Burroughs

Computer Management System (CMS)

Data Communications Subsystem

REFERENCE MANUAL

DCSS

PRICED ITEM

Burroughs

Computer Management System (CMS)

Data Communications Subsystem

REFERENCE MANUAL

Printed in U.S.A. August 1980 1090909

This edition includes the information released under the following:
PCN 1090909-001 (February 22, 1979)

Any comments or suggestions regarding this publication should be forwarded to Systems Documentation, Technical Information Organization, TIO-East, Burroughs Corporation, P.O. Box CB7, Malvern, Pennsylvania 19355.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT)

TABLE OF CONTENTS (CONT)

TABLE OF CONTENTS (CONT)

TABLE OF CONTENTS (CONT)

TABLE OF CONTENTS (CONT)

TABLE OF CONTENTS (CONT)

TABLE OF CONTENTS (CONT)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1

GENERAL INFORMATION

GENERAL

This reference manual covers the total data communications network on the Burroughs Computer Management Systems (CMS). The syntax and semantics of each of the languages (NDL, MPL, and COBOL) are not extensively documented here; rather this manual helps explain and define the interfaces between the three languages and the data communications subsystem. The specific syntax and semantics may be found in the following reference manuals:

CMS COBOL Reference Manual, form 2007266
CMS MPL Reference Manual, form 2007563
CMS NDL Reference Manual, form 1090925

DOCUMENTATION CONVENTIONS

Throughout this document, bit 15 is defined as the most significant bit (2 to the 15th power) and bit 0 as the least significant bit (2 to the 0th power).

Items in the document marked RESERVED are considered "don't cares" to the user but, in fact, are unmolested by the DC subsystem.

Items marked as IMPLEMENTATION DEPENDENT are also considered don't cares to the user but may be manipulated by some implementation groups.

For bit flags which denote availability or presence of some condition;

Set	=	Available/Present
Reset	=	Unavailable/Absent

For those bit flags having dual meaning, the first meaning is the reset condition; the second meaning is the set condition. For example,

BIT X = ASCII/EBCDIC = 0/1

All use of the term "task number" within this document refers to the external mix number of the task.

SYSTEM LIMITATIONS

The number of stations, lines, queues, and related items is limited due to the NDLSYS file layout. Current limitations within the data comm subsystem are:

1. The number of different terminals must be less than 128.
2. The number of different modems must be less than 256.
3. The number of lines must be less than 256.
4. The number of subnet queues must be less than 256.
5. The number of stations depends on the type of station as follows:
   a. If all stations are non-BDLC and do not use extended tallies, the number of stations must be less than 1,130.
   b. If all stations are BDLC or use extended tallies, there must be less than 820 stations.

MESSAGE CONTROL SYSTEM INTERFACE

The message control system (MCS) is the central portion of the data communications system and, as such, has total control over the operations performed by the system. It has sole responsibility for allowing or denying accesses by user programs to portions of the data comm network; message switching, either on a global basis (automatic message routing) or on a message-by-message basis (context switching); error handling; and reconfiguration of the data comm network. In addition, the MCS may perform such functions as auditing and log-in/out validation, depending on the requirements of the system.

In order to perform data comm functions on a system, it is necessary that one, and only one, MCS be present in the system. The MCS runs as a normal task (with a few minor exceptions) under the standard operating system, and is written in a high-level language. This language is MPLII, augmented by certain constructs required to perform MCS-specific tasks. These language constructs are described in Section 2.

OPERATOR INTERFACE

The MCS may receive input from, or send output to, the system operator's console using the normal **ACCEPT** and **DISPLAY** constructs. Unsolicited SPO input messages are preceded with the characters DC. The system control language recognizes these characters as defining a message intended for the MCS, and alerts the data communications subsystem. The message text, after being stripped of the DC characters, is transferred into a data communications subsystem message space, which is ultimately placed on the MCS queue.

The interpretation of the message text is defined by the particular MCS.

There are two cases in which a DC input message may be refused by the system. They are:

1. There is no MCS in the mix.
2. There is no available DC message space in which to transfer the message text.

The system informs the operator of the appropriate condition by displaying either **DC INVALID** or **DC NOSPACE** on the system operator's console.

DC ERRORS

Within the MCS communicate descriptions which appear elsewhere in this document, the phrase "an error is monitored" is used to indicate a communicate failure. The ultimate effect of such a failure is determined by the <error option> available to most communicates, where:

<error option> ::= <empty> |, ERROR

If a communicate is unsuccessful and the <error option> = empty and event number and a corresponding error message are printed and the MCS must be discontinued. The vehicles used for discontinuing an MCS are the DS or DP SPO input messages:

DS <mix> <program name>
DP <mix> <program name>

If a communicate is unsuccessful and the <error option> = ERROR, the most significant eight bits of the communicate result field (fetch value) are set equal to @20@ and the remaining 16 bits are set equal to the appropriate event number. Fetch value is then made available for the programmer to interrogate.

The event numbers assigned for use by the DC subsystem range in value from 200 through 299. This range has been further divided into the following categories:

200-254 272-274 282-284 303-304	Implementation Independent Errors
255-269	Implementation Dependent Errors B 80
270-271 279-281	Implementation Dependent Errors B 800/CP 9400
285-299	Implementation Dependent Errors B 1800
300-302 305-320	Implementation Dependent Errors CP 9500

The following is a list of the currently defined event numbers and the corresponding messages.

Event Number	Message
200	DC ERROR BAD MESSAGE TYPE The type field in the message header contains a value > 12. ERROR is returned after: QUEUE
201	DC ERROR BAD STATION NO A reference has been made to LSN > STATION COUNT-1. ERROR is returned after: ALLOW.OUTPUT CLEAR CONTINUE.STATION DISALLOW.OUTPUT QUEUE REDEFINE.STATION ROUTE.OUTPUT SET.INPUT.LIMIT STATION.DESCRIPTION STATION.STATUS
202	DC ERROR BAD QUEUE REF A reference has been made to an invalid queue. Certain communicates are restricted to certain queues, therefore, the queue referenced may exist but be invalid in this context. ERROR is returned after: CLEAR DEQUEUE QUEUE QUEUE DEPTH RECALL ROUTE.INPUT ROUTE.OUTPUT SET.QUEUE.LIMIT
203	DC ERROR BAD SUBNET NO A reference has been made to an <SUBN> SUBNET COUNT-1. ERROR is returned after: ALLOW.INPUT DISALLOW.INPUT SUBNET.DESCRIPTION SUBNET.STATIONS
204	DC ERROR TEXT SIZE TOO BIG The text length field is the message header was set > message length field. ERROR is returned after: WRITE.HEADER
205	DC ERROR NULL MREF An attempt has been made to perform a function on a null message reference, in other words, one

which does not refc-ence a message space. ERROR is returned after:
COPY.TEXT
QUEUE
READ.HEADER
READ.TEXT
WRITE.HEADER
WRITE.TEXT

206	DC ERROR BYTE INDEX TOO BIG The starting byte index of a text transfer is illegal. For the source message the index must be less than text length. For the destination message, the index must be less than message length. ERROR is returned after: COPY.TEXT READ.TEXT WRITE.TEXT
207	DC ERROR BAD TASK NO The task number referenced is not currently in the mix or is outside the range of the mix table. ERROR is returned after: ALLOW.INPUT ALLOW.OUTPUT CONTINUE.TASK DISALLOW.INPUT DISALLOW.OUTPUT SET.OUTPUT.LIMIT TASK.NAME
208	DC ERROR BAD LINE NO The LLN referenced is greater than LINE.COUNT-1. ERROR is returned after: QUEUE LINE.DESCRIPTION LINE.STATIONS LINE.STATUS REDEFINE.LINE REDEFINE.STATION
209	DC ERROR BAD MODEM NO A reference has been made to an LMN greater than MODEM.COUNT-1. ERROR is returned after: MODEM.DESCRIPTION REDEFINE.LINE REDEFINE.STATION
210	DC ERROR BAD TERMINAL NO A reference has been made to LTN greater than TERMINAL.COUNT-1. ERROR is returned after: TERMINAL.DESCRIPTION REDEFINE.STATION
211	DC ERROR NO SPACE No message space available to execute the communicate. ERROR is returned after: CLEAR RECALL
212	DC ERROR STATION NOT ATTACHED An attempt was made to make an unattached station ready. ERROR is returned after: QUEUE
213	DC ERROR COMM NOT IMPLEMENTED The last communicate issued is not unimplemented on this CMS system.
214	DC ERROR LIMIT NOT ALLOWED A queue limit of 0 has been specified. ERROR is returned after: SET.INPUT.LIMIT SET.OUTPUT.LIMIT SET.QUEUE.LIMIT

Event numbers 220-228 occur during configuration.

Even Number	Message
220	DC ERROR STATION ALREADY ATTACHED The LLN of an attached station has been set to a value other than @FF@. ERROR is returned after: REDEFINE.STATION
221	DC ERROR ATTRIBUTE MISMATCH The new attributes of the station or line are inconsistent with the existing network definition. ERROR is returned after: REDEFINE.STATION REDEFINE.LINE
222	DC ERROR DIRECT CONNECT LINE An attempt was made to assign a modem to a direct-connect line. ERROR is returned after: REDEFINE.LINE
223	DC ERROR FULL DUPLEX MISMATCH Attribute mismatch of full duplex terminal. ERROR is returned after: REDEFINE.LINE REDEFINE.STATION
224	DC ERROR INCOMPLETE VARIABLE The length of the parameter area, to be used for reconfiguration, is insufficient. ERROR is returned after: REDEFINE.LINE REDEFINE.STATION
225	DC ERROR IMPROPER LINE CONDITION The line being redefined is not in the required state of not-ready and, for a switched line, not switched busy or not connected. ERROR is returned after: REDEFINE.LINE
226	DC ERROR MESSAGES QUEUED Messages are queued for output to the station referenced by REDEFINE.STATION. Messages are queued for output to a station on the line referenced by REDEFINE.LINE. ERROR is returned after: REDEFINE.STATION REDEFINE.LINE
227	DC ERROR NO VACANCY ON LINE The MAXSTATIONS statement in the line section of NDL defines the maximum number of stations which may be attached to a particular line. An attempt has been made to attach a station to a line which already has MAXSTATIONS.
228	DC ERROR SPEED MISMATCH The speed specified for a station, when either redefining the station or attaching the station to a line, does not match the speeds of the other stations on that line.
229	DC ERROR QUEUE FULL The MCS has attempted to queue a message which would cause the queue count field of the station table or subnet table to overflow.
230	DC ERROR NDL DCP MISMATCH The data comm loader has detected an inconsistancy between the NDL code file and the DCP code file. Possibly the DCP code file was not generated from this NDL code file.

231-248	RESERVED FOR EXPANSION
249	DC LOAD/EOJ BAD NDL PRIORITY CLASS The NDLSYS file does not have the correct value in the priority class field of the PPB @ 3180@.
250	DC LOAD/EOJ FAILURE DISK ERROR The NDLSYS or DCP file cannot be read because of a disk I/O failure.
251	DC LOAD/EOJ FAILURE NDL DATA ERROR The NDLSYS file either has a line with address invalid for the B 800 or specifies an amount of required memory which is insufficient for the tables and buffers declared.
252	DC LOAD/EOJ FAILURE INSUFFICIENT MEMORY The memory space required field of the preset data in the NDLSYS file specifies more space than the MCP can provide.
253	DC LOAD/EOJ FAILURE CANNOT CLOSE NDL FILE Performing the close communicate on the NDLSYS file has failed.
254	DC LOAD/EOJ FAILURE CANNOT OPEN NDL FILE Performing the open communicate on the NDLSYS file has failed. For example: 1. The file is not on disk 2. The file has the wrong file type.
272	DC ERROR PROCESSOR NUMBER INVALID A Load/Reload specifies an invalid DCP.
273	DC ERROR PROCESSOR BUSY A reload specifies a DCP which is busy.
274	DC ERROR PROGRAM FILE NAME INVALID The DCP file name specified in a reload is not defined in the NDLSYS file.
282	DC LOAD/EOJ FAILURE CANNOT CLOSE DCP FILE Performing close communicate on the DCP file has failed.
283	DC LOAD/EOJ FAILURE CANNOT OPEN DCP FILE Open communicate on DCP file has failed for one of the following reasons: 1. The file is not on disk. 2. Bad file type. 3. The file is larger than the DCP memory.
284	DC LOAD/EOJ FAILURE DC* NOT ON SYSTEM The specified DCP has not been warmstarted.
303	DC DCP* NOT LOADED This message will be displayed subsequent to detection of a DCP related load error, to indicate the DCP is in error.
304	DC NO DCPs LOADED This message is displayed if no DCPs have been loaded. This condition is fatal to the data comm load.

BUFFER LIMITING

In order to prevent a task, or group of tasks, from monopolizing the use of message spaces, the ability of a task to allocate a message space is limited by the capacity of the servicing task to process and de-allocate the message space. This is accomplished by placing changeable limits on the depth of station and subnet queues, and also by giving the MCS the ability to delay or suspend input from a particular task or station.

Messages may pass through the system by six essentially different routines:

TASK → MCS

These are output messages from user tasks with the MCS participating. The DC subsystem maintains an output count and an output limit for each task whose output is directed to the MCS. If a task attempts to issue an output message and its output count is greater than, or equal to, its output limit, message space is not allocated and the task is suspended. The count is incremented with each output attempt and is decremented when the MCS issues a CONTINUE.TASK communicate. The limit is initially set by the DC subsystem, but can be altered by the MCS by means of the SET.OUTPUT.LIMIT communicate.

STATION → MCS

These are input messages from DC stations with the MCS participating. The DC subsystem maintains an unprocessed input count and an unprocessed input limit for each station whose input is directed to the MCS. If a station attempts to input a message and its input count is greater than, or equal to, its input limit, message space is not allocated and the input is refused. The count is incremented when the message space is added to the MCS queue, and decremented when the MCS issues a CONTINUE.STATION communicate. The limit is initially set to two by the NDL compiler, but can be altered by the MCS by means of the SET.INPUT.LIMIT communicate.

TASK → STATION

These are output messages from user tasks without MCS participation. The DC subsystem maintains a queue count and a queue limit for each station queue. If a task attempts to issue an output message to a station whose queue count is greater than, or equal to, the queue limit, message space is not allocated and the task is suspended. The count is incremented when an item is added to the queue and decremented when an item is removed. The limit is initialized to two by the NDL compiler, but can be altered by the MCS by means of the SET.QUEUE.-LIMIT communicate.

STATION → SUBNET QUEUE

These are input messages from a DC station without MCS participation. The DC subsystem maintains a queue count and a queue limit for each subnet queue. If a station attempts to input a message and the subnet queue's count is greater than or equal to its queue limit, message space is not allocated and the input is refused. The count is incremented when an item is added to the subnet queue and decremented when an item is removed. The limit is initialized to two by the DC subsystem, but can be altered by the MCS by means of the SET.QUEUE.LIMIT commmunicate.

MCS → SUBNET QUEUE

The MCS may add an item to any subnet queue. The queue count is automatically incremented each time an item is added. The only time that the MCS is denied is when the queue is full. That is, the addition of the item causes an overflow of the queue count field.

MCS → STATION

The MCS may add an item to the queue of any station which is attached to a line. The station queue count is automatically incremented each time an item is added. It should be noted that all items intended for a station must be queued to the NDL queue rather than to a particular station queue. This is done to provide a common interface to the entire NDL process. The only time that the MCS is denied is when the queue is full. That is, the addition of the item causes an overflow of the queue count field.

QUEUE REFERENCES

<queue reference> ::= <expression>

The 16-bit value of <queue reference> has the following format in order to identify the MCS, NDL, subnet, and station queues:

4 BITS	12 BITS
QUEUE TYPE	QUEUE NUMBER

The queue type has these values:

0 (0000)	= MCS QUEUE
1 (0001)	= NDL QUEUE
2 (0010)	= SUBNET QUEUE
3 (0011)	= STATION QUEUE

If the queue type indicates the MCS or the NDL queue, then queue number must be zero.

If the queue type indicates a station queue, then queue number should contain the appropriate logical station number.

If the queue type indicates a subnet queue, then queue number should contain the appropriate logical subnet number.

MESSAGE DECLARATIONS

Message declarations declare one or more variables of type message reference which, when set, hold references to data comm message spaces.

There exists in the machine an area called the message reference table, which holds references to message spaces which are accessible by the MCS. The size of each entry in the message reference table is four bytes. One unique value must be reserved to designate a null or unset reference.

MESSAGE SPACE HANDLING

The MCS programmer must use extreme caution in handling DC message spaces. Carelessness could seriously affect DC throughput and, in the extreme, could cause thrashing.

The DC subsystem operates out of a predetermined amount of system memory. However, any time it senses that all DC message spaces are in use, it attempts to claim more system memory for its own. This implies two things:

1. DC input is suspended until message space becomes available, and
2. The amount of virtual memory available for overlayable data segments is decreased.

Repeated occurrences of this situation will eventually diminish the overlay area to the extent that thrashing is unavoidable.

To prevent such problems, follow these guidelines:

1. Do not hold message spaces in message references or subnet queues unless absolutely necessary.
2. Transfer data out of DC message spaces as soon as possible.
3. Use the RELEASE.MESSAGE SPACE communicate instead of waiting for space to be released automatically.
4. Do not use a message space that is capable of holding more text than is necessary (some message types don't require any text space at all).

5. Try to keep the MCS queue empty - it may contain releasable message space and/or important information concerning the status of a station or a line.

6. Do not issue an unrestricted number of output messages - the status of a station or a line may change before the messages are transmitted.

7. Set reasonable limits on the depth of station and subnet queues.

8. Monitor the NOSPACE bit of input message headers.

9. Do not set the message header MCS-flag unless you are interested in the results of both successful and unsuccessful output attempts.

RULES OF DATA TRANSFER

Any time the DCSS performs a data move, the following rules apply:

1. Characters are moved in a left to right fashion.
2. The data is left-justified in the destination area.
3. If the size of the source area is larger than the size of the destination area, the data is right truncated.
4. If the size of the destination area is larger than the size of the source area, the excess destination characters are not space filled.
5. For some communicates, the programmer may specify the length of the move (byte length). However,
   a. If the move is from a DC buffer to a user data segment, the actual length of the move is the smallest of:
      Byte length.
      Message Header Text.Length.
      Number of bytes available from the beginning of the data area to the end of the data segment.
   b. If the move is from a user data segment to a DC buffer, the actual length of the move is the smallest of:
      Byte length.
      Message Header Message.Length.
      Number of bytes available from the beginning of the data area to the end of the segment.
   c. If the move is from one DC buffer to another, the actual length of the move is the smallest of:
      Byte length.
      (Source) Message Header Text.Length.
      (Destination) Message Header Message-.Length.
6. In any case, no indication of the actual number of characters moved is returned to the MCS programmer.
7. The COBOL programmer, on the other hand, may interrogate the CD area TEXT.LENGTH field after a receive operation to find out how many text characters have been moved into his data area.
8. The MPL application programmer may use DC.TEXTLENGTH to determine how many text characters have been moved into his data area.

Network Error Handling

The transmission and reception of data comm messages is performed at the NDL level. NDL is also responsible for first level error handling, for example, retransmission of a message. The NDL usually retries a message a finite number of times. If, within this finite number of retries, successful transmission/reception is not achieved, the error is reported to a higher level — the MCS. In order to utilize the DC subsystem effectively, the MCS programmer must be aware of the events which occur during the reporting process.

When an error of the above type occurs, a message is placed on the MCS input queue by the DC subsystem. The message header result and event fields indicate the cause of the error and should always be examined by the MCS programmer. The message header type field is dependent on the state of the NDL process at the time the error was detected. Input messages result from errors detected in NDL line control or receive request; output messages from transmit request. In the case of input messages, the associated text, if any, represents a partically received messsage and may usually be discarded. In the case of output messages, the associated text must be saved in order to preserve the correct output sequence.

SECTION 2

MCS FUNCTIONS

This section deals with the MCS constructs in MPL and their use. For more detailed explanations, refer to the CMS MPLII Reference Manual, form 2007563.

ALLOW.INPUT

ALLOW.INPUT (<queue number>, <task number> <error option>);

This is a procedure which causes the task referenced by <task number> to become "attached" to the subnet queue specified by <queue number>. That is, the task is allowed to reference the subnet queue for input.

If the task had been waiting for a response to an attachment request regarding <queue number>, the appropriate "attached" indicator is set, and the task is made ready to run.

If the task has not been waiting for a response to an attachment request regarding <queue number>, this is a NO-OP.

If the <queue number> specifies an undefined subnet queue, an error is monitored.

If the task number is 0 or greater than 9, an error is monitored.

ALLOW.OUTPUT

ALLOW.OUTPUT (<station number>, <task number> <error option>);

This is a procedure which causes the task referenced by <task number> to become "attached" to the station specified by <station number>. That is, the task is allowed to reference the station for output.

If the task had been waiting for a response to an attachment request regarding <station number>, the appropriate "attached" indicator is set, and the task is made ready to run.

If the task had not been waiting for a response to an attachment request regarding <station number>, this is a NO-OP.

If the <station number> specifies an undefined station, an error is monitored.

If the <task number> is 0 or greater than 9, an error is monitored.

CLEAR

CLEAR (<queue reference> <error option>);

This ia a procedure which performs an automatic RELEASE.MESSAGE.SPACE on any messages on the station or subnet queue specified by <queue reference>.

