Pacsat Protocol: File Transfer Level 0

Jeff Ward, G0/K8KA
Harold E. Price, NK6K

ABSTRACT

This document specifies Version 0 of the File Transfer Level 0 (FTL0) protocol designed for use on store-and-forward satellites (PACSATs). The protocol provides procedures for transferring binary files to and from a server computer using an error-corrected communication link.

Further to basic file transfer facilities, FTL0 provides file selection and directory procedures. Because a server may contain many files, only some of which are of interest to each client, the client may "select" a subset of the server's files. Subset selection is based on a flexible logical equation involving a large number of file characteristics. The scope of directory requests or file download requests can be limited to those files currently selected.

This protocol is designed to work specifically with files in which the actual "data" or "body" of the file is preceded by a standard "header", specified in the PACSAT File Header Definition document.

1.0 BACKGROUND

Terrestrial packet-radio bulletin board systems (PBBSs) in the amateur radio service generally provide a character-based, human-to-machine interface through which user's command the PBBS software. A similar interface has been adopted on the FUJI-OSCAR store-and-forward (PACSAT) satellites. Through the character-based interface, the user selects messages, requests directory listings, and views, stores and deletes messages. All messages are stored as ASCII text files, and the PBBS maintains a directory entry for each message. The directory contains message source, destination, title, time of storage, size, and status. The PBBS also stores data relating to each user station, such as the most recent message each user station has listed or viewed.

Since messages and PBBS commands contain only ASCII characters, the user's station equipment can be as basic as an ASCII terminal connected to a **terminal node** controller (TNC). This simplicity **makes the** current system attractive, and has lead to its widespread adoption.

These current PBBS systems also have several drawbacks, stemming from the text-based user interface and from the lack of a flexible, informative message header. All messages must be in ASCII, only simple ASCII message headers are supported, and only a few search-keys are provided for message selection.

These limitations are particularly damaging in the PACSAT environment. where the user base is large, the communications bandwidth is restricted, and the onboard computer software must be compact and highly reliable.

The File Transfer Level 0 protocol defined here, and the PACSAT File Header Definition presented elsewhere, combine to provide a PBBS protocol suite which overcomes the limitations of the current text-based protocols.

The proposed protocols are based on a different model of a store-and-forward messaging network. The PBBS is a "server" and the user stations are "clients". The server stores files, each of which contains a "header" of known format, followed by a "body". The header holds information about the file and about the body, including (but not limited to) the file length, creation date, destination, source, keywords, and body compression technique. The body is the data portion of the file.

which can be any digital information, e.g. binary, ASCII, FAX, digitised voice, etc.

The user station, or "client" is assumed to be under computer control. The client sends files to the server and receives files from the server over an AX25 data link, using an automated server/client file transfer protocol. The client software interprets and displays the file headers and bodies for the human user. When the user wishes to upload a file to the server, client software must add suitable standardized header fields to the file, and it may also compress or otherwise prepare the file for uploading.

Each user is generally interested in only a small subset of the files available on a server. Using a powerful selection facility the client can select an interesting subset of the files on the server for directory listing or downloading. The selection can be based on any combination of items in the file header, thus user's software can assure that the user only "sees" files which are likely to be of interest.

The server might not keep a user data base, and the client software must store any important information concerning the user's last server session. For example, to implement a command equivalent to the PBBS command "list new", the client software must store the message number, or date and time of the of the last access to the server.

Since this protocol is also useful for terrestrial servers, we use the terms "server" and "client" throughout, instead of "PACSAT" and "groundstation". The protocol does depend on the presence of the standard PACSAT file header, this is to implement file locking or other features. A version of the protocol that can be used by terrestrial stations without the PACSAT headers is under consideration.

FTL0 includes several features that make the protocol somewhat more complex than some others in use in the amateur packet network. These features include simultaneous bidirectional file transfer, file locking by forwarding gateways, and multiple destination forwarding. As is sometimes the case, the actual implementation is smaller than the specification. The pseudo-code provided in Appendix B and C show that the protocol is actually quite straightforward. The authors also plan to make the prototype source code available on a public BBS.

1.1 FTL0

The FTLO protocol can support full-duplex message transfer, in which a single data link (AX.25 connection) is used simultaneously to upload one file and to download another file. Client software does not necessarily have to implement this capability. The protocol provides for continuation of uploads and downloads interrupted by loss of the data link (generally caused by LOS at a satellite groundstation), and includes confirmation handshakes at all critical stages of file transfer. An FTL0 file transfer should withstand interruption at any stage.

It is assumed that the client software uses the transparent mode of an AX.25 TNC and that the AX.25 connection represents an error-free channel. Other data links which present an ordered, error corrected byte stream may be used. Although the link may be terminated at any time, it is assumed that the link will not pass corrupted or out-of-order bytes, and that processes at both ends of the link will be notified if the link is terminated. The PACSAT File Header standard provides overall checks on the integrity of files transferred to and from fully-compliant servers.

FTL0 commands and responses form simple "packets" inside the byte stream on the communications link. Specifically, in AX25, an FTL0 packet may be transmitted in one or more AX.25 I frames; neither the server nor the client is aware of the AX.25 I frame boundaries. The packet format is described in 2.0 below.

1.2 Associated Documents

PACSAT Data Specification Standards defines the meaning of data types and structures within this specification.

The PACSAT File Header Definition defines the file header fields which must be pre-pended to files before uploading to a fully-compliant FTL0 server. (FTL0 procedures for terrestrial servers without PACSAT file headers are under development.)

1.3 Terminology

server	is the PACSAT or tbe PBBS.
client	is tbe user station.
commands	are transmitted from the client to tbe server
responses	are transmitted from the server to the client
uplink	is data flow from the client to the server
downlink	is data flow from the server to the client
packet	is a sequence of bytes as defined in 2.0 below
frame	is an AX25 level 2 I frame
data link	generally an AX.25 level 2 link, but may be a link using any protocol which provides an ordered, error-checked, data transparent byte stream.

