ERCC Communications Notes

Network

NETWORK

Protocols

PROTOCOLS

note	author	date	title/subject

TCPs/Workstations

TCPs/WORKSTATIONS

TCP/WORKSTATIONS note TCP/WS-1

TCP/WS DOCUMENTATION

Scott Currie
23rd June 1980

The file ERCM08.NSICONFIGL contains a list of the hardware and software configuration for every NSI TCP/Workstation in the RCO Network. As it is rather large I shall not distribute it, but the file is permitted.

Detailed documentation on the structure of TCPs is available from me - on a loan basis.

Front Ends

FRONT ENDs

note	author	date	title/subject

Nodes

NODEs

note	author	date	title/subject

PSS

PSS

PSS

note: PSS-2

Requirements of a PSS service

John Butler
11th August 1980

*   *   *

This note is a copy of a discussion paper prepared for the Joint Network Team in March, which formed part of the submission to the JNT for a Gateway machine.

*   *   *

This paper examines some of the requirements of an ERCC PSS service and how these may be met. Points for discussion are:

Facilities provided

Protocols

Availability

Reliability

Ease of development

Ease of maintenance

Security

Size

Facilities provided

The ERCC PSS service must initially provide for interactive access to and from remote sites through PSS and must ultimately allow RJE traffic and file transfer. It is anticipated that interactive traffic will form the bulk of traffic through PSS.

Protocols

The service must allow terminals using RCO protocols to use PSS and vice versa, and so must perform the following protocol conversions:

RCO HDLC <-> X25 level 2

RCO NSI <-> X25 level 3

RCO ITP <-> X3/28/29

RCO ITP <-> TS29 + Transport Station

It must also be transparent to RCO ITP, FTP and some RJE protocol yet to be specified. It must also allow for the possibility of RCO NET moving to X25 at some future time.

Availability

The PSS service must be available to the whole of RCONET and should not depend on a particular mainframe to provide a service.

Reliability

The PSS service must be reliable.

1) Hardware should be used which has proved itself in a communications environment.

2) Software should be written in a high level language, preferably one which supports the handling of complex data structures.

3) An operating system should be used which has proved itself in a communications enviroment and which is well understood.

4) Support software should be available to allow fast diagnosis of problems and fast alterations to code. This implies

    a) Software should be capable of being compiled on the machine on which it will run.

    b) Utilities should provide for:

        Straightforward system generation and loading
        Intelligent dump analysis
        Monitoring

5) The hardware plus operating system should provide adequate inter-process protection.

6) Software should be written such that different functional units are completely distinct - ie. the function of a protocol conversion system or gateway should not be combined with that of a switch, front-end or mainframe. Ideally the PSS service should be provided via a gateway located in a separate machine.

Security

Users will be charged for use of the PSS service so the service will have to maintain accounts.

Accounting files must be secure and free from interference.

Size

As a rough estimate a PSS service will require about 40K words of code divided equally between

Low-level protocol conversion
High-level protocol conversion
Operator I/O and accounting

There will be additional overheads due to supervisory software and buffer space.

Conclusions

1) The ERCC PSS service should consist of a Gateway system running in a dedicated machine. There is very little to be gained and a good deal to be lost by combining the gateway with any other piece of communications software.

2) This machine should be a PDP11 running DEIMOS. There is a good deal of PDP11 expertise in ERCC and PDP11's have proved themselves in virtually every communications environment. The DEIMOS operating system has similarly proved itself in the EMAS 2900 front-ends, the ERTE terminal emulator, a large RJE workstation and now in the new generation of communications switches. There is a good deal of knowledge of DEIMOS within the Centre and a good deal of support software for PDP11 DEIMOS systems including a PDP11 IMP compiler fully up to the standard required.

3) The machine should use currently available communications hardware. It is tempting to wait for more advanced communications devices. However these would require Post Office approval which would take time and effort. Such devices would be new and untried.

4) The machine will require 64K words of store to accommodate the Gateway Software.

5) The smallest machine that meets these requirements is a 64 Kwd PDP11. PDP 11/03, PDP 11/04 are unsuitable because they lack the memory management facility needed by DEIMOS. PDP 11/23 is unsuitable as it uses a Q-bus to which it is impossible to connect currently available HDLC hardware.

   The machine will require two DUP11 communications interfaces, two floppy disk drives (for accounting/accrediting information and for use during development) and a console.

6) The Gateway software should be written entirely in Edinburgh. Any attempt to write an RCONET handler on to the back of a purchased PSS handler would run into serious difficulties.

   1) The purchased package will be written in a 'foreign' language and will run under an unfamiliar operating system. In order to maintain it here we either have to import an operating system and compiler thus abandoning the DEIMOS/IMP system with all its advantages or we have to translate it into IMP which rather defeats the object of the exercise.

   2) The PSS level 2 and 3 handlers form only a small part (20%) of the Gateway software. It would not be worth compromising a perfectly workable comms software support system for a small piece of code.

   3) There is a good deal of expertise to be gained in writing X25 software - something which has not yet been done in the Centre.

Summary

1) The PSS Gateway should be a dedicated Machine.

2) It should make use of the considerable ERCC investment in PDP11 communications software ie. it should be written in IMP and run under DEIMOS.

3) The Gateway machine must therefore be a 64K word PDP 11/34.

PSS note: PSS-3

X25 GATE TO TASK INTERFACE

John Butler
21st July 1980

This note defines the interface between the two "Gate" modules and higher level tasks within the PSS gateway. The position of this interface and its relationship to adjoining modules is described in Gateway Note PSS-1. The interface is intended to be the X25 equivalent of the NSI Gate-to-Task interface designed by Brian Gilmore (DEIMOS note 3).

The Interface resembles the Study Group 3 transport service in that it uses the same primitives and the same call setup and cleardown procedures. It differs in that data is passed in blocks not bytes and that there is an explicit flow control mechanism across the interface. These differences were introduced to avoid gross inefficiencies in passing data through the Gateway.

The interface is designed to be as far as possible protocol independant, and to allow efficient transfer of data. The Data packet therefore incorporates a dummy record which allows blocks with different packet header lengths to be handled without having to move data or obtain knowledge of the header length.

The content of the data field does not form part of this specification. This specification does not contain provision for multiplexing data streams, so this must be done via a higher level protocol if required.

Value	Function*	Purpose	I	S1	S2	S3
15	Ack	Flow Control	Acknowledgement*	Undefined	Undefined	Undefined
16	Connect	Call Establishment	Quality	Calling Address*	Calling Address	Comment
17	Accept	Call Acceptance	Quality	Recall Address	Comment	Undefined
18	Disconnect	Call Disconnection	Reason	Location	Comment	Undefined
19	Reset	Resynchronisation	Reason	Location	Comment	Undefined
20	Address	Address Transfer	Qualifier	Address	Undefined	Undefined
21	Data	Data transfer	Acknowledgement*	Dummy*	Data*	Undefined
22	Expedited	Priority data	Data*	User Data	Undefined	Undefined

(Mandatory fields are indicated by *)

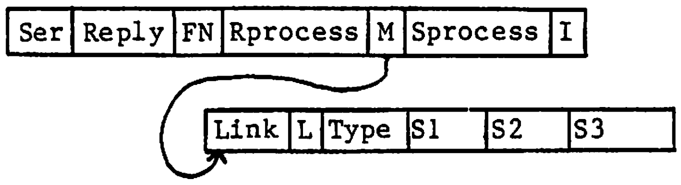
Interface Specification

```
%Recordformat PF(%byteinteger Ser,Reply,FN,Rprocess,
                   %record(MEF)%name M, %byteinteger Sprocess, I)

%Recordformat MEF(%Record(MEF)%name Link, %byteinteger L,Type,
                    %string (255) S1, S2, S3)
```

Parameter Area:

Data Area:



where:

Ser	= Service Number of recipient task
Reply	= Service Number of sending task
FN	= Function number
Rprocess	= Process number within recipient task. If the packet is identified by the sender's process alone this field will be zero.
M	= Pointer to the data area if one exists, Null otherwise (ie. value of M = 0).
Sprocess	= Process number within sending task. If the packet is identified by the recipients process alone, this field will be zero. Rprocess = 0 and Sprocess = 0 together constitute a fault.
I	= Numerical qualifier. Value depends on Function.
Link	= Link for queueing data packets together.
L	= Total length of data field - ie. the total length of the three strings, plus three for the 3 length bytes. A value of L not equal to this sum constitutes a fault.
Type	= Total length of block. This field is for the exclusive use of Buffer Manager and should not be altered.
S1,S2,S3	= The content of these strings depends on the function. These are distinct strings each with their own length byte and are not concatenated. The null string is represented by a single byte of value 0.