The system may require a message space to perform this procedure. If no message space is available, the procedure is not executed, the most significant eight bits of fetch value are set equal to @40@, and the remaining 16 bits are set equal to the event number corresponding to *nospace*.

If the <queue reference> designates the MCS queue or the NDL queue, an error is monitored (bad queue reference).

If the <queue reference> designates an undefined station or subnet queue, an error is monitored (bad station number or bad subnet number).

If the <queue reference> designates a station queue for which the corresponding station is not attached to a line, an error is monitored (station not attached).

CONTINUE.STATION

CONTINUE.STATION (<station number> <error option>);

This is a procedure which allows the system to continue accepting input from a station whose input is routed to the MCS by decrementing the station's "unprocessed input count."

No action is taken if the station's unprocessed input count is 0.

Issue one CONTINUE.STATION for each such message processed by the MCS. Otherwise, input attempts from the station are unsuccessful and "nospace" conditions are reported.

If the <station number> specifies an undefined station, an error is monitored.

CONTINUE.TASK

CONTINUE.TASK (<task number> <error option>);

This a procedure which allows a task to continue issuing "send" messages to a station with output routed to the MCS, by decrementing the task's output count.

No action is taken if either the task's unprocessed output count is 0 or the referenced task is not a data comm task.

One CONTINUE.TASK should be issued for each send message processed by the MCS. Otherwise, the task may be suspended until one is issued or until the route indication is changed.

If the task has been suspended for issuing too many send messages, and the task's output count is now less than the output limit, the task is made ready to run.

If the task number is invalid, (that is, out of range or not currently executing) an error is monitored (bad task number).

COPY.TEXT

COPY.TEXT (<message variable>,<starting byte>, <message variable>,<starting byte>, <byte length> <error option>);

This is a procedure which causes the text of the message space referenced by the first specified <message variable>, starting at the first specified <starting byte> for a length of <byte length> to be placed in the text area of the message space referenced by the second specified <message variable>, starting at the second specified <starting byte>.

If either <message variable> is null, an error is monitored.

If the source <starting byte> is greater than the source TEXT.LENGTH, or if the destination <starting byte> is greater than or equal to the destination MESSAGE.LENGTH, an error is monitored.

The normal rules of data transfer apply.

The contents of the TEXT.LENGTH field of the message headers are not automatically updated as a result of this communicate.

DCP.DESCRIPTION

DCP.DESCRIPTION (<processor number>, <variable> <error option>;

This is a procedure which fills the <variable> with a list of the program file names and the number of terminals associated with each of the program file names declared for this processor number in the NDL program. Each entry in the list is a two-byte number (0-65535) followed by a 12-character (space-filled) name. The format of the information is described in the interrogate layouts section.

An error is monitored when the <processor number> is invalid or unused by this NDL program.

The <variable> must be of type character.

The normal rules of data transfer apply.

DCP.PROCESSORS

DCP.PROCESSORS

This function returns the highest defined logical DCP number plus 1. The DCP number is incremented by 1 to make it 1 rather than 0 relative.

DCP.PROGRAM.COUNT

DCP.PROGRAM.COUNT (<processor number>)   A Function

This is a function which returns the number of program file names declared for this <processor number> in the NDL program. This number is zero if the <processor number> is not used in this NDL program. If the <processor number> is invalid (greater than 1), a value of @FFFF@ is returned.

This function is used in relation to the DCP.DESCRIPTION procedure.

DCP.PROGRAM.NAMES

DCP.PROGRAM.NAMES (<variable>);

This procedure fills the <data variable> with a list of the program file names of the program loaded into each processor. The order of the names is according to the processor number. If a processor is not used by this NDL program, its position in the list is space-filled. Each position is 12 characters long and space-filled to complete any name which is less than 12 characters.

The <variable> must be of type character.

The normal rules of data transfer apply.

DCP.PROGRAM.TERMINALS

DCP.PROGRAM.TERMINALS (<processor number>, <variable>,<program name>,<error option>);

This is a procedure which fills the <variable> with a list of the terminals declared for this <processor number> and <program name> in the NDL program. Each entry in the list is a two-byte logical terminal number (0-65535).

The number of items in this list reflects the number of terminals returned with this <program name> by the DCP.DESCRIPTION interrogate for this <processor number>.

If the <processor number> is invalid, an error is monitored. Likewise, if the <program name> is incorrect, an error is monitored.

The normal rules of data transfer apply.

DCP.RELOAD

DCP.RELOAD (<processor number>,<program name><error option>);

This procedure causes the data communications processor identified by <processor number> to be loaded with the program file identified by <program name>.

If the <processor number> is invalid (greater than 1) or is not used by this NDL program, an error is monitored. Likewise, if the <program name> is incorrect, an error is monitored.

The <program name> must be of type character.

The <processor number> being reloaded must be in an idle state, or else an error is monitored.

DEQUEUE

DEQUEUE (<message variable>,<queue reference> <error option>);

This is a built-in procedure which causes the top message on the subnet queue specified by <queue reference> to be unlinked and a reference to it to be filled into the <message variable>. If the message variable is not initially null, the message space originally referenced is "released" before the new message is acquired. If the queue is empty, the message reference is left as null. Any <queue reference> other than a valid subnet queue causes an error.

DISALLOW.INPUT

DISALLOW.INPUT (<queue number>,<task number> <error option>);

This is a procedure which causes the task referenced by <task number> to become unattached from the subnet queue specified by <queue number>. That is, the task is not allowed to reference the subnet queue for input. If the task had been suspended because it was necessary for the DC communicate handler to issue an Attach Queue message to the MCS regarding the specified <queue number>, the task is made ready to run.

Unless the <queue number> specifies a valid subnet queue, an error is monitored.

If the <task number> is 0 or greater than 9, an error is monitored.

DISALLOW.OUTPUT

DISALLOW.OUTPUT (<station number>,<task number> <error option>);

This is a procedure which causes the task referenced by <task number> to become unattached from the station specified by <station number>. That is, the task is not allowed to reference the station for output. If the task had been suspended because it was necessary for the DC communicate handler to issue an attach station message to the MCS regarding this station, the task is made ready to run.

If the <station number> specifies an undefined station, an error is monitored.

If the <task number> is 0 or greater than 9, an error is monitored.

EXCHANGE.REFERENCE

EXCHANGE.REFERENCE (<message variable>, <message variable>);

EXCHANGE.REFERENCE causes the contents of the first specified <message variable> to be exchanged with the contents of the second specified <message variable>.

FETCH.MESSAGE

FETCH.MESSAGE (<message variable> <wait option>);
<wait option>::= <empty>, NOWAIT

This is a procedure which causes the top message on the MCS queue to be unlinked and a reference to it to be filled into the <message variable>. If the message reference was not initially null, the message space originally referenced is released (returned to the free pool and its contents lost) before the new message is acquired.

If wait option = <empty>, and the MCS queue is empty, the MCS is suspended until the MCS queue becomes active.

If wait option = NOWAIT, and the MCS queue is empty, the <message variable> is left as null and control is immediately returned to the MCS.

GET.MESSAGE.SPACE

GET.MESSAGE.SPACE (<message variable>, <byte length>);

This is a procedure which acquires a message space capable of holding <byte length> text characters and fills the <message variable> with a reference to it.

If the <message variable> is not initially null, the referenced space is released.

If an insufficient amount of message space is available, the message variable is left as null.

LINE.COUNT

LINE.COUNT A Function

This is a function which returns the number of data communication lines defined in the NDL program.

LINE.DESCRIPTION

LINE.DESCRIPTION (<line number>,<variable> <error option>);

This is a procedure which causes the definition of the line referenced by <line number> to be placed in the <variable>. The format of the information is described in the interrogate layouts section of this document.

If the <line number> designates an undefined line, an error is monitored.

The <variable> must be of type character.

The normal rules of data transfer apply.

LINE.NUMBER

LINE.NUMBER (<line address>) A Function
<line address> ::= <expression>

This is a function which, given a physical line address in <line address>, returns the corresponding logical line number. If the <line address> is not defined in the NDL program, a value of @FFFF@ is returned.

LINE.STATIONS

LINE.STATIONS (<line number>,<variable> <error option>);

This is a procedure which places in the <variable> the logical station numbers of the stations attached to the line referenced by <line number>. The format of the information is described in the Interrogate Layouts section of this document.

NOTE
No more than 100 stations may be attached to a line at any given time.

If the <line number> designates an undefined line, an error is monitored.

The normal rules of data transfer apply.

LINE.STATUS

LINE STATUS (<line number>, <variable> <error option>);

This is a procedure which causes the current status of the line referenced by <line number> to be placed in the <variable>. The information format is described in the Interrogate Layouts section of this document.

If the <line number> designates an undefined line, an error is monitored.

The normal rules of data transfer apply.

MODEM.COUNT

MODEM.COUNT A Function

This function returns the number of modems defined in the NDL program.

The NDL compiler always generates two dummy modem tables for direct connect and BDI lines. Modem 0 is assigned to any direct connect line. Modem 1 is assigned to any BDI line. Therefore, the value returned by MODEM.COUNT is always equal to : (Number of explicitly defined modems) + 2.

MODEM.DESCRIPTION

MODEM.DESCRIPTION (<modem number>, <variable> <error option>);
<modem number>::= <expression>

This procedure causes the definition of the modem referenced by <modem number> to be placed in the <variable>. The format of the information is described in the interrogate layouts section of this document.

If the <modem number> designates an undefined modem, an error is monitored.

The <variable> must be of type character.

The normal rules of data transfer apply.

NULL

NULL (<message variable>) A Function

This is a functional S-OP which returns a true value if the <message variable> is null, and a false value otherwise. A <message variable> is null if it does not reference a valid message space.

QUEUE

QUEUE (<message variable>,<queue reference> <error option>);

This procedure causes the message referenced by the <message variable> to be added to the tail of the queue specified by <queue reference>. The <message variable> is then set to null.

If the <message variable> was already null, an error is monitored (NULL MREF).

If the <queue reference> designates any of the following:

1. A station queue.
2. An undefined subnet queue.
3. The MCS queue with non zero queue number.
4. The NDL queue with non zero queue number.

an error is monitored (BAD QUEUE REF).

If the <queue reference> designates a queue for which the queue count is equal to 255, an error is monitored (QUEUE FULL).

If the <queue reference> designates the NDL queue and the appropriate DCP is in a hardware error state, the message is returned to the MCS queue with result field equal to the DC HARDWARE error. However, if the MCS queue count equals 255, the message is queued. Instead, an error is monitored (QUEUE FULL).

If the <queue reference> designates the NDL queue, the message header must satisfy the following conditions:

1. MESSAGE.TYPE field must be less than 12 (else, BAD, MSG, TYPE).
2. For the following messages the MESSAGE-.LINE field must contain a valid logical line number (else, BAD LINE NUMBER):

   MAKE LINE READY/NOT READY
   DIALOUT
   IMMEDIATE LINE NOT READY

   Also, the line must have at least one station attached (else, BAD MSG TYPE).
3. For the following messages the MESSAGE.ST-ATION field muust contain a valid logical station number (else, BAD STATION NUMBER).

   OUTPUT
   PRIORITY OUTPUT
   ENABLE/DISABLE INPUT
   MAKE STATION READY/NOT READY

   Also, the designated station must be attached to a line (else, STATION NOT ATTACHED).
4. If the MESSAGE.TYPE = DIALOUT, the test length must be non zero (else, BAD MSG TYPE).

If the <queue reference> designates a subnet queue and the logical station number field of the message header is invalid, an error is monitored (BAD STATION NUMBER).

QUEUE.DEPTH

QUEUE.DEPTH (<queue reference>) A Function

This is a function which returns a value indicating the number of messages on the queue specified by <queue reference>.

If the <queue reference> designates the NDL queue, an undefined station, or an undefined subnet queue, a value of @FFFF@ is returned.

If the <queue reference> designates the MCS queue, then queue number must be zero or a value of @FFFF@ is returned.

READ.HEADER

READ.HEADER (<message variable> <variable> <error option>);

This procedure causes the header information of the message space referenced by the <message variable> to be placed in the <variable>.

If the <message variable> is null, an error is monitored.

The <variable> must be of type character.

The normal rules of data transfer apply.

READ.TEXT

READ.TEXT (<message variable>, <starting byte>, <byte length>, <variable> <error option>);

This is a procedure which causes the text contained in the message space referenced by <message variable> starting at byte <starting byte> for a length of <byte length> to be placed into the <variable>.

If the <starting byte> is greater than the TEXT-.LENGTH given in the message header, an error is monitored.

If the <message variable> is null, an error is monitored.

The <variable> must be of type character.

The normal rules of data transfer apply.

RECALL

RECALL (<queue reference> <error option>);

This is a procedure which causes all messages on the referenced station queue or subnet queue to be delinked and placed on the MCS queue, followed by an end-recall message.

If no message space is available to formulate the end-recall message, the procedure is not executed, the most significant eight bits of fetch value are set equal to @40@, and the remaining 16 bits are set equal to the event number corresponding to "nospace."

Messages recalled from a station's (type bits) output save queue are marked with a result field = "recalled from output save queue."

Messages recalled from a station queue are marked with a result byte = "recalled from station" and the end-recall message is of type "end recall from station."

Messages recalled from a subnet queue are marked with a result byte = "recalled from subnet queue" and the end-recall message is of type "end recall from queue".

If the <queue reference> designates any of the following, an error is monitored:

1. The MCS queue.
2. The NDL queue.
3. An undefined station queue.
4. An undefined subnet queue.
5. A station queue for which the corresponding station is not attached to a line.

REDEFINE.LINE

REDEFINE.STATION (<line number>,<variable> <error option>);

This is a procedure which allows the programmer to change certain characteristics of the line referenced by <line number>. The format of the information supplied in the variable is described in the Interrogate Layouts section. The variable must be of type character.

If the <line number> designates an undefined line, an error is monitored.

If the system cannot perform the redefinition because of some inconsistency in the data, an error is monitored.

REDEFINE.STATION

REDEFINE.STATION (<station number>,<variable> <error option>);

This is a procedure which allows the programmer to change certain characteristics of the station referenced by <station number>. The format of the information supplied in the variable is described in the Interrogate Layouts section of this document. The variable must be of type character.

If the <station number> designates an undefined station, an error is monitored.

If the system cannot perform the redefinition because of some inconsistency in the data, an error is monitored.

RELEASE.MESSAGE.SPACE

RELEASE.MESSAGE.SPACE (<message variable>);

This is a procedure which causes the message space referenced by the <message variable> to be returned to the available pool, and the <message variable> to be marked null. If initially it was null, this is a NO-OP.

ROUTE.INPUT

ROUTE.INPUT (<station number>,<queue reference> <reroute> <error option>);
<reroute ::= <empty> |, REROUTE

This is a procedure which causes all subsequent input messages from the station referenced by <station number> to be placed onto the queue specified by <queue reference>.

If the previous and new routing specify the same queue, then no action is taken.

If the previous routing was to the MCS queue and the new routing is to a subnet queue, then any and all "non special" input messages from the station are delinked from the MCS queue and placed on the subnet queue. A non special input message has a message header result field of zero.

For each message moved from the MCS queue to the subnet queue, the station's unprocessed input count is decremented and the subnet queue count is incremented for the destination subnet queue. No checks are made to prevent the subnet queue count from exceeding the subnet queue limit.

If the previous routing was to a subnet queue, the new routing is to a different subnet queue, and the reroute option was specified, then any and all input messages from the station are delinked from the first queue and placed on the second queue.

For each message removed from a subnet queue, the subnet queue count is decremented. For each message placed on the MCS queue, that station's unprocessed input count is incremented, and for messages placed on another subnet queue, that subnet queue's count is incremented. No checks are made to prevent the subnet queue or unprocessed input counts from exceeding the limits.

The input order of the messages is always maintained.

If the <queue reference> specifies the NDL queue, any station queue or an undefined subnet queue, an error is monitored.

If the <queue reference> specifies the MCS queue but queue number is not zero, an error is monitored.

If the <station number> specifies an undefined station, or one which is not attached to a line, an error is monitored.

Checks are made to prevent the destination queue from exceeding 255 entries in the process of executing this communicate. The rerouting is performed until the destination queue is full; then, rerouting is discontinued and an error result (QUEUE FULL) is returned to the MCS. Routing paths are not modified until the last message is successfully rerouted.

ROUTE.OUTPUT

ROUTE.OUTPUT (<station number>,<queue reference> <error option>);

This is a procedure which causes all subsequent output intended for the station referenced by <station number> to be placed onto the queue specified by <queue reference>. If the station's output is to be routed to the NDL queue and had been routed to the MCS queue, the MCS queue is scanned and all messages of type "send" for the station are delinked from the MCS queue and linked to the NDL queue after their message types have been changed to "output". The order of the messages is maintained, and the appropriate queue depths are updated.

For each message rerouted from MCS queue to the NDL queue, the apppropriate task output count is decremented and, ultimately, the station queue count must be incremented. Any SEND messages that contain an invalid task number are not rerouted.

Checks are made to prevent the destination queue from exceeding 255 entries in the process of executing this communicate. The rerouting is performed until the destination queue is full; then, rerouting is discontinued and an error result (QUEUE FULL) is returned to the MCS. Routing paths are not modified until the last message is successfully rerouted.

In the case when routing is changed from the NDL queue to the MCS queue, no messages are rerouted.

If the <queue reference> specifies any station queue or any subnet queue, an error is monitored.

If the <station number> specifies an undefined station, or one which is not attached to a line, an error is monitored.

SET.INPUT.LIMIT

SET.INPUT.LIMIT (<station number>,<limit> <error option>); limit < 128

This is a procedure which causes the system's buffer limiting mechanism to restrict the number of input messages accepted from a station, whose input is routed to the MCS, to be less than the <limit>. That is, the station's unprocessed input limit is set equal to <limit>.

Limiting value is updated regardless of whether station input is currently directed to the MCS.

If <limit> is greater than 127, the limiting value is set to 127.

If the <station number> designates an undefined station, an error is monitored.

SET.OUTPUT.LIMIT

SET.OUTPUT.LIMIT (<task number>,<limit> <error option>); limit < 128

This is a procedure which causes the system's buffer limiting mechanism to restrict the number of output messages issued by a user task to any station whose output is routed to the MCS to be less than the <limit>. That is, the task's output limit is set equal to <limit>.

If the task number is invalid, an error is monitored. If the <limit> is greater than 127, the limiting value is set to 127.

If the specified task is not executing, the results of this communicate are undefined.

If the specified task has been suspended for issuing too many SEND messages and the new limit is now greater than the output count, the task is made ready to run.

SET.QUEUE.LIMIT

SET.QUEUE.LIMIT (<queue reference>,<limit> <error option>); limit < 128

This is a procedure which causes the system's buffer limiting mechanism to restrict the number of items placed on the queue to less than the <limit>. That is, the appropriate queue limit is set equal to <limit>.

If the <queue reference> designates the MCS queue or the NDL queue and the queue number is zero, SET.QUEUE.LIMIT becomes a NO-OP.

If the <queue reference> designates the MCS queue or the NDL queue and queue number is non-zero, an error is monitored.

If the <queue reference> designates an undefined station or an undefined subnet queue, an error is monitored.

If <limit> is greater than 127, the limiting value is set to 127.

If a new limit is set for a station queue, and there are tasks suspended on output (MCS non-participating) to that station, and if the new limit is greater than the old, those tasks are made ready to run.

STATION.COUNT

STATION.COUNT   A Function

This is a function which returns the number of stations defined in the NDL program.

STATION.DESCRIPTION

STATION.DESCRIPTION (<station number>,<variable> <error option>);

This procedure causes the definition of the station referenced by <station number> to be placed in the <variable>. The format of the information is described in the Interrogate Layouts section of this document.

If the <station number> designates an undefined station, an error is monitored.

The <variable> must be of type character.

The normal rules of data transfer apply.

STATION.NUMBER

STATION.NUMBER (<station name>)
<station name> ::= <expression>   A Function

This is a function which, given a character string in <station name>, returns the corresponding logical station number.

If the given character string is not a station name known to the system, a value of @FFFF@ is returned.

If the given character string is less than 12 characters in length, spaces are automatically added to produce a 12 character <station name>.

If the given character string is more than 12 characters in length, only the first 12 characters are used as the <station name>.

The <expression> must be of type character.

STATION.STATUS

STATION.STATUS (<station number>,<variable> <error option>);

This is a procedure which causes the current status of the station referenced by <station number> to be placed in the <variable>. The format of the information is described in the interrogate layouts section of this document.

If the <station number> designates an undefined station, an error is monitored.

If the <station number> references an unattached station, the bit representing STATION READY has no meaning.

The normal rules of data transfer apply.

SUBNET.COUNT

SUBNET.COUNT A Function

This is a function which returns the number of files (subnet queues) defined in the NDL program.

SUBNET.DESCRIPTION

SUBNET.DESCRIPTION (<queue number>, <variable> <error option>);

This is a procedure which causes the definition of the subnet queue referenced by <queue number> to be placed in the <variable>. The format of the information is described in the Interrogate Layouts section of this document.

If the <queue number> designates an undefined subnet queue, an error is monitored.

The <variable> must be of type character.

The normal rules of data transfer apply.

SUBNET.STATIONS

SUBNET.STATIONS (<queue number>,<variable> <error option>);

This is a procedure which places in the <variable> the logical station numbers of the stations defined to be associated with the subnet queue referenced by <queue number>. The format of the information is described in the Interrogate Layouts section of this document.

NOTE
No more than 100 stations may be associated with a particular subnet queue at any given time.

If the <queue number> designates an undefined subnet queue, an error is monitored.

