inclusive. This is a

response to the OPEN command.

IDO - Illegal data opcode. *(fatal)*
A packet appeared on the data connection with a bad opcode. The user should only use opcode 200 in a CHARACTER transfer, and only opcode 300 in a BINARY transfer.

IFH - Illegal file handle. *(command)*
The specified file handle was for READ when a file handle for WRITE was expected, or vice versa. This is a response to the FILEPOS or SET-BYTE-SIZE command.

IOC - Input/Output condition. *(restartable)*
This error code is a catch-all for a variety of conditions, mostly file-system dependent, such as running out of room on a medium, or getting an error report from a memory device. It can happen at any time during a file transfer.

IPO - Illegal packet opcode. *(fatal)*
The server received an packet with an illegal opcode, or received a synchronous or asynchronous mark when it did not expect one.

IRF - Illegal request format. *(command)*
The command was not well-formed. This is a response to any command.

ISC - Illegal SET-BYTE-SIZE channel. *(command)*
The channel was not a BINARY channel. This is a response to a SET-BYTE-SIZE command.

NCN - Null command name. *(command)*
The command name field was empty.

NER - Not enough resources. *(command)*
The server has run out of some resource. This can happen if it runs out of I/O channels (or JFNs) with which to communicate with its operating system in the OPEN command, if the system runs out of Chaosnet connections in the DATA-CONNECTION command, or if it cannot find the user-specifid information in the LOGIN command.

NLI - Not logged in. *(command)*
The user end is not logged in. This is a response to the OPEN command.

TMI - Too much information. *(command)*
A response or error packet overflowed.

UFH - Unknown file handle. *(command)*
The file handle in the command is not known.

UKC - Unknown command. *(command)*
The command name is not known.

UNK - User not known. *(command)*
The user is not known. This is a response to the LOGIN command.

UOO - Unknown OPEN option. *(command)*
The option is not known. This is a response to the OPEN command.

xxx - (Anything.)
Servers may send any other code, for errors that are not covered above. User ends should be prepared to find anything in the "erc" field. When system calls fail, the ITS server sometimes sends error codes based on an internal table used in the

"ERR: device", and the TOPS-20 server uses the first letters of the first three words of the error message returned by an ERSTR call. However, the TOPS-20 GJFX24 ("file not found") and OPNX2 ("file does not exist") errors will always be reported as FNF errors (see above).

1.8 Notifications

A notification packet (opcode 203 octal) is sent on the control connection to notify the user of some event, such as that the server is scheduled to be shut down at some certain time. The data part of the packet is a character string to be displayed to the user. The user end should somehow report this message.