Description of Functions

The various Functions are summarised in the facing Table, and described in more detail in the following paragraphs. Undefined fields may take any value. Trailing undefined strings will not be included in the length field L. Unused but otherwise defined fields must be set to a sensible default as defined in this specification except for trailing null strings which may be omitted. If the data area is missing this will be taken to imply that S1, S2 & S3 assume their default values (the null string).

Connect

This function requests that a call be established from the "calling address" to the "called address" with a specified "quality". Sprocess must be specified and R process is undefined. The default quality is zero. The form of the address fields is specified in Appendix 1.

Accept

This function indicates acceptance of a call. "Quality" and "recall address" are included for compatibility with the Transport Service and always take their default values (zero and null respectively).

Disconnect

This function indicates disconnection of a call. "Reason" contains a qualifier for diagnostic purposes and has default value zero. Location always takes its default value null.

Data

This function transfers data across the interface. The "dummy" field is used to reduce the need for moving data to accommodate different packet header lengths amd will be constructed by lower levels such that the dummy packet plus the length byte of the "data" field sit exactly over the packet header. The Acknowledgement field contains four subfields.

N(R)	M	N(S)	Q

N(S) is a sequence number - modulo 7 - which starts at 0 and is incremented each time a data packet is sent.

N(R) is a response number - also modulo 7 - which represents the next packet expected. This starts at 0 and is incremented each time a frame is received.

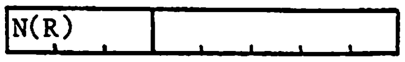
N(R) and N(S) are used to implement flow control using the principle adopted in X25 level 3, i.e. N(S)-N(R) modulo 7 must not exceed a predefined window, currently 2.

M is an indication of more data. If set, this indicates that the packet is not complete and should be taken in conjunction with one or more subsequent packets (See PSS Technical Guide Section 3, Paragraph 1.3.4)

Q is a qualifier bit. If set this indicates to the lower levels that this is a control packet (See PSS Technical Guide Section 3, Paragraph 1.3.5)

Ack

This packet performs flow control. The acknowledgement field contains one subfield.



N(R) is a response number as described above. An Ack will be sent immediately on receipt of a data packet unless previous acks or other data packets are queued, in which case redundant acks will be suppressed and the acknowledgement "piggybacked".

Reset

This function indicates resynchronisation: N(R) and N(S) are set to zero. Reason contains diagnostic information as to why the Reset has occurred. Location and Comment are currently set to Null.

Expedite

This function indicates priority data or interrupt. If the Expedite function carries a data area, then the interrupt is qualified by the content of the "user data" field. If not, then the interrupt is qualified by the "data" byte in the parameter area. The default value for this parameter is 0.

Address

This function is included for compatibility with the Transport Service. All fields take the default values 0 and Null. The function is currently defined to have no effect.

Procedural Considerations

Functions do not have end-to-end significance.

A connection is established by transmission of a connect packet in one direction, followed by transmission of an accept in the other. A Connection may be refused by sending a disconnect in place of the accept. An established connection may be disconnected by sending a disconnect in each direction across the interface. Data may be sent after the Connect but before receipt of the Accept. This data will be lost if the connection is subsequently refused.

A connection may be cleared down before it is established by sending a disconnect after the connect. The connection will be fully cleared down when the disconnect is acknowledged by another disconnect, possibly preceded by an Accept.

Procedures are those adopted by the study group 3 Transport service and that document should be consulted for further details.

APPENDIX

Address formats

Addresses are specified at three points within the gateway.

1) Within connections arriving from RCO. These are specified as three-byte groups containing the binary value of node, terminal and stream (N T S).

2) Within connections arriving from PSS. These are specified as packed BCD strings.

3) By human operators and users. These are specified as character strings.

Addresses will cross the gate-task interface as human-readable character strings.

RCO addresses will therefore be specified as

address::= N<number> T<number> (λ| S<number>)

<number> is the node, terminal or stream number as an unsigned decimal number with lead zeroes suppressed.

PSS addresses will initially be specified as

address ::= <number>.

Compilers

COMPILERS

COMPILER NOTES — note IMP77-1

IMP77 Cookbook

Bill Hay
17th June 1980

This is a set of recipes for converting to the new IMP77. I have used ECCE macro's fairly extensively, and can now convert fairly large programs (>1000 statements) in about 30 minutes, with one or two compilations to catch the more difficult cases.

1. 'CONST' NAMES

The construction:

```
%CONSTINTEGERNAME FRED = K'200'
```

must be changed to:

```
%CONSTINTEGERNAME FRED == K'200'.
```

I let the compiler find these (ie show them as faults).

2. STRINGS

Strings must be in DOUBLE quotes. For TCP/WS software, strings can be converted by the ECCE command:

```
(F/PRINTSTRING/(F/'/S/"/)2)*
```

3. CYCLES WITH CONTROL VARIABLES

The construct:

```
%CYCLE I=1,1,N
```

has been changed to:

```
%FOR I=1,1,N %CYCLE
```

The following procedure fixes this semi-automatically:

```
%X = F/%CYCLE/

%Y = S/%FOR/R*I/;/L*T/%FOR/F/;/I/%CYCLE/
```

Do X commands until a CYCLE with control variables is found, then a Y command - and repeat.

4. 'ELSEIF'

In a statement of the form "%FINISH.....%START", these two keywords must now be either both present or both absent. The statement "%ELSEIF.....%START" is therefore wrong, and must be replaced either by "%FINISHELSEIF.....%START" or by "%ELSEIF.....". Thus for example the construct:

```
                %IF <condition> %START

                         . . . .

                %ELSEIF <condition> %START

                         . . . .

                %ELSEIF <condition> %START

                         . . . .

                %FINISH
```

has become:

```
                %IF <condition> %START

                         . . . .

                %ELSEIF <condition>

                         . . . .

                %ELSEIF <condition>

                         . . . .

                %FINISH
```

Most occurrences of this are caught by the ECCE command:

```
            (F/%ELSEIF/D/%START/)*
```

However there is still a problem in the following case:

```
                %IF <condition> %START

                         . . . .

                %ELSEIF <condition> %START

                         . . . .

                %ELSESTART       |
                                 |
                         . . . . }  or   %ELSE . . . .
                                 |
                %FINISH          |
```

This has to become:

```
%IF <condition> %START

        . . . .

%ELSEIF <condition>

        . . . .

%FINISHELSESTART   |
                   |
        . . . .    } or    %FINISHELSE . . . .
                   |
%FINISH            |
```

But let the compiler find this.

(See pg. 6.3 of the IMP77 Manual).

5. COMMENTS

Semicolons are not now treated as terminators in comment lines - that is, comments starting with a '!' now terminate at the following NEWLINE, and not at a ';'. This may cause problems if comments appear before valid statements on the same line. The solution is to enclose the comment in curly brackets: {}. The following ECCE macro's are useful:

```
%X = (F/!/T/;/\M)*

%Y = E-1I/}/(LV/!/\)*EI/{/
```

The X command will display possible candidates: those which need to be converted can be changed with the Y command.

6. CONTROL

%CONTROL has changed. The most useful values are:

1 - do not enforce type-checking of %RECORDNAME assignments.

256 - generate code for systems without the extended instruction set.

7. ADDRESS MAPPING

The new IMP77 contains useful built-in mapping functions, such as 'ADDR', 'INTEGER', and 'RECORD'. These should be used instead of mapping record names with alternative formats.

8. IMP77 MANUAL

The new IMP conforms much more closely than the old IMP (WIMP) to the description in the IMP77 manual (report CSR-19-77 from the Dept. of Computer Science, May '79).

Deimos

DEIMOS

DEIMOS — note DEIMOS-1ii

BOOTRL

Bill Hay
31st July 1980

Introduction

BOOTRL is a bootstrap which allows named files to be loaded from a DEIMOS disk into a PDP11 for execution in stand-alone mode. It is most useful on the development machine, as development systems can be transmitted to the RL01 disc from EMAS using the development machine under DEIMOS (see note DEIMOS-4), or put on to disk from paper tape (see note DEIMOS-2); and then loaded and executed using BOOTRL.

Loading BOOTRL

At present BOOTRL is loaded from the general development disk as follows:

1. Press 'CNTRL' and 'HLT' buttons simultaneously.
2. Press 'CLR'.
3. Press "1".
4. Press 'LSR'.
5. Press 'CNTRL' and 'BOOT'.
6. The console will print '@'. Type DL(CR). The bootstrap now loads and prints:

```
BOOT RL V0.0
*
```

The '*' is the command prompt.

Commands

Note: 1. No type-ahead is possible.
2. The only permitted line-editing is RUBOUT.