The normal rules of data transfer apply.

SUBNET.STATUS

SUBNET.STATUS (<queue number>, <variable>, <error option>);

This procedure causes the current status of the subnet referenced by <queue number> to be placed in the <variable>. If the <queue number> designates an undefined subnet queue, an error is monitored (BAD SUBNET NUMBER).

SUBNET.NUMBER

SUBNET.NUMBER (<subnet name>)
<subnet name> ::= <expression>   A Function

This is a function which, given a character string in <subnet name>, returns the corresponding subnet queue number.

If the given character string is not a subnet queue name known to the system, a value of @FFFF@ is returned.

If the given character string is less than 12 characters in length, spaces are automatically added to produce a 12 character <subnet name>.

If the given character string is more than 12 characters in length, only the first 12 characters are used as the <subnet name>.

The <expression> must be of type character.

TASK.NAME

TASK.NAME (<task number>,<variable> <error option>);

This is a procedure which, given a <task number>, places in the <variable> the corresponding symbolic task name.

If the <task number> is 0 or greater than 9, an error is monitored.

If the <task number> is within range, but there is no such task in the mix, the <variable> is space filled.

The normal rules of data transfer apply.

The <variable> must be of type character.

TASK.NUMBER

TASK.NUMBER (<task name>)
<task name> ::= <expression>   A Function

This function returns the lowest task number found in the mix table that corresponds to the given <task name>.

If the given character string is not a task name known to the system, a value of @FFFF@ is returned.

If the given character string is less than 12 characters in length, spaces are automatically added to produce a 12 character <task name>.

If the given character string is more than 12 characters in length, only the first 12 characters are used as the <task name>.

The <expression> must be of type character.

TASK.STATUS

TASK.STATUS (<task number>, <variable>, <error option>);

This procedure causes the status of the task referenced by <task number> to be placed in the <variable>.

If the <task number> is invalid, an error is monitored <BAD TASK NUMBER>.

If the <task number> is valid but is not currently executing, byte 0 of the variable is set to @FF@.

TERMINAL.COUNT

TERMINAL.COUNT   A Function

This is a function which returns the number of terminals defined in the NDL program.

TERMINAL.DESCRIPTION

TERMINAL.DESCRIPTION (<terminal number>, <variable> <error option>);

This procedure causes the definition of the terminal referenced by <terminal number> to be placed in the <variable>. The format of the information is described in the interrogate layouts section of this document.

If the <terminal number> designates an undefined terminal, an error is monitored.

The <variable> must be of type character.

The normal rules of data transfer apply.

WRITE.HEADER

WRITE.HEADER (<message variable>,<variable> <error option>);

This procedure causes the data contained in the <variable> to be placed in the header of the message space referenced by <message variable>. Although the "message.length" field is accessible by the programmer, any attempt to change the contents of the field is ignored.

If the <message variable> is null, an error is monitored.

If an attempt is made to set TEXT.LENGTH to a value greater than MESSAGE.LENGTH, an error is monitored.

The <variable> must be of type character.

The normal rules of data transfer apply.

WRITE.TEXT

WRITE.TEXT (<message variable>,<starting byte>, <byte length>, <variable> <error option>);

This procedure causes the text contained in the <variable> to be placed in the message space referenced by <message variable>, starting at <starting byte> for a length of <byte length>.

If the <message variable> is null, an error is monitored.

If the <starting byte> is greater than the "message length" given in the message header, an error is monitored.

The <variable> must be of type character.

The normal rules of data transfer apply.

The contents of the TEXT.LENGTH field of the message header are not automatically updated as a result of this communicate.

SECTION 3
INTERROGATE LAYOUTS

This section deals with data communications interrogates that are performed by certain MCS communicates. This section is referenced by Section 2 and references Section 4. Each line in the layout diagrams represents two characters unless otherwise specified.

DCP.DESCRIPTION

NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NUMBER OF TERMINALS	
. . .	. . .
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NUMBER OF TERMINALS	

NAME: 12 bytes
Program file name.
NUMBER OF TERMINALS: 2 bytes
The number (0-127) of terminals declared to be associated with this program file name.

DCP.PROGRAM.TERMINALS

LOGICAL TERMINAL NUMBER
LOGICAL TERMINAL NUMBER
. .
. .
. .
LOGICAL TERMINAL NUMBER

LOGICAL TERMINAL NUMBER: 2 bytes
The logical terminal number(s) (0-126) associated with a program file name.

DCP.PROGRAM.NAMES

NAME 0	NAME 0
NAME 0	NAME 0
NAME 0	NAME 0
NAME 0	NAME 0
NAME 0	NAME 0
NAME 0	NAME 0
. . .	. . .
NAME N	NAME N
NAME N	NAME N
NAME N	NAME N
NAME N	NAME N
NAME N	NAME N
NAME N	NAME N

NAME : 12 bytes
DCP program file name. One 12-byte name for each data comm processor from zero to N.

LINE.DESCRIPTION

LINE ADDRESS	
TYPE	
MAX ENTRIES	MAX STATIONS
MODEM	

For an explanation of the items see Line Table Layout.

LINE.STATIONS

This is a list of the logical station numbers attached to this line. Each logical station number is two bytes long. The number of items returned is dependent upon the maxstations value of the LINE-

.DESCRIPTION interrogate. However, maxstations never exceeds 100.

LOGICAL STATION NUMBER
LOGICAL STATION NUMBER
LOGICAL STATION NUMBER
. .
LOGICAL STATION NUMBER

LINE.STATUS

STATUS

STATUS: 2 Bytes 16 Bits

15 LINE QUEUED
14 RESERVED
13 STANDBY
12 LINE READY
11 RATE SELECT
10 LINE CONNECTED
9 SWITCHED BUSY
8 LINE BUSY
7 AUXILIARY LINE QUEUED
6 RESERVED
5 RESERVED
4 RESERVED
3 RESERVED
2 RESERVED
1 RESERVED
0 AUXILIARY LINE BUSY

MODEM.DESCRIPTION

TYPE
SPEED
NOISE DELAY
TRANSMIT DELAY

For an explanation of the items, see Modem Table Layout.

Noise delay and transmit delay are in normal binary form (not one's or two's complement).

REDEFINE.LINE

TYPE
MODEM

For an explanation of the items, see Line Table Layout.

REDEFINE.STATION

LOGICAL LINE NO	RUN MODE BITS
END CHARACTER	DELETE CHARACTER
BACKSPACE CHAR	WRU CHARACTER
CONTROL CHARACTER	STATION FREQUENCY
TRANSMIT ADDR 1	TRANSMIT ADDR 2
TRANSMIT ADDR 3	RECEIVE ADDR 1
RECEIVE ADDR 2	RECEIVE ADDR 3
TYPE	
SPEED	
MODEM	TERMINAL
RETRY	

For an explanation of the items, see Station Table Layout.

STATION.DESCRIPTION

NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
LOGICAL LINE NO	RUN MODE BITS
END CHARACTER LINE	DELETE CHARACTER
BACKSPACE CHARACTER	WRU CHARACTER
CONTROL CHARACTER	STATION FREQUENCY
TRANSMIT ADDRESS 1	TRANSMIT ADDRESS 2
TRANSMIT ADDRESS 3	RECEIVE ADDRESS 1
RECEIVE ADDRESS 2	RECEIVE ADDRESS 3
TYPE	
SPEED	
MODEM	TERMINAL
MCS DATA BITS	
WIDTH	PAGE
DIGIT COUNT	PHONE NUMBER
PHONE	NUMBER
PHONE	NUMBER
PHONE	NUMBER
ORIGINAL RETRY	

For an explanation of the items, see Station Table Layout and Extended Station Table Layout.

STATION.STATUS

STATUS	INPUT QUEUE NUMBER
UNPROCESSED INPUT LIMIT	UNPROCESSED INPUT COUNT
STATION QUEUE LIMIT	STATION QUEUE COUNT

7 STATION QUEUED
6 RESERVED
5 RESERVED
4 RESERVED
3 RESERVED
2 STATION ATTACHED
1 ENABLED INPUT
0 STATION READY

INPUT QUEUE NUMBER -1 BYTE

This contains the subnet number to which input from the station is to be routed; @FF@ if routing is to MCS input queue. (If STATION ATTACHED is false, STATION READY has no meaning.)

SUBNET.DESCRIPTION

NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NUMBER OF STATIONS	

NAME: 12 bytes

Symbolic subnet (file) queue name, left-justified with space filler.

NUMBER OF STATIONS: 2 bytes/binary the number of stations declared to be associated with this subnet queue.

SUBNET.STATIONS

LOGICAL STATION NUMBER
LOGICAL STATION NUMBER
LOGICAL STATION NUMBER
. .
. .
. .
LOGICAL STATION NUMBER

This is a list of the logical station numbers associated with this subnet queue. Each logical station number is two bytes long. The number of items returned depends on the "number of stations" field returned by the SUBNET.DESCRIPTION interrogate. The number of stations never exceeds 100.

SUBNET.STATUS

SUBNET QUEUE LIMIT	SUBNET QUEUE COUNT

SUBNET QUEUE LIMIT – 1 BYTE

This is the current maximum number of messages which may be queued on this subnet.

SUBNET QUEUE COUNT – 1 BYTE

This is the number of unprocessed messages on this subnet queue.

TASK. STATUS

STATUS	QUEUE REFERENCE
QUEUE REFERENCE	LIMIT
COUNT	

STATUS: 5 BYTES – 40 BITS

1. Byte 0 = @FF@ if the specified task is not in the mix or is not a user data comm job.
2. Otherwise, byte 0:
   = @00@ if task is not waiting.
   = @01@ if task is waited on QUEUE LIMIT.
   = @02@ if task is waited on RECEIVE.
   = @03@ if task is waited on ATTACH.
   = @04@ if task is waiting for space.
3. Bytes 1, 2:
   = QUEUE REFERENCE if waited for LIMIT, RECEIVE, or ATTACH.
   = @FFFF@ if waiting for space.
4. Byte 3 = TASK OUTPUT LIMIT.
5. Byte 4 = TASK OUTPUT COUNT.

TERMINAL.DESCRIPTION

<table>
<tr><td colspan="6">RUN MODE BITS</td></tr>
<tr><td colspan="2">TR COUNT
OR
SV QUEUE LIMIT</td><td colspan="2">T–AD COUNT</td><td colspan="2">R–AD COUNT</td></tr>
<tr><td colspan="3">SYNC CHARACTER</td><td colspan="3">PARITY MASK</td></tr>
<tr><td colspan="6">STANDARD TIMEOUT</td></tr>
<tr><td colspan="6">TURNAROUND DELAY</td></tr>
<tr><td colspan="6">AUXILIARY LINE CONTROL POINTER</td></tr>
<tr><td colspan="6">LINE CONTROL POINTER</td></tr>
<tr><td colspan="6">RECEIVE REQUEST POINTER</td></tr>
<tr><td colspan="6">TRANSMIT REQUEST POINTER</td></tr>
<tr><td colspan="6">TRANSLATION TABLE POINTER</td></tr>
<tr><td colspan="6">MAXIMUM INPUT SIZE</td></tr>
<tr><td colspan="3">ADAPTER INFO</td><td colspan="3">NUMBER OF BUFFERS</td></tr>
<tr><td colspan="6">TYPE</td></tr>
<tr><td colspan="6">SPEED</td></tr>
<tr><td colspan="6">STOP BITS</td></tr>
<tr><td colspan="6">MCS DATA BITS</td></tr>
<tr><td colspan="3">WIDTH</td><td colspan="3">PAGE</td></tr>
<tr><td colspan="3">CARRIAGE CHARACTER</td><td colspan="3">LINEFEED CHARACTER</td></tr>
<tr><td colspan="3">HOME CHARACTER</td><td colspan="3">CLEAR CHARACTER</td></tr>
</table>

For an explanation of the items, see Terminal Table Layout and Extended Terminal Table Layout.

Standard timeout, turnaround delay, and maximum input size are in normal binary form (not one's or two's complement).

SECTION 4
NDL TABLES

GENERAL

This section deals with the tables that are used by the network definition language to control the various portions of the data communications subsystem. Some of these tables are also used to obtain information for the interrogate layouts described in Section 3. Each line of an NDL table diagram represents two characters unless otherwise specified.

LINE TABLE LAYOUT

LINE DESCRIPTOR	
LINE TALLY (1)	LINE TALLY (0)
MAX ENTRIES	MAX STATIONS
AUX LINE TALLY (1)	AUX LINE TALLY (0)
LINE ADDRESS	
LOGICAL LINE NO	MODEM
TYPE	
AUX LINE DESCRIPTOR	
STATION TALLY (0)	STATION DESCRIPTOR (0)
STATION TABLE POINTER (0)	
↓	↓
STATION TALLY (N−1)	STATION DESCRIPTOR (N−1)
STATION TABLE POINTER (N−1)	

LINE DESCRIPTOR
2 Bytes

This field consists of the 16 one-bit flags listed below. Bits 14, 10, 7, 6, 5, 4, and 3 are set according to information supplied in the NDL program. The remaining bits are initialized to zero by the NDL compiler.

15 LINE QUEUED
14 DIALOUT CAPABLE
13 STANDBY
12 LINE READY
11 RATE SELECT
10 LINE CONNECTED
9 SWITCHED BUSY
8 LINE BUSY
7 LINE PULSE/ACU DIALOUT
6 SWITCHED
5 FULL DUPLEX
4 DISCONNECT ON LOSS OF CARRIER
3 ASYNCHRONOUS
2 RESERVED
1 LINE TOG (1)
0 LINE TOG (0)

LINE TALLY (1)

1 Byte/Binary

Contains the NDL byte variable known as LINE TALLY (1). It is initialized to zero by the NDL compiler.

LINE TALLY (0)

1 Byte/Binary

Contains the NDL byte variable known as LINE TALLY (0). It is initialized to zero by the NDL compiler.

MAX ENTRIES

1 Byte/Binary

Contains the value specified in the NDL program for MAXSTATIONS. It equals the maximum number of stations that may be attached to this line at the same time. MAX ENTRIES can never exceed 100.

MAXSTATIONS

1 Byte/Binary

Contains the run time value MAXSTATIONS. It equals the number of stations currently attached to this line. It is initialized by the compiler according to information supplied in the NDL program. Like MAX ENTRIES, MAXSTATIONS can never exceed 100.

AUXILIARY LINE TALLY (1)

1 Byte/Binary

Contains the NDL byte variable known as AUX

LINE TALLY (1). It is initialized to zero by the NDL compiler.

AUXILIARY LINE TALLY (0)

1 Byte/Binary

Contains the NDL byte variable known as AUX LINE TALLY (0). It is initialized to zero by the NDL compiler.

LINE ADDRESS

2 Bytes/Binary

Contains the line's physical address. This field is initialized by the NDL compiler.

LOGICAL LINE NUMBER

1 Byte/Binary

Contains the logical number that has been assigned to this line by the NDL compiler.

MODEM

1 Byte/Binary

Contains the logical number of the modem that is attached to this line.

TYPE

2 Bytes

Contains the 16 one-bit flags listed below. The flags are initialized by the compiler according to information supplied in the NDL program.

15 SPECIAL
14 BITS
13 BDI
12 TELEX
11 STANDBY TRUE
10 STANDBY OPTION
9 LOW/HIGH RATE
8 RATE SELECT
7 MODEM
6 DISCONNECT ON LOSS OF CARRIER
5 LINE PULSE/ACU DIALOUT
4 DIALOUT
3 DIALIN
2 ASCII/EBCDIC SYNC CHARACTER
1 ASYNCHRONOUS
0 FULL DUPLEX

AUX LINE DESCRIPTOR

2 Bytes

Contains the 16 one-bit flags listed below. This field is initialized to zero by the NDL compiler.

15 AUX LINE QUEUED
14 AUX LINE TOG (0)
13 AUX LINE TOG (1)
12 RESERVED
11 RESERVED
10 RESERVED
9 RESERVED
8 AUX LINE BUSY
7 RESERVED
6 RESERVED
5 RESERVED
4 RESERVED
3 RESERVED
2 RESERVED
1 RESERVED
0 RESERVED

STATION TALLIES

1 Byte Each/Binary

For each of the line's stations, one byte is allocated to contain the NDL byte variable, STATION TALLY. These bytes are initialized to zero by the NDL compiler.

STATION DESCRIPTORS

1 Byte Each

For each of the line's stations, one byte is allocated to contain the eight one-bit flags listed below. The NDL compiler sets bits 7 through 4 to zero, bits 3, 2 and 1 according to information supplied in the NDL program, and bit 0 to binary 1.

7 STATION QUEUED
6 RESERVED
5 RESERVED
4 RESERVED
3 MYUSE OUTPUT
2 MYUSE INPUT
1 ENABLED INPUT
0 STATION READY 1

STATION TABLE POINTERS

2 Bytes Each/Binary

For each of the line's stations, two bytes are allocated to contain a pointer to the appropriate station table.

On disk, this field contains a logical station number supplied by the NDL compiler.

In memory, it contains the absolute address of a station table. This value is inserted by the DC loader at DC initialize time.

STATION TABLE LAYOUT

LOGICAL LINE NO	RELATIVE STATION NO
END CHARACTER	LINE DELETE CHARACTER
BACKSPACE CHARACTER	WRU CHARACTER
CONTROL CHARACTER	STATION FREQUENCY
TRANSMIT ADDRESS–2	TRANSMIT ADDRESS–1
RUN MODE BITS	TRANSMIT ADDRESS–3
RECEIVE ADDRESS–2	RECEIVE ADDRESS–1
RECEIVED	RECEIVE ADDRESS–3
RECEIVE TRANSMISSION NO./OUTPUT SAVE QUEUE HEAD	
TRANSMIT TRANSMISSION NO./OUTPUT SAVE QUEUE TAIL	
LOGICAL STATION NO	
UNPROCESSED INPUT LIMIT	UNPROCESSED INPUT COUNT
ORIGINAL RETRY	RETRY
TALLY (1)	TALLY (0)
TALLY (2)	TOGGLES (7→0)
OPTIONS	EVENTS
EVENTS	
INITIATE RECEIVE DELAY	
ACTIVE TRANSMIT DELAY	
STATION QUEUE HEAD	
STATION QUEUE TAIL	
QUEUE LIMIT	QUEUE COUNT
ATTACHED STATUS	
WAIT STATUS	
SUBNET QUEUE ADDRESS	
RESERVED	LINE PRIORITY CODE
TYPE	
SPEED	
MODEM	TERMINAL
TALLY (4)	TALLY (3)
TALLY (6)	TALLY (5)
TALLY (8)	TALLY (7)
TALLY (10)	TALLY (9)
TALLY (12)	TALLY (11)
TALLY (14)	TALLY (13)
TALLY (16)	TALLY (15)
TALLY (18)	TALLY (17)
OUTPUT SAVE QUEUE COUNT	INPUT SAVE QUEUE COUNT
INPUT SAVE QUEUE HEAD	
INPUT SAVE QUEUE TAIL	

LOGICAL LINE NUMBER

1 Byte/Binary

Contains the logical number assigned to this station's line by the NDL compiler. If the station is not initially attached to a line, this field should contain all ones.

RELATIVE STATION NUMBER

1 Byte/Binary

Contains the station's relative position within the list of stations in the line table. It is initialized by the NDL compiler.

END CHARACTER

1 Byte/ASCII

Contains the ASCII value of the end character specified in the NDL program.

LINE DELETE CHARACTER

1 Byte/ASCII

Contains the ASCII value of the line delete character specified in the NDL program.

BACKSPACE CHARACTER

1 Byte/ASCII

Contains the ASCII value of the backspace character specified in the NDL program.

WRU CHARACTER

1 Byte/ASCII

Contains the ASCII value of the WRU character specified in the NDL program.

CONTROL CHARACTER

1 Byte/ASCII

Contains the ASCII value of the control character specified in the NDL program.

STATION FREQUENCY

1 Byte/Binary

Contains the value specified in the NDL program for frequency.

TRANSMIT ADDRESS

3 Bytes/ASCII

Contains the transmit address characters as specified in the NDL program. A maximum of three characters may be used. Zeroes appear for any character that is not specified.

RUN MODE BITS

1 Byte

Contains the eight one-bit flags listed below. Flags 0 through 3 are initialized to zero by the NDL compiler while flags 4 through 7 are set according to values specified in the NDL program. These bits represent the state of the station at the start of the run.

7 MYUSE OUTPUT —
6 MYUSE INPUT —
5 SECOND STOP BIT —
4 ENABLE INPUT —
3 IMPLEMENTATION DEPENDENT*
2 RESERVED
1 RESERVED
0 RESERVED

*Used by B 800 as the route output indicator (0 = route to MCS).

RECEIVE ADDRESS

3 Bytes/ASCII

Contains the receive address characters as specified in the NDL program. A maximum of three characters may be used. Zeros appear for any character that is not specified.

RECEIVE TRANSMISSION NUMBER

2 Bytes

Contains the receive transmission number in four-bit binary coded decimal form. A maximum of three digits may be used. The entire field is initialized to zero by the NDL compiler.

NOTE

For input messages, this field is moved to the message header at NDL terminate time.

TRANSMIT TRANSMISSION NUMBER

2 Bytes

Contains the transmit transmission number in four-bit binary coded decimal form. A maximum of three digits may be used. The entire field is initialized to zero by the NDL compiler.

OUTPUT SAVE QUEUE HEAD

2 Bytes/Binary

For stations of type bits, contains the absolute address of the next output or priority output message still unacknowledged by the remote station. This field is initialized to zero by the NDL compiler.