1.4 Overview of Operation

To allow simultaneous uploading and downloading. FTL0 clients and servers implement two state machines. One is related to file uploads, and the second to all other possible commands. FLT0 is not symmetric; the protocol state machines implemented by the server are necessarily different from those implemented by the clients.

Dividing the data stream into packets allows for modular software implementation and the multiplexing of commands and data on a single data link. The packet format is simple. Every packet begins with 2 bytes of control information, which can be followed by 0 to 2047 bytes of data. The packet may arrive in many AX.25 I frames. No packet synchronization or checksumming is provided, since the data link (AX.25 connection) provides a notionally error-free connection. Full file error checks are performed at tbe completion of file transfers.

It is assumed that neither the client software nor the server software is aware of or can control data link (AX25) frame boundaries.

A FTL0 session starts when an data link is established between tbe server and the client. The Server will send a LOGIN response when the connection is established.

To allow the user to specify a group of files for directory listings or downloading, the client software implement may implement menus or some other user interface.

Once the user has indicated the desired files, the client software converts the user's preferences into an equation , such as

((SOURCE = "G0K8KA*") && (KEYWORD = "*FTL0*")) && (FILE NUMBER > 3215).

And then encodes the equation in a postfix, binary format as a SELECT command. When the server receives a SELECT command it builds a list of files for which the equation evaluates to TRUE. As a response to the SELECT command, the server informs tbe client bow many files are in the selection list.

When the client sends a one of the directory commands, the server responds by transmitting directory information for the next 10 files on the selected list. Each subsequent directory command results in the server transmitting directory information for a further 10 files from the selection list, until the list is exhausted.

Specific files or files from the selection list may be requested by the client for downloading. The download procedure consists of two command/response cycles. On the first cycle, the client requests the file and the server responds by transmitting the file as a series of DATA packets followed by an END_OF DATA packet. The server then waits for the client to acknowledge receipt of the file. When the server receives the acknowledgement, it transmits a final handshake and the download is finished.

To upload a file to the server, the client sends an UPLOAD command telling the server how large the file is. The server responds with a handshake which either tells the client to abort or to proceed. If instructed to proceed, the client sends the entire file as a series of DATA packets followed by an DATA_END packet. Tbe server receives the entire file, and then either acknowledges the transfer or sends an error indication.

Because PACSAT may go over tbe horizon in the midst of a transaction, downloading and uploading can be continued during several sessions with tbe server. Wben a download is interrupted, the client must retain the file offset at which communications was interrupted. For interrupted uploads, the server stores the appropriate offset.

1.5 File Identification

Although servers may identify files using some kind of system file name (PACSATs use the MS-DOS standard of an 8 character body and a 3 character extension), FTL0 does not use server file names to identify files. Instead, the server assigns every file a 32-bit file number. This is to insure that two files with different contents but the same name will never be confused. Thus, every message which a user uploads to PACSAT (or another FTL0 server) will have a unique identifier.

1.6 Gateway Procedures

Some messages on PACSAT will be destined for gateway stations which will introduce the messages into other networks. It is essential that only one gateway download and relay each message. To accommodate this, FTL0 provides locked downloads. Only one gateway at a time can perform a locked download of a given file for a given destination. If a locked download is interrupted, this will be indicated in the PACSAT File Header of the file, so that other gateways can tell when a file is in the process of being delivered.

PACSAT File Headers allow files with multiple destinations, and separate locks are implemented for each destination. See Appendix A for further information.

1.7 Delivery Registration

A simple form of delivery registration is supported by FTL0. In the final handshake after receiving a file, the client can command the server to modify the contents of the PACSAT File Header to show the AX.25 address of the client and the time at which the download was completed. It is intended that this facility be used if the file is specifically addressed to the client station, not for files of general interest. If the file has multiple destinations, one "registration" is supported for each destination. See Appendix A for further information.

1.8 State Variables

In order to support full-duplex operations, the server and the client maintain 2 state variables, one relating to the uploading of files and the other relating to all other operations (selecting, directories, and downloading). These are called the uplink state and the downlink state variables.

Appendix B provides pseudo-code definitions of the state transitions for sewers, Appendix C is for clients.

2.0 PACKET FORMAT

An FTL0 packet is a sequence of information bytes preceded by a two byte header. After establishment of a data link, the first byte delivered to either client or server is the first byte of a packet header. The packet header informs the client/server the type of the packet and the number of information bytes which will follow. There may be between 0 and 2047 information bytes, inclusive. After reception of the final information byte of a packet, the client/server expects the next byte to be the first byte of another packet header. Such a stream is shown below, where [<data >] indicates that there may be no data bytes.

<length lsb><hl>[<info>...]<length lsb><hl>[<info>...]
|----First FTL0 packet---|--Second FTL0 packet---|

2.1 Header Format

```
struct FTL0_PKT (
    unsigned char length_lsb;
    unsigned char hl;
}
```

< length_lsb > - 8 bit unsigned integer supplying the least significant 8 bits of data length.

< hl > - an 8-bit field.

bits 7-5 contribute 3 most significant bits to data length.

bits 4-O encode 32 packet types as follows:

0	DATA
1	DATA_END
2	LOGIN RESP
3	UPLOAD_CMD
4	UL GO RESP
5	UL-ERROR_RESP
6	UL-ACK RESP
7	UL_NAK-RESP
8	DOWNLOAD_CMD
9	DL ERROR RESP
10	DL-ABORTED RESP
11	DL-COMPLETED _RESP
12	DL-ACK CMD

13 DL NAK_CMD
14 DIR SHORT_CMD
15 DIR-LONG_CMD
16 SELECT CMD
17 SELECT-RESP

2.2 Information Length

The <information length> value is formed by pre-pending bits 7-5 of the <h1> byte to the <length lsb> byte. <information length> indicates how many more bytes will be received before the beginning of the next packet. If <information length> is 0, there are no information bytes.