Commands to load and execute file

L<filename> - loads DEIMOS file <filename> into store.

G - GO - start executing at current start address. Fails if start address is not set.

R<filename> - RUN - same as 'L' followed by 'G'.

S start address - Set or reset start address (which must be in OCTAL). Prints old and new address.

Commands to examine and modify store

Once a program is loaded using the 'L' command, store can be examined and modified before running the program, as follows:

C	- CLEAR - sets store locations from 0 to 140000 to zero.
E low:{high}	- EXAMINE - prints a section of store from 'low' to 'high', 8 words to a line. If 'high' is omitted just one line is printed, giving store locations ('low') to ('low'+16).
M address newcontents	- MODIFY. Prints out old and new contents of this address.
=	- repeat the last 'E' command. Useful after several 'M' commands, to check them.
+	- repeats the last 'E' command, but for the section of store following that displayed by that previous 'E' command.
-	- repeats the last 'E' command, for the section of store preceding that shown in the last 'E' command.
B address	- Set Base for 'E' and 'M' commands. 'E' commands are normally relative to store location 0. Using the 'B' command the addresses can be made relative to the location address - useful for example when working with load maps.
F value	- prints out all store addresses whose contents are "value", between 0 and 160000. Note that the values printed are not relative to the base.

Other Features

ABS loader

The ABS loader is automatically loaded with BOOTRL. The start address is 157500.

TU58 loader and dumper

The TU58 loader and dumper are held on disk, as LOAD(50) and DUMP(50). To write a TU58 with a TCP image do the following:

1. BOOTSTRAP BOOTRL
2. L fname (fname is TCP image on disk)
3. L DUMP(50)
4. S 157000
5. G

The PDP11 will HALT when the core image has been written.

Building BOOTRL

All sources are in BOOTSRC. To build it, proceed as follows:

```
        IMP11 BOOTSRC_BOOT,BOOT#REL
        IMP11 BOOTSRC_TTY,TTY#REL
        IMP11 BOOTSRC_RL01,RL01#REL
        LINK11
        .ALONE 140000 156776
        BOOT#REL
        TTY#REL
        RL01#REL
        .END
        BOOT#OBJ
```

A utility is provided to copy BOOTRL on to the second load site on the RL01. This is built as follows:

```
        IMP11 BOOTSRC_WBOOT,WBOOT#REL
        LINK11
        WBOOT#REL
        ERCM09.IMP77PY#REL
        .END
        WBOOT#OBJ
```

The OBJ files must be sent to the RL01 by loading the DEIMOS/RJE package, and then SENDING the OBJ file to .LP155. They will arrive as VLP???(16) (where ??? are digits). They should be copied to a safe area. Assuming they are copied with the names WBOOT and BOOT, then the command

```
                WBOOT BOOT
```

will write the bootstrap to the second load site.

DEIMOS

note: DEIMOS-2

READING PAPER TAPES ON TO A DEIMOS DISC

Brian Gilmore
2nd July 1980

A program is available to read either Binary or Source tapes on to a DEIMOS disc. It is called 'READPR' (source READPS(13)).

It is run by the command:

```
>READPR /<filename>
```

The program then prompts 'BINARY FILE?'; reply 'yes' or 'no', as follows:

'Y(CR)' - for all bytes to be transferred unchanged from the tape.

'N(CR)' - for all non-binary files; the effect is to strip all characters of code value less than 32, except for NL.

On reaching the end of the tape the program will stop. The file is now in <filename> on the DEIMOS disc, and can for example be loaded and run from the disc using 'BOOTRL' (see DEIMOS note 1).

DEIMOS note: DEIMOS-3

GATE-to-TASK Interface

Brian Gilmore
4th July 1980

This note describes the interface between GATE and a TASK which wishes to do any of the following:

a) Establish a connection
b) Receive incoming connections
c) Send/Receive data through said connections
d) Send/Receive messages.

The interface operates by the sending of messages as records, 'P', in the format:

```
%BYTEINTEGER SER, REPLY, FN, PORT,
%RECORD (MEF) %NAME MES,
%BYTEINTEGER SO, S1.
```

or - for non-data transfers -

```
%BYTEINTEGER SER,REPLY,FN,PORT,FACILITY,FLAG,NODE,TERM
```

where

SER	= service number of receiving TASK
REPLY	= service number of sending TASK
FN	= function code
PORT	= User or GATE port reference
SO	= various uses (see below)
S1	= returns GATE port numbers etc. (see below)

MES is a record name of format (MEF):

```
%RECORD (MEF) %NAME LINK,
%BYTEINTEGER LEN, TYPE,
%RECORD (NSI1F) NSL.
```

where LEN = length of the data block,
TYPE=0 is 256 byte block,=64 is 64 byte block, and
format NSI1F is:

```
%BYTEINTEGER FN, SUFL, ST, SS, SN, DN, DT, DS, FLAG, UFL,
%BYTEINTEGERARRAY A(0:238)
```

(ie. this is an NSI block format. Note that short block formats can be mapped on to this if required).

The various functions available in the two directions across the interface are as follows:

<u>TASK to GATE</u>:

1 - ENABLE FACILITY

2 - DISABLE FACILITY

3 - CALL REPLY

4 - ENABLE INPUT

5 - PUT OUTPUT

6 - CLOSE CALL

7 - ABORT CALL

8 - OPEN CALL

9 - OPEN MESSAGE

<u>GATE to TASK</u>:

2 - INCOMING CALL

3 - INPUT RECEIVED

4 - OUTPUT TRANSMITTED

5 - CALL CLOSED

6 - CALL ABORTED

7 - OPEN REPLY A

8 - OPEN REPLY B

9 - MESSAGE

10 - MESSAGE REPLY

In detail, these functions are as follows (NSI equivalents in brackets):

Interface: TASK to GATE

ENABLE FACILITY	:	P_S1 = facility number. Requests that all incoming calls to this facility number be sent to the TASK - will supercede any other TASK which has enabled the facility.
DISABLE FACILITY	:	P_S1 = facility number.
CALL REPLY	:	(CONNECT RESPONSE or SENDMESSAGE RESPONSE) P_S1 = reply showing success/fail: = 0 - reject, with a fail flag of 128. (both CONNECT and SENDMESSAGE) = 128 - set SUFL=0 in reply to a message. (SENDMESSAGE only) # 0 - accept the connection, using P_S1 as NSI_flag. (CONNECT only) This call is used by the TASK to reply to an "INCOMING CALL" or "MESSAGE" sent by GATE.
ENABLE INPUT	:	(SENDBLOCK RESPONSE) This call is used by the TASK to acknowledge an "INPUT RECEIVED" call from GATE, and hence allow the remote end to transmit another packet.
PUT OUTPUT	:	(SENDBLOCK) P_MES == block of data to be transmitted.
CLOSE CALL	:	(SENDBLOCK + DISCONNECT) To send a last block of data. GATE will reply with "CALL CLOSED" if it is OK, or "CALL ABORTED" if it failed to close normally.
ABORT CALL	:	(STATUS DISCONNECT) to close the connection (P_MES == null!). You will get "CALL ABORTED" in reply.
OPEN CALL	:	(CONNECT). P_PORT = TASK's reference number. This will cause GATE to allocate a port, the number of which it will return as P_S1 in the return call "OPEN REPLY A". P_S1 = 0 means that GATE is full - try again later. When the connection is established GATE will send "OPEN REPLY B" and the TASK must determine the success/failure of the connection. Thereafter all port references must use the GATE port number.
OPEN MESSAGE	:	(SEND MESSAGE) P_PORT = TASK's reference number MES == message When GATE receives the response a "MESSAGE REPLY" will be sent back with P_S0 = TASK's reference port, MES == message reply. The TASK has to determine success/failure. (Remember to free the buffer).

Interface: GATE to TASK.

INCOMING CALL	:	(CONNECT from GATE to TASK)
		P_S0 = forward/reverse buffer limit
		P_S1 = source terminal number
		P_PORT = GATE port
		Requests that you accept or fail a call.
		P_MES points to the connect packet which you may reference for extra information, but you must *not* free it. The TASK must return a "CALL REPLY" to GATE.

INPUT RECEIVED	:	P_MES == incoming block of data. (TASK must free block)
OUTPUT TRANSMITTED	:	(SENDBLOCK RESPONSE) Enables TASK to send another block of data.
CALL CLOSED	:	GATE has received a Send Block + Disconnect, and "CALL CLOSED" is sent AFTER the last incoming data block. The TASK should reply "CLOSE CALL" or "ABORT CALL" to GATE.
CALL ABORTED	:	Obvious. Must reply with ABORT CALL.
OPEN REPLY	:	(CONNECT RESPONSE) This GATE reply to an "OPEN CALL" from the TASK consists of two separate parts: OPEN REPLY A and OPEN REPLY B OPEN REPLY A contains the TASK reference number in P_PORT, and also a GATE Port number which *must* be quoted in all subsequent transactions (and GATE responses). This GATE Port number is in P_S1: it is zero if either GATE has no free ports or if the line is down. OPEN REPLY B contains the success/fail flag in P_S1.