Output and priority output messages are queued to this save queue by the NDL construct, "terminate save", executed in the transmit request set.

OUTPUT SAVE QUEUE TAIL

2 Bytes/Binary

For stations of type bits, contains the absolute address of the most recent message still unacknowledged by the remote station. This field is initialized to zero.

LOGICAL STATION NUMBER

2 Bytes/Binary

Contains the logical number assigned to this station by the NDL compiler.

UNPROCESSED INPUT LIMIT

1 Byte/Binary

Contains the maximum value that unprocessed input count is allowed to have. It is initialized to two by the NDL compiler.

UNPROCESSED INPUT COUNT

1 Byte/Binary

Contains the number of input messages from this station, routed to the MCS, that have been accepted by the DC-firmware, but have not yet been processed by the MCS. The field is initialized to zero by the NDL compiler.

ORIGINAL RETRY

1 Byte/Binary

Contains the retry value specified in the NDL program for this station. The maximum value the user may assign to original retry is 254. The value 255 is reserved for system use.

RETRY

1 Byte/Binary

Contains the run time retry count for input messages to this station. It is initialized to the retry value as specified for the original retry.

NOTE

This field is moved to the message header at NDL terminate time. The retry value for output messages, however, is maintained in the message header, not the station table.

TALLIES

Three 1-Byte Fields/Binary

Each field contains one of the NDL station tally byte variables. All three bytes are initialized to zero by the NDL compiler.

TOGGLES

1 Byte

Contains the eight NDL bit variables listed below. The NDL compiler initializes this byte to zero.

7 TOGGLE (7)
6 TOGGLE (6)
5 TOGGLE (5)
4 TOGGLE (4)
3 TOGGLE (3)
2 TOGGLE (2)
1 TOGGLE (1)
0 TOGGLE (0)

OPTIONS

1 Byte

Contains the NDL options flags for input messages to this station. The flags are moved to the message header at NDL terminate time. The NDL compiler initializes this field to zero.

NOTE

The options flags for output messages are maintained in the message header.

7 LINEFEED
6 CARRIAGE
5 PAPERMOTION
4 PAGE
3 SKIP
2 TRANSPARENT
1 BLOCK
0 SPACE

EVENTS

Bytes

Contains the NDL events flags for input messages to this station. The flags are moved to the message header at NDL terminate time. The NDL compiler initializes all three bytes to zero.

NOTE

The events flags for output messages are maintained in the message header.

23 NAK RECEIVED
22 NAK ON SELECT
21 NO SPACE
20 TERMINATE ERROR
19 DISCONNECT
18 TERMINATE NO LABEL
17 ADAPTER FAULT
16 MODEM NOT READY
15 CONTROL CHARACTER RECEIVED
14 WRU CHARACTER RECEIVED
13 TRANSMISSION NUMBER ERROR
12 MESSAGE LENGTH EXCEEDED
11 EVENT 1
10 FORMAT ERROR
9 BCC ERROR
8 ADDRESS ERROR
7 SYNCHRONOUS TRANSMISSION UNDERFLOW
6 BREAK ON TRANSIT
5 LOSS OF CARRIER
4 CHARACTER PARITY ERROR/INVALID FRAME
3 BREAK ON RECEIVE/IDLE
2 BYTE OVERFLOW-SERVICED TOO LATE
1 STOP BIT ERROR/ABORT
0 TIMEOUT

INITIATE RECEIVE DELAY

2 Bytes/Binary Two's Complement

Contains the noise delay value specified in the NDL program for this station's modem.

ACTIVE TRANSMIT DELAY

2 Bytes/Binary Two's Complement

Contains the greatest of the following values:

Terminal turnaround delay
Station modem noise delay
Line modem transmit delay

It is initialized by the NDL compiler.

STATION QUEUE HEAD

2 Bytes/Binary

Contains the absolute address of the next input or output message to be processed by this station. It is initialized to zero by the NDL compiler. The use of this field is implementation dependent.

STATION QUEUE TAIL

2 Bytes/Binary

Contains the absolute address of the final message that is currently waiting to be processed by this station. It is initialized to zero by the NDL compiler. The use of this field is implementation dependent.

QUEUE LIMIT

1 Byte/Binary

Contains the maximum value that queue count is allowed to have. It is initialized to two by the NDL compiler. The use of this field is implementation dependent.

QUEUE COUNT

1 Byte/Binary

Contains the number of message spaces of type output that are currently linked to the station queue. It is initialized to zero by the NDL compiler. The use of this field is implementation dependent.

ATTACHED STATUS

2 Bytes

Contains run time information as to whether or not this station is attached to a particular task. If it is, the bit corresponding to the task's ID is set by the DC firmware. The field is initialized to zero by the NDL compiler.

15 TASK ID 15
14 TASK ID 14
13 TASK ID 13
12 TASK ID 12
11 TASK ID 11
10 TASK ID 10
9 TASK ID 9
8 TASK ID 8
7 TASK ID 7
6 TASK ID 6
5 TASK ID 5
4 TASK ID 4
3 TASK ID 3
2 TASK ID 2
1 TASK ID 1
0 TASK ID 0

The use of this field is implementation dependent.

WAIT STATUS

2 Bytes

Contains run time information as to whether or not a particular task is waiting until queue count becomes less than queue limit.

If a task is being waited, the bit corresponding to that task's ID is set by the DC firmware. The field is initialized to zero by the NDL compiler.

15 TASK ID 15
14 TASK ID 14
13 TASK ID 13
12 TASK ID 12
11 TASK ID 11
10 TASK ID 10
9 TASK ID 9
8 TASK ID 8
7 TASK ID 7
6 TASK ID 6
5 TASK ID 5
4 TASK ID 4
3 TASK ID 3
2 TASK ID 2
1 TASK ID 1
0 TASK ID 0

The use of this field is implementation dependent.

SUBNET QUEUE ADDRESS

2 Bytes/Binary

Contains a pointer to the subnet queue to which this station's input is routed. If input is routed to the MCS, this field contains all ones.

This field is initialized to all ones by the NDL compiler and is updated by the DC communicate handler. The use of this field is implementation dependent.

LINE PRIORITY CODE

1 Byte/Binary

Contains an eight-bit code indicating the speed of this station's line. It is initialized by the NDL compiler according to the line priority chart.

TYPE

2 Bytes

Contains the 16 one-bit flags listed below which are set according to information supplied in the NDL program.

15 SPECIAL
14 BITS
13 BDI

12 TELEX
11 RESERVED
10 RESERVED
9 RESERVED
8 RESERVED
7 MODEM
6 RESERVED
5 RESERVED
4 RESERVED
3 TALLIES
2 ASCII/EBCDIC SYNC CHARACTER
1 ASYNCHRONOUS
0 FULL DUPLEX

SPEED

2 Bytes

Indicates the frequency to be used with this station. The valid speeds are listed as follows by bit position. Note that the bits take on different meanings for synchronous and asynchronous speeds.

The appropriate bits are set by the NDL compiler.

Asynchronous	Synchronous
15-Reserved	Reserved
14-38,400 BPS	Reserved
13-19,200 BPS	Reserved
12-9,600 BPS	Reserved
11-4,800 BPS	Reserved
10-2,400 BPS	Reserved
9-1,800 BPS	Reserved
8-1,200	Reserved
4-600 BPS	9,600 BPS
6-300 BPS	7,200 BPS
5-200 BPS	4,800 BPS
4-150 BPS	3,600 BPS
3-110 BPS	2,400 BPS
2-100 BPS	2,000 BPS
1-75 BPS	1,200 BPS
0-50 BPS	600 BPS

BPS=Bits per second.

MODEM

1 Byte/Binary

Contains the logical number assigned to this station's modem by the NDL compiler.

TALLY (3) THROUGH TALLY (18)

16 Fields, 1 Byte/Binary

Each field contains one of the 16 extra byte variables required by a station of type bits. The extra tallies are assigned via the NDL TALLIES statement. These tallies cannot be stored in the message header. All bytes are initialized to zero by the NDL compiler.

OUTPUT SAVE QUEUE COUNT

1 Byte/Binary

Used only by stations of type bits. Contains the number of messages currently in the output save queue. It is initialized to zero by the NDL compiler.

INPUT SAVE QUEUE COUNT

1 Byte Binary

Used only by stations of type bits. Contains the number of messages currently in the input save queue. It is initialized to zero by the NDL compiler.

INPUT SAVE QUEUE HEAD

2 Bytes/Binary

For stations of type bits, contains the absolute address of the next input message from the station to be acknowledged. This field is initialized to zero by the NDL compiler. Input messages are queued to the input save queue by the NDL instruction TERMINATE SAVE when executed in the receive request set.

INPUT SAVE QUEUE TAIL

2 Bytes/Binary

For station of type bits, contains the absolute address of the latest message still unacknowledged to the remote station. This field is initialized to zero by the NDL compiler.

TERMINAL

1 Byte/Binary

Contains the logical number assigned to this station's terminal by the NDL compiler.

MODEM TABLE LAYOUT

TYPE
SPEED
NOISE DELAY
TRANSMIT DELAY

TYPE

2 Bytes

Contains the 16 one-bit flags listed below. The flags are initialized by the compiler according to information supplied in the NDL program.

15 SPECIAL
14 RESERVED
13 RESERVED
12 RESERVED
11 RESERVED
10 STANDBY OPTION
9 RESERVED
8 RATE SELECT
7 MODEM
6 DISCONNECT ON LOSS OF CARRIER
5 ANSWERTONE NEEDED
4 DIALOUT
3 DIALIN
2 RESERVED
1 ASYNCHRONOUS
0 FULL DUPLEX

SPEED

2 Bytes

Indicates the frequency to be used with this modem. The valid speeds are listed below by bit position. Note that the bits take on different meanings for synchronous and asynchronous speeds.

The appropriate bits are set by the NDL compiler.

	Asynchronous	Synchronous
15	Reserved	Reserved
14	38,400 BPS	Reserved
13	19,200 BPS	Reserved
12	9,600 BPS	Reserved
11	4,800 BPS	Reserved
10	2,400 BPS	Reserved
9	1,800 BPS	Reserved
8	1,200 BPS	Reserved
7	600 BPS	9,600 BPS
6	300 BPS	7,200 BPS
5	200 BPS	4,800 BPS
4	150 BPS	3,600 BPS
3	110 BPS	2,400 BPS
2	100 BPS	2,000 BPS
1	75 BPS	1,200 BPS
0	50 BPS	600 BPS

NOISE DELAY

2 Bytes/Binary - Two's Complement

Contains the noise delay as specified for this modem in the NDL program.

TRANSMIT DELAY

2 Bytes/Binary - Two's Complement

Contains the transmit delay as specified for this modem in the NDL program.

TERMINAL TABLE LAYOUT

RUN MODE BITS		
TR COUNT OR SV QUEUE LIMIT	T–AD COUNT	R–AD COUNT
SYNC CHARACTER	PARITY MASK	
STANDARD TIMEOUT		
TURNAROUND DELAY		
AUXILIARY LINE CONTROL POINTER		
LINE CONTROL POINTER		
RECEIVE REQUEST POINTER		
TRANSMIT REQUEST POINTER		
TRANSLATION TABLE POINTER		
MAXIMUM INPUT SIZE		
ADAPTER INFO	NUMBER OF BUFFERS	
TYPE		
SPEED		
STOP BITS		

RUN MODE BITS

2 Bytes

Contains the 16 one-bit flags listed below. Flags 7, 5, and 2 are initialized to zero by the NDL compiler. The remaining flags are set according to information supplied in the NDL program.

15 VERTICAL
14 HORIZONTAL
13 NO TRANSLATE
12 BCC ONES
11 FULL DUPLEX
10 TRANSPARENT
9 CASESHIFT
8 BCC/CRC
7 RESERVED
6 BITS
5 RESERVED
4 MOD8/MOD128
3 ODD/EVEN PARITY
2 SUMMED PARITY
1 CRC-1/(ECMA)
0 SYNC/ASYNC

TR-COUNT

1 Byte/Binary

Contains the number of digits to be used in the receive and transmit transmission numbers. It is initialized by the NDL compiler.

SV QUEUE LIMIT

1 Byte/Binary

For terminals of type bits, contains the number of messages allowed to be queued to the output save queue. It is initialized to the value specified in the NDL SAVE statement.

T-AD COUNT

4 Bits/Binary

Contains the number of characters to be used in the transmit address. It is initialized by the NDL compiler.

R-AD COUNT

4 Bits/Binary

Contains the number of characters to be used in the receive address. It is initialized by the NDL compiler.

SYNC CHARACTER

1 Byte

For terminals not of type bits. Contains the sync character in either ASCII or EBCDIC form as defined in the NDL. For bits terminals, contains the flag sequence value of Hex 7E.

PARITY MASK

1 Byte

Contains a mask character that has a binary 1 in each bit position that contains data (excluding parity) in a normal data character. The remaining bit positions contain a binary 0. The field is initialized by the NDL compiler.

STANDARD TIMEOUT

2 Bytes/Binary Two's Complement

Contains the timeout value specified in the NDL program.

AUXILIARY LINE CONTROL POINTER

2 Bytes/Binary

On disk, this field contains the logical number of the control set specified for this terminal's AUX line control.

In memory, this field is updated by the DC loader to contain the absolute address of the appropriate control set.

If no AUX line control is specified for this terminal, the field contains all ones.

TURNAROUND DELAY

2 Bytes/Binary Two's Complement

Contains the turnaround time value specified in the NDL program.

LINE CONTROL POINTER

2 Bytes/Binary

On disk, this field contains the logical number of the control set specified for this terminal's line control.

In memory, this field is updated by the DC loader to contain the absolute address of the appropriate control set.

RECEIVE REQUEST POINTER

2 Bytes/Binary

On disk, this field contains the logical number of the request set specified for this terminal's receive request.

In memory, this field is updated by the DC loader to contain the absolute address of the appropriate request set.

If no receive request is specified for this terminal, the field contains all ones.

TRANSMIT REQUEST POINTER

2 Bytes/Binary

On disk, this field contains the logical number of the request set specified for this terminal's receive request.

In memory, this field is updated by the DC loader to contain the absolute address of the appropriate request set.

If no transmit request is specified for this terminal, the field contains all ones.

TRANSLATION TABLE POINTER

2 Bytes/Binary

On disk, this field contains the logical number of the translation table specified for this terminal.

In memory, this field is updated by the DC loader to contain the absolute address of the appropriate table.

If no translation table is specified for this terminal, the field contains all ones.

MAXIMUM INPUT SIZE

2 Bytes/Binary One's Complement

Contains the size in bytes of the largest message that can be inputted from this terminal as specified in the NDL program.

ADAPTER INFO

1 Byte

Contains information used by the DC firmware to condition the hardware. The field is initialized by the NDL compiler from information supplied in the NDL program.

7 RECEIVE PARITY
6 EVEN PARITY
5 ASYNCHRONOUS
4 CHARACTER SIZE
3 CHARACTER SIZE
2 TRANSMIT PARITY
1 RESERVED
0 BINARY 1

NOTE

Character size is a two-bit code indicating the number of bits (including parity) to be used in a normal data character. It is specified as follows:

11 FIVE-BIT CHARACTER
10 SIX-BIT CHARACTER
01 SEVEN-BIT CHARACTER
00 EIGHT-BIT CHARACTER

NUMBER OF BUFFERS

1 Byte/Binary One's Complement

Contains the number of DC buffers needed to hold a message (header plus text) for this terminal. It is computed by the NDL compiler as follows:

1. Let Y equal this terminal's maximum input size
2. Let Z equal 2* (DC-buffer-size-1)
3. Let N equal the integer value (Y + 37)/Z
4. Then "number of buffers" equals the one's complement of N.

TYPE

2 Bytes

Contains the 16 one-bit flags listed below. Flags 15, 14, 13, 12, 8, 7, 3, 2, and 1 are set by the compiler according to information supplied in the NDL program. The remaining flags are initialized to zero.

15 SPECIAL
14 BITS
13 BDI
12 TELEX
11 RESERVED
10 RESERVED
9 RESERVED
8 DIRECT
7 MODEM
6 RESERVED
5 RESERVED
4 RESERVED
3 TALLIES
2 ASCII/EBCDIC SYNC CHARACTER
1 ASYNCHRONOUS
0 FULL DUPLEX

SPEED

2 Bytes

Indicates the frequency to be used with this terminal. The valid speeds are listed below by bit position. Note that the bits take on different meanings for synchronous and asynchronous speeds.

For synchronous terminals, only one bit indicating the maximum speed may be set.

For asynchronous terminals, multiple bits may be set indicating that several speeds are possible.

The appropriate bits are set by the NDL compiler.

Asynchronous	Synchronous
15 RESERVED	RESERVED
14 38,400 BPS	RESERVED
13 19,200 BPS	RESERVED
12 9,600 BPS	RESERVED
11 4,800 BPS	RESERVED
10 2,400 BPS	RESERVED
9 1,800 BPS	RESERVED
8 1,200 BPS	RESERVED
7 600 BPS	9,600 BPS
6 300 BPS	7,200 BPS
5 200 BPS	4,800 BPS
4 150 BPS	3,600 BPS
3 110 BPS	2,400 BPS
2 100 BPS	2,000 BPS
1 75 BPS	1,200 BPS
0 50 BPS	600 BPS

STOP BITS

2 Bytes

For asynchronous terminals, it is possible to specify several speeds. For each speed, it is possible to select either one or two stop bits. This field indicates the number of stop bits associated with each speed according to the speed's bit position as defined above. If the bit is reset, one stop bit is used. If the bit is set, two stop bits are used. This field is initialized by the NDL compiler.

FILE TABLE LAYOUT

INDEX TO LFN-0's LSN-LIST
LFN-0 NUMBER OF STATIONS
LFN-0's LSN-LIST
. . .
. . .
INDEX TO LFN-(N-1)'s LSN-LIST
LFN-(N-1) NUMBER OF STATIONS
LFN-(N-1)'s LSN-LIST
. . .
. . .
. . .

INDEX TO LFN-X'S LSN-LIST

2 Bytes/Binary

Contains the index (byte offset divided by 2) from the base of the file table of the list of logical station numbers of the stations which are associated with logical-file-number X.

LFN-X NUMBER OF STATIONS

2 BYTES/BINARY

Contains the number of stations which are associated with logical-file-number X.

LFN-X'S LSN-LIST

Each Entry Is 2 Bytes/Binary

Contains the list of the logical station numbers of the stations which are associated with logical-file-number X.

EXTENDED STATION TABLE LAYOUT

MCS DATA BITS	
WIDTH	PAGE
DIGIT COUNT	PHONE NUMBER
PHONE NUMBER	
PHONE NUMBER	
PHONE NUMBER	

MCS DATA BITS

2 Bytes

Contains information specified in the NDL program for this station which may be of interest to the MCS.

15 SPO
14 LOGIN
13 WRAPAROUND
12 RESERVED
11 RESERVED
10 RESERVED
9 RESERVED
8 RESERVED
7 RESERVED
6 RESERVED
5 RESERVED

4 RESERVED
3 RESERVED
2 RESERVED
1 RESERVED
0 RESERVED

WIDTH

1 Byte/Binary

Contains the station width as specified in the NDL program.

PAGE

1 Byte/Binary

Contains the station page size as specified in the NDL program.

DIGIT COUNT

4 Bits/Binary

Contains the number of digits in a given phone number. Digit count may range from 0 to 15 and is initialized by the NDL compiler.

PHONE NUMBER

15 Four-Bit Binary Coded Decimal Digits

Contains the station's phone number as specified in the NDL program.

EXTENDED TERMINAL TABLE LAYOUT

MCS DATA BITS	
WIDTH	PAGE
CARRIAGE CHARACTER	LINEFEED CHARACTER
HOME CHARACTER	CLEAR CHARACTER

MCS DATA BITS

2 Bytes

Contains information specified in the NDL program for this station which may be of interest to the MCS.

15 RESERVED
14 RESERVED
13 WRAPAROUND
12 SCREEN
11 BLOCKED
10 TRANSPARENT CAPABLE
9 RESERVED
8 RESERVED
7 RESERVED
6 RESERVED
5 RESERVED
4 RESERVED
3 RESERVED
2 RESERVED
1 RESERVED
0 RESERVED

WIDTH

1 Byte/Binary

Contains the value specified in the NDL program as the terminal's width.

PAGE

1 BYTE/BINARY

Contains the value specified in the NDL program as the terminal's page size.

CARRIAGE CHARACTER

1 Byte/ASCII

Contains the ASCII value of the carriage character specified in the NDL program.

LINEFEED CHARACTER

1 Byte/ASCII

Contains the ASCII value of the linefeed character specified in the NDL program.

HOME CHARACTER

1 Byte/ASCII

Contains the ASCII value of the home character specified in the NDL program.

CLEAR CHARACTER

1 Byte/ASCII

Contains the ASCII value of the clear character specified in the NDL Program.

SECTION 5
MESSAGE HEADER

INTRODUCTION

This section covers the format and use of the message header. The message header is a means of communicating status and events between the MCS and the NDL program.

The layout of the message header below shows two characters per line. The header that the MCS sees (the one sent by the MCS and the one received by the MCS) is only 35 bytes in length. The first five bytes are never used or seen by the MCS.

