APPENDIX -1- Commands of the line editor EDIT (FBBDOS)

The FBBDOS offers a small line editor allowing the process of ASCII files. It can help the distant SysOp maintening the system files of the software.

The commands are straightforward, and for a better efficiency, they can be chained on a single line.

List of the commands:

? : Gives the list of the editor's commands.

A : Adds one line after the current line. The text of the line follows the command, and must be terminated with the character /. The character / can be imcluded into the character string provided it is preceded by a character \. To include a \ character into the string, this character must be sent twice (\\). The line pointer is on the inserted line.

B : places the line pointer at the beginning of the file.

E : places the line pointer at the end of the file.

F : Search for the first occurence of the character string as specified after the command, starting from the current line. The character string should be terminated by the character /. The character / can be inserted into a string under the condition that a \ character is leading. The \ character must be sent twice if it is to be considered as a valid character. The line pointer stands on the line where the string has been found.

I : Inserts a line ahead of the current line. The text of the line follows the command and must be terminated by the character /. The character / can be included into the string provided it is preceded by a character \. The character \ must be sent twice if it is to be considered as a valid character. The line pointer stands on the inserted line.

K : Suppresses the number of lines specified before the command, starting from the current line. If the number is not stated, the value 1 is taken as a default.

L : moves the pointer of the number of lines specified before the command. This number can be positive or negative.

N : Enables or disables the numbering of the lines.

P : Displays the number of lines specified before the command, starting at the current line.

R : Search for the first occurrence of the character string specified after the command, and replace it by the second character string. The character strings must be terminated by a character /. The / character can be included provided it is preceded by a \ character. The \ character must be sent twice if it is to be taken into account as a valid character. The line pointer stays on the modified line.

S : Saves the modified file.

Q : Exits the editor. Take care, the file is not automatically saved. Think about doing it before you quit.

Examples :

```
EDIT>B5L10P
```

places the pointer at the beginning of the file, advances 5 lines, displays 10 lines.

EDIT>B4L6K-2L10P

Places the pointer at the beginning of the file, advances 4 lines, deletes 6 lines, steps back 2 lines and displays 10 lines.

EDIT>BFBonjour/K-2L5P

Sets the pointer at the beginning of the file, searches for "Bonjour", suppresses the line (containing "Bonjour"), steps back 2 lines, and displays 5 lines.

EDIT>RBonjour/Au revoir/-2L5P

Searches for "Bonjour", replaces by "Au revoir", steps back 2 lines and displays 5 lines.

EDIT>BFBonjour/IC'est une new line with a \/inside/-1L3P

Sets the pointer at the beginning of the file, searches for "Bonjour", inserts a new line, steps back 1 line, and displays 3 lines. This lines includes a character /. A leading \ indicates that it is to be conside red as a valid character and not as a delimiter.

EDIT>SQ

Saves, and quits the editor.

APPENDIX -2- Commands corresponding to the PK232 host-mode (as per F6AIW)