MESSAGE	:	(SENDMESSAGE from GATE to TASK).
		P_S0 = NSI_flag.
		P_S1 = source terminal number.
		P_PORT = GATE port.
		P_MES points to the SENDMESSAGE packet, which you may reference for information - eg. the Node and Stream numbers - but you must *not* free it. The TASK should send a "CALL REPLY" to GATE in response.

MESSAGE REPLY	:	This is the GATE response to a TASK's "OPEN MESSAGE" call. P_Mes points to a block containing the SM reply. You may inspect it for the success/fail field, and *must* free the block.

DEIMOS note: DEIMOS-4

Connecting DEIMOS to RCOnet

Brian Gilmore
5th July 1980

There is a package of programs within DEIMOS which simulates an NSI workstation, thus enabling a DEIMOS system to connect to the RCOnet. The programs are called by loading a file called 'GO'. This is usually in file system 60, but for some machines it is in file system 15. Thus the initial commands are:

```
¥LOGON 60   (or 15)

¥LOAD GO
```

When it has finished loading, 'GO' prints the message "SYSTEM LOADED". Several other messages are also printed; these are produced by the protocol handler and the network link:

When the protocol handler has got the line up it prints the message:

```
PRTO: LINE UP
```

GATE will now attach to the network, and prints the message:

```
ATTACHED OK
```

The link to the network is provided by the program 'NSIW'. When this starts up it prints the message:

```
LP:ENABLED
SM:DISABLED
```

showing that: i) the 'LP' facility is ready for use (ie. the system can accept files from the network), and
ii) Send Messages from the Network are discarded (this is for compatibility reasons on the 2900 FEPs).

Thus the first command that should be given is "SM/ENABLE", to allow messages to be received from the network. It should be noted that <u>all</u> commands to 'NSIW' consist of a 2-character mnemonic, followed by '/' and a command-specific text. The basic commands are:

SM/ENABLE	- allow console messages to be printed.
LP/ENABLE	- allow the LP facility (ie. accept files).
TT/<ADDRESS>	- set up an interactive connection to a host.
TT/KILL	- terminate an interactive connection to a host.
SM/<ADDRESS>	- send a message to a network terminal.
OP/<ADDRESS>	- send a message to a mainframe OPER.
CR/<specific text>	- send file(s) to a mainframe or terminal.
LP/NULL	- provides a fast bucket for testing.
LP/FILE	- enables the default disc file names to be changed.

* * *

DETAILED DESCRIPTION

1. Interactive Terminal Use

A user first logs on to the mainframe by using the command "TT/" followed by the required host name - ie:

"TT/2980" or "TT/2970" or "TT/2972"

or by specifying the network address - for example "TT/N80T80 " (the SPACE is necessary). The host will then prompt for "USER:" and "PASS:" as usual.

If 'NSIW' succeeds in connecting to the host it produces the message "TT:CONNECTED". If it fails it prints the message "TT:CONNECT FAILS n", where 'n' is the NSI fail code. On logging off, 'NSIW' terminates the connection and prints "TT:ABORTED".

Note: (i) that at no time is type-ahead allowed.
(ii) that a very crude INTerrupt scheme is implemented as:

TT/INT n, where 'n' is the interrupt character.

However using "TT/INT A" will probably cause more problems than it cures. If in extreme difficulties, type "TT/KILL" which terminates the session.

* *

2. Sending messages and Operator Commands

Messages can be sent to remote terminals, nodes or FEPs by using the command

```
SM/<address> text
```

where <address> is 'NxTy' for Node 'x' and Terminal 'y'; hence the command

```
SM/N9T23 message
```

sends 'message' to SLOW DEVS. In the case of FEPs, the host name can be used - for example "SM/2980 Hello".

The command OP/<address> text" is identical except that it will send a message to the mainframe; for example to log on to the 2980:

```
OP/2980 LOGON T155     - or, identically,
OP/N80T80 LOGON T155
```

The 'Nx' may be omitted, in which case the node number defaults to zero.

If a message is to be sent to a stream other than 2 (console) or 11 (the mainframe), a stream number may be appended to the address as 'Sz'.

In all cases, if the message is accepted at the destination, the message "SM:OK" is printed; if it is not accepted, the message is "SM:FAILS 'n'".

* *

3. Receiving files from the Network on to the Disc.

The 'LP' facility allows other terminals or mainframes to send files to DEIMOS, and these are written to the disc. The files are given the default filenames 0.VLP000(16), 0.VLP001(16), etc., but some older versions may use 0.VLP100(16), 0.VLP101(16),.... (The '0' is the disc unit number, and '(16)' is the file system. Within that file system and on that disc, the incoming files are named VLP000, VLP001, etc.).

The 'LP' facility is enabled on start-up; it is disabled or re-enabled by typing the 'toggle' command "LP/ENABLE".

It should be noted that if a file is aborted in transfer for any reason, the message "LP:ABORTED" is printed and the facility is left in the 'idle' state - which means that it must be 'enabled' before another file can be received. The status of the facility can be ascertained by typing "LP/STATUS".

The 3-character name of the default file may be changed (the numbers in the filename cannot be changed - they are always 000, 001, etc) by the command "LP/FILE", and then on the prompt 'LP:BASE FILE' giving the new name - that is:

```
LP:BASE FILE: 0.xxx000(56)
```

If a test 'bucket' is required, then the command "LP/NULL" should be issued; all files are then thrown away until a new "LP/FILE" command is issued.

* *

4. Sending Files into the Network

These commands are in the form "CR/" followed by the appropriate text, as follows:

CR/<address> FILE	- to send one file to a mainframe (the file must contain its own JCL).
CR/<address> FILES n	- to send 'n' files to a mainframe as a single unit.
CR/<address> LP	- to send a file to facility '4' at 'address'.
CR/<address> BINARY LP	- to send a binary file to facility '4'.
CR/<address> FE	- to send a file to facility '9' on the FEP at 'address'; ONLY USE THIS COMMAND IF YOU KNOW WHAT YOU'RE DOING!!

'NSIW' replies as follows:

CR: connect fails 'n'	- if it fails to connect. In the case of an LP file it will print:
CR: WILL KEEP TRYING	- and do so until it is successful.
CR: CONNECTED	- when successfully connected.
CR: FILENAME:	- this is the prompt for the file which is to be sent, and follows any of the above 'CR/' commands. Reply either with the name of a DEIMOS file (eg. FRED(16) or 1.FRED(16)), or '.TT' for input from the console. There are no further prompts in the latter case, which must be terminated with "ctrl+D" ('EOT'). 'NSIW' will respond with either 'CR:FILE DONE' (if there are more files to come), or 'CR:FINISHED' (on completion). In the latter case it will terminate the connection.

This method of input is useful for sending files to SPOOLR, since - with the 'CR/<addr> FILES 2' command - the JCL can be typed in from the console for attaching to the front of the file. Similarly there is a file called 'LP23' on some machines which contains the necessary JCL to have a file printed on .LP23. Thus (with user commands underlined):

```
CR/2980 FILES 2
CR:CONNECTED
CR FILE:.TT
//DOC DEST=FILE,USER=ERCM03,PASS=????,NAME=NEWFILENAME(CR)('EOT')(CR)
CR:FILE DONE
CR FILE:DISCFILE
CR:FINISHED, nnn CHARS
```

or:

```
CR//2980 FILES 2
CR:CONNECTED
CR FILE:LP23
CR:FILE DONE
CR FILE:DISCFILE
CR:FINISHED, nnn CHARS
```

At any time the user can type "CR/STATUS" to find out what 'CR' is doing.

Don't forget to terminate the session with "OP/<addr> LOGOFF" !!