MESSAGE HEADER LAYOUT

MESSAGE LINK *	
BUFFER LINK *	
PROCESSOR	LINE
RESULT	TYPE
TASK MCS FLAG	
STATION	
OPTIONS	EVENTS
EVENTS	
RESERVED	SUBNET QUEUE
TEXT LENGTH	
MESSAGE LENGTH	
SKIPCONTROL	RETRY
RESERVED	TRANSMISSION NUMBER
TRANSMISSION NUMBER	
TALLY (1)	TALLY (0)
TALLY (2)	TOGGLES (7→0)
DATE (YEAR)	DATE (MONTH)
DATE (DAY)	TIME (HOURS)
TIME (MINUTES)	TIME SECONDS
MCS DATA	

* The size of this field is implementation-dependent.

MESSAGE HEADER

The message header is always the first 40 bytes of the first buffer associated with each message. All messages contain all of the message header fields described. However, not all fields are meaningful for all messages. Furthermore, not all fields are read/write accessible by the user, any such field is marked as being either NOT ACCESSIBLE BY USER or READ ONLY BY THE USER. All other fields are read/write.

Message Link

Not accessible to the user.

Contains the absolute address of the next message in this queue, or zero if this is the last message in the queue.

Buffer Link

Not accessible to the user.

Contains the absolute address of the next buffer used for this message, or zero if this is the only buffer used for this message.

Processor
1 Byte/Binary

Not accessible to the user.

Contains the data communications processor number associated with this message.

Line
1 Byte/Binary

Contains the logical line number associated with this message.

Result
1 Byte/Binary

Contains an index value indicating any special conditions associated with this message. The defined values for result are:

0 - complete and successful
1 - line not ready
2 - station not ready
3 - control or WRU flag set
4 - recalled from station
5 - recalled from subnet queue

6 - station not attached
7 - unable to initiate
8 - invalid network request
9 - DC hardware error
10 - DIALIN received
11 - recalled from output save queue

Type

1 Byte/Binary

Contains a value indicating the message type as follows:

0 - Maintenance
1 - Input
2 - Output
3 - Priority output
4 - Enable input
5 - Disable input
6 - Make station ready
7 - Make station not-ready
8 - Make line ready
9 - Make line not-ready
10 - Dialout
11 - Immediate line not-ready
12 - Recover*
13 - Deallocate*
14 - Dialin
15 - SPO input
16 - End recall from queue
17 - End recall from station
18 - Attach queue
19 - Attach station
20 - Enable queue
21 - Enable station
22 - Disable queue
23 - Disable station
24 - send
25 - Task detach
26 - Line Marker *
27 - Deallocate Space *

*Implementation dependent

Task

1 Byte/Binary

Contains the number of the task in which the message originated, valid values for user tasks range from 1 through 9.

MCS Flag

1 Byte

Indicates by the setting of the least significant bit that the MCS is to be notified of the results of this output message:

1. Only if errors occur (bit = 0).
2. Whether or not errors occur (bit = 1).

Station

2 Bytes/Binary

Contains the logical station number associated with this message.

Options

1 Byte

Contains the eight, one-bit flags listed here, which are available for use by the NDL program and the DC firmware.

7 - LINEFEED
    Output a linefeed character
6 - CARRIAGE
    Output a carriage return character
5 - PAPERMOTION
    Move paper before printing
4 - PAGE
    Advance page
3 - SKIP
    Skip to channel
2 - TRANSPARENT
    Message contains transparent text characters
1 - BLOCK
    One block (but not the last) of a multi-block message
0 - SPACE
    Advance line(s)

These flags are intended to be used in forms control; however, their actual meaning, if any, is determined by the NDL programmer.

Events

3 Bytes

Contains 24 one-bit flags listed below, which are set by the data communications subsystem to indicate conditions which occurred on the line while processing this message.

23 - NAK RECEIVED
22 - NAK ON SELECT
21 - NO SPACE
20 - TERMINATE ERROR
19 - DISCONNECT
18 - TERMINATE NO LABEL
17 - ADAPTER FAULT
16 - MODEM NOT READY
15 - CONTROL CHARACTER RECEIVED
14 - WRU CHARACTER RECEIVED
13 - TRANSMISSION NUMBER ERROR
12 - MESSAGE LENGTH EXCEEDED

11 - EVENT 1
10 - FORMAT ERROR
9 - BCC ERROR
8 - ADDRESS ERROR
7 - SYNCHRONOUS TRANSMISSION UNDERFLOW
6 - BREAK ON TRANSMIT
5 - LOSS OF CARRIER
4 - CHARACTER PARITY ERROR/INVALID FRAME
3 - BREAK ON RECEIVE/IDLE
2 - BYTE OVERFLOW-SERVICED TOO LATE
1 - STOP BIT ERROR/ABORT
0 - TIMEOUT

If one or more of the following flags is set, the line associated with this message is implicitly made not-ready and the appropriate value placed into the result field:

DISCONNECT
ADAPTER FAULT
MODEM NOT READY

If one or both of the following flags has been set, the station associated with this message has been implicitly made not-ready and the appropriate value has been placed into the result field:

TERMINATE ERROR
TERMINATE NO LABEL

Events during DIALOUT

Contains eight, one-bit flags, listed below, which are set by the data communications subsystem to indicate conditions occurring on the line while processing a dialout message.

23 - RESERVED
22 - RESERVED
21 - INVALID OR NO ANSWERTONE AFTER PULSE DIALING (NON-ACU MODEM)
20 - PREMATURE CONNECTION (U.K. ONLY)
19 - ACR BUT NO DSS AFTER ACU-DIALOUT
18 - FIRST PND WAS SENSED. BUT SUBSEQUENT PNDS WERE NOT
17 - ACR WITHOUT FIRST PND OR RING AFTER DTR/CRQ IS RAISED
16 - PWI WAS RESET OR DLO WAS SET, AT START OF ACU-DIALOUT

Subnet Queue

1 Byte/Binary

Contains the subnet queue number associated with this message.

Text Length

2 Bytes/Binary

Contains the number of text characters present in this message.

Message Length

2 Bytes/Binary

Read only by the user and contains the total number of bytes of space available for text in this message. Its value is always greater than, or equal to, the value of text length.

Skip Control

1 Byte/Binary

This contains a value to be used in connection with the options field. (For example, it may contain the number of lines that are to be skipped.)

Like options, the actual meaning of skip control is determined by the NDL programmer.

Retry

1 Byte/Binary

Contains the NDL retry count associated with this message. The maximum value the user may assign to retry is 254. The value 255 is reserved for system use.

Transmission Number

3 Bytes/ASCII

Contains three ASCII characters indicating the transmission number (000 through 999) received with this input message.

Tallies

3 Bytes

Three separate eight-bit binary fields, with use and meaning determined by the NDL programmer in cooperation with the user.

Toggles
1 Byte

Eight one-bit flags, with use and meaning determined by the NDL programmer in cooperation with the user.

Date
3 Bytes

Contains the data relevant for this message. It is given as six binary coded decimal digits in the form YYMMDD (Year, month, day).

For input messages, this field is filled by the DC firmware when the message is received.

For output messages, it is the user's responsibility to fill this field if it is so desired.

Time
3 Bytes

Contains the time of day relevant for this message. It is given as six binary coded decimal digits in the form HHMMSS (hours, minutes, seconds).

For input messages, this field is filled by the DC firmware when the message is received.

For output messages, it is the user's responsibility to fill this field if it is so desired.

MCS Data
2 Bytes/Binary

This field is provided for the use and convenience of the MCS only. It is initialized to zeroes on incoming messages by the DC firmware and is unaltered at all other times.

VALID MESSAGE HEADER FIELDS

Message header field information is given in figures 5-1 and 5-2.

DATA COMMUNICATIONS MESSAGE TYPES

Directive Type Messages

INPUT/OUTPUT

These messages are queued to the bottom of the station queue which belongs to the station referenced in the message header. Input messages are only queued to half-duplex stations defined as MYUSE-INPUT. Output messages are only queued to stations defined as MYUSE-OUTPUT. A rejected input/output message is returned to the MCS with result equal to "unable to initiate".

If a message is queued, the station is marked queued. If the station is ready and the line is ready, the line is marked queued. (See also Priority-Output, Enable-Input, Disable-Input, and Make-Station-Ready.)

If the station is ready and the line is ready, connected, and not busy (primary), line-control is initiated.

PRIORITY-OUTPUT

This message is treated just like a standard output message, except that it is queued to the top of the station queue.

Priority-output messages are used in error handling. If an unrecovered or unrecoverable error occurs while the line is engaged in a write request for some station, the error is reported in the header of the output (or priority-output) message when it is returned to the MCS. The line or station is also made not-ready, whichever is appropriate. In order to reinitiate the output in its proper sequence before other output in the station queue, the MCS may resubmit the message in error as PRIORITY-OUTPUT before making ready the line or station.

ENABLE-INPUT/DISABLE-INPUT

The messages are queued to the bottom of the station queue which belongs to the station referenced in the message header. ENABLE-INPUT messages are only queued to stations defined as MYUSE-INPUT. Otherwise, they are returned to the MCS with result equal to Unable To Initiate.

If a message is queued, the station is marked queued. If the station is ready and the line is ready, the line is also marked queued. If the station is ready and the line is also ready, connected, and not busy (primary), line control is initiated.

Like input and output messages, ENABLE-INPUT/DISABLE-INPUT is processed by the initiate-request statement in line control. At that time, the station is enabled/disabled and the message is returned to the MCS with result equal to Complete and Successful.

Messages Initiated by the MCS

FIELD \ MESSAGE TYPE	MAINTENANCE	INPUT	OUTPUT	PRIORITY OUT	ENABLE IN	DISABLE IN	MAKE STATION READY	MAKE STATION NOT READY	MAKE LINE READY	MAKE LINE NOT READY	DIALOUT	IMMEDIATE LINE NOT READY	RECOVER	DEALLOCATE	DIALIN	SPO INPUT	END RECALL QUEUE	END RECALL STATION	ATTACH QUEUE	ATTACH STATION	ENABLE QUEUE	ENABLE STATION	DISABLE QUEUE	DISABLE STATION	SEND	TASK DETACH
LINE									*	*	*	*														
RESULT																										
TYPE		*	*	*	*	*	*	*	*	*	*	*														
TASK																										
MCS FLAG			*	*																						
STATION		*	*	*	*	*	*	*																		
OPTIONS			X	X																						
EVENTS																										
SUBNET Q																										
TEXT LENGTH			*	*							*															
MSG LENGTH		S	S	S	S	S	S	S	S	S	S	S														
SKIPCONTROL			X	X																						
RETRY		X	X	X																						
TRANS NO		X	X	X																						
TALLIES		X	X	X																						
TOGGLES		X	X	X																						
DATE			U	U																						
TIME			U	U																						
MCS DATA		U	U	U	U	U	U		U		U	U														

LEGEND: * = Must be supplied by the user.
X = May be supplied by the user (if required by NDL).
U = May be supplied by the user (if he wishes).
S = Supplied by the DC subsystem.
Blank = Don't care.

Figure 5-1. Messages Initiated by the MCS

Messages Found in the MCS Queue

FIELD \ MESSAGE TYPE	MAINTENANCE	INPUT	OUTPUT	PRIORITY OUT	ENABLE IN	DISABLE IN	MAKE STATION READY	MAKE STATION NOT READY	MAKE LINE READY	MAKE LINE NOT READY	DIALOUT	IMMEDIATE LINE NOT READY	RECOVER	DEALLOCATE	DIALIN	SPO INPUT	END RECALL QUEUE	END RECALL STATION	ATTACH QUEUE	ATTACH STATION	ENABLE QUEUE	ENABLE STATION	DISABLE QUEUE	DISABLE STATION	SEND	TASK DETACH
LINE	*								*	*	*	*			*											
RESULT	*	*	*	*	*	*	*	*	*	*	*	*			*											
TYPE	*	*	*	*	*	*	*	*	*	*	*	*			*	*	*	*	*	*	*	*	*	*	*	*
TASK	*																		*	*	*	*	*	*	*	*
MCS FLAG	*		*	*																						
STATION		*	*	*	*	*	*	*										*		*		*		*	*	
OPTIONS	*	X	X	X																					X	
EVENTS	*	*	*	*																						
SUBNET Q																	*		*		*		*			
TEXT LENGTH		*	*	*							*					*					*	*	*	*	*	
MSG LENGTH		S	S	S	S	S	S	S	S	S	S	S			S	S	S	S	S	S	S	S	S	S	S	S
SKIPCONTROL		X	X	X																					X	
RETRY		X	X	X																					*	
TRANS NO		X	X	X																						
TALLIES		X	X	X																						
TOGGLES		X	X	X																						
DATE		*	U	U																						
TIME		*	U	U																						
MCS DATA		U	U	U	U	U	U		U		U	U														

LEGEND: * = Contains valid information.
X = May contain valid information (if it was supplied by NDL).
U = May contain valid information (if it was supplied by the user).
S = Supplied by the DC subsystem.
Blank = Don't care.

Figure 5-2. Messages Found in the MCS Queue

The ENABLE flag for each station, maintained by the Enable/Disable messages, controls the NDL enable-input feature. While the ENABLE flag is on, read request logic can be entered by an Initiate-Enable-Input statement in line-control or by a Terminate-Enable-Input statement in a write request.

MAKE-STATION-READY

The station referenced in the message is made ready, and the message is returned to the MCS with result equal to Complete and Successful.

If the referenced station is actively associated with a read/write request, the line/primary may have the STATION-NOT-READY-PENDING flag set from a previous Make-Station-Not-Ready message for that same station. If so, the PENDING flag is reset and the Make-Station-Not-Ready message is thereby countermanded, never to be returned to the MCS. Note, however, that the message space of the pending message has already been returned to the available buffer pool. (See Make-Station-Not-Ready.)

If the line is ready, the line is marked queued. If the line is ready, connected, and not busy (primary), line-control is initiated.

MAKE-STATION-NOT-READY

The station referenced in the message header may or may not be made not-ready immediately, depending on whether the station is actively associated with a read/write request. That is, the station remains ready if:

1. It is currently referenced by the line/primary station variable,
2. The line is busy, and
3. The line/primary is executing S-code in a request set.

If the station is active, the line/primary STATION-NOT-READY-PENDING flag is set. The message is passed to the available buffer pool with type equal to Discard. Later, when pending states are resolved, (when line busy is reset or during the execution of certain Terminate S-OPs), message space is allocated with a header filled with line number, station number, and type equal to MAKE-STATION-NOT-READY. Logic then proceeds as follows.

If the station is immediately made not-ready or if the STATION-NOT-READY-PENDING state is being resolved, the Make-Station-Not-Ready message is returned to the MCS with result equal to STATION-NOT-READY. The station is marked not ready, (primary) and line control is initiated:

1. I the line/primary station is executing S-code in a request set, and
2. If line busy is reset.

Any message space held by the line/primary is then queued to the top of the station queue of the line/primary station. If the auxiliary side of a full-duplex line is executing S-code in a request set for that station, AUX space is returned to the available buffer pool and AUX line-control is initiated.

MAKE-LINE-READY

If the line is switched-busy, the message is returned to the MCS with result equal to UNABLE TO INITIATE. Otherwise, the message is returned to the MCS with result equal to COMPLETE AND SUCCESSFUL.

If the line is already ready and not-busy, the line is not affected. If the line is ready and busy, the line's NOT-READY-PENDING flag is reset, thereby countermanding a previous Make-Line-Not-Ready message for that line. (The countermanded MAKE-LINE-READY may or may not be returned to the MCS, depending on system implementation.)

If the line is connected and not-ready, the line is made ready and (primary) line-control is initiated.

If not connected and not-ready, DIALIN logic is initiated as follows.

The line is made ready. While the line is awaiting an incoming call, it is kept not-switched-busy to allow interruption by a Dialout, Make-Line-Not-Ready, or Immediate-Line-Not-Ready message.

The line table resident variables, LINE TOG [0] and LINE TOG [1], are reset when a line (which was previously not ready) is made ready.

CALL-RECEIVED

As soon as DATASET READY (DSR, CC) is true and an incoming call is detected, the line is marked switched-busy. On successful completion of the call, the line is marked connected and not switched-busy. Message space is sent to the MCS with a header filled with line-number, type equal to DIALIN, and result equal to Complete and Successful.

On an unsuccessful DIALIN (when a LOSS OF CARRIER EQUAL TO DISCONNECT line fails to detect carrier), message space is allocated with a header filled with line-number and type equal to DIALIN. Disconnect logic is then invoked (see LINE-ABORT).

MAKE-LINE-NOT-READY

If the line is switched-busy, the Make-Line-Not-Ready message is returned to the MCS with result equal to UNABLE TO INITIATE.

If the line is ready and busy, the line's NOT-READY-PENDING flag is set, and the message is returned to the available buffer pool with type equal to DISCARD. Later, when pending states are resolved (see also Make-Station-Not-Ready), message space is allocated with a header filled with line-number and type equal to MAKE-LINE-NOT-READY. Logic then proceeds.

If the line is immediately made not-ready or if the LINE-NOT-READY-PENDING state is being resolved, the Make-Line-Not-Ready message is returned to the MCS with result equal to Line-Not-Ready.

If the line/primary was executing S-code in a request set and therefore has message space, that message space is queued to the top of the station-queue of the line/primary's station. Message space held by the auxiliary side of a full-duplex line is returned to the available buffer pool with type equal to DISCARD. The auxiliary is then idled. The line is then made not-ready.

DIALOUT

For DIALOUT, the line must be:

1. Dialout capable,
2. Not busy,
3. Not switch-busy,
4. Not connected.

If these criteria are not met, the message is returned with result equal to Unable To Initiate.

The line is made ready (if not already), and made switched-busy for the duration of the DIALOUT. DATASET READY (DSR,CC) is checked to see if the line has physically accepted an incoming call due to a previous Make-Line-Ready message. If the line is DATASET-READY, the dialout message is returned with result equal to DIALIN-RECEIVED. Logic then proceeds to CALL-RECEIVED. (See Make-Line-Ready.)

If an incoming call is not detected, the line goes off-hook and appropriate dialout logic is selected, depending on whether the line has an automatic calling unit (ACU) or a modem with dialout capability. The phone number to be dialed is contained in the text of the message. The number of digits to dial is indicated by the first four-bit, binary digit of the text. If the operator is dialing to a BPO Telex station, the last digit of the phone number must be a C.

If the call is successfully completed and answered, the dialout message is returned to the MCS with result equal to Complete and Successful. The line is marked connected and (primary) line-control is initiated.

Unsuccessful Dialout

If the call was either not completed or not correctly answered, disconnect logic is invoked (see Line-Abort).

Call Collision with BPO Telex

The dialout message is returned with result equal to DIALIN-RECEIVED. Logic then proceeds to CALL-RECEIVED. (Refer to Make-Line-Ready.)

IMMEDIATE-LINE-NOT-READY (SWITCHED DISCONNECT)

If the line is switched-busy, the IMMEDIATE-LINE-NOT-READY message is returned to the MCS with result equal to Unable to Initiate.

If the line/primary was interrupted while executing S-code in a request set and therefore has message space, the space is queued to the top of the station queue. For the auxiliary side of a full-duplex pair, the space is returned to the available buffer pool and the auxiliary is idled.

LINE-ABORT

If the line/primary was executing S-code in a request set and therefore has message space, that message space is queued to the top of the station-queue. Message space held by the auxiliary side of a full-duplex line is returned to the available buffer pool with type equal to DISCARD. The auxiliary is idled.

If the line is switched, the line is physically disconnected. The line is made ready and switched-busy until DATASET-NOT-READY has been achieved.

Whether the line is switched or leased, it now becomes not-ready. The IMMEDIATE-LINE-NOT-READY (or DIALIN, or DIALOUT) message is returned to the MCS with result equal to Line-Not-Ready.

RECOVER/DEALLOCATE

For the station referenced in the message header, the head/tail pointers of the station queue are copied into the message-header and then are cleared in the station table. Station-queue is reset. The recover/deallocate message is returned to the data communications controller, which discards each formerly queued message (for deallocate) or forwards each message to the MCS (for recover). Recover/deallocate misses any space not on the station queue. Should the station be executing S-OPS of a request set when the recover/deallocate is received, the space used by the request set is missed. To counteract this situation, the station should first be made not-ready.

Messages types RECOVER and DEALLOCATE are not seen by the user but are used within the data communications subsystem as a result of the RECALL/CLEAR communicate.

The text of the message contains data entered by the operator and directed to the MCS.

End Recall From Queue

Having recalled all input (from the subnet queue), a message of type END RECALL FROM QUEUE is placed on the MCS queue with result equal to COMPLETE AND SUCCESSFUL.

End Recall From Station

Having recalled all output (from the station queue), a message type END RECALL FROM STATION is placed on the MCS queue with result equal to COMPLETE AND SUCCESSFUL.

Maintenance

When a data communication hardware error occurs, the CMS subsystem generates a maintenance message and forwards it to the MCS.

SECTION 6
NDL PROGRAM FILE

GENERAL

This section outlines and describes the NDL object code file used by the system. This is only a description of the NDL object file. For a description of the NDL source, refer to the CMS NDL Reference Manual, form 1090925.

This section of the document describes the disk format of the NDL program file. Descriptions and initial values of individual fields are given where appropriate.