8B	8BITCONV	AU	AAB	AB	ABAUD	AG	ACHG	AA	ACRDISP
AK	ACRPACK	AT	ACRRTTY	AE	ADDRESS	AD	ADELAY	AI	ALFDISP
AP	ALFPACK	AR	ALFRTTY	AL	ALIST	AM	AMTOR	AC	ARQ
AO	ARQTMO	AS	ASCII	AY	ASPECT	AW	AWLEN	AV	AX25L2V2
AX	AXDELAY	AH	AXHANG	BA	BAUDOT	BE	BEACON	BI	BITINV
BK	BKONDEL	BT	BTEXT	CL	CANLINE	CP	CANPAC	CX	CASEDISP
CU	CBELL	CC	CCITT	CF	CFROM	CB	CHCALL	CD	CHDOUBLE
CH	CHSWITCH	CK	CHECK	CQ	CMDTIME	CM	CMSG	CI	CODE
CN	COMMAND	CE	CONMODE	CO	CONNECT	CY	CONPERM	CG	CONSTAMP
CI	CPACTIME	CR	CRADD	CT	CTEXT	CW	CWID	DS	DAYSTAMP
DA	DAYTIME	DC	DCDCONN	DL	DELETE	DF	DFROM	DI	DISCONNE
DW	DWAIT	EA	EAS	EC	ECHO	ES	ESCAPE	FA	FAX
FN	FAXNEG	FE	FEC	FL	FLOW	FR	FRACK	FS	FSPEED
FU	FULLDUP	GR	GRAPHICS	HB	HBAUD	HD	HEADERLN	HI	HID
HO	HOST	HP	HPOLL	ID	ID	IL	ILFPACK	IO	IO
JU	JUSTIFY	KI	KISS	LR	LEFTRITE	LO	LOCK	MX	MAXFRAME
MB	MBX	MC	MCON	MD	MDIGI	MM	MEMORY	MI	MFILTER
MF	MFROM	MH	MHEARD	MN	MONITOR	MO	MORSE	MP	MSPEED
MR	MRPT	MS	MSTAMP	MT	MTO	MA	MYALIAS	ML	MYCALL
MG	MYSELCAL	MK	MYALTCAL	NE	NEWMODE	NO	NOMODE	NR	NUCR
NF	NULF	NU	NULLS	OK	OK	OP	OPMODE	PA	PACKET
PL	PACLEN	PT	PACTIME	PR	PARITY	PS	PASS	PX	PASSALL
PE	PERSIST	PP	PPERSIST	PC	PRCON	PF	PRFAX	PO	PROUT
PY	PRTYPE	RW	RAWHDLC	RB	RBAUD	RC	RCVE	RE	RECEIVE
RX	RXREV	RD	REDISPLA	RL	RELINK	RS	RESET	RP	RESPTIME
RT	RESTART	RY	RETRY	RF	RFEC	SE	SELFEC	SP	SENDPAC
SI	SIGNAL	SL	SLOTTIME	SQ	SQUELCH	SR	SRXALL	ST	START
SO	STOP	TB	TBAUD	TC	TCLEAR	TM	TIME	TR	TRACE
TW	TRFLOW	TI	TRIES	TD	TXDELAY	TF	TXFLOW	TX	TXREV
UN	UNPROTO	UR	USERS	US	USOS	VH	VHF	WI	WIDESHFT
WO	WORDOUT	WR	WRU	XW	XFLOW	XM	XMIT	XO	XMITOK
XF	XOFF	XN	XON						

APPENDIX -3- Process number and on-line help identification.

The various processing functions contained in the software are identified by three numbers displayed in the status banner. The first of those three numbers is the main process system (BBS, FBBDOS, Satellite Computation, etc...), the second number is the process function (in the BBS, list, message sending, etc..) and the third number is a sub-function (record of the message title, message, etc..)

A complete description of these numbers would be useless and time consuming. They are mainly used for debugging purpose. The first number is also used to identify the help block out of the x.HLP file. Upon receipt of the "?" or the "H", the software searches the x.HLP file for a line of corresponding to the language in use, and of the format @@ number word in which "number" stands for the current processing level, and "word" stands for the word following the command "?" or "H".

Example : you are inside FBBDOS, and you type in the command "? EDIT", the help block searched for must begin with the line:

```
@@ 9 EDIT
```

It may happen that a block matches several search keywords. It is enough to specify the various words separated by the character "|" (vertical bar), WITH NO SPACE.

```
@@ 9 EDIT|EDITEUR
```

List of the processing levels :

0 Connection.
2 Qra-Locator.
3 Statistics.
4 Informations.
5 Nomenclature.
6 Satellite Orbital Computation.
9 FbbDos.
11 Telefone Modem
14 BBS.
15 Forward.
16 GatewaySysop page.
17 YAPP.
18 Conference.

APPENDIX -4- Recording a message.

A message can be left by a user or within a forwarding connection. The recording mechanism is always the same.

The recording command is always like :

Sx desti @ bbs < exped $ ident + filename

Only recipient field is mandatory, all other fields are optional.