*   *

DEIMOS

note: DEIMOS-5

BLOCK ALLOCATION ON DEIMOS DISCS

Brian Gilmore
19th August 1980

The disc block allocation for the current implementations of DEIMOS is as follows:

BLOCKS	DISC TYPE								
	RK05		RL01		RX02		AMPEX etc.		RL02
	octal	(dec)	octal	(dec)	octal	(dec)	octal	(dec)	
BOOT block:	0	(0)	0	(0)	15	(13)	0	(0)	0
first System									
start:	1	(1)	1	(1)	16	(14)	1	(1)	1
end:	76	(62)	76	(62)	112	(74)	76	(62)	76
Block list									
start:	100	(64)	100	(64)	130	(88)	100	(64)	100
end:	147	(103)	217	(143)	140	(96)	477	(319)	337
DIRECTORY									
start:	150	(104)	220	(144)	141	(97)	1100	(576)	340
end:	247	(167)	317	(207)	240	(160)	1177	(639)	437
User Blocks									
first:	400	(256)	400	(256)	241	(161)	1500	(832)	500
last:	10767	(4599)	21757	(9199)	1750	(1000)	175000	(64000)	19439
second System									
start:	10770	(4600)	21760	(9200)					
end:	11066	(4662)	22056	(9262)					
Dump Site:	11100	(4672)							

FILE SYSTEM HANDLER PRIMITIVES

Brian Gilmore
19th August 1980

This note describes how a task may access files without using the standard IMP I/O routines.

The interface operates by sending messages as records, 'P', in the format:

```
     %BYTEINTEGER SER,REPLY, %INTEGER FN, %C
     %RECORD(file descriptor)%NAME B, %INTEGER C
```

The format 'file descriptor' is as follows:

```
                                          BYTE
     %RECORDFORMAT FILE DESCRIPTOR(%INTEGER UNIT,FSYS, %C
                                   %BYTEINTEGERARRAY NAME (0:5))
```

where 'UNIT' is the logical unit number of the file to be accessed/created, 'FSYS' is the file system number (0-K'77') and 'NAME' contains the 6 character name of the file (padded out with spaces).

A valid file descriptor should be passed in all cases.

Operations on the file system are carried out by a number of functions (put in FN) and are detailed below:

0 - Examine

1 - Get Next

2 - Destroy

3 - Create

4 - Append

5 - Rename

6 - Rename Temp File

7 - Rename Fsys

8 - Get DIR BLOCK Number

Examine

This call is used to determine if a file (described by P_B) exists and what its first Block Number is.

```
On reply, P_FN = 0 - file does not exist.
               = n - 1st block of the file is 'n'.
P_C is not used.
```

Get Next

This call is used to find out the second and subsequent block numbers of a file. On the call, P_C = last block accessed.

```
On reply, P_FN = 0 - no more blocks in file.
               = n - next block in file is 'n'.
```

Destroy

This call destroys the file, described by P_B (P_C not used).

```
On reply, P_FN = 0 - file has been destroyed.
               = 1 - file does not exist.
               = -1 - file was corrupt (still destroyed).
```

Create

This call creates a one-block file as defined by P_B, (P_C not used).

```
On reply, P_Fn = 0 - failed to create, no free blocks
                     or directory full.
               = n - 'n' is the block number of the one-
                     block file.
```

Note: If the file already exists, a second file of the same name is created; this will cause problems with 'examine'.

Append

This call is used to extend the length of a file by one block. On the call P_C = last block of the file.

```
On reply, P_FN = 0 - failed - no free blocks.
               = n - n is the new block.
```

Rename

This call uses a different call format, viz:

```
%BYTEINTEGER SER,REPLY,%INTEGER FN,%RECORD(FILE DESCRIPTOR) %C
            OLD NAME,NEW NAME
```

and renames the file described by 'OLDNAME' into the file described by 'NEWNAME'. The call will fail if the unit and/or FSYS are different, if 'OLD NAME' does not exist, or if 'NEWNAME' exists already.

```
On reply, P_FN = 0 - call successful.
               # 0 - call failed.
```

Rename Temp

This call renames a temporary file (eg. #FRED) to a permanent file (eg. FRED), destroying the old copy of the permanent file (eg. FRED) if necessary. P_C is not used.

```
On reply, P_FN = 0 - call succeeded
               = -1 - temp file does not exist.
```

Rename Fsys

Rename Fsys is the same as 'RENAME' except that it is prepared to 'move' a file from one file system to another. This call was put in specifically for the Appleton Tower System and its use elsewhere is not recommended.

Get DIRECTORY BLOCK Number

This call returns the directory block number for the specified file (just using the UNIT and FSYS parts). It should be used by any program that wishes to access directories, since the position of the directory blocks depends on the type of disc being used (see DEIMOS note 5).

On reply, P_FN = directory block number.

Info

INFO

INFO | note: INFO-1

PRELIMINIARY FUNCTIONAL SPECIFICATION

Scott Currie
24th July 1979

This document is an expansion of the brief specification written in April 1979 for the purpose of fund allocation, and is the result of consultation with interested parties within the ERCC.

The Network Information Station (INFO) is the data collection centre for the network, its primary function being to determine the state of terminals on the network, either by polling those terminals or by receiving status reports, and - in the case of failures - to alert the operations staff.

INFO will also permit on-line debugging of specific network terminals and will gather data for statistical analysis. It is envisaged that INFO will simply store this data for analysis elsewhere.

Data will be gathered by INFO in two ways, primarily by polling terminals using NSI SENDMESSAGE packets on a reserved stream, but a facility will also be provided to accept asynchronous messages from terminals.

Note that INFO is only an information centre and NOT a control centre. A malfunction of INFO will not affect normal operation in the network.

INFO will supply information to four types of user:

- the "ordinary" interactive console or RJE station user
- the operation staff
- the systems development staff (chiefly communications), and
- ERCC management.

User Facilities

INFO will appear as a HOST on the network. In response to commands it will present status reports to the user in a suitable format, in particular as follows:

a) A list of HOSTs on the Network and their status (eg. UP/DOWN), FEP status and number of users.

b) The above information for a specific HOST plus any current "message of the day" input by the operations staff.

c) The status of the network Nodes (and Node to Node connections).

d) The status of the TCP's.

e) The status of specific RJE terminals.

Operator Facilities

INFO will drive a VDU device at a central point, convenient to the operations staff, and will provide a continuous display of the status of mainframes, FEP's, nodes, TCP's, and possibly RJE stations, plus the current broadcast message.

Any failure of the major components displayed will result in that status "flashing" on the screen, probably with an audible warning, the operator cancelling the warning after action has been taken. The status displayed will be gathered by INFO polling the necessary terminals on the Network - it is envisaged that everything be polled within a 2 minute period so there will be a small delay in detecting failures.

The "broadcast message" is a facility by which operators can send messages from INFO, both globally (to RJE and interactive consoles) and selectively.

The operations staff will from time to time require more detailed information from specific nodes and terminals. The following requirements have been identified.

a) Nodes	- display current noticeboard and the status of lines and terminals.
b) Mainframes	- input appropriate operator commands to the mainframes (though this is really a control function).
c) FEP's	- status of lines.
d) TCP's	- determine state of active buffers.

Another operational requirement is the recording of all RJE/mainframe jobs such that these jobs can be monitored, and a list played back or printed if required. This may be seen as a mainframe requirement however.

Systems Facilities

The following facilites would be "priveleged", ie. protected by passwords at least. The facilities required for Nodes, TCP's and FEP's are all similar in nature, viz:

a) A general debugging package to examine core tables, and possibly modify them.

b) Regular statistics-gathering by polling terminals and accumulating data for later analysis.

c) A method of invoking, interrogating and decoding Node Internal traces.

Management Facilites

The statistics gathering function of INFO mentioned above can be extended to provide more accurate and comprehensive data on the Network than has been available up to now eg.

- measuring reliability in terms of up/down times of, say, TCP and RJE stations.

- load statistics on specific lines.

Hardware Considerations

INFO can be split into two broad categories of operation: a "what's up or down" facility; and a more detailed debugging/statistics gathering facility.

It is relatively simple to provide the status information required by polling the terminals involved, indeed a prototype INFO service is now available which provides the status of the TCP's and 2970 Front End. The prototype, which is based on NSI workstation software, also contains a TCP Debugging Package which has proved its worth over the last few months.

There is however a considerable increase in complexity to support such facilities as general statistics gathering, node tracing and job recording, and it is doubtful that a small system (the current INFO in say an LSI-11/03) could support all these - certainly not all at once.

It may therefore be prudent to develop the system further on a PDP-11 with memory management (LSI-11/23) under the DEIMOS system, to permit future expansion. Such a system would require floppy discs and cassette tape for large scale statistics gathering.

Conclusions

The basic facilities required could be provided by the current INFO system in an LSI-11/03 with 32K words of store, DUV-11, a VDU and possibly a TU-58 cassette tape for mass storage.

A more expandable system would be an INFO under DEIMOS in an LSI-11/23 with initially 32K words of store, DUV-11, a VDU, RX02 floppy discs and TU-58 cassette tape. This system would then be self-supporting.

The interactive debug/stats gathering from remote ring stn seems a good idea Alan

INFO

note: INFO-2

PROPOSED SOFTWARE STRUCTURE AND INTERFACES

Scott Currie
7th July 1980

INFO will consist of several tasks running under the DEIMOS system, the most important being outlined below.