NDL PROGRAM PARAMETER BLOCK

Bytes	Use
Byte 0	(Implementation Level number) - Binary 00
Bytes 1-12	(Program Name) - NDLSYS
Bytes 13-24	(S-Language Name) - NDL S-LANG
Bytes 25-31	(Interpreter Pack-Id) - 0000000
Bytes 32-43	(Interpreter Name) - NDL.INTERP
Bytes 44-55	(Compiler Name) - NDL COMPILER
Bytes 56-61	(Compiler Date) - YYMMDD
Bytes 62-63	(Priority Class) - Binary 3180
Byte 64	(Data Segment For Initiating Message) - Binary FF
Bytes 65-67	(S-Program Start Address) - Binary 000000
Bytes 68-69	(Program Segment Table Length) - Binary 0030
Bytes 70-71	(PST Location) - Binary 0002
Bytes 72-73	(Data Segment Table Length) - Binary 0066
Bytes 74-75	(DST Location) - Binary 0003
Bytes 76-77	(TCB Present Area Length) - Binary 0000
Bytes 78-79	(TCB Preset Area Address) - Binary 0000
Bytes 80-81	(Stack Length) - Binary 0000
Bytes 82-83	(CCB Preset Area Length) Binary 0000
Bytes 84-85	(CCB Preset Area Address) - Binary 0000
Bytes 86-87	(TCB Preset Extension Length) - Binary 0000
Bytes 88-89	(Internal File Name Block Length) - Binary 0000
Bytes 90-91	(Internal File Name Block Address)- Binary 0000
Bytes 92-179	(TCB Preset Area Values) - All binary zeros

NDL PROGRAM SEGMENT TABLE

This segment contains descriptors pointing to the various program segments. Each descriptor is six bytes long and is structured as follows:

Bytes 0 and 1	Binary 0 (Indicating an ordinary, overlayable, read-only code segment)
Bytes 2 and 3	Relative disk address of the program segment
Bytes 4 and 5	Length in bytes of the program segment

The descriptors are arranged within the segment as follows:

Descriptor 0	Control Sets, Format A
Descriptor 1	Control Displacements, Format A
Descriptor 2	Request Sets, Format A
Descriptor 3	Request Displacements, Format A
Descriptor 4	Control Sets, Format B
Descriptor 5	Control Displacements, Format B
Descriptor 6	Request Sets, Format B
Descriptor 7	Request Displacements, Format B

NDL PROGRAM SEGMENT DESCRIPTIONS

Control Sets - Format A

This segment has all of the S-code resulting from control sets referenced in the NDL program. The control sets are arranged within the segment by logical-control-set number.

Logical-control-sets numbers are assigned in the order in which the control sets are referenced in the NDL program. Note that each pair of S-code data bytes is followed by a pair of bytes having the binary value 8000 if the data is a relative address, and the value 0000 otherwise. The 8000/0000 byte pair occupies disk space but not memory space.

Relative addresses are relative to the base of the (control sets-format A) segment, and are specified in terms of byte displacement divided by 4.

Control Displacements - Format A

Bytes 0 and 1 contain the number of control sets referenced in the NDL program.

Bytes 2 and 3 contain the relative address of logical-control-set 0.

Bytes 4 and 5 contain the relative address of logical-control-set 1; and so on.

Request Sets - Format A

The segment contains all of the S-code resulting from request sets referenced in the NDL program. The request sets are arranged within the segment by logical-request-set number.

Logical-request-set numbers are assigned in the order in which the request sets are referenced in the NDL program.

Note that each pair of S-code data bytes is followed by a pair of bytes having the binary value 8000 if the data is a relative address, and the value 0000 otherwise. The 8000/0000 byte pair occupies disk space but not memory space.

Relative addresses are relative to the base of the (request sets-format A) segment, and are specified in terms of byte displacement divided by 4.

Request Displacements - Format A

Bytes 0 and 1 contain the number of request sets referenced in the NDL program.

Bytes 2 and 3 contain the relative address of logical-request-set 0.

Bytes 4 and 5 contain the relative address of logical-request-set 1, and so on.

Control Sets - Format B

This segment contains all of the S-code resulting from control sets referenced in the NDL program. The control sets are arranged within the segment by logical-control-set number.

Logical-control-set numbers are assigned in the order in which the control sets are referenced in the NDL program.

Relative addresses are specified in terms of byte displacement from the base of the (control sets - format B) segment.

Control Displacements - Format B

Bytes 0 and 1 contain the number of control sets referenced in the NDL program.

Then bytes 2 and 3 contain the relative address of logical-control-set 0; bytes 4 and 5 contain the relative address of logical-control-set 1; and so on.

Request Sets - Format B

The segment contains all of the S-code resulting from request sets referenced in the NDL program. The request sets are arranged within the segment by logical-request-set number.

Logical-request-set numbers are assigned in the order in which the request sets are referenced in the NDL program.

Relative address is specified in terms of byte displacement from the base of the (request sets - format B) segment.

Request Displacements - Format B

Bytes 0 and 1 contain the number of request sets referenced in the NDL program.

Then bytes 2 and 3 contain the relative address of logical-request-set 0; bytes 4 and 5 contain the relative address of logical-request-set 1, and so on.

NDL DATA SEGMENT TABLE

This segment contains descriptors pointing to the various data segments. Each descriptor is six bytes long and is structured as follows:

Bytes 0 and 1 - Binary 0 (indicating an ordinary, overlayable, read-only data segment)

Bytes 2 and 3 - Relative disk address of the data segment

Bytes 4 and 5 - Length in bytes of the data segment.

The descriptors are arranged within the segment as follows:

Descriptor	Meaning
Descriptor 0	Preset Data
Descriptor 1	Line Tables
Descriptor 2	Line Table Displacement List
Descriptor 3	Station Tables
Descriptor 4	Station Table Displacement List
Descriptor 5	Modem Tables
Descriptor 6	Terminal Tables
Descriptor 7	File Tables
Descriptor 8	Extended Station Tables
Descriptor 9	Extended Terminal Tables
Descriptor 10	Station Name Table
Descriptor 11	File Name Table
Descriptor 12	Translation Tables
Descriptor 13	Translation Table Displacement List
Descriptor 14	Line Priority Chart
Descriptor 15	Line Speed Table
Descriptor 16	DCP-terminals Format A
Descriptor 17	Source Statement Occurrence
Descriptor 18	DCP-Terminals Format B

PRESET DATA

Bytes 0 and 1 - memory space required - contains the amount of space (in bytes) required for run time DC memory structures. It is computed as follows:

MEMORY = 8L + SUM(E) + 29SJ + 40SK + 15T + 5F + R/4 + C/4 + X/2 + B*N
Where:
L = Number of lines defined in the NDL program
SUM(E) = Sum, over all lines, of 2 times the Max entries value defined for each line
SJ = Number of stations defined in the NDL program which are not of type bits or type tallies
SK = Number of stations defined as type bits or type tallies in the NDL program
T = Number of terminals defined in the NDL program
F = Number of files defined in the NDL program
R = Length in bytes of the (request sets - format A) program segment
C = Length in bytes of the (control sets - format A) program segment
X = Length in bytes of the data segment containing translate tables.
B = DC Buffer size - defined below
N = Mimimum buffer count - defined below

Bytes 2 and 3 - DC Buffer Size

If a buffer value has been specified in the DCP section of the NDL program, then DC buffer size equals the integer value (buffer + 1)/2. Otherwise, the integer value (X + 41)/2 is used, where X equals the smallest maximum input size specified in the program.

Bytes 4 and 5 - Minimum Buffer Count

If a buffer value has been specified in the DCP section of the NDL program, then minimum buffer count equals this value. Otherwise, the following algorithm is used:

Let N(X) = Number of buffers needed to hold a message for terminal (X).
Let S = Sum, over all attached stations, of each station's terminal's N(X) value.
Let L = Number of defined lines.
Then minimum buffer count = (2 * S) + L.

If necessary, the compiler forces this value to be equal to, or greater than, 4.

Bytes 6 and 7 - Station Count

Contains the number of stations defined in the NDL program.

Byte 8 - File Count

Contains the number of files defined in the NDL program.

Byte 9 - Line Count

Contains the number of lines defined in the NDL program.

Byte 10 - Modem Count

Contains the number of modems defined in the NDL program.

Byte 11 - Terminal Count

Contains the number of terminals defined in the NDL program.

Bytes 12-13 - Additional Buffer Count

Contains the number of additional buffers allowed to the data comm subsystem, over and above the minimum buffer count.

Bytes 42-43 - Reserved for NDL Postprocessor

Byte 44 - DCP Count

Contains the number of data comm processors defined in the NDL program.

Byte 45 - Highest DCP Number

Byte 46 - Station Table Maximum Length

Byte 47 - Reserved

Byte 48 - N - DCP Data List

This bit contains N 18-byte entries; one entry for each DCP, from DCP 0 to the highest DCP number declared. Entries for undefined DCPs within this range are initialized to spaces.

Each entry consists of the following fields:

1. DCP MEMORY REQUIREMENT
   Two bytes/binary
   Memory = (SUM(5) * (STL + 4)) + (SLI*L)

Where:
SUM(S) = Sum of the max entries for each line on this DCP.
STL = Station table length (maximum).
SLI = Size of a line table with 0 stations.
L = Number of lines defined, in NDL, for this DCP.

2. DCP MEMORY SIZE
Two bytes/binary
Contains the memory size of this DCP as defined in the DCP(N) memory statement. If no DCP memory statement is specified, the NDL compiler supplies the value 6,144. If other than 6,144 is specified, this field is set to all ones.

3. DCP NUMBER OF LINES
One byte/binary
Contains the number of lines on this DCP.

4. DCP NUMBER OF FULL-DUPLEX LINES
One byte/binary
Contains the number of full-duplex lines on the DCP.

5. DCP LOAD FILE NAME
12 bytes/characters
Contains the program file name to be loaded into the DCP at data comm load time. If no DCP TERMINAL statement is specified, either NDLDCP or BDLDCP is supplied by the NDL compiler: if memory equals 6144, NDLDCP; if memory is greater than 6144, BDLDCP.

LINE TABLES

This segment contains the line tables generated by the compiler on a one-to-one basis with the lines defined in the NDL program. The tables are arranged within the segment by logical line number.

Logical line numbers are assigned in the order in which the lines are defined in the NDL program.

Line Table Displacement List

Bytes 0 and 1 contain the number of lines defined in the NDL program.

Bytes 2 and 3 contain a pointer to logical-line-table 0; bytes 4 and 5 contain a pointer to logical-line-table 1; and so on.

Pointers are relative to the line table segment base and are specified in terms of byte displacement divided by 2.

STATION TABLES

This segment contains the station tables generated by the compiler on a one-to-one basis with the stations defined in the NDL program. The tables are arranged within the segment by logical station number.

Logical station numbers are assigned according to the alphabetical order of the programmer specified station names.

Station Table Displacement List

BYTES 0-1 contain the number of stations declared in NDL.

BYTES 2-3 contain a pointer to logical-station-table 0. Each succeeding two-byte field contains a pointer to the next logical station table.

Pointers are relative to the station table segment base and are specified in terms of byte displacement divided by 2.

MODEM TABLES

This segment contains the modem tables generated by the compiler on a one-to-one basis with the modems defined in the NDL program. The tables are arranged within the segment by logical modem number.

NOTE

Two dummy modem tables for direct-connect are automatically generated by the compiler to aid in the reconfiguration process.

Logical modem numbers are assigned in the order in which the modems are defined in the NDL program.

TERMINAL TABLES

This segment contains the terminal tables generated by the compiler on a one-to-one basis with the terminals defined in the NDL program. The tables are arranged within the segment by logical terminal number.

Logical terminal numbers are assigned in the order in which the terminals are defined in the NDL program.

FILE TABLE

This segment contains a table generated by the compiler comprised of information on the files defined in the NDL program.

Logical file numbers are assigned according to the alphabetical order of the programmer-specified file names.

EXTENDED STATION TABLES

This segment contains the extended station tables which are generated and arranged exactly like the station tables.

EXTENDED TERMINAL TABLES

This segment contains the extended terminal tables which are generated and arranged exactly like the terminal tables.

STATION NAME TABLE

This segment contains a table of the programmer-specified station names. The names are arranged alphabetically within the table, and each entry is 12 bytes long, space filled on the right, if necessary.

FILE NAME TABLE

This segment contains a table of the programmer-specified file names. The names are arranged alphabetically within the table, and each entry is 12 bytes long, space filled on the right, if necessary.

TRANSLATION TABLES

This segment contains the translation tables referenced and/or defined in the NDL program. The tables are arranged within the segment by logical-translation-table number.

Logical-translation tables are referenced in the NDL program.

TRANSLATION TABLE DISPLACEMENT LIST

Bytes 0 and 1 contain the number of translation tables referenced in the NDL program.

Bytes 2 and 3 contain a pointer to logical-translation-table 0; bytes 4 and 5 contain a pointer to logical-translation-table 1; and so on.

Pointers are relative to the translation table segment base, and are specified in terms of byte displacement divided by 2.

LINE PRIORITY CHART

This segment contains the table of constants given in tables 6-1 and 6-2. Each entry in the table is one byte long. The left digit contains a line speed code, and the right digit contains a line priority code.

It is intended that this chart be used at reconfiguration time to assign proper priority to the line being redefined.

LINE SPEED TABLE

This segment contains a table of logical line numbers arranged by line speed, the higher speed lines appearing first. Each entry is one byte long.

DCP TERMINALS FORMAT A

This segment contains the program file names and associated terminal lists for each of two DCPs (DCP 0 and DCP 1). CMS systems with more than two DCPs reference the segment for DCP Terminals Format B.

Byte 0 - The number of program files defined for DCP 0.

Byte 1 - The number of program files defined for DCP 1.

If either field is zero, there are no program file lists and no program terminals lists for that particular DCP.

Bytes 2-N - DCP 0 program file list
DCP 1 program file list
DCP 0 program terminals lists
DCP 1 program terminals lists

PROGRAM FILE LIST

NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NUMBER OF TERMINALS	
TERMINALS LIST POINTER	
. . .	. . .
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NUMBER OF TERMINALS	
TERMINALS LIST POINTER	

Table 6-1. Asynchronous Line Priority Chart

ASYNCHRONOUS SPEED	BITS PER CHARACTER INCLUDING STOPBITS	MILLISECONDS PER CHARACTER	TABLE VALUE
50	11/10	--	00/00
	9/8	-160.00	00/20
	7/–	140.00/–	20/00
75	11/10	146.67/133.33	10/10
	9/8	120.00/106.67	10/10
	7/–	93.33/–	11/00
100	11/10	110.00/100.00	20/20
	9/8	90.00/80.00	21/21
	7/–	70.00/–	22/00
110	11/10	100.00/90.91	30/31
	9/8	81.82/72.73	31/32
	7/–	63.64/–	32/00
150	11/10	73.33/66.67	42/42
	9/8	60.00/53.33	43/43
	7/–	46.47/–	43/00
200	11/10	55.00/50.00	53/53
	9/8	45.00/40.00	54/54
	7/–	35.00/–	54/00
300	11/10	36.67/33.33	64/64
	9/8	30.00/26.67	65/65
	7/–	23.33/–	65/00
600	11/10	18.33/16.67	75/75
	9/8	15.00/13.33	76/76
	7/–	11.67/–	76/00
1200	11/10	9.17/8.33	86/86
	9/8	7.50/6.67	87/87
	7/–	5.83/	87/00
1800	11/10	6.11/5.56	97/97
	9/8	5.00/ 4.44	98/98
	7/–	3.89/ –	99/00
2400	11/10	4.58/ 4.17	A8/A8
	9/8	3.75/ 3.33	A9/A9
	7/–	2.92/ –	AA/00
4800	11/10	2.29/ 2.08	BA/BA
	9/8	1.87/ 1.67	BB/BB
	7/–	1.46/ –	BB/00
9600	11/10	1.15/ 1.04	CC/CC
	9/8	.94/ .83	CD/CD
	7/–	.73/ –	CD/00
19200	11/10	.57/ .52	DE/DE
	9/8	.47/ .42	DE/DE
	7/–	.36/ –	DE/00
38400	11/10	.29/ .26	EF/EF
	9/8	.23/ .21	EF/EF
	7/–	.18/ –	EF/00

NAME: 12 bytes
DCP program file name.
NUMBER OF TERMINALS: two bytes
The number of terminals declared to be associated with the program file.
TERMINALS LIST POINTER: two bytes
Self-relative index to list of the terminals associated with the program file.

PROGRAM TERMINALS LIST

LOGICAL TERMINAL NUMBER: two bytes
This list contains the logical numbers for the terminals which were declared in NDL for the program file which points to this list. There are as many of these lists for a DCP as there are program files declared for that DCP.

LOGICAL TERMINAL NUMBER
LOGICAL TERMINAL NUMBER
. . .
LOGICAL TERMINAL NUMBER

SOURCE STATEMENT OCCURRENCE

This segment within the NDLSYS file is generated by the NDL compiler to inform the post-processor program (non-interpretive program file generator) of

Table 6-2. Synchronous Line Priority Chart

SYNCHRONOUS SPEED	BITS PER CHARACTER	MILLISECONDS PER CHARACTER	TABLE VALUE
600	-8	-13.34	00/06
	7/6	11.66/10.00	06/06
	5/-	8.34/-	06/00
1200	-8	-6.67	00/07
	7/6	5.83/5.00	07/08
	5/-	4.17/-	08/00
2000	-8	-4.00	00/08
	7/6	3.50/3.00	09/09
	5/-	2.50/-	0A/00
2400	-8	-3.33	00/09
	7/6	2.92/2.50	0A/0A
	5/-	2.08/-	0A/00
3600	-8	-2.22	00/0A
	7/6	1.94/1.67	0B/0B
	5/-	1.39/-	0B/00
4800	-8	-1.67	00/0B
	7/6	1.46/1.25	0B/0C
	5/-	1.04/-	0C/00
7200	-8	-1.11	00/0C
	7/6	.97/.83	0C/0D
	5/-	.69/-	0D/00
9600	-8	.83	00/0D
	7/6	.73/.63	0D/0D
	5/-	.52/-	0E/00

the occurrence of certain source statements within the NDL program.

POINTER TO 1ST REQUEST SET INFORMATION
CONTROL SET INFORMATION
CONTROL SET INFORMATION
. . .
CONTROL SET INFORMATION
REQUEST SET INFORMATION
REQUEST SET INFORMATION
. . .
REQUEST SET INFORMATION

CONTROL/REQUEST set information: two bytes

Each two-byte entry is considered as a set of 16 (1-bit) flags, each indicating the presence/absence of a particular S-Op within the control/request set.

The flags represent the occurrence of the following S-Ops. the flags are numbered right to left, flag 15 being the left-most bit position.

15 - LINE BUSY = TRUE/FALSE
14 - LINE BUSY = TOGGLE
13 - AUX LINE BUSY = TRUE/FALSE
12 - AUX LINE BUSY = TOGGLE
11 - BINARY = TRUE/FALSE
10 - TERMINATE BLOCK
9 - SYNCS = TRUE/FALSE
8 - CRC = TRUE/FALSE
7 - SHIFT = UP/DOWN/MIDDLE
6 - STATION = LIT/VARIABLE
5 - USE OF LCHAR
4 - RECEIVE WAIT
3 - RECEIVE TEXT
2 - BACKSPACE
1 - UNUSED
0 - UNUSED

There are as many information items as there are control and request sets in the NDL program. The items are in the order of the logical numbers assigned to the control sets and request sets by the NDL compiler.

DCP TERMINALS FORMAT B

This request contains information concerning DCP program files and their associated terminals.

BYTE 0 - DCP COUNT
Contains the number of DCPs defined in NDL.

Byte 1-N - DCP data directory
DCP data
Program terminal lists

DCP Data Directory

Contains T two-byte entries where T is the total number of DCPs from DCP 0 to the highest DCP numbers declared. Each two-byte entry is a segment base relative pointer to the appropriate DCP data structure. If a DCP is not required by the NDL file, the directory entry is set to all ones.

DCP Data

One for each DCP specified. This consists of a one-byte program file count followed by a series of 15-byte entries as follows.

NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NAME	NAME
NUMBER OF TERMINALS	TERMINAL LIST POINTER
TERMINAL LIST POINTER	

NAME : 12 Bytes
Contains the DCP program file name.
NUMBER OF TERMINALS: One byte
Contains the number of terminals declared to be associated with the program file.
TERMINAL LIST POINTER: Two bytes
Contains a segment base relative pointer to the terminal list associated with this program file.

Note
There is one of the previous entries for each DCP program file associated with this DCP.

PROGRAM TERMINAL LISTS

Following DCP data are the various terminal lists associated with the individual DCP program file. Each list is an array of two-byte logical terminal numbers.

SECTION 7

COBOL DATA COMMUNICATIONS

GENERAL

This section deals with the COBOL constructs for data communications and their use. For more detailed information on the syntax and semantics of these communicates, refer to the CMS COBOL Reference Manual, form 2007266.

COBOL COMMUNICATION DESCRIPTIONS

A communication description (CD) serves to specify the interface area between the system, the MCS and a COBOL program.

Two types of communication descriptions are required, one for input and one for output.

Input CD

The input communication description defines an interface area where information relating to input messages is passed between the data comm subsystem, the MCS, and a COBOL program.

SYMBOLIC QUEUE

This field is used to pass the symbolic name of a queue to the data communications subsystem and the MCS. If a queue name which has not been defined to the system is used, it is regarded as an error and an error code is returned in the status key field. The symbolic queue must be left-justified with space filler.

SYMBOLIC SUB-QUEUE 1, 2, 3

The system does not support sub-queues and an error code is returned in the status key if the field contains any character other than spaces.

MESSAGE DATE

The message date field has the format YYMMDD (year, month, day). Its contents represent the date on which the system recognizes that the message is complete.