3.0 LOGON

As soon as a data link is established between the client and the server, the client is considered to be logged on. The server sends a LOGIN_RESP packet with one byte of flags and a 4-byte timestamp.

When the client receives the LOGIN_RESP packet the server is initialized and ready to begin transactions.

3.1 Initiation of Session

An FTL0 session is initiated when a data link is established between the client and the server.

3.2 Server Login Response Packet

When the data link is established the server transmits a LOGIN RESP packet.

Packet: LOGIN RESP
Information: 5 bytes

```
struct LOGIN_DATA {
    unsigned long login_time:
    unsigned char login_flags;
}
```

<login time> - a 32-bit unsigned integer indicating the number of seconds since January 1, 1970.

<login_flags> - an 8-bit field.

bit:76543210
xxxxSHVV

Bit 3, the SelectionActive bit, will be 1 if there is already an active selection list for this client. The SelectionActive bit will be 0 if there is no active selection for this client already available.

Bit 2, the HeaderPFH bit, will be 1 if the server uses and requires PACSAT File Headers on all files. If the HeaderPFH bit is 1, the flag PFHserver used in the following definition should be considered TRUE.

The HeaderPFH bit will be 0 if the Server does not use PACSAT File Headers. If the HeaderPFH bit is 0, the modified procedures specified in Section 7 should be used.

Bits 1 and 0 form a 2-bit FTL0 protocol version number. The version described here is 0.

3.3 Server Login Initialization

Upon transmitting the LOGIN_RESP packet the server should initialize its state variables to UL CMD WAIT and DL CMD WAIT.

3.4 Client Login Initialization

Upon receiving the LOGIN RESP packet, the client should initialize its state variables to UL CMD OK and DL CMD OK.

4.0 SELECT

The select command is the mechanism through which the client can search the server for desirable files. The human user can search the files on the server based on any information contained in the PACSAT File Header. To achieve this goal, the implementation and specification go through several levels of abstraction which must be followed closely.

The user's desires form an equation. The equation might simply be "the file is to G0K8KA and it was stored after my last logon". Or it might be more complex, e.g. "the file is less than 8 kbytes, and it was created after 9 July 1990, and it has 'landrover" as one of its keywords". Humans would naturally express this in "infix" notation:

(FILE_SIZE < 8k) && (CREATE_DATE > 9 July 1990) && (KEYWORDS = = "*landrover*")

The server uses "postfix" notation, like an RPN calculator: and wants an equation like:

(FILE_SIZE < 8k) (CREATE_DATE > 9 july 1990) && (KEYWORDS = = "*landrover*") &&

To convert a postfix logical equation into a SELECT command. the client must encode the logical operators, the relational operators the header item names, and tbe values to compare against. Tbe header item names and comparison values are encoded just as they are in tbe PACSAT File Header definition. The logical operators and relational operators are encoded as single bytes. Tbe complete syntax is defined below.

4.1 Client Issuing Select Command

When the client's downlink state variable is DL_CMD OK (e.g. the client is not involved in another select, file download, or directory download), the client may transmit a SELECT CMD packet.

Packet: SELECT CMD
Information: variable length SELECTION

SELECTION is defined recursively by the following structures and rules.

```
struct SELECTION (
      struct LVALUEx equation;
      unsigned char end-flag;
}
```

< end flag > - 0x00

<equation > is recursively defined by two structures LVALUEO and LVALUE1:

```
struct LVALUEO {
      struct LVALUEx t1;
      struct LVALUEx t2;
      unsigned char lop;
}
```

c t1 > and < t2 > are furtber LVALUEO or LVALUE1 structures.

< lop> is and 8-bit field representing a logical operator:

bit 76543210
0000Oool is AND
00000010 is OR

```
struct LVALUE1 (
      unsigned char relop;
      unsigned int item id;
      unsigned char length;
      unsigned char constant [ ];
}
```

< relop > is an 8-bit field encoding a relational operator:

b i t
0

b i t	Relational Operator
000	equal to
001	greater than
010	less than
011	not equal
100	greater than or equal to
101	less than or equal to
110	reserved
111	reserved

bit 321Q	Type of Comparison
0000	treat <constant > as a multi-byte unsigned integer (valid for <length > 1,2, and 4).
0001	treat <constant > as a multi-byte signed integer (valid for < length > 1,2,and 4).
0010	treat <constant > as an array of unsigned char
0011	treat <constant > as an array of ASCII characters, convert to lower case before making comparison.
0100	treat <constant > as an array of ASCII characters, convert to lower case before comparison and interpret wildcard characters.

ALL OTHER VALUES RESERVED.

<item id > is a 16-bit unsigned integer identifying one of the PACSAT Header Definition header items.

<length> is the number of bytes in the <constant[ ]> array.

<constant[ ] > is an array of bytes which are compared against the header item identified by < item_id > , using the specified relational operator and comparison type.

4.2 **Response to Select Command**

T'he server responds with one of the packets from 4.2.1 or 4.2.2 and returns its downlink state to DL **CMD** OK, ready to accept another command from the client.

4.2.1 Successful Selection

If the server can interpret the SELECT_CMD packet, it responds with a SELECT RESP packet

Packet: SELECT_**RESP**
Information: 2 bytes
unsigned int no_of_files

<no_of_files> is a 16 bit unsigned integer telling how many files have been selected. It may be 0.

4.2.2 Unsuccessful Selection

If the server cannot interpret the SELECT CMD packet, it responds with a DL ERROR-RESP packet

Packet: DL ERROR_RESP
Information: 1 byte
unsigned char err code

<err_code> is ER_POORLY_FORMED SEL if the selection equation could not be parsed by the server because of a syntax error.

5.0 DOWNLOAD

To receive a file from the server, the client uses the download command. This command can be used to download a specific file, to continue the downloading of a specific file, or to download the next file in the selection list.