N.B. In this note P is a record with format:

(%byteinteger SER,REP,%integer A,B,C)

or

(%byteinteger SER,REP,A1,A2,B1,B2,C1,C2)

GATE : NSI handler (bridge) - already exists but will be amended to pass Sendmessage Responses with data back to the originating tasks. The interface to GATE is the subject of DEIMOS note 3.

POLL : Module to do the basic regular polling of TCP's and probably Nodes, possibly FEP's. It will maintain a small database and will release information on demand to other tasks with parameters as follows:

a) Message from task for TCP type terminal

```
P_A : TCP NODE NO.
P_B : TCP Terminal No.
P_C : 1.
```

Reply to task is:

```
P_A1 : -2 - illegal
       -1 - down
        0 - up
```

P_A2, P_B1,B2, P_C1,C2 : no. of users on each host (up to 5 Hosts).

To pick up TCP base addresses as above with P_C=2.

```
Response P_A=host address
         P_B=console address
         P_C=port address
```

b) for other terminals - to be defined.

<u>ITP</u>	: Handles user requests to INFO. Will send out first INFO prompt and select task to handle the user's requirements. Thereafter will pass data to/from user and specified task. User inputs will be the subject of a later note. Interface to other tasks is the subject of note 3.
TCP DEBUG	:
<u>NODE DEBUG</u>	: These are privileged tasks which require a password from the user. They may access the POLL information or send their own messages through GATE under user control.
<u>USER</u>	: this will handle the normal user requirements - usually just the polling information, plus the message of the day for a given host.
<u>DISPLAY</u>	: will construct the local VT100 displays (several pages probably) with access to other tasks (via ITP). Will handle operator input.
<u>STATS</u>	: statistics gatherers to access GATE and/or other tasks.

INFO | note:; INFO-3

ITP TO TASK INTERFACE

Scott Currie
7th July 1980

When a user logs on to INFO the ITP task will allocate a port and send a HELLO message to the USER task, which will have a few less ports than the ITP task.

The USER task will output a title and service prompt (or busy message) and will deal with normal user enquiries as detailed below. If a user requests a special task the USER task will send a CHANGE TASK message to the ITP task which will load the required task and send it a HELLO message.

The slot in the USER task will be marked as occupied and any log-off will be sent to the USER task as well as the current task.

All tasks will send data as requested in lengths of less than 121 bytes. To save buffers it is recommended that tasks do not queue buffers to send but fill them on request.

N.B. All tasks other than the USER task will handle one console at a time.

The interface is defined assuming the record formats specified in the DEIMOS note on the GATE/TASK interface (DEIMOS note 3).

HELLO	: Sent from ITP to TASK. PORT is the ITP reference number which must be used in all messages for that console.
DATA IN	: From ITP to task. LEN = length of data. This is one line as typed in. The task must free the buffer.
INT IN	: From ITP to task. LEN = length of data. The user has typed an INT. The task must free the buffer.
SEND DATA	: From ITP to task: requests data out or a prompt.
DATA OUT	: From task to ITP. LEN = length of data. Maximum length = 120 (it will be truncated).
SEND PROMPT	: From task to ITP. LEN = length of prompt (<16 characters). N.B. there is no type ahead at the user level.
LOGOFF	: Either direction. From ITP to task, it means the user has typed Setmode EOT. From task to ITP (in response to SEND DATA) it will cause the user to be logged off (N.B. any log-off message must be sent as data previous to a LOGOFF). The task should cease to exist.
CHANGE TASK	: From task to ITP. Requests that the user be transferred to another task. SO = task identifier. Current identifiers are: 0 = User task 1 = TCP debugger 2 = Node debugger. Normally this will mean that the current task should cease to exist.
SETMODE	: From task to ITP. The data must be in the Setmode format (a routine will be provided to do this). This is used, for example, to turn echoing on/off.
ILLEGAL MESSAGE	: From ITP to task. SO = function sent to ITP. This probably implies a wrong port number etc.

Local Networks
(ring, etc.)

LOCAL NETWORKS

KENT

note: KENT-1

THE KENT NETWORK

Stephen Binns
30th July 1980

Configuration

We have currently 2 workstations: a PDP11/20 with 3 DM multiplexors and 2 character printers; and a PDP11/10 with 3 DH multiplexors, paper tape station, plotter and two printers. Later this summer we will configure in an 11/34 with DZ multiplexors.

A second 11/10 runs RJE work to ULCC and OXFORD (VME/B). The system is capable of taking output from EMAS as pseudo-cards to be sent to London or Oxford, and output from London may be spooled into EMAS. This system is written in Macro-11 for historical reasons. We plan to replace it with an X25 gateway when the remote hosts we want to talk to support PSS compatible X25. The system will also act as a ring to PSS gateway.

We have recently started work on a Z80 based workstation, capable of supporting 8 terminals and a character printer. This is being programmed in BCPL as a re-implementation of the IMP workstation, and we are aiming for a prototype by October 1980.

The 'hosts' at Kent are a 2960 with an 11/34 front end and a VAX running UNIX. The VAX doesn't have a front end, although we plan in the medium term to interpose a KMC11-based DMA controller between the VAX and the ring. This is being developed at Cambridge for an 11 system.

All the workstations and the hosts are connected via the Cambridge ring. The workstations are booted from the FEP round the ring; there is a dump facility back to the FEP and thence to EMAS. On EMAS there is a simple dump analyser for the 11's.

So far with only one host we have not needed a name server. A version of the Cambridge Z80 name server is now being tested to enable host addresses to change for maintenance and breakdown cover.

Software

The initial (January 1980) EMAS service utilised NSI protocol on top of byte stream protocol on the ring. There was one byte stream between each workstation and the front end. Since 14th July we have been running our new NSI-less software, in which a byte stream is used instead of an NSI stream; after some initial teething problems it seems to work satisfactorily. This means that GATE in the FEP disappears and a functionally equivalent (from above) byte stream protocol handler takes its place. The RJE and ITP handlers in the front end needed slight modification. BRIDGE has been completely re-written for the workstation. It now incorporates the functions of TRUNK, which has ceased to exist as a separate module. CONSOLE has been slightly modified to improve the UNIX user interface. 'Raw Mode' (single character, remote echo) access for UNIX will not be provided because of the load it would impose on the workstations and the VAX. However as this facility was mainly used within the editor for line-editing, it has been moved to the workstation. It provides enhanced local editing of input lines and the facility for the host to prime the input buffer with text to be edited at the workstation and then re-input.

We have also implemented bulk setmode and getmode for UNIX. We are considering withdrawing the local setmode (ctrl+A) and enforcing setmodes by the host.

Monitor

MONITOR

note: MONITOR-1

SHIFTA - simulates bit-shifting errors

Nick Stroud
7th July 1980

There is a new program on the monitor which reads in a string of up to 16 bytes and prints them out as they would appear if shifted to the left by 'n' bits, n=1-7. This simulates the effect of dropped bits on a communications line, or - by reading upwards from the bottom of the output listing - the effect of inserted bits.

The program is called 'SHIFTA', and is loaded in the usual way for the monitor - that is, '(ESC).SHIFTA(CR)'. The command level prompt is "WHAT NEXT?". On first entering the program type 'D' (NO CR or SPACE), followed by the bytes to be shifted. These bytes are specified in the same way as in the monitor programs - that is:

for HEX - prefix the first byte with 'X' (hex is the default mode).

for OCTAL - prefix the first byte with 'O'.

for DECIMAL - prefix EVERY byte with 'T'.

for LITERAL - prefix each byte with 'L', then specify the byte exactly as it appears on a (SYMBOLIC) monitor listing - eg 'DLE', 'A', 'n'. The actual code value taken in this input mode depends on the monitor's setting - ASCII/EBCDIC - set via the toggle command 'E' in SETMODE.

Separate each byte with a SPACE, and terminate the string with CARRIAGE RETURN. The program then prints the number of bytes and their bit pattern, followed by 8 lines of data - showing the original string, then the result of shifting this one bit to the left (equivalent to one bit dropped), then two bits to the left, and so on, up to seven bits. The string is 'wrapped around', in that the last byte is made up with the most significant bits from the first byte.

The other commands now available are as follows:

D: type in a new string of data.

E: enhance bytes as specified (in reverse video or blink).

F: enhance all BSC control codes.

M: show the enhanced bytes.

O: set OCTAL mode for all input/output.

(continued...)

S: enter SETMODE to change display parameters. The options are:

C: control code translation ON/OFF (toggle).

E: ASCII/EBCDIC interpretation of data (toggle). Default is ASCII.

O: select OCTAL for all i/o.