MESSAGE TIME

The message time field has the format

FORMAT OF INPUT CD AREA

	DESCRIPTION	COMMENT	POSITION
01	DATA-NAME	–	–
02	DATA-NAME PIC X(12)	SYMBOLIC QUEUE	1-12
02	DATA-NAME PIC X(36)	SYMBOLIC SUB-QUEUE	13-48
02	DATA-NAME PIC 9(6)	MESSAGE DATE	49-54
02	DATA-NAME PIC 9(8)	MESSAGE TIME	55-62
02	DATA-NAME PIC X(12)	SYMBOLIC SOURCE	63-74
02	DATA-NAME PIC 9(4)	TEXT LENGTH	75-78
02	DATA-NAME PIC X(1)	END KEY	79
02	DATA-NAME PIC X(2)	STATUS KEY	80-81
02	DATA-NAME PIC 9(6)	MESSAGE COUNT	82-87 STOP
		QUEUE NUMBER	88-89
		STATION NUMBER	90-91

HHMMSSTT (hours, minutes, seconds, hundredths of a second). Its contents represent the time at which the system recognizes that the message is complete. The hundredths of a second part of the field is always 00. If the program is being executed on a system without a clock, the time is always presented as 24000000.

The time and date fields are only updated by the system during the successful execution of a receive statement and reflect the time and date the incoming message was accepted by the system and not the time it was executed.

SYMBOLIC SOURCE

During the execution of a receive statement, the system places in the symbolic source field the symbolic name of the station that is the source of the message being transferred.

TEXT LENGTH

The system places in the text length field the number of character positions filled as a result of the execution of the receive statement.

END KEY

The contents of the end key field are set during the execution of a receive statement according to the following rules:

1. If an end-of-group has been detected, end key = 3.
2. If an end-of-message has been detected, end key = 2.
3. If less than a message is transferred, end key = 0 (the message was truncated).

STATUS KEY

The contents of the status key field are set during the execution of receive, accept message count, enable input, and disable input statements. The status key values are listed in figure 7-1.

MESSAGE COUNT

The contents of the message count field indicate the number of messages that exist in a queue. The field is only updated as part of the execution of an Accept statement with the count phrase.

QUEUE NUMBER

The field queue number is provided to allow the system to minimize the overhead of name-to-number translation. Wherever symbolic queue or symbolic sub-queue are changed by the program, this field is changed to @FFFF@. This field is not accessible to the user.

STATION NUMBER

The field station number is provided to allow the system to minimize the overhead of name-to-number translation. Wherever symbolic source is changed by the program, this field is changed to @FFFF@. This field is not accessible to the user.

OUTPUT CD

The output communication description (CD) defines an interface area where information relating to output messages is passed between the COBOL program, the MCS, and the data communications subsystem.

FORMAT OF OUTPUT CD AREA

	DESCRIPTION	COMMENT	POSITION
01	DATA-NAME	–	–
02	DATA-NAME PIC 9(4)	DESTINATION COUNT	1-4
02	DATA-NAME PIC 9(4)	TEXT LENGTH	5-8
02	DATA-NAME PIC X(2)	STATUS KEY	9-10
02	DATA-NAME PIC X(1)	ERROR KEY	11
02	DATA-NAME PIC X(12)	SYMBOLIC DESTINATION	12-23 STOP
		STATION NUMBER	24-25

STATUS KEY CONDITIONS

RECEIVE	SEND	ACCEPT	ENABLE INPUT	ENABLE OUTPUT	DISABLE INPUT	DISABLE OUTPUT	STATUS KEY	
X	X	X	X	X	X	X	00	NO ERROR DETECTED. ACTION COMPLETED.
	X			X		X	20	DESTINATION UNKNOWN OR ACCESS DENIED BY MCS. NO ACTION TAKEN FOR UNKNOWN DESTINATION. ERROR KEY INDICATES UNKNOWN.
X		X	X		X		20	QUEUE UNKNOWN OR ACCESS DENIED BY MCS. NO ACTION TAKEN.
	X			X		X	30	CONTENT OF DESTINATION COUNT INVALID. NO ACTION TAKEN.
	X						50	CHARACTER COUNT GREATER THAN LENGTH OF SENDING FIELD. NO ACTION TAKEN.
X	X	X	X	X	X	X	91	MCS/DC SUBSYSTEM NOT AVAILABLE.

Figure 7-1. Status Key Conditions

DESTINATION COUNT

The destination count field indicates the number of symbolic destinations to be used from the destination table. (Field error key and symbolic destination comprise the destination table).

The system permits only one destination to be specified and if the destination count has any value other than 1, an error condition is indicated in the status key field and execution of the statement is terminated.

TEXT LENGTH

The system interprets the text length field as the number of characters to be sent when executing a send statement.

STATUS KEY

The contents of the status key field are set during the execution of Send, Enable Output, and Disable Output. The Status Key values are listed in figure 7-1.

ERROR KEY

The error key field, when equal to 1, indicates that the symbolic destination is unknown or not accessible by this program. The status key field is set to a value indicating the appropriate condition. Otherwise, the error key field is set to 0.

SYMBOLIC DESTINATION

The symbolic destination field is used to pass the symbolic name of the destination station while executing Send, Enable Output, and Disable Output statements.

STATION NUMBER

The station number field allows the system to minimize the overhead of name-to-number translation. Wherever symbolic destination is changed by the COBOL program, the station number field is changed to @FFFF@. This field is not accessible to the user.

COBOL DATA COMM STATEMENTS

To ensure some degree of system integrity, all COBOL data comm statements cause a check to be performed as to whether this user is currently "attached" to a designated symbolic queue (input) and/or a designated symbolic destination (output).

In the event the COBOL program is not currently attached to the appropriate queue or destination, an attach message is formulated and placed on the MCS queue. The format of these messages is:

For ACCEPT, DISABLE INPUT, ENABLE INPUT AND RECEIVE:

TYPE	=	"ATTACH QUEUE"
TASK	=	TASK NUMBER
TEXT LENGTH	=	12
SUBNET QUEUE	=	SUBNET QUEUE NUMBER
TEXT	=	SYMBOLIC QUEUE NAME

For DISABLE OUTPUT, ENABLE OUTPUT, AND SEND:

TYPE	=	"ATTACH STATION"
TASK	=	TASK NUMBER
STATION	=	LOGICAL STATION NUMBER
TEXT LENGTH	=	12
TEXT	=	SYMBOLIC DESTINATION (STATION NAME)

The COBOL program is waited until the MCS performs an Allow or Disallow communicate.

ACCEPT

ACCEPT <cd-name> MESSAGE COUNT;

Accept causes the depth (number of entries) of the subnet queue identified by symbolic queue to be inserted into MESSAGE.COUNT.

Before this can be done, Accept must check that the symbolic queue is known, that the symbolic subqueue is space-filled, and that the task is attached to the symbolic queue. If the task is not attached, Accept attempts to rectify the situation by issuing an Attach Queue message to the MCS and waiting for the reply. Any failure causes the STATUS.KEY to be set to 20.

Successful execution (and therefore a meaningful MESSAGE.COUNT) is indicated by a STATUS.KEY of 00.

ENABLE INPUT

ENABLE INPUT <cd-name> WITH KEY <identifier/literal>;

Enable Input invokes a function defined by the

MCS, by sending an Enable Queue message to the MCS.

Before this can be done, Enable Input must check that the symbolic queue is known, that the symbolic sub-queue is space-filled, and that the task is attached to the symbolic queue. If the task is not attached, enable input attempts to rectify the situation by issuing an Attach Queue message to the MCS and waiting for the reply. Any failure causes the STATUS.KEY to be set to 20.

If these tests succeed and the task is attached, status key is set to 00 and an Enable Queue message sent to the MCS. The semantics of Enable Input are defined by the MCS; in particular, key validation is performed by the MCS and thus there is no mechanism which allows for the rejection of the key.
MESSAGE TO MCS:
ENABLE INPUT

TYPE	=	"Enable Queue"
TASK	=	Task Number
Subnet Queue	=	Subnet Queue Number
Text Length	=	13-22
Text	=	12 Characters of Queue Name, Followed by Information Defined by "Key".

DISABLE INPUT

DISABLE INPUT <cd-name> WITH KEY <identifier/literal>;

Disable Input invokes a function defined by the MCS, by sending a Disable Queue message to the MCS.

Before this can be done, disable input must check that the symbolic queue is known, that the symbolic sub-queue is space filled, and that the task is attached to the symbolic queue. If the task is not attached, disable input attempts to rectify the situation by issuing an Attach Queue message to the MCS and waiting for the reply. Any failure causes the STATUS.KEY to be set to 20.

If these tests succeed and the task is attached, status key is set to 00 and a Disable Queue message is sent to the MCS. The semantics of Disable Input are defined by the MCS; in particular, key validation is performed by the MCS and thus there is no mechanism which allows for the rejection of the key.
MESSAGE TO MCS:
DISABLE INPUT

TYPE	=	"Disable Queue"
Task	=	Task Number
Subnet Queue	=	Subnet Queue Number
Text Length	=	13-22
Text	=	12 Characters of Queue Name, Followed by Information Defined by "Key".

ENABLE OUTPUT

ENABLE OUTPUT <cd-name> WITH KEY <identifier/literal>;

Enable Output invokes a function defined by the MCS, sending an Enable Station message to the MCS.

Before this can be done, several checks must be made:

If the destination count is not equal to 1, then STATUS.KEY is set to 30.
If the symbolic destination is not known to the system, then STATUS.KEY is set to 20 and ERROR.KEY is set to 1.
If the task is not attached to the symbolic destination, then Enable Output attempts to rectify this situation by issuing an Attach Station message to the MCS and waiting for the reply. If attachment is denied, then STATUS.KEY is set to 20 and ERROR.KEY is set to 1.

If these tests succeed and the task is attached, status key is set to 00, error key to 0, and an Enable Station message is sent to the MCS. The semantics of Enable Output are defined by the MCS; in particular, key validation is performed by the MCS and thus there is no mechanism which allows for the rejection of the key.
MESSAGE TO MCS:
ENABLE OUTPUT

Type	=	"Enable Station"
Task	=	Task Number
Station	=	Logical Station Number
Text Length	=	13-22
Text	=	12 Characters of Station Name, Followed by Information Defined by "Key".

DISABLE OUTPUT

DISABLE OUTPUT <cd-name> WITH KEY <identifier/literal>;

Disable Output invokes a function defined by the MCS, by sending a Disable Station message to the MCS.

Before this can be done, several checks must be made:

If the destination count is not equal to 1, then STATUS.KEY is set to 30.

If the symbolic destination is not known to the system, then STATUS.KEY is set to 20 and ERROR.KEY is set to 1.

If the task is not attached to the symbolic destination, then disable output attempts to rectify this situation by issuing an Attach Station message to the MCS and waiting for the reply. If attachment is denied, then STATUS.KEY is set to 20 and ERROR.KEY is set to 1.

If these tests succeed and the task is attached, status key is set to 00, error key to 0, and a Disable Station message is sent to the MCS. The semantics of Disable Output are defined by the MCS; in particular, key validation is performed by the MCS and thus there is no mechanism which allows for the rejection of the key.

MESSAGE TO MCS:
DISABLE OUTPUT

Type	=	"Disable Station"
Task	=	Task Number
Station	=	Logical Station Number
Text Length	=	13-22
Text	=	12 Characters of Station Name, Followed by Information Defined by "Key".

RECEIVE

RECEIVE <cd-name> MESSAGE INTO <identifier>{; NO DATA <statement> |

Receive attempts to read a message from the queue specified by symbolic queue. If successful, the message text is moved to the data area and information about the message is assembled in the input CD area.

Before this can be done, receive must check that the symbolic queue is known, that the symbolic subqueue is space-filled, and that the task is attached to the symbolic queue. If the task is not attached, receive attempts to rectify the situation by issuing an Attach Queue message to the MCS and waiting for the reply. Any failure causes the STATUS.KEY to be set to 20.

If the symbolic queue is empty, and the NO DATA phrase is specified, then receive sets the fetch value to @100000@ and exits to allow execution of the NO DATA statement.

If the symbolic queue is empty, and the NO DATA phrase is absent, then the task is suspended until a message appears on the queue.

The message text is moved to the data area left-justified, without space fill, and the text length field is set to reflect the size of the message. The message data and message time are updated with the quantities implied by their names. Symbolic source is updated to the name of the station where the message originated. If the message exceeds the length of the data area, the message is truncated and the end key set to 0. If the message is detected as being the last of a group, end key is set to 3, otherwise it is a 2.

Status key is set to 00 to indicate successful execution.

SEND

```
SEND cdname [FROM identifier-1] { WITH EMI }
                                { WITH EGI }

[ { BEFORE }  ADVANCING  { { identifier-2 }  { LINE  } } ]
[ { AFTER  }             { { integer      }  { LINES } } ]
[                        { { mnemonic name }           } ]
[                        { { PAGE          }           } ]
```

Send attempts to dispatch a message ultimately to the station named by symbolic destination in the output CD area. The message is actually sent either to the MCS or to the appropriate station depending on the history of routing directives (ROUTE.OUTPUT communicates) issued by the MCS.

Before this can be done, several checks must be made:

1. If the destination count is not equal to 1, then STATUS.KEY is set to 30.
2. If the symbolic destination is not known to the system, then STATUS.KEY is set to 20 and ERROR.KEY is set to 1.
3. If the task is not attached to the symbolic destination, then send attempts to rectify this situation by issuing an Attach Station message to the MCS and waiting for the reply. If attachment is denied, then STATUS.KEY is set to 20 and ERROR.KEY is set to 1.
4. If the TEXT.LENGTH exceeds the size of the data area given by identifier-1, then the message is not sent and STATUS.KEY is set to 50.

If insuffcent buffer space is available for the SEND, the task is suspended until space becomes available.

The amount of space required for the SEND message includes the CMS message header. All valid header fields for this message type are initialized. The RETRY field is set equal to the value of ORIGINAL RETRY, found in the station table of the destination station.

If output is to be directed to the MCS and the task's output count is greater than, or equal to, its output limit, the task is suspended until the MCS issues a CONTINUE.TASK communicate or until the route indication is changed. Messages directed to the MCS are marked with TYPE = SEND.

The phrase WITH EMI/EGI indicates that the contents of identifier-1 are to be associated with an end of message indicator (EMI) or an end-of-group indicator (EGI). WITH EMI implies that this message is one of a group of messages and that the final message of the group is sent using the phrase WITH EGI. The implication is that a single message (not one of a group) should always be sent using the phrase WITH EGI. Note that this phrase is eventually mapped into the block bit of the message header options field.

The advancing phrase is encoded in the message header options and SKIP.CONTROL fields. The ultimate effect of this action is defined by the MCS and the associated line procedures.

SKIP CONTROL (CPA BYTES 3, 4)

Encodings of the skip control fields are:

BYTE 3

Bit 7 = 1
Bit 6 = 1
Bit 5 = 0 = Print Before Papermotion
1 = Print After Papermotion
Bit 4 = 0 = Do Not Advance To New Page
1 = Advance To New Page
Bit 3 = 0 = Do Not Skip To Channel
1 = Skip to Channel
Bit 2 = 0
Bit 1 = 0
Bit 0 = 0 = Do Not Advance Line(s)
1 = Advance Line(s)

If either bit 3 or bit 0 of byte 3 is set, byte 4 contains the line count or channel number as appropriate.

VARIANT [FROM IDENTIFIER-1]

The system is notified of the absence of the FROM <identifier-1> phrase by the data area size (CPA bytes 12-13) being zero.

STATUS.KEY is set to 00 to indicate successful completion.

SECTION 8
RECONFIGURATION

GENERAL

On CMS systems, *reconfiguration* means to alter the NDL descriptions of some characteristics of the data comm lines and station. Two MPLII data comm communicates in the MCS program, REDEFINE.LINE and REDEFINE.STATION, are used to perform these alterations. Thus, in CMS systems, it is possible for an MCS program to modify the data comm network which it controls. The reconfiguration changes are made to the temporary NDL tables used during execution of an MCS program. This means that the reconfiguration is temporary; that is, the next time the same NDL program is used, the network is in the original configuration.

REDEFINE.LINE

TYPE
MODEM

For an explanation of the items, see Line Table Layout.

The REDEFINE.LINE communicate alters the description of a line. The line's logical number and a data area containing the desired changes are the parameters of the communicate.

The data variable contains the values to be inserted into the alterable fields of the line table. If the data variable is not at least three bytes long, an error is monitored.

The characteristics of a line which may be altered are:

- Modem
- Transmission Method (type bit 1)
- Form of Sync Character (type bit 2)
- Dialin Capability (type bit 3)
- Dialout Capability (type bit 4)
- Dialout Device (type bit 5)
- Action on Loss of Carrier (type bit 6)
- Rateselect Capability (type bit 8)
- Rate (type bit 9)
- Standby Capability (type bit 10)
- Use of Standby Speeds (type bit 11)

Success in redefining these fields does not depend on their current values, but on the definitions of related parts of the network. These conditions are described in the Network part of this section.

Another factor is that the system cannot be actively using the line to be redefined. That is, there can be no messages for that line or any station on it in the NDL request queue and the line must be not-ready. Switched lines must also be not-switched-busy and not-connected. If either of these requirements is not met, an error is monitored.

Note that the use of modems characteristics (type bit 7) may not be altered. Its value in the type word to be inserted must be the same as its current value. Also, the current value for the use of modems characteristic must be modems (as opposed to no modems or direct-connect). If either of these conditions is not met, an error is monitored.

If the line can be legally redefined to have the given values, the line table alterable fields and (for any station on the line) some station table fields are reinitialized. The fields in the line table which are reinitialized are:

- Line Descriptor, including resetting line toggles 0 and 1
- Line Tally (0)
- Line Tally (1)
- Aux Line Tally (0)
- Aux Line Tally (1)
- Aux Line Descriptor

If there are stations on the line, the following fields are reinitialized in those station tables:

- Active Transmit Delay
- Run Mode (second stop bit)
- Line Priority Code

REDEFINE.STATION

For an explanation of the items, see Station Table Layout.

LOGICAL LINE NO.	RUN MODE BITS
END CHARACTER	LINE DELETE CHARACTER
BACKSPACE CHARACTER	WRU CHARACTER
CONTROL CHARACTER	STATION FREQUENCY
TRANSMIT ADDR. 1	TRANSMIT ADDR. 2
TRANSMIT ADDR. 2	RECEIVE ADDR. 2
RECEIVE ADDR. 2	RECEIVE ADDR. 3
TYPE	
SPEED	
MODEM	TERMINAL
ORIGINAL RETRY	

The REDEFINE.STATION communicate alters the description of one station. The parameters of the communicate include the station's logical number and a data area containing the desired changes.

The data variable contains the values to be inserted into the alterable fields of the station table. If the data variable is not at least 21 bytes long, an error is monitored.

The characteristics of a station which may be altered are:

- Logical Line Number
- Myuse Output Capability (run mode bit 7)
- Myuse Input Capability (run mode bit 6)
- Use of Second Stop Bit (run mode bit 5)
- Allowing of Input (run mode bit 4)
- End Character
- Line Delete Character
- Backspace Character
- WRU Character
- Control Character
- Station Frequency
- Transmit Address Characters
- Receive Address Characters
- Form of Duplex (type bit 0)
- Use of Modems (type bit 7)
- Use of Telex (type bit 12)
- BDI Mode (type bit 13)
- Speed
- Modem
- Terminal
- Original Retry

The actions taken in performing the REDEFINE.STATION communicate depend on the value of the logical line number currently in the station table and the value of the logical line number to be inserted into the table. One action is to remove a station from a line. In this case, no changes are made to the station table except for the logical line number. This action is taken when the line number in the table is a valid logical line number and the line number to be inserted is @FF@.

Another action is to add a station to a line and update all the alterable fields of the station table. This is done when the line number in the table is @FF@ and the line number to be inserted is a valid logical line number.

The last action is to change all the alterable fields of the station table for a station which is on a line. In this case, the line number in the table and the line number to be inserted are the same valid logical line number. If both are valid logical line numbers but they are different, an error is monitored. In the case that both the line number in the station table and the line number to be inserted are @FF@, no action is taken.

For all other alterable fields, success in redefining them does not depend on their current definition but on the definition of related parts of the network. These conditions are described in the Network section.

Another factor in the success of redefining a station is that the system cannot be actively using that station or the line involved in the redefinition. This means that there can be no messages for that line or any station on it in the NDL request queue and the line must be not ready. For this type bits stations, both save queues must be empty. Also, if the station is being removed from the line, the station cannot be in the Queued condition. Additionally, if the station is being added to a line, there must be room for it; which means that in the line table the current value of the Max Stations field is less than the current value of the Max Entries field. If any of these requirements is not met, an error is monitored.

If the REDEFINE.STATION action requested is legal, the station table and line table are updated and reinitialized according to the particular action. When removing the station from the line, the only alterable field which is changed is the logical line number. Also, in the station table, the attached status field is reinitialized. In the line table, the Max Stations field is decremented and the station's entry in the line vector is removed.

For the actions of adding the station to the line or changing the station on the line, all the alterable fields are changed. Also, these Line Table fields are reinitialized:

- Line Desc (full duplex)
- Station Tally of This Station
- Station Description of This Station

In the station table, these fields are reinitialized:

- Line Priority Code
- Receive Transmission Number

- Transmit Transmission Number
- Tally (0)
- Tally (1)
- Tally (2)
- Tally (3) through Tally (18)
- Toggles
- Options
- Events
- Initiate Receive Delay
- Active Transmit Delay

The output routing of the station is also reinitialized.