5.1 Client Initiates Download

When the client's uplink state variable is DL_ CMD_ OK, the client may send the DOWNLOAD CMD packet.

Packet: **DOWNLOAD_CMD**
Information : 9 bytes of arguments with the following structure

```
struct (
    unsigned long file_no;
    unsigned long byte_offset;
    unsigned char lock-destination;
}
```

<file no> is a 32-bit binary integer uniquely identifying a file on the server. Two reserved values have special meanings: If <file-no> is equal to Oxffffffff, the server will attempt to use the next file in the current selection list, **moving** from older files to newer files. If <file_no> is equal to 0, the server will attempt to use the next file in the current selection list, **moving** from newer files toward older files. If <file_no > is not one of the reserved values, the server attempts to download the file which it associates with <file no x

<byte offset > is a 32-bit unsigned binary integer giving-the offset from the beginning of the file at which the download should begin. If the client is continuing an aborted download, this argument should be set to the number of bytes previously received. Otherwise it should be 0.

<lock_destination > is an 8 bit unsigned binary integer. It will generally be set to 0 by non-gateway stations.

[If <lock destination > is not 0, it indicates that the client wishes to conduct a locked download for the destination numbered by clock destination > . If the <lock_destination > exists and is not already locked, the client's AX.25 address is placed in the PACSAT File Header AX25 DOWNLOADER item associated with the specified DESTINATION item. Destination numbering is defined in section x.x]

5.2 Server Responses to the Download Command

When it receives a DOWNLOAD_CMD packet the server determines whether the download is possible. This may include determination of the desired file number from the current select list.

5.2.1 Downlinking File Data

If the download is possible the server transmits the file data beginning <byte_offset > bytes from the beginning of the file. If <byte_offset> is greater than or equal to the file length, no data is transmitted. The file data bytes are transmitted as information in **DATA** packets. These packets may be any size from 0 to 2047 bytes.

[If < lock destination > is not-zero, the server sets the <DOWNLOAD_TIME> associated clock_destination > before beginning to send DATA packets.]

Once the server has transmitted a DATA packet containing the last byte of the file, or if the <byte offset> was greater than or equal to the file length, the server transmits a single DATA_END packet. After transmitting the end packet, the server expects to receive a DL ACK CMD or DL NAK CMD.

5.2.2 Failure of Download Command

If the server cannot service the download command, it responds with a DL ERROR RESP packet.

Packet: DL ERROR RESP
Information: 1 byte
unsigned char err code;

<err_code > is an S-bit unsigned binary integer will be one of:

ER SELECTION EMPTY if the selection list has no more files in it or no select command was previously received.

ER NO_SUCH_FILE NUMBER if the file identified by < file no > do& not exist.

ER_ALREADY LOCKED - <lock_destination> was not 0, and the file is already locked for the specified destination.

ER_NO_SUCH_DESTINATION
<lock destination > was not 0, and the file has fewer-than <lock destination > DESTINATION items in its PACSAT File Header.]

5.3 Client **Accepting** or **Rejecting Downloaded File**

5.3.1 Successful Download

Upon reception of the DATA_END packet from 'the server, the client performs any possible checks on the received file. If the file passes the checks, the client transmits a DL ACK CMD packet.

Packet: DL_ACK_CMD
Information: 1 byte
unsigned char < register-destination >

<register destination > is an 8-bit unsigned integer identify& one of the destinations in the PACSAT File Header of the downloaded . file. <register_destination > should be 0 if the client does not wish to register receipt of the file.

Upon receiving the DL_ACK CMD packet, the server completes the download process.

[If <register destination > is not 0, the appropriate DESTINATION item in the PACSAT File Header is located and the associated AX25 DOWNLOADER and DOWNLOAD TIME items are updated. If clock destination > was non-zero in the DOWNLOAD COMMAND, the server locates the appropriate DESTINATION item in the PACSAT File Header and updates the associated DOWNLOAD TIME item.]

5.3.2 Unsuccessful or Aborted Download

If the client wishes to abort the download before receiving the DATA END packet from the server, or finds that the downloaded file fails some integrity test after reception of the DATA END packet, the client sends the DL NAKCMD packet.

Packet: DL_NAK CMD
Information: none-

Upon receiving this packet, if the server has not already sent the DATA_END packet, it will do so and proceed to the final unsuccessful download handshake (5.4.2). If DL NAK CMD is received after the server has transmitted& DATA END packet, the server proceeds to the final download handshake.

5.4 Final Download Handshake

5.4.1 Completion of Successful Download

[If the server receives the DL_ACK_CMD, it removes any <lock destination > lock from the file and sets <AX25_DOWNLOADER > and <DOWNLOAD_TIME> for the specified <DESTINATION > item.]

[If <register destination > in the DL ACK CMD is non-zero, the server attempts-to Set the <AX25_DOWNLOADER > and <DOWNLOAD_TIME> items associated with the specified <DESTINATION > item. If the

<register destination > does not exist, the server transmits; DL ABORTED RESP packet.]

If <register destination > was 0, the server transmits a DL_COMPLETED_RESP packet, and returns its downlink state variable to DL CMD OK.

Packet: DL_COMPLETED_RESP
Information: none.

This response tells the client that the server has completed processing the DL_ACK_CMD. The client downlink state variable should be set to DL CMD OK.

If the L2 link fails before the client receives the DL_DONE_RESP, the client cannot be sure the DL_ACK_CMD was processed. This is important if the client had specified a <lock destination > or a <register destination >. In either of these cases, the client should treat the download as incomplete and continue it later. This assures that the PACSAT File Header has been properly modified. If both <lock destination > and < register destination > were 0, the client can consider the downloadcomplete.

5.4.2 Completion of Unsuccessful Download

A download is "unsuccessful" if the server receives the DL NAK_CMD from the client or if < register destination > is non-zero and indicates a non-exist&t destination.