Appending a filename is a possibility reserved to the sysop. The name must be complete, including logic unit and complete path (C:\FBB\SYSTEM\TEST.TXT).

When receiving the command line, a first test checks if a route exists when a route has been specified, or if the message must be automatically routed when no route was specified.

The title of the message is then asked to the user.

If the title is a missing, the message is canceled and the user returns to the main menu.

The text of the message is then asked to the user.

The software checks possible preamble lines. These lines give informations on the previous BBS having routed this message. They all begin by R: on first column. The BBS callsign is given behind the @ character within the preamble line. All adjacent BBS mentioned in this preamble will be included in the "already forwarded" list, and will not be concerned by this message. This list specific to each message can be displayed with the $ or FN command followed by the message number.

When receiving a /EX in first column, or a Ctrl Z, a message number is then assigned, The BID (or MID if private) and the list of adjacent BBS concerned by this message are created. All these informations are sent to the user when acknoledging the message.

In case of disconnection before the /EX or Ctrl Z, the whole message will be lost, and the texts already stored are deleted.

All informations about the message (sender, recipient, route, MID, title, etc...) are stored in the DIRMES.SYS file. The text of the message is stored in a sub-directory of the MAIL directory. The sub-directory is MAILn where n is the last digit of the message number. The name of the file corresponds to the message number 123 is M_000123.MES, the number is 6 digits wide, in this case it is in the sub-directory MAIL3.

The message number uses a long integer (32 bits), the number boundary is very far (more than 4 billions !).

APPENDIX -5- Tricks and tips.

This rubric is yours, more than mine. I'll try to insert there all tricks you will tell me.

A/ Using DesqView (c).

There is no particular problem when using DesqView. The minimum window size is 500 KB. You MUST use communication drivers, like ESS, COMBIOS or MBBIOS, as the software does not dispose of the whole process time.

B/ Communication errors displaying.

An error counter can be displayed just right of the date, on the first line of the screen. If you are using TNC2 with WA8DED software, these errors can be minor, but with PK232, error recovery is more difficult, and the system may reboot.

With a correct operation of your system, this counter will not appear, or exceptionnaly. If errors are displayed, they can result from :

- Using DOS 4.0 or 5.0 : The keyboard driver of these versions is very slow. You must use the ESS driver for RS232 (or COMBIOS).

- A too hight baudrate, or RS232 defective cables. The baudrate can be selected down to 4800 Bds. It is not a good idea to go down 4800 Bds, as the performance of the software should be lower.

- Change the LM324 fitting out some TNC RS232 line drivers by a TL074 or TL084.

- HF detection in the TNC. Errors and resynchronizations will appear when the transmitter is running. There is no real cure, you must investigate.

You can also use communication drivers like ESS, COMBIOS or MBBIOS if you are not still using them.

C/ Repetition of the last message number.

The last message number displayed, read, killed, etc... can be utilised again with the # (pound) character. This short-cut allows as for an example to read a message after a list or to suppress it just after its reading.

Example :

```
F6FBB BBS > R 12351
The message is displayed ...
F6FBB BBS > K #
Message #12351 killed.
F6FBB BBS >
```

APPENDIX -6- FBB Forward Protocole.

FBB software includes two forward protocoles. The first one is standard with MBL/RLI protocole. The second one was developped to allow efficiency, particularly on long links where propagation time of data are long. The exchange of commands is reduced to a minimum, and not acknoledged to get time. The data transfer direction is changed every block of data, a block of data holding up to five messages. This uses the "pipeline" effect of long links (Nodes and digipeaters), and gain some time over short links (HF...).

FBB protocole is very simple in its principle. It is based on MID/BID usage. The identification is made by the F letter in the SID (system type identifier contained in square brackets). All command lines must start in first collumn with the 'F' character. All command lines are ended by a return (CR) character.

Suppose I call another BBS to forward some mail. When I connect another BBS using FBB protocole, I will receive the SID followed by a text and the prompt (">"). If the SID contains the F flag, I will send immediately my SID and the first proposal.