P: turn PARITY bit ON/OFF (toggle) - ie. use or drop the parity bit in interpreting ASCII characters.

S: turn SYMBOLIC mode ON/OFF (toggle) - ie. display either the character or its code value.

U(n): display 'n' as <numeric ASCII EBCDIC>.

X: select HEX for all i/o.

?: review current SETMODE settings.

(SP),(CR): exit from SETMODE.

W: print the results on the line printer (no need to use command 'Z' first, but printer stays on subsequently. Use 'Z' to turn it off).

X: set HEX mode for all input/output.

Y: print the complete ASCII/EBCDIC conversion table.

Z: turn line printer ON/OFF (toggle).

(ESC): terminate program and restore MUSS operating system.

These commands are available in the program: type 'H' (or make a mistake) at command level.

The printer listing overleaf illustrates the use of 'SHIFTA':

```
LP ON

WHAT NEXT? D=LSP =LEOT =LSTX =C2 =LSP =4D =F3 =LNAK =FF =FE

NUMBER OF BYTES =    000A
00100000 00000100 00000010 11000010 00100000 01001101 11110011 00010101 11111111
11111110
 HEX
..........................................
20  04  02  C2  20  4D  F3  15  FF  FE
..........................................
40  08  05  84  40  9B  E6  2B  FF  FC
..........................................
80  10  0B  08  81  37  CC  57  FF  F8
..........................................
00  20  16  11  02  6F  98  AF  FF  F1
..........................................
00  40  2C  22  04  DF  31  5F  FF  E2
..........................................
00  80  58  44  09  BE  62  BF  FF  C4
..........................................
01  00  B0  88  13  7C  C5  7F  FF  88
..........................................
02  01  61  10  26  F9  8A  FF  FF  10
..........................................

WHAT NEXT? S
SM-S              < SYMBOLIC  7 BITS  HEX >
SM-P              < SYMBOLIC  8 BITS  HEX >
SM-C              < SYMBOLIC  8 BITS CONTROL ONLY  HEX >
SM-               < SYMBOLIC  8 BITS CONTROL ONLY  HEX >


NUMBER OF BYTES =    000A
00100000 00000100 00000010 11000010 00100000 01001101 11110011 00010101 11111111
11111110
 SYMBOLIC  8 BITS CONTROL ONLY  HEX
..........................................
SP  EOT STX C2  SP  4D  F3  NAK FF  FE
..........................................
40  BS  ENQ 84  40  9B  E6  2B  FF  FC
..........................................
80  DLE VT  RS  81  37  CC  57  FF  F8
..........................................
NUL SP  SYN XC1 STX 6F  98  AF  FF  F1
..........................................
NUL 40  2C  22  EOT DF  31  5F  FF  E2
..........................................
NUL 80  58  44  HT  BE  62  BF  FF  C4
..........................................
SOH NUL B0  88  XC3 7C  C5  7F  FF  88
..........................................
STX SOH 61  DLE 26  F9  8A  FF  FF  DLE
..........................................

WHAT NEXT? Z
```

Miscellaneous

MISCELLANEOUS

note	author	date	title/subject

Utilities

UTILITIES

The following Comms group utilities are available via
OPTION.SEARCHDIR=ERCM03.COMMSDIR:

COMPARE file1,file2: to find out if two files are identical. If they are not, COMPARE stops at the first difference and prints out the surrounding bytes, indicating the discrepancy. Better than TEXTCOMPARE as it works with any files (not just character files).

DIGIT: for on-line translation of a number specified in decimal, octal or hex, into octal decimal and hex. A 'pseudo command level' can be regained from within DIGIT which allows this number-translation to be performed without coming out of commands such as EDIT, LOOK & PATCH.

SUPERSNAP file: for the detailed examination and alteration of any of the user's files, at the byte level.

ZAP: to tidy away the dead-wood files left by WIMP & IMP77.

COMPARE

George Howat
18th June 1980

COMPARE was written to find out whether or not two files were identical - in particular to see if a master file has been updated - and has since been extended to show the first mismatch in non-identical files. It is particularly useful for comparing two binary files, since John Wexler's TEXTCOMPARE cannot do this. The command and the two possible results are as follows:

```
Command:COMPARE file1,file2
   Lengths of files: file1 = n, file2 = n
   No mismatch found


Command:COMPARE file1,file3
   Lengths of files: file1 = n, file3 = p
   Mismatch in files at I=n
   Byte in file1 = a(dec) = b (hex)
   Byte in file3 = c(dec) = d(hex)
   xxxxxxxxxx
   xxxxxwwwww
        !            (the first difference is marked).
```

*   *   *

DIGIT

George Howat
23rd June 1980

DIGIT displays 16-bit numbers in decimal, octal and hex. The number is specified in the format 'aN' where 'N' is the number, and 'a' is 'O' for octal or 'X' for hex. If 'a' is omitted then 'N' is treated as a decimal.

DIGIT can be called in two ways: with a number, to translate the number and return to command level immediately; and without any parameter, to remain in DIGIT until the specific command to exit. The latter form not only allows further numbers to be translated without having to type the DIGIT command for each, but also incorporates a 'pseudo command level' through which all the usual EMAS commands can be executed. Thus for example access to the number-translator could be obtained during an EDIT or LOOK thus:

run DIGIT

enter pseudo command level (by typing '>')

EDIT/LOOK as usual (NO BRACKETS! - ie. LOOK file, not LOOK(file)).

come out of the pseudo command level (by typing '<') to return to the number-translator.

return to the EDIT/LOOK via '>'.

.
.
.
.

exit from the EDIT/LOOK.

exit from DIGIT ('E' or '*').

An example of the simple case (user commands underlined):

```
Command:DIGIT 12
D = 12 O = 000014 X = 000C

Command:DIGIT XABC
D = 2748 O = 005274 X = 0ABC
```

And an example of the clever case:

```
Command:DIGIT
>12                 ('>' is DIGIT's prompt)
D = 12 O = 000014 X=000C
>XABC
D = 2748 O = 005274 X = 0ABC
>>                  ('>' to enter pseudo command level)
>>LOOK OUTPUTFILE       ('>>' is the prompt at pseudo command level)
Look:M/DUMP:/
DUMP: 000000 017677 000005 036010 000177 004376
Look:E              (exit from LOOK)
>><                 ('<' to return from pseudo command level to DIGIT)
>017677
D = 8127 O = 017677 X = 1FBF
>05
D = 5 O = 000005 X = 0005
>0036010
D = 15368 O = 036010 X = 3C08
>E or *             (exit from DIGIT)
Command:            (back to normal)
```

*   *   *

EMAS HASH COMMANDS

Roderick McLeod
21st July 1978

The 2980 '#' commands are diagnostic aids provided by the EMAS group. There is no guarantee of their long term support.

Command Structure

Each command has a # as its first character. Note that hash commands do not invoke the Loader and there is no one-to-one correspondence between each hash command and an external routine. Therefore hash commands cannot be called from programs.

The parameters to hash commands include numeric constants. These can be typed as decimal integers in the normal way, or as hexadecimal numbers, in which case they should be preceded by an 'X'. For example, the following two commands would have the same effect:

```
#SWORD(X840000,256)

#SWORD(X840000,X100)
```

* In all relevant cases lengths are expressed in bytes.

Individual command descriptions

The hash commands available are as follows:

#ACR

prints out the current ACR level.

#CONNECT(filename)

connects the file, if possible in write mode, otherwise in read mode. Prints the connect address. Can also be used for a member of a partitioned data set, but the member is always connected in READ mode. Since a file connected by #CONNECT remains connected on return to command level, it is advisable to disconnect it explicitly when any operations upon it have been completed.

#DEC(hex value)

converts the hexadecimal value to decimal.

#DUMP(address,numberofbytes)

dumps the specified area to the line printer.

#DUMPFILE(filename, offset of start, bytes)

dumps the specified area of the file to the line printer.

#HEX(decimal value)

converts the decimal value to hexadecimal.

#PCOM(integer)

prints out the value of the specified location in COMREG.

#PMESS(integer)

prints out the Subsystem error message associated with the specified fault number.

#PVM

prints table of connected files.

#QUIT

logs off even when the session directory is corrupted.

#REGS

prints out registers at the time of the most recent failure contingency. Note that information on the last four contingencies is held; the three previous ones can be obtained using #REGS(-1), #REGS(-2) and #REGS(-3).

#SBYTE(address,value)

sets the specified byte to value. Clearly the byte must be accessible in write mode. This can be achieved by connecting the file using #CONNECT.

#SCOM(integer,value)

sets the specified location in COMREG to value.

#SETBASE(filename)

sets the name of the basefile to be used for subsequent sessions. If the parameter is omitted the default Subsystem basefile will be used.