If clock destination > was non-zero, the lock is removed- and < DOWNLOAD-TIME > is updated.

The server transmits the DL ABORTED RESP packet.

Packet: DL_ABORTED_RESP
Information: n o n e

The server downlink state variable returns to the DL CMD OK state.

Upon reception of the DL ABORTED RESP the client downlink state -variable returns to DL CMD OK.

6.0 DIRECTORY

The client uses directory commands to get information about files on the server. Servers conforming to the PACSAT File Header Definition send PACSAT File Headers as directory entries. "Long" directory entries are the complete PACSAT File Header and "short" directory entries are only the Mandatory File Header items.

The client can request a directory entry for a specific file or for the files in the selection list. The selection list can be scanned from oldest to newest files or from newest to oldest files.

6.1 Initiating a Directory

The client may request a directory whenever the client downlink state variable is DL CMD OK. To request a directory, the client transmits either a DIR LONG CMD packet or a DIR-SHORT CMD packet.

Packet: DIR LONG CMD
or DIR-SHORT_CMD
Information: 4 bytes
unsigned long file no;

<file no> is a 32-bit unsigned binary integer identifying a file or files on the server. There arc two reserved values: 0 and Oxffffffff. If <file-no> is Oxffffffff, the server will send directory entries for the next 10 files in the current selection list, moving from older files to newer files. If <file no > is 0, the server will send directory entries for the next 10 files in the current selection list, moving from newer files toward older files.

6.2 Server Responses to Directory Commands

6.2.1 Successful Directory Request

If the directory request can be serviced (there are files in the selection list, or a specified file exists) the server will transmit one or more PACSAT File Headers in a series of DATA packets followed by a DATA_END packet. A maximum of 10 file headers will be transmitted for each DIR command.

6.2.2. Failed Directory Request

If the directory request cannot be serviced, a DL ERROR RESP packet is transmitted.

Packet: DL ERROR_RESP
Information: 1 byte
unsigned char err code;

<err code> will be one of:

ER SELECTION_EMPTY - if the <file_no> was-0 or 0xffffffff, and either there are no files left in the selection list or there is no selection list.

ER_NO_SUCH_FILE NUMBER - if <file no> is not 0 or 0xffffffff -and no file identified by <file no> exists on the server.

7. UPLOAD

7.1. Initiating an Upload

The client can initiate an upload any time the client's upload state variable is UL CMD OK.

Packet: UPLOAD_CMD
Information: 8 bytes

```
struct {
    unsigned long continue_file_no;
    unsigned long file_length;
}
```

<continue file no> - a 32-bit unsigned integer identifying the file to continue. Used to continue a previously-aborted upload. Must be 0 when commencing a new upload.

< file-length > - 32-bit unsigned integer indicating the number of bytes in the file.

7.2 Server Responses to Upload requests

Downlink Packet: UL GO RESP or UL ERROR RESP

7.2.1 Successful Upload Request

Packet: UL GO_RESP
Information! 8 bytes

```
struct {
    unsigned long server file no:
    unsigned long byte-offset;
}
```

<server file no> is a 32-bit unsigned binary integer identifying the client's file on the server.

<byte offset > is a 32-bit unsigned binary integer number of bytes from the beginning of the file at which the client should begin file transmission. This will be 0 if the <continue file no > was zero.

After receiving the UPLOAD_PROCEED_RESP packet, the client should advance to the data transmitting state described in Section 7.3.

7.2.2 Unsuccessful Upload Request

If the server cannot process the upload request, it will transmit an UL ERROR RESP packet.

Packet: UL_ERROR_RESP
Information: 1 byte
unsigned char err code;

<err code > must be one of:

ER_NO_SUCH_PILE_NUMBER if <continue file no> is not 0 and the file identified by <con&e file no> does not exist. Continue is not possible-

ER BAD CONTINUE if <continue_file_no> is not-0 and-the <file length > does not agree with the <file length > previously associated with the file identified by <continue file no > . Continue is not possible.

ER PILE COMPLETE if <continue file no> is not 0 and the file identified by < continue_file_no> was completely received on a previous upload. Note - receipt of this command should be accepted by the client as confirmation of file receipt by the server.

ER_NO ROOM if the server does not have room for the file.

After transmitting the CJL_ERROR_RESP packet. the server's uplink state variable is set to UL CMD OK.

7.2.3 Continuation of Uploads

Any file for which the client receives a UL_GO_RESP should be continued until a UL_ACK_ RESP or a non-recoverable UL_ERR_RESP or UL_NAK_RESP for that file is received. The server should be prepared to continue reception of any file for which it has transmitted a UL_GO_RESP. Some file numbers will be allocated by the server and lost by link

failure before the UL GO_RESP reaches the client; a 32-bit file number space provides sufficient scope for some lost file numbers. To avoid saving partial files for these lost file numbers, the server should not reserve space for a message until it has received at least one DATA packet from the client.

73. Data Uplinking Stage

The client now uplinks the bytes from the file, in DATA packets. The uplinking should begin <byte offset> bytes from the start of the file. After transmitting a DATA packet containing the last byte in the file, the client should transmit a DATA_END packet. The client should also transmit a DATA_END packet if the <byte_offset > was greater than or equal to the file length.

The server may attempt to terminate the upload by transmitting a UL_NAK_RESP at any time during the upload. If the client receives a UL_NAK RESP packet before transmitting a DATA END packet the client must send a DATA-END packet.

7.4. Completion of Upload

7.4.1 Successful Upload Completion

When the server receives the DATA END packet it will check the integrity of the file as far as possible. If the chocks pass, the server will downlink a UL ACK RESP packet.

Packet: UL_ACK_RESP
Information: none

After transmitting the UL_ACK_RESP the server uplink state variable is UL_CMD OK. After receiving the UL_ACK_RESP, the-client uplink state variable is UL CMD OK.