Proposals looks like :

```
FB P F6FBB FC1GHV FC1MVP 24657_F6FBB 1345
F> HH
```

FB : Identifies the type of the command (proposal)
P : Type of message (P = Private, B = Bulletin).
F6FBB : Sender (from field).
FC1GHV : BBS of recipient (@field).
FC1MVP : Recipient (to field).
24657_F6FBB : BID ou MID.
1345 : Size of message in bytes.
F> : End of proposal.
HH is optional. It is the checksum of the whole proposal in hexadecimal.

ALL the fields are necessary. This kind of command must hold seven fields. If a field is missing upon receiving, an error message will be send immediately followed by a disconnection.

A proposal can handle up to five FB command lines. If the total size of messages seems to be too important, the proposal can handle less lines. In FBB software, a parameter is defined in INIT.SRV file to tell the maximum size of the message block. It is set by default to 10KB.

Exemple of proposal :

```
FB P F6FBB FC1GHV.FFPC.FRA.EU FC1MVP 24657_F6FBB 1345
FB P FC1CDC F6ABJ F6AXV 24643_F6FBB 5346
FB B F6FBB FRA FBB 22_456_F6FBB 8548
F> HH
```

This proposal is limited to three FB lines, as the amount of messages overran the 10KB limit.

When receiving the proposal, the other BBS will reject, accept or defer the message. This command is made by a FS line :

```
FS -+=
```

This means :

- I don't want the first message (-).
- I need the second message (+).
- I defer the third message, as I'm still receiving it.

It should interesting to defer a message if you are still receiving it on a other channel, or if you think that the size is to big, or for another reason. The message should be proposed again at the next connection.

FS line MUST have as many +,-,= signs as lines in the proposal.

When receiving the FS lines, I can send the block of messages. Each message is made with the title on the first line, the text, and a Ctrl Z in the last line. The is no blank line between the messages.

```
Title of 2nd message
Text of 2nd message
.....
^Z
```

When the other BBS has received all the asked messages, it acknoledges by sending its proposal, and the system is reversed.

If it has no message to send, it only sends a line :

FF

This line must not to be followed by a F>.

If the other hand has no message, it sends a line :

FQ

and asks for the disconnection.

Example :

```
F6FBB                          FC1GHV
-----------------------------------------------------------------

Connects FC1GHV

                               Connected

                               [FBB-5.11-FHM$]
                               Bienvenue a Poitiers, Jean-Paul.
                               >

[FBB-5.11-FHM$]     (F6FBB has the F flag in the SID)
FB P F6FBB FC1GHV.FFPC.FRA.EU FC1MVP 24657_F6FBB 1345
FB P FC1CDC F6ABJ F6AXV 24643_F6FBB 5346
FB B F6FBB FRA FBB 22_456_F6FBB 8548
F> HH

                               FS +-+ (accepts the 1st and the 3rd).

Title 1st message
Text 1st message
......
^Z
Title 3rd message
```

```
Text 3rd message
......
^Z
                                FB P FC1GHV F6FBB F6FBB 2734_FC1GHV 234
                                FB B FC1GHV F6FBB FC1CDC 2745_FC1GHV 3524
                                F> HH

FS -- (Don't need them, and send immediately the proposal).
FB P FC1CDC F6ABJ F6AXV 24754_F6FBB 345
F> HH
                                FS + (Accepts the message)

Title message
Text message
......
^Z
                                FF (no more message)

FB B F6FBB TEST FRA 24654_F6FBB 145
F> HH
                                FS + (Accepts the message)

Title message
Text message
......
^Z
                                FF (still no message)

FQ (No more message)

Disconnection of the link.
```

In this example, FBB protocole is used as the two BBS were identified by the F flag in the SID. If F6FBB had sent the SID [FBB-5.11-MH$] when answering FC1GHV, the protocole should be the standard MBL/RLI.

All callsigns are only examples !