#SNAP(address,bytes)

dumps the specified area on .OUT. The amount printed on a line will depend on the current setting of OPTION ITWIDTH.

#SNAPCODE(address,bytes,out)

decompiles code and prints it on the device or file specified. Default is .OUT.

#SNAPCH(address,bytes)

prints the specified area as ISO characters on .OUT.

#SSTRING(address,"string")

sets the string at the address specified to the string given. Note that the string should be enclosed in double quote characters.

#SWORD(address,value)

sets the word at the address specified to value.

* * *

SUPERSNAP

George Howat
18th June 1980

SUPERSNAP provides a tool for the detailed examination of an EMAS file. It can also be used as a simple editor by providing a means to alter specified bytes or words. It incorporates all the facilities of the "hash" commands #CONNECT, #SNAP, #SNAPCH, #SWORD and #SBYTE (in a neater format), as well as adding some new features. (For a brief description of the '#' commands, see the UTILITIES note 'EMAS HASH COMMANDS'). EMAS files begin with a header followed by data, if any; all is accessible to SUPERSNAP.

On typing the command 'SUPERSNAP filename' an attempt is made to connect the file in write mode. If this attempt fails there will be either a subsystem message giving the reason, or a message that the file cannot be connected in write mode. In the latter case the SUPERSNAP commands which alter the file will not work, although the file contents can still be examined.

After connection, the following information is provided:

'CONAD' - the address at which the file is connected.
'DS' - the data start address in the file.
'DE' - the data end address - actually the address of the first free location in the file.
'DL' - the data length, in hex and decimal (DE-DS).

The following SUPERSNAP facilities are available (see Table for command format):

ALTER B: replace one byte in the file (equivalent to EMAS #SBYTE)

ALTER W: replace one (32-bit) word in the file (EMAS #SWORD)

DUMP: dump a section of the file - specified as a start address together with either an end address or a number of bytes or words. The result is in both hex and character formats (an improvement over EMAS #SNAP), and can be directed to a file or output device if required.

FIND: search through the file for a particular sequence - specified either as a character string in the format C"....." or in hex, as Xxxxx.

HEADER: print the file's header - which includes such information as the type and size of the file, the amount and format of the data in it, and the date and time when it was last altered.

RECALL: recall the connect address (CONAD) and the start, end and length of the data (DS, DE and DL).

END: exit from SUPERSNAP.

The commands and their arguments for each of these SUPERSNAP operations are as follows:

OPERATION	COMMAND	1st ARG	2nd ARG	3rd ARG
Alter byte	AB or ALTERB	start address	replacement (hex)	—
Alter word	AW or ALTERW	start address	replacement (hex)	—
Dump	D or DUMP	start address	end address	file/device (default=.OUT)
OR	D	start address	mB (==m bytes)	file/device
OR	D	start address	nW (==n words)	file/device
End	E or END	—	—	—
Find text	F or FIND	start address	text	—
Show header	H or HEADER	—	—	—
Recall conad	R or RECALL	—	—	—

(the arguments are separated by SPACES - NOT commas).

There are several addresses which are used regularly when examining a file: these are the fixed addresses 'CONAD' (the file's connect address), 'DS' (the data start address), & 'DE' (the data end address); and the variable addresses returned after a FIND or DUMP operation. SUPERSNAP allows these addresses to be specified as mnemonics, to save the bother of looking up and re-typing the hex address each time. Thus for example the entire file (together with its header) can be printed by the command:

```
DUMP CONAD DE .LP
```

When dumping out small sections of a file, the mnemonic 'D+' can be used as a start address, and means 'follow straight on from the previous dump'. Thus the commands:

```
DUMP CONAD 16B .LP
DUMP D+ 16B .LP
```

will print the first 16 bytes of the file, then the second 16 bytes. The command 'D?' returns the current value of 'D+'.

The mnemonics 'P' and 'P+' can be used as addresses in conjunction with the FIND command, to represent - respectively - the address of the start of the found text, and that address plus one. The mnemonic 'T' can be used to mean 'the text of the previous FIND command', when a search is to be continued. They are used as follows:

FIND CONAD C"OLD"	(ie. search the whole file for the first occurrence of the string "OLD")
FIND P+ T	(move on to the next occurrence of the same text)

```
        ALTERB P X20          (replace the 'O' of this "OLD" with a SPACE
                              (hex 20))
```

The command 'P?' is available to discover the current setting of the FIND pointer.

* * *

In summary the SUPERSNAP commands are:

AB a b - replace the byte at address 'a' with 'b'.

AW c d - replace the word at address 'c' with 'd'.

D i j k - dump the file from address 'i' to address 'j', on device 'k'.

D i mB k - dump 'm' bytes, starting at address 'i'.

D i nW k <u>OR</u> D i n k - dump 'n' words, starting at address 'i'.

D? - gives the start address represented by the mnemonic 'D+' in the command 'dump D+ 10W'.

E - exit from SUPERSNAP.

F p q - find the first occurrence of the text 'q' (C"....." or Xxxxx) following the address 'p'.

H - for the EMAS file header.

P? - for the address of the FIND pointer: this is the address at which replacement text will be inserted by the commands 'AB' & 'AW' if the mnemonic 'P' is used for the address.

R - recall the connect address and the data's start address, end address and length ('CONAD', 'DS', 'DE' & 'DL').

And the following address mnemonics are available:

CONAD - the address at which the file is connected.

DE - the address of the first free location in the file - ie. the location following the last data entry in the file.

DS - the address of the first data byte in the file.

D+ - as the start address in a DUMP command for the dump to follow on from the previous one.

P - the start address of the field found by the last successful FIND.

P+ - used in a FIND command to continue a search from the address at which the previous FIND command was successful.

* * *

The following example illustrates the use of SUPERSNAP:

```
Command:LIST TEST

THIS IS A TEST FILE 1
THIS IS A TEST FILE 2

Command:SUPERSNAP TEST
CONAD=00A80000 DS=00A80020 DE=00A8004D DL=0000002D (= 45)

Op: HEADER
(00A80000)    0000004D 00000020 00001000 00000003    M
(00A80010)    00000000 29A8E374 00000000 00000000    ) t

Op: DUMP X00A80020 X00A8004D
(00A80020)    0A544849 53204953 20412054 45535420    THIS IS A TEST
(00A80030)    46494C45 20310A54 48495320 49532041    FILE 1 THIS IS A
(00A80040)    20544553 54204649 4C452032 0A000000    TEST FILE 2

Op: D X00A80020 X00A8004D
(00A80020)    0A544849 53204953 20412054 45535420    THIS IS A TEST
(00A80030)    46494C45 20310A54 48495320 49532041    FILE 1 THIS IS A
(00A80040)    20544553 54204649 4C452032 0A000000    TEST FILE 2

Op: D DS 10B
(00A80020)    0A544849 53204953 20412054 45535420    THIS IS A TEST

Op: F CONAD C"THIS"
(00A80020)    0A544849 53204953 20412054 45535420    THIS IS A TEST
Start address: 00A80021   End address: 00A80024

Op: P?
P = X00A80021

Op: F P+ T
(00A80030)    46494C45 20310A54 48495320 49532041    FILE 1 THIS IS A
Start address: 00A80037   End address: 00A8003A

Op: P?
P = X00A80037

Op: ALTERB P X30
(00A80030)    46494C45 20310A30 48495320 49532041    FILE 1 OHIS IS A

Op: D P 1W
(00A80030)    46494C45 20310A30 48495320 49532041    FILE 1 OHIS IS A

Op: RECALL
CONAD=00A80000 DS=00A80020 DE=00A8004D DL=0000002D (= 45)

Op: AW X00A80035 X20202020
Address is not full word aligned.

Op: AW X00A80034 X20202020
(00A80030)    46494C45 20202020 48495320 49532041    FILE HIS IS A

Op: END

Command:LIST TEST

THIS IS A TEST FILE    HIS IS A TEST FILE 2
```

WHO

George Howat
25th June 1980

The command 'WHO' shows which members of the Comms Group are logged on to the 2980 at the time of asking.

The command 'WHO name' returns the specific reply 'yes' or 'no' according to whether or not (name) is logged on to the 2980. ("name" can be BILL, JOHN, SCOTT, BRIAN, GEORGE, NOEL or NICK).

ZAP

George Howat 18th June 1980

WIMP and IMP77 leave several files lying around. If all seven of us IMP77 together we'd leave around a meg of dead files lying around. The command ZAP removes this garbage, by destroying the following files:

```
SS#01
SS#02
SS#03
SS#LIST

SS#IMP77O
SS#IMP77D

IMP#INT

TEMP
```

(No message is produced for those which don't exist).

* * *