7.4.2 Failure Caused by Server Rejecting Upload

The server may reject an upload while the client is sending DATA packets (due to file system problems on the server) or after the client has sent the DATA END packet (due to corruption of the file).

If the server must abort the upload while receiving DATA packets or after receiving the DATA_ END checks fail, it sends the UL NAK RESP packet.

Information: 1 byte
unsigned char err code;

<err code> must be one of:

ER_BAD_HEADER - The file either has no PFH, or has a badly-formed PFH.

ER HEADER_CHECK - The PFH checksum failed.

ER BODY_CHECK - The PFH body checksum failed.

ER_NO_ROOM - The server ran out of room for file storage before the upload was complete. The server will implement procedures to avoid frequently running out of room, but this cannot be guaranteed.

After transmitting the UL NAK_RESP packet, the server uplink state variable is UL_ CMD_OK. After receiving the UL_ERROR_RESP, the client uplink state variable is UL CMD OK.

7.4.3 Link Failure During Upload

If the data link fails before the server receives the DATA END packet, the server retains the offset of the next byte needed for the file. This value will be transmitted by the server as <byte offset > if the client Later continues the upload by specifying the < server-file_no > in the <continue file no> of an UPLOAD_CMD packet. If the offset is not 0, the server may safely retain a temporary file containing the data received so far. If the server retains files for which the <byte offset > is 0, these should be purged after some reasonable time, since the client may not know the <server file no> of the file.

8.0 TERMINATION OF SESSION

The session is terminated by closing the data link. Either the client or the server may decide to close the link when uplink state is UL_ CMD_OK and downlink state is DL_CMD_OK. The link is also terminated if unexpected packets are received.

If the link terminates when any state machine is not in the CMD_OK state, appropriate actions are taken.

APPENDIX A - Use of the PACSAT File Header

The PACSAT File Header Definition defines a standard header which will be found at the beginning of every file downloaded from PACSAT (or any fully compliant FTL0 server).

One purpose of the PACSAT File Header is to allow clients to determine the status of files on the server. Specifically, the originating client may wish to find out if the file has been downloaded by its intended destination, and a Gateway client may wish to find out if a message still needs to be downloaded for relay to a particular destination. Both of these tasks are done by examining a set of PACSAT File Header Items comprising <DESTINATION >, <AX25_DOWNLOADER > and < DOWNLOAD TIME > .

A gateway will examine <DESTINATION > to see if it can relay or deliver the file. If so, it will examine <DOWNLOAD_TIME> and <AX25 DOWNLOADER> to determine the delivery status of the message:

If <DOWNLOAD TIME> is zero, the message still needs to be relayed.

If <DOWNLOAD TIME> is non-zero, but <AX25_DOWNLOADER> is blanks, the message is in the process of being downloaded, and the download started at <DOWNLOAD TIME >.

If <DOWNLOAD_TIME> is non-zero, and <AX25 DOWNLOADER> is non-blank, the message-has been downloaded for relay to the <DESTINATION > .

A similar (but simpler) procedure is used to see if a specific destination station has "registered receipt" of a file. When receipt is registered (at the end of a successful download), c AX25_DOWNLOADER > and <DOWNLOAD_TIME> will both be correctly set.

For these checks to work correctly, the downloading clients must use the proper command procedures. Gateways must set the correct <lock_destination > in their DOWNLOAD CMD packets, and clients must set the correct < register_destination > in the DL_ACK CMD. The <lock destination > procedure should be implemented- by all gateways. The < register_destination> procedure is optional for client implementations.

The proper number for <register_destination > or clock_destination > is determined as follows. If a file has only one <DESTINATION > item, it is destination 1. Any other destinations are numbered sequentially in order of occurrence in the PACSAT File Header.

APPENDIX B - Server State Transition Definitions

SERVER STATE DEFINITIONS

The following pseudo code defines the transitions of the Server state machines in FTL0. One state machine is used for file uploading. The other state machine is used for all other procedures.

It is assumed that the data link will be terminated if there have been no packets transmitted or received for a period of time (TBD). If the link is terminated by this "activity timeout" both state machines receive a "Data Link Terminated" event. "Data Link Terminated" is also generated if the link fails for any reason.

When an unexpected packet is received by one of the state machines, that state machine terminates the data link and returns to an uninitialized state. The second state machine is notified of this through a 'Data Link Terminated" event, and also returns to an uninitialized state. At this point, the FTL0 processing for this client ceases until the establishment of another data link.

STARTUP

State variables are created when a data link is established between client and server. The server executes the following:

```
Transmit LOGIN_RESP packet.
ul_state:= UL CMD OK
dl state : = DLCMD-OK
```

SERVER UPLINK STATE MACHINE

This machine receives only frames which are relevant to the file uploading process : UPLOAD_CMD, DATA, and DATA END. It also is executed upon timeout of the input timer or termination of the level 2 link (from local or remote causes).

```
state UL CMD OK {

   EVENT: Receive UPLOAD CMD packet{
      if ok to upload
         Send UPLOAD_GO_RESP packet.
         ul_state <- UL DATA RX
      else
         Send UL_ERROR_RESP packet.
         ul state <- UL CMD OK
   }

   EVENT: DEFAULT{
      if EVENT is not "data link terminated"
          Terminate data link.
      ul state <- UL UNINIT



state UL DATA RX {

   EVENT: Receive DATA packet {
      Try to store data.
      if out of storage {
         Transmit UL NAK_RESP packet.
         Close file.
         Save file_offset and file number.
         ul state < - UL ABORT
      }
      else {
         Update file_offset.
         ul state <- UL DATA RX
      }
   }

   EVENT: Receive DATA END packet {
      Close file.
      if file passes checks {
```

```
            Transmit UL_ACK_RESP packet
            ul state c- UL CMD OK
        }
        else {
            file_offset <- 0
            Save file_offset and file_number
            Transmit UL_NAK_RESP packet
            ul state <- UL CMD OK
        }
    }

    EVENT: DEFAULT{
        if (file_offset > 0)
            Save file_offset and file number.
        if EVENT is not "data link terminated"
            Terminate data link.
        ul state <- UL UNINIT
    }
}

state UL ABORT {

    EVENT: Receive DATA packet
        ul state <- UL ABORT

    EVENT: Receive DATA_END packet
        ul state <- UL CMD OK

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        ul state <- UL UNINIT
    }
}
```