APPENDIX -7- Extension to the protocole. Compressed forward FBB.

The protocole utilized for the transfer of ascii files compressed is an extension to the existing protocole. The compressed forward is validated by the presence of the letter B in the SID [FBB-5.12-BFHM$]. The transfer of compressed files can only take place under FBB protocole. The presence of the letter B in the SID without the F letter will remain without effect.

The only difference as regard to the standard protocol is the submit line. It can specify the type of data contained in the compressed message. FA means that the transfer will be an ascii compressed message. FB means that the message will be a binary compressed file (this last possibility is not yet implemented in the version 5.12).

The submission of an ascii message will be in the form :
FA P FC1CDC F6ABJ F6AXV 24754_F6FBB 345

The submission of a binary file will be in the form :
FB P FC1CDC F6ABJ F6AXV 24754_F6FBB 345

The transfered data are of a specific format. The transfer will be done in binary mode. This last one is derived of the YAPP protocol which is very reliable. All transfer is made of a header, a block of data, an end of message and a checksum. Each transfer is equivalent to the transfer of one message of the standard protocol and shall not be followed by a control Z, the end of file specifier is defined in another way.

Format of header for an ascii compressed message (submission FA) :

<SOH> 1 byte = 01 hex
Length of the header 1 byte = Length from the title,
including the two <NUL> characters.
Title of the message 1 to 80 bytes
<NUL> 1 byte = 00 hex
Offset 1 to 6 bytes
<NUL> 1 byte = 00 hex

Format of header for a binary compressed file (submission FB) :

<SOH> 1 byte = 01 hex
Length of the header 1 byte = Length from the filename,
including the two <NUL> characters.
Name of the file 1 to 80 bytes
<NUL> 1 byte = 00 hex
Offset 1 to 6 bytes
<NUL> 1 byte = 00 hex

As to follow the french regulation, the title of the message or the file name are transmitted in ascii, not compressed.

The offset is also transmitted in ascii and specifies the offset at which the data should be inserted in the file (in case of a fragmented file). In the version 5.12, this parameter is not utilized and is always equal to zero.

A data block contains from one to 256 bytes. It begins by two bytes which specify the format.

Data block format :

<STX> 1 byte = 02 hex
Number of data 1 byte = 00 to ff hex. (00 if length = 256 bytes).

Data bytes 1 to 256 bytes

The last data block is followed by the end of file specifier and the checksum.

End of file specifier format :

<EOT> 1 byte = 04 hex
Checksum 1 byte = 00 a ff hex

The checksum is equal to the sum of all the data bytes of the transmitted file, modulo 256 (8 bits) and then two's complemented.

The checking of the checksum is very simple :

The sum of the datas from the file and the checksum received modulo 256 (anded with FF) shall be equal to zero.

In case of a checksum error, the message or the file is not taken to account and the system issues a disconnect request after having sent the comment :

*** Checksum error

APPENDIX -8- Format of the ACK messages.

The ACK messages on receiving have a simple and compact format. The aim is to have a message as short as possible in order to avoid an unnecessary usage of the network.

The title of the message is the title of the original message with a leading "ACK:". Example :

ACK:Title of the original message.

These ACK messages are true messages strickly speaking. They carry the origin, the destination, the route and the MID but they are of a particular type, the type A (private are of type P, bulletins of type B, etc...). This difference allows the routing of these messages without the lines "R:". This is done again with the aim of avoiding an excessive load by data which are of no use in this case.

To keep the compatibility with the existing forwarding protocol, the type of these messages is changed to P (private) if the receiving BBS of the forwarding does not know the type of ACK messages (specified in the SID [FBB-5.12-ABFHM$] by the letter A). In this case, the ack message will continue on its route as a private message.

The ACK messages are of the following form :

```
ACK:Message test. <-- Title of message
Msg FD1CDC@F6FBB - 22-dec 17:28z <-- Text of message
```

It tells that the message that you had sent to FD1CDC at F6FBB and whose title is "Message test" has been received in the BBS F6FBB on 22 dec at 12:28 GMT.