SERVER DOWNLINK STATE MACHINE

This machine receives all packets other than UPLOAD CMD, DATA, and DATA END.

```
state DL CMD OK {

    EVENT: Receive SELECT_CMD packet {
        if syntax is correct {
            Form select list.
            Transmit SELECT_RESP packet.
            dl state <- DL CMD OK
        }
        else {
            Transmit DL_ERROR_RESP packet.
            dl state <- DL CMD OK
        }
    }
```

```
    EVENT: Receive DIR LONG_CMD or DIR SHORT CMD packet {
        if there are directories to send
            dl_state <- DL DIR DATA
        else {
            Transmit DL_ERROR_RESP packet.
            dl state <- DL CMD OK
        }
    }

    EVENT: Receive DOWNLOAD_CMD packet {
        lf request can be serviced {
            if <lock_destination> < > 0
                set PFH DOWNLOAD_TIME.
            dl state c - DL FILE DATA
        }
        else {
            Transmit DL_ERROR_RESP packet.
            dl state <- DL CMD OK
        }
    }

    EVENT: Receive DL_NAK_CMD packet
        dl state <- DL CMD OK

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl state <- DL UNINIT
    }
}

state DL_DIR_DATA {
    if there is more directory data to send {
        Transmit directory data in DATA packets.
        dl state <- DL DIR DATA
    }
    else {
        Transmit DATA_END packet.
        dl state < - DL CMD OK
    }

    EVENT: Receive DL_NAK CMD packet{
        Transmit DATA_END packet.
        dl state <- DL CMD OK.
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl state <- DL UNINIT
    }
}
```

```
state DL FILE DATA {

    if there is more file data to send
        Transmit file data in DATA packets.
        dl state < - DL FILE DATA
    }
    else {
        Transmit DATA_END packet.
        dl state <- DL FILE END
    }

    EVENT: Receive DL NAK_CMD packet {
        If lock destination-c > 0
            unlock lock_ destination.
        Transmit DATA_END packet.
        Transmit DL_ABORTED_RESP packet.
        dl state <- DL CMD OK
    }

    EVENT: DEFAULT {
        If lock_destination < > 0
            unlock lock_ destination.
        if EVENT is not "data link terminated"
            Terminate data link.
        dl state < - DL UNINIT
    }
}

state DL FILE END {

    EVENT: Receive DL_NAK_CMD packet {
        dl state c - DL_CMD_OK
        Transmit DL_ABORTED_RESP packet.
        If lock_destination < > 0
            unlock lock destination.
    }

    EVENT: Receive DL_ACK_CMD packet {
        if register destination < > 0 and register_destination exists {
            set PFH AX25 DOWNLOADER
            set PFH DOWNLOAD TIME
            Transmit DL COMPLETED_RESP packet.
        }
        else if register_destination < > 0 and it doesn't exist {
            Transmit DL ABORTED RESP.
        }
        else if lock_destination < > 0 {
            unlock lock_destination.
            set PFH AX25_DOWNLOADER.
            set PFH DOWNLOAD TIME.
            Transmit DL COMPLETED RESP packet.
        }
        dl state < - DL CMD OK
    }
```

```
    EVENT: DEFAULT {
        If lock_destination < > 0
            unlock lock_destination.
        if EVENT is not "data link terminated"
            Terminate data link.
        dl state <- DL UNINIT
}
```

APPENDIX C - Client State Transition Definitions

Client Pseudo-code State diagrammes.

The following diagrammes indicate how an FTL0 client should react to different events when in different states.

"User Requests" come from the process using the FTL0 state machine.

Receive packets come from the data link and transmit packets go to the data link.

It is assumed that the data link will be terminated if there have been no packets transmitted or received for a period of time (TBD). If the link is terminated by this "activity timeout" the state machines receive a "Data Link Terminated" event. "Data Link Terminated" is also generated if the link fails for any reason.

UPLINK STATE MACHINE

The client uplink state machine controls the file uploading process. File downloading and all other processes are controlled by the downlink state machine.

The state of the uplink is indicated by the ul state variable.

The uplink state machine processes the following events:

```
    EVENT: User Requests Uplink
    EVENT: Data Link Terminated
    EVENT: Receive UL_GO_RESP
    EVENT: Receive UL_ERROR_RESP
    EVENT: Receive UL_ACK_RESP
    EVENT: Receive UL_NAK_RESP
```

Where the "EVENT: DEFAULT" is indicated, this includes all events on the above list which have not already been processed.

```
State UL UNINIT {

    EVENT: User Requests File Upload
        Refuse.

    EVENT: DEFAULT
        ignore.
}
```

```
State UL CMD OK {

    EVENT: User Requests File Upload {
        Transmit UL_CMD packet, setting <continue file no> if necessary.
        ul state <- UL WAIT
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        ul state <- UL UNINIT
    }
}

State UL WAIT {

    EVENT: Receive UL GO RESP packet. {
        Associate <server-file number > with the file.
        Mark the file CONTINUE.
        ul state <- UL DATA
    }

    EVENT: Receive UL_ERROR_RESP packet. {
        If error is unrecoverable, mark the file IMPOSSIBLE.
        ul state <- UL CMD OK
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        ul state <- UL UNINIT
    }
}

State UL_DATA {
    If there is data to send {
        Transmit DATA packet.
        ul state <- UL DATA.
    }
    else {
        Transmit DATA_END packet.
        ul state <- UL END.
    }

    EVENT: Receive UL_NAK_RESP packet. {
        if error is unrecoverable mark file IMPOSSIBLE.
        Transmit DATA_END packet.
        ul state <- UL CMD OK.
    }
```

```
    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        ul state <- UL UNINIT
    }
}

State UL END {

    EVENT: Receive UL_ACK_RESP packet. {
        Mark file COMPLETED.
        ul state <- UL CMD OK
    }

    EVENT: Receive UL_NAK_RESP packet. {
        If error is unrecoverable mark file IMPOSSIBLE.
        ul state <- UL CMD OK
    }
}
```