APPENDIX -9- Replacement characters or wildcards.

Most of the search commands or list commands and some configuration files as well, accept replacement characters or wildcards.

Character	Replaces
@	a letter
?	an alphanum character (letter or number)
=	a printable character
#	a numeral character or the # character
*	a string of printable characters.
&	a dot followed by printable characters. (equiv to .*)

APPENDIX -10- Programming technics for servers.

The servers are exec programs (.COM or .EXE). They are compact and fast. They will work as the function of the messages which are addressed to them.

They should be compact because the available memory to run their application is limited (check the information Ok:nnnn in the status window). They should be fast because they are executed in the MsDos environment which is not multi-task.

The programming language can be of any kind provided that it could be compiled and that it is able to read parameters which are given appended in the command line.

I wrote three servers in TurboC but I have no equivalent in TurboPascal or in TurboBasic, since I usually don't write in these languages. The working principle remains always the same whatever language is utilized.

The program is called with the following manner from the MsDos (Example for the REQDIR.COM command) :

```
C> REQDIR.COM TEMP.$$$
```

TEMP.$$$ is the name of the file in which the message addressed to REQDIR is located. It is necessary to read the name of this file in the command line, as the one can change from one call to another.

The file TEMP.$$$ contains the message with the following format :

```
SP REQDIR < F6FBB
Title of message
Text of message line 1
Text of message line 2
...
Text of message last line.
/EX
```

The server should then eventually work as a function of the contents of this message.

The server can read and make use of the configuration file of the BBS software (in particular INIT.SRV) to execute its process.

If the server generates a return message, it should be APPENDED to the incoming mail file to the BBS. The name of this file can be found in INIT.SRV. Take care : it is necessary to open the incoming mail file in APPEND as to add the answer at the end of the file. If it is not done this way, the messages which could be waiting in this file are destroyed.

The incoming mail file is tested each and every minute, except in the case of the usage of a service, where it is tested right after.

The format of the messages in the incoming mail file is identical to the format of the file given to the server. Several messages can be written sequentially in the file. There should not be blank lines or separations between the messages. The routing fields (@ field), and the originator (< field) should mandatory be specified. The originator field is the callsign of the BBS which is taken from the INIT.SRV file.

Example of server REQFIL written in C language.

```
/*
 * REQFIL.C Server example.
 *
 * This server is called with a command line like this :
 *
 * REQFIL.COM FILE
 *
 * FILE is the filename of the message to be answered.
 *
 *
 * This server answers to a message like this :
 *
 * SP REQFIL < FC1EBN
 * TEST.TXT @ F6ABJ
 * Text is not necessary
 * /EX
 *
 * by a message like this
 *
 * # <- This is a local message
 * SP FC1EBN @ F6ABJ < F6FBB <- command line
 * Req File : TEST.TXT <- subject
 * Contents of the file <- text
 * etc.....
 * /EX <- end of text (must be in 1st column)
 *
 * Appent to mail in bbs file.
 *
 *
 * The server receives from FBB software 1 argument :
 *
 * argv[1] = Name of the file including the message received from
 * FBB software.
 *
 * ============================================
 * The server must APPEND its answer to MAIL.IN
 * file to avoid destroying existing mail.
 * ============================================
 *
 * As this server opens the INIT.SRV file, it must be in the same
 * directory.
 *
 */

 #include <stdio.h>
 #include <fcntl.h>
 #include <sys/stat.h>

 /* Offsets of parameters from INIT.SRV */

 #define BBS_CALL 1
 #define USER_DIR 8
 #define MAIL_IN 14

 main(int argc, char **argv)
 {
 #define LINE 80
   int end = 0;
   int index = 0;
   FILE *fptr;
```

```
char buffer[LINE];
char sender[LINE];
char route[LINE];
char file[LINE];
char bbs_call[LINE];
char base_dir[LINE];
char mail_in[LINE];

if (argc != 2) exit(1);            /* Check the number of arguments */

/* The first task is to open and then read the message */

fptr = fopen(argv[1], "rt") ;      /* Open the received message */
if (fptr == NULL) exit(1);

fgets(buffer, LINE, fptr);         /* Read the command line */
sscanf(buffer, "%*s %*s %*s %s\n", sender);

*file = *route = '\0';
fgets(buffer, LINE, fptr);         /* Read the subject */
strupr(buffer);                    /* Capitalize */

/* Scan dir and route */
sscanf(buffer, "%[^@\n]%[^\n]", file, route);

fclose(fptr);                   /* All needed is read in the message */

/* We must get some informations from the INIT.SRV file */

fptr = fopen("INIT.SRV", "rt"); /* Open the file */
if (fptr == NULL) exit(1);

/* Scan the file to get the requested lines. */
while (!end) {
  fgets(buffer, LINE, fptr) ;
  if (*buffer == '#') continue; /* Comments ignored */

  switch (++index) {

  case BBS_CALL:
    sscanf(buffer,"%[0-9A-Za-z]", bbs_call);
    break; /* Callsign */

  case USER_DIR:
    sscanf(buffer,"%s\n", base_dir);
    break; /* Users directory */

  case MAIL_IN :
    sscanf(buffer,"%s\n", mail_in);
    end = 1; /* Mail in file */
    break;

  }
}

fclose(fptr);

/* Append the answer to mail in file */
/* Mail in file is opened in appent text mode */

if (fptr = fopen(mail_in, "at")) {

  /* Tell that this is a message from this BBS */
```

```
    fprintf(fptr, "#\n");

    /* Send command line */
    fprintf(fptr, "SP %s %s < %s\n",
    sender, route, bbs_call);

    /* Send subject and requested file */
    send_file(fptr, base_dir, file);

    /* Send end of message */
    fprintf(fptr, "/EX\n");

    /* That's all ! */
    fclose(fptr);
  }
  exit(0);                                /* Tell BBS all is correct */
}

int points(char *ptr)   /* Looks for a ".." sequence in the path */
{
  while (*ptr) {
    if ((*ptr == '.') && (*(ptr+1) == '.')) return(1);
    ++ptr;
  }
  return(0);          /* ".." not fond ! */
}

send_file(FILE *fptr, char *base_dir, char *filename)
{
#define BUF_SIZE 1000
  int fd;
  int nb;
  char path[256];
  char buffer[BUF_SIZE];
  char last_char;

  sprintf(path, "%s%s", base_dir, filename);       /* Complete path */

  fprintf(fptr, "ReqFil 1.1 : %s\n", filename);       /* Subject */

  if ((!points(path)) && ((fd = open(path, O_RDONLY | O_TEXT)) != -1)) {
    while (nb = read(fd, buffer, BUF_SIZE)) {
      fwrite(buffer, nb, 1, fptr);
      last_char = buffer[nb-1];
    }
    close(fd);

    /* Be sure /EX will be in first column */
    if (last_char != '\n') fputc('\n', fptr);

  }
  else fprintf(fptr, "File not found !\n");
}
```

APPENDIX -11- Contents of directories.

These files and directories are mandatory. Other files may reside but will be created by the system or by the sysop.

Directory FBB :

```
[BIN]         BAT, COM and EXE files.
[BINMAIL]     Compressed messages.
[DOCS]        Files used for DOCS command.
[OLDMAIL]     Archives of messages.
[MAIL]        Ascii messages.
[PG]          COM and EXE files for PG command.
[SYSTEM]      System files.
EPURMESS INI  Init file for EPURMESS.
INIT     SRV  Init file for SERV.
NEWDOC   SYS  Configuration for NEWDOC server.
```