DOWNLINK STATE MACHINE

The client's downlink state machine processes the following events:

```
    EVENT: User requests download
    EVENT: User requests directory
    EVENT: User requests selection
    EVENT: User requests abort
    EVENT: Receive DL_ABORTED_RESP packet.
    EVENT: Receive DL_COMPLETED RESP packet.
    EVENT: Receive SELECT_RESP packet.
    EVENT: Receive LOGIN_RESP packet.
    EVENT: Receive DATA packet.
    EVENT: Receive DATA END packet.
    EVENT: Receive DL ERROR RESP packet.
```

The only user request which generates and event when not in DL_CMD_OK state is "user requests abort", which is processed in the DL_DATA state. All other requests are ignored (perhaps queued) until dl state is DL CMD OK.

```
State DL UNINIT{
}

State DL CMD OK{

    EVENT: User requests download {
        Transmit DL CMD packet.
        dl state := Di WAIT.
    }

    EVENT: User requests directory {
        Transmit DIR_LONG CMD or DIR SHORT_CMD packet.
        dl state := DL_DIR      WAIT.
    }
```

```
    EVENT: User requests selection {
        Transmit SELECT_CMD packet.
        dl state := DL SEL.
    }

    EVENT: User requests abort {
        ignore.
        dl state : = DL CMD OK
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl state <- UL UNINIT


State DL WAIT{

    EVENT: Receive DL_ERROR_RESP packet {
        dl state <- DL CMD OK.
    }

    EVENT: Receive DATA packet. {
        Store data.
        Mark file INCOMPLETE.
        dl state C- DL DATA.
    }

    EVENT: Receive DATA_END packet. {
        If file at client is ok {
            Mark file DATA OK.
            Save <continue_offset > equal to file length.
            Transmit DL ACK CMD.
        }
        Else {
            Mark file INCOMPLETE.
            Save <continue offset > of 0.
            Transmit DL NAK CMD.
        }

        dl state <- DL END.
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated*'
            Terminate data link.
        dl state <- UL UNINIT
    }
}
```

```
State DL DATA{

    EVENT: User requests abort {
        Save current offset as <continue_offset >.
        Transmit DL_NAK_CMD.
        dl state c - DL ABORT.
    }

    EVENT: Receive DATA packet. {
        Store data.
        dl state <- DL DATA.
    }

    EVENT: Receive DATA_END packet. {
        If file is ok at client
            Transmit DL ACK CMD packet.
        Else
            Transmit DL_NAK_CMD packet.
        dl state <- DL END
    }

    EVENT: DEFAULT{
        Save current offset as <continue offset >.
        if EVENT is not "data link term&ted"
            Terminate data link.
        dl state <- UL UNINIT
    }
}

State DL END{

    EVENT: Receive DL_ABORTED_RESP packet. {
        dl state <- DL CMD OK.
    }

    EVENT: Receive DL_COMPLETED RESP packet. {
        Mark file COMPLETE.
        dl state <- DL CMD OK.
    }

    EVENT: DEFAULT{
        if EVENT is not "data link terminated"
            Terminate data link.
        dl state <- UL UNINIT
    }
}

State DL ABORT{

    EVENT: Receive DATA packet. {
        Ignore.
        dl state <- DL ABORT.
    }
```

```
    EVENT: Receive DATA END packet. {
       dl state <- DL END.-
    }

    EVENT: DEFAULT{
       if EVENT is not "data link terminated"
           Terminate data link.
       dl state <- UL UNINIT
    }
}

State DL DIR WAIT{

    EVENT: Receive DATA packet. {
       Send directory data to user.
       dl state <- DL DIR DATA.
    }

    EVENT: Receive DL_ERROR_RESP packet. {
       dl state <- DL CMD OK.
    }

    EVENT: DEFAULT{
       if EVENT is not "data link terminated"
           Terminate data link.
       dl state <- UL UNINIT
    }
}

State DL DIR DATA{

    EVENT: Receive DATA packet. {
       Send directory data to user.
       dl state <- DL DIR DATA
    }

    EVENT: Receive DATA_END packet.{
       dl state <- DL CMD OK
    }

    EVENT: DEFAULT{
       if EVENT is not "data link terminated"
           Terminate data link.
       dl state <- UL UNINIT
    }
}
```

```
State DL SEL{

   EVENT: Receive DL ERROR RESP packet. {
      dl state := D L UNINIT
   }

   EVENT: Receive SELECT_RESP packet. {
      dl state := DL UNINIT
   }

   EVENT: DEFAULT{
      if EVENT is not "data link terminated"
          Terminate data link.
      dl state <- UL UNINIT
   }
}
```

APPENDIX D - Error Codes

Value	m e
1	ER_ILL_FORMED CMD
2	ER BAD CONTINUE
3	ER-SERVER_FSYS
4	ER-NO_SUCH_FILE_NUMBER
5	ER_SELECTION_EMPTY
6	ER MANDATORY FIELD MISSING
7	ER-NO PFH -
8	ER-POORLY FORMED SEL
9	ER-ALREADY LOCKED
10	ER-NO_SUCH_DESTINATION
11	ER SELECTION E M P N
12	ER-FILE_COMPLETE
13	ER-NO_ROOM
14	ER_BAD_HEADER
15	ER HEADER_CHECK
16	ER-BODY CHECK