Directory BIN :

```
APPEL    BAT  Batch file running the software.
CLEANUP  COM  Program to clean MAIL and BINMAIL directory.
CUT      COM  Copy/paste utility.
EPURMESS COM  Message processing.
EPURWP   COM  White Pages processing.
ESS      COM  RS232 driver.
ESSKAM   COM  KAM RS232 driver.
FBBIOS   COM  MODEM driver.
FV       COM  FV shareware used for LIST command in FbbDos.
LOGSTAT  EXE  Log statistics.
MAINTINF COM  INF.SYS maintenance program.
MAINTREJ COM  REJET.SYS maintenance program.
MKINF514 COM  Change INF.SYS for 5.13 to 5.14
NEWDOC   COM  NEWDOC server.
REQDIR   COM  REQDIR server.
REQFIL   COM  REQFIL server.
SATUPDAT EXE  Automatic update of satellite database.
SERV     EXE  FBB software EXE file.
SETUSER  COM  set a user owner of a FbbDos file.
SLEEP    EXE  waits for some seconds.
TLABEL   COM  processes label file of FbbDos.
```

Directory BINMAIL :

```
[MAIL0]        [MAIL1]        [MAIL2]       [MAIL3]         [MAIL4]
[MAIL5]        [MAIL6]        [MAIL7]       [MAIL8]         [MAIL9]
```

Directory MAIL :

```
[MAIL0]        [MAIL1]        [MAIL2]       [MAIL3]         [MAIL4]
[MAIL5]        [MAIL6]        [MAIL7]       [MAIL8]         [MAIL9]
```

Directory SYSTEM :

```
[FWD]         Directory for FORWARD.SYS includes
[LOG]         Directory for FBBLOG.nn files.
[LANG]        Directory for .TXT .HLP .ENT language files.
[SAT]         Directory for satellite files
[WP]          Directory for White pages database
BALISEn  SYS  Beacon file.
BBS      SYS  BBS callsigns
CRON     SYS  Hour work.
INITTNCn SYS  Initialization of TNC.
FORWARD  SYS  Forwarding description.
LANGUE   SYS  Language description
MAINTn   SYS  Housekeeping text.
```

```
MEMO     SYS  CtrlF0 to CtrlF9 aliases
PASSWD   SYS  Password file.
PORT     SYS  Port description.
PROTECT  SYS  Directories protection.
REJET    SYS  Messages reject.
SWAPP    SYS  Swap file.
```

APPENDIX -12- CONNECTION FILTERING.

FBB software allows filtering on connection. Filtering is not done by the BBS software but by external programs developped by users.

Connection filter may be interactive and allows to incorporate some features like dedicated information for predifined callsigns, password filtering, etc...

I did not develop such programs, but this is an open door to many applications.

The C_FILTER program must be found by the PATH of MsDos. Its extension can be COM or EXE, and it must be little and fast as multitasking is stopped during the activity of this program. If this program is not found, it will not be called until the BBS is rebooted.

When receiving the connection, the C_FILTER program (if found) is called with some arguments including a level number. This number is incremented each time the program is called in the same connection session. The first time the level number will be 0.

The line arguments given to the C_FILTER program are :

- Callsign (format as F6FBB-8).
- Level number (0 is the first time, up to 99).
- Flags of the user (binary number as user's mask of INIT.SRV).
- New : Flag indicating if the user is unknown in the BBS database.
- Record number of the user in INF.SYS.
- Received data (each word is a new argument).

The C_FILTER program ends with an exit value. This value is very important and tells the BBS what to do :

0 : end of session and return to the BBS menu.
1 : the program will be called again and the level number is incremented.
2 : the user will be disconnected.

```
/*
 * C_FILTER.C
 *
 * Connection filter for FBB BBS software. (C) F6FBB 1991.
 *
 */

#include <stdio.h>

/*
 * Connexion filter called for each connection.
 * All datas sent to stdout will be sent to the user.
 *
 * Filter is called with some arguments on the command line :
 * C_FILTER CallSign Level Flags New Record ReceivedData....
 *
 * The return value tells the BBS if C_FILTER must be called again or not
 * 0 if the BBS can go on,
 * 1 if the C_FILTER must be called again
 * 2 if the user must be disconnected.
 *
 * Callsign is in the FORM CALLSIGN-SSID (F6FBB-0).
```