Some additional reinitialization and updating are done for the add action. In the line table, the Max Stations field is incremented and an entry is added to the station vector. These fields are reinitialized in the station table:

- Relative Station Number
- Unprocessed Input Limit
- Unprocessed Input Count
- Queue Limit
- Subnet Queue Address

NETWORK

The conditions which a redefinition must meet concerning related network parts are ones which result in a network description that is legal under the requirements of CMS NDL. With only one exception, these conditions exclusively involve the descriptions of the parts of the one line affected by the specific redefinition. Those descriptions are: the line description, the description of the line modem, the descriptions of any stations on the line, the descriptions of those stations' modems, and the descriptions of their terminals. Some network conditions are met automatically because many parts of the network cannot be redefined. The conditions for which the redefinition must be validated are:

- All of the descriptions must specify the same transmission method - Asynchronous or Synchronous
- All of the modem descriptions and the line description must specify the same action on Loss of Carrier, Rateselect Capability, and Standby Capability
- If the line description specifies Dialin, the lines modem description must specify Dialin
- If the line description specifies Dialout, either the line description specifies ACU or the Line Modem Description specifies Dialout, but not both
- If the line description does not specify Dialout, it must not specify ACU
- If the line description specifies Telex, it must not specify the Standby Option
- If the line description specifies Telex, and if the station and terminal descriptions specify Telex, then the line description must not specify the Rateselect Option
- For each station, the station description and its terminal description must specify the same use of Telex
- All of the terminal descriptions must specify the same use of Telex
- If the station and terminal descriptions specify Telex, the line description must specify Telex
- If the line description specifies BDI, then all of the station and terminal descriptions must specify BDI
- If a line description specifies Bits, then it must also specify use of Modems and Synchronous transmission
- All corresponding terminal, station, and line descriptions must specify the same use of BDI
- All corresponding terminal, station, and line descriptions must specify the same use of Bits
- All modem and station descriptions and line descriptions must specify the same use of Modems
- All of the station and terminal descriptions and the line description must specify the same form of Sync Characters - ASCII or EBCDIC
- All corresponding terminal, station, and line descriptions must specify the same Duplex Capability
- If the station and terminal descriptions specify Full Duplex, all of the modem descriptions and the line description must specify Full Duplex
- If the line description specifies BDI, the line description and all of the modem descriptions must specify the same Duplex Capability as is specified by the station and terminal descriptions
- If the line description specifies Direct-Connect and if the line description does not specify BDI, then, for each terminal, the terminal description must specify Direct-Connect
- If line description specifies Modems, then, for each terminal, the terminal description must specify Modem
- If a terminal description specifies Tallies, then each associated station description must specify Tallies
- For each station, the station description must specify exactly as many non-null Transmit Address Characters as the terminal description specifies in the Transmit Address Count and the Receive Address Count
- For each station, if its terminal description specifies no Receive Request Set, the station description must not specify Myuse Input
- For each station, if its terminal description spec-

ifies no Transmit Request Set, the station description must not specify Myuse Output

- For each station, if the station description does not specify Myuse Input, it must not specify Enable Input
- If the line description specifies Synchronous, then, for each station, the station description must not specify Second Stop Bit
- If the line description specifies Asynchronous, then, for each station, the station description must specify the same number of Stop Bits as its terminal description specifies for line Speed
- If the line description specifies Asynchronous, all station descriptions must specify the same number of Stop Bits
- If the line description specifies Synchronous, then, for each station, the station description must specify one speed within the Synchronous range
- If the line description specifies Synchronous, all of the modem descriptions must specify the same Speed or Speeds
- If the line description specifies Synchronous, then, for each station, the station description must specify a Speed greater than, or equal to, the highest Speed specified in the modem descriptions
- If the line description specifies Synchronous, then, for each terminal, the terminal description must specify a Speed greater than, or equal to, the highest Speed specified in the modem descriptions
- If the line description specifies Asynchronous, and if the line description specifies both the Rateselect and Standby Options, then, for each station, the station description must specify three Speeds within the Asynchronous range
- If the line description specifies Asynchronous, and if the line description specifies the Rateselect Option but not the Standby Option, then, for each station, the station description must specify two Speeds within the Asynchronous range
- If the line description specifies Asynchronous, and if the line description does not specify the Rateselect or Standby Options, then, for each station, the station description must specify one Speed within Asynchronous range
- If the line description specifies Asynchronous, then each corresponding station and terminal description must specify the sameSpeed or Speeds
- If the line description specifies Asynchronous, then all of the station descriptions must specify the same Speed or Speeds
- If the line description specifies Telex, for each terminal description that does not specify Telex but specifies Horizontal Parity; CRC, BCC, Ones, or Summed Parity must not be specified
- If the line description specifies Telex, then, for each terminal, if the terminal description does not specify Telex, then if the terminal description has Horizontal Parity (Terminal Run Mode), then the terminal description must not specify CRC, BCC, Ones, and Summed Parity
- All of the terminal descriptions must specify the same use of Vertical Parity (Terminal Run Mode), Use of Translation (Terminal Run Mode), Use of Case Shift (Terminal Run Mode), Transmit Address Count, Receive Address Count, Sync Character, Parity Mask, Auxiliary Line Control Set, Line Control Set, Adapter Info, and Translation Table
- If the line description specifies Asynchronous, then, at each Speed specified by the station descriptions, all of the terminal descriptions must specify the same number of Stop Bits
- If a line description specifies Bits, then all corresponding terminal descriptions must specify the same modulus
- If a station description specifies Bits, then the transmit and receive address for that station must be different

The one unusual condition is that not all of the terminals described in the NDL program are necessarily allowed to communicate with a particular DCP program file. The possible terminals are those named for a DCP's current program file in that DCP's terminal description. The particular DCP is specified by the line address in the description of the line involved in the redefinition.

If the redefinition fails to meet these conditions, an error is monitored.

ERRORS

Below are the events which are reported by unsuccessful REDEFINE LINE communicates:

```
@00D0@   208 DC ERROR BAD LINE NO
@00D1@   209 DC ERROR BAD MODEM NO
@00DD@   221 DC ERROR ATTRIBUTE MISMATCH
@00DE@   222 DC ERROR DIRECT CONNECT LINE
@00DF@   223 DC ERROR FULL DUPLEX MISMATCH
@00E0@   224 DC ERROR INCOMPLETE VARIABLE
@00E1@   225 DC ERROR IMPROPER LINE CONDITION
@00E2@   226 DC ERROR MESSAGES QUEUED
@00E4@   228 DC ERROR SPEED MISMATCH
```

These are the events which are reported by unsuccessful REDEFINE STATION communicates:

```
@00C9@   201 DC ERROR BAD STATION NO
@00D0@   208 DC ERROR BAD LINE NO
@00D1@   209 DC ERROR BAD MODEM NO
```

@00D2@	210 DC ERROR BAD TERMINAL NO
@00DC@	220 DC ERROR STATION ALREADY ATTACHED
@00DD@	221 DC ERROR ATTRIBUTE MISMATCH
@00DF@	223 DC ERROR FULL DUPLEX MISMATCH
@00E0@	224 DC ERROR INCOMPLETE VARIABLE
@00E1@	225 DC ERROR IMPROPER LINE CONDITION
@00E2@	226 DC ERROR MESSAGES QUEUED
@00E3@	227 DC ERROR NO VACANCY ON LINE
@00E4@	228 DC ERROR SPEED MISMATCH

SECTION 9
MPLII USER DATA COMMUNICATIONS

GENERAL

A user data comm interface, similar to COBOL in nature, is provided within the MPLII language through a set of built-in procedures and functions. This interface provides an identical COBOL interface to the data comm subsystem.

DC.ACCEPT

DC.ACCEPT (<queue name>, <result>);

This built-in procedure is used to set the value of <result> to the fixed value of the count of messages on the subnet queue specified by <queue name>.

The status key within the input CD can be tested to determine the validity of the value of <result>. The <queue name> must be of type characters.

Refer to ACCEPT in the COBOL Data Comm Statements section.

DC.ENABLE.INPUT

DC.ENABLE.INPUT (<queue name>, <password>);

This built-in procedure invokes an MCS-defined function by sending an ENABLE QUEUE message to the MCS.

The <queue name> and <password> must be of type character.

The success or failure of the operation can be checked by interrogating the status key of the input CD area. Only the first 10 characters of <password> are significant. Refer to ENABLE INPUT in the COBOL Data Comm Statements section.

DC.ENABLE.OUTPUT

DC.ENABLE.OUTPUT (<station name>, <password>);

This built-in procedure invokes an MCS-defined function by sending an ENABLE STATION message to the MCS.

The <station name> and <password> must be of type character.

The success or failure of the operation can be checked by interrogating the status key of the output CD area. Only the first 10 characters of <password> are significant. Refer to ENABLE OUTPUT in the COBOL Data Comm Statements section.

DC.DISABLE.INPUT

DC.DISABLE.INPUT (<queue name>, <password>);

This built-in procedure invokes an MCS-defined function by sending a DISABLE QUEUE message to the MCS.

The <queue name> and <password> must be of type character.

The success or failure of the operation can be checked by interrogating the status key of the input CD area.

Only the first 10 characters of <password> are significant.

Refer to DISABLE INPUT in the COBOL Data Comm Statements section.

DC.DISABLE.OUTPUT

DC.DISABLE.OUTPUT (<station name>, <password>);

This built-in procedure invokes an MCS-defined function by sending a DISABLE STATION message to the MCS.

The <station name> and <password> must be of type character.

The success or failure of the operation can be checked by interrogating the status key of the output CD area.

Only the first 10 characters of <password> are significant. Refer to DISABLE OUTPUT in the COBOL Data Comm Statements section.

DC.RECEIVE

DC.RECEIVE (<queue name>,<destination>, <char count><wait option>);
<char count> ::= <expression>
<wait option> ::= <empty> |, NOWAIT

This built-in procedure is used to remove the top message from the queue specified by <queue name> and copy its text to the data field specified by <destination>. The number of characters moved is the smaller of the fixed value given by <char count> and text length of the message.

If the specified queue is empty, the program is waited until a message is placed on the queue, unless the NOWAIT option is specified, in which case, control passes to the next statement.

The input CD area contains information about the message and can be interrogated by use of the provided built-in functions.

The <queue name> and <destination> must be of type character. Refer to RECEIVE in the COBOL data comm statements section.

DC.SEND

DC.SEND (<station name>,<source>,<char count> <eom option> <before/after option> <line control> <NOWAIT option>);
<eom option> ::=, EMI | <empty>
<before/after option> ::=, BEFORE | <empty>
<line control> ::=, PAGE |, LINE |, LINE (<expression>) | <empty>
<NOWAIT option> ::=, NOWAIT | <empty>

This built-in procedure is used to send a message to the station specified by <station name>.

The text of the message is obtained from the data field specified by <source>. The number of characters moved is given by the fixed value of <char count>.

The message is assumed to be the last of a logical group of messages unless the EMI (end-of-message indicator) is specified.

If <line control> is specified, the station should have carriage control capabilities. PAGE specifies an advance to top of the next page. LINE (<expression>) causes N lines to be skipped; where N is the fixed value of <expression>.

If the before option is specified, the carriage control information is actioned before the message text is printed.

The output CD area contains information about the message and can be interrogated by use of the built-in functions.

If the <NOWAIT option> is specified, and the send will exceed a currently active queue limit, control is returned to the program with a fetch value of @100001@.

If <NOWAIT option> is specified and the subsystem lacks sufficient buffer space to accommodate the send, control is returned to the program with a fetch value of @100000@.

The <source> and <station name> must be of type character. Refer to SEND in the COBOL Data Comm Statements section.

INPUT RELATED FUNCTIONS

The input CD is implicity defined by the MPL interpreter.

DC.NODATA

This built-in function returns a true value if the FETCH VALUE of the preceeding communicate was equal to @000000@. It may be used after either a DC.RECEIVE or a DC.SEND. After a DC.RECEIVE with the NOWAIT option specified, DC.NODATA returns a true value if the specified queue was empty; otherwise, it returns a false value.

After a DC.SEND with the NOWAIT option specified a true value, this indicates that the message was not sent. A false value indicates the SEND was successful.

The value returned by DC.NODATA is meaningless if the last communicate was not either a DC.RECEIVE or a DC.SEND with the NOWAIT option specified.

DC.INPUT.STATUS

This built-in function returns a fixed value indicating whether or not there were any abnormal conditions associated with the last input-related data comm communicate (DC.ACCEPT, DC.ENABLE.-INPUT, DC.DISABLE.INPUT, or DC.RECEIVE).

The values are:

0	No errors; action completed.
20	Queue unknown or access denied by MCS; no action taken.
91	MCS/data comm subsystem not available; no action taken.

DC.ORIGIN

This built-in function returns a descriptor of type character, size 12 bytes. Its value is the symbolic source field (station name) of the input CD.

DC.TEXTLENGTH

This built-in function returns a fixed value which is the binary equivalent of the text length field of the input CD.

DC.DATE

This built-in function returns a descriptor of type character, size six bytes. Its value is the message date field of the input CD.

DC.TIME

This built-in function returns a descriptor of type character, size eight bytes. Its value is the message time field of the input CD.

DC.ENDKEY

This built-in function is used to interrogate the end key field of the input CD.

The fixed value returned is meaningful only if the last data comm communicate was a DC.RECEIVE.

The values are:

0	The specified text length is less than the number of text characters in the message.
2	This message is not the last of a logical group of messages.
3	This message is the last of a logical group of messages.

OUTPUT RELATED FUNCTIONS

The output CD is implicitly defined by the MPL interpreter.

DC.OUTPUT.STATUS

This built-in function returns a fixed value indicating whether or not there were any abnormal conditions associated with the last output-related data comm communicate (DC.ENABLE.OUTPUT, DC.DISABLE.OUTPUT, DC.SEND).

The values are:

0	No errors; action completed
20	Destination unknown or access denied by MCS. No action taken.
50	Character count greater than length of sending field. No action taken.
91	MCS/data comm subsystem not available.

DC.ERROR.KEY

This built-in function returns the fixed value of the error key field of the output CD.

SECTION 10
B 80-DEPENDENT FEATURES

GENERAL

This section contains a description of those features of the CMS data communications subsystem which are unique to the B 80 series.

B 80 implementation-dependent error messages are:

255 DC INVALID
256 DC ERROR LOAD FAILURE BAD COMPILATION
257 DC ERROR LOAD FAILURE BAD COMPILATION
258 DC ERROR LOAD FAILURE NOT ENOUGH MESSAGE SPACE
259 DC ERROR LOAD FAILURE CANNOT EXECUTE NDL PROGRAM
260 DC ERROR LOAD FAILURE MISSING OR INVALID CONTROLLER
261 DC ERROR LOAD FAILURE BAD COMPILATION

EXPLANATIONS

255 DC INVALID
The operator has entered a DC message when no MCS is running. This event does not set fetchvalue or fetchmessage.

256 DC ERROR LOAD FAILURE BAD COMPILATION
This event is returned and data comm load aborted if:
(SUBNET.COUNT * 16)+1 is greater than 2000. That is, if there is insufficient space for the number of subnet queues defined.

257 DC ERROR LOAD FAILURE BAD COMPILATION
This event is returned and the data comm load aborted if:
(STATION.COUNT * 12) is greater than 2000. That is, if there is insufficient space for the number of stations defined.

258 DC ERROR LOAD FAILURE NOT ENOUGH MESSAGE SPACE
This event is returned and the data comm load aborted if there is insufficient space declared in the NDL preset data for the system queue header and at least one message. Insufficient message space is declared to be:
<(STATION.COUNT+SUBNET.COUNT+2) x12+176

259 DC ERROR LOAD FAILURE CANNOT EXECUTE NDL PROGRAM
This occurs if the load of the NDL interpreter was not caused by an MCS load.

260 DC ERROR LOAD FAILURE MISSING OR INVALID CONTROLLER
This occurs if the system detects that a line channel/subchannel does not contain a valid data comm controller, of if the transmission method of the controller is incompatible with that declared for the line in NDL (for example, an async controller for a line declared as sync in NDL). The load of data comm is aborted.

261 DC ERROR LOAD FAILURE BAD COMPILATION
Insufficient space has been allocated to the NDL interpreter by the MCP. Load of data comm is aborted.

The following error messages are outside the range of B 80-dependent errors. They refer to restrictions which will be lifted in the future.

369 DC ERROR LOAD FAILURE - FULL DUPLEX LINE NOT IMPLEMENTED
This event is returned and the load of the data comm aborted if the NDLSYS contains a full duplex line.

379 DC ERROR LOAD FAILURE - TELEX LINE NOT IMPLEMENTED
This event is returned and the load of data comm aborted if the NDLSYS contains a Telex line.

SECTION 11

B 800-DEPENDENT FEATURES

GENERAL

This section contains a description of the features of the CMS data communications subsystem which are unique to the B 800 series.

B 80 implementation-dependent error messages are:

270	DC ERROR 7PM PARITY DC* XXXX
271	DC ERROR SPM PARITY DC* YYYY
279	DC LOAD/EOJ FAILURE DC* SPM PARITY ERROR XXXX
280	DC LOAD/EOJ FAILURE DC* 7PM PARITY ERROR YYYY
281	DC LOAD/EOJ FAILURE DC* NO RESPONSE (DC* indicates processor in error) (XXXX = four-digit 7PM address) (YYYY = four-digit line address) (7PM indicates DCP micromemory) (SPM indicates scratchpad memory)
279	DC LOAD/EOJ FAILURE DC* SPM PARITY ERROR XXXX
280	DC LOAD/EOJ FAILURE DC* 7PM PARITY ERROR YYYY
281	DC LOAD/EOJ FAILURE DC* NO RESPONSE (DC* indicates processor in error) (XXXX = four-digit 7PM address) (YYYY = four-digit line address) (7PM indicates DCP micromemory) (SPM indicates scratchpad memory)

Detailed explanations of the messages follow.

	TEXT
270	DC ERROR 7PM PARITY DC* XXXX The DCP has failed because of a microprogram memory parity error.
271	DC ERROR SPM PARITY DC* YYYY The DCP has failed because of a scratchpad memory parity error.
279	DC LOAD/EOJ FAILURE DC* SPM PARITY ERROR Load of DCP has failed because of scratchpad memory parity error.
280	DC LOAD/EOJ FAILURE DC* 7PM PARITY ERROR Load of the specified DCP has failed because of a DCP microprogram memory parity error.
281	DC LOAD/EOJ FAILURE DC* NO RESPONSE The DCP specified in a load or reload has failed to complete handshake after load process.

B 800 SCRATCH PAD MEMORY

A formatted SYSDUMP shows an analysis of the data comm memory space if an MCS was running at the time of the clear/start.

The last part of this analysis is a breakout of the scratch pad memory for each line (figure 11-1).

Pages 0 and 1 of the scratch pad memory dump are valid for half-duplex memory. Pages 0, 1, 2, and 3 are valid for full-duplex memory.

Columns headed by ''D---'' are the columns containing the actual data described in the figure. The columns headed by ''S---'' contain the status word for the read of the previous data word. This status word should be 0000 (good status). Any non-zero status word indicates an error in the DCP.

The following description describes the content of page 0 of scratch pad memory. Page 2 uses the same mnemonics, but is for the full-duplex auxiliary line. These descriptions deal with the individual bytes shown in figure 11-2.

Bytes 0 and 1, M-PTR-L and M-PTR-M

This field contains the absolute DCP microaddress used by the manager to store the nonerror return address.

Byte 2, LINE-NO

This field contains the physical address of the communication line associated with this set of SPM.

Byte 3, ID

This field contains the DCP number associated with this DCP.

Byte 4, DS-DESC

This field contains the current copy of the data set descriptor used by hardware. DC-DESC is only

```
PROCESSOR NUMBER: ZERO
PROCESSOR PORT: 0C
LINE NUMBER: C0
PAGE NUMBER: C/1

ADDR     D----S--- D----S--- D----S--- D----S--- D----S--- D----S--- D----S--- D----S---
0C00     007A C000 000C 00C0 0004 0000 0202 00C0 FC18 C000 FFBC C000 FFFF C000 0071 00C0
0C10     E710 C0C0 DB9F C0C0 0293 0000 000C C0C0 0C41 C000 003E 00C0 EC06 00C0 4C09 00C0
0C20     0C00 C000 006C 00C0 DB08 C000 DBAE C0C0 C603 C000 000C 0000 FFD7 C0C0 FFB1 C0C0
0C30     C603 C0C0 C616 C0C0 0500 C000 8501 C0C0 C603 C000 DBD8 00C0 1500 C0C0 CBDC 00C0

PAGE NUMBER: 2/3

ADDR     D----S--- D----S--- D----S--- D----S--- D----S--- D----S--- D----S--- D----S---
0C00     0C00 C000 000C C0C0 FF00 C000 00FF C0C0 0C00 0000 000C 00C0 0C00 00C0 0C0C 00C0
0C10     0C00 C000 000C C0C0 D963 C000 000C C0C0 0C00 C000 000C C0C0 0C00 00C0 0C0C 00C0
0C20     0C00 C000 006C 00C0 0C00 C000 DBAE C0C0 0C00 C000 000C 00C0 0C00 C0C0 0C0C 00C0
 **

LINE NUMBER: C1
PAGE NUMBER: C/1

ADDR     D----S--- D----S--- D----S--- D----S--- D----S--- D----S--- D----S--- D----S---
0C00     1621 C0C0 0001 C0C0 0005 C000 00FF 00C0 0C00 C000 0002 0000 FFFF C0C0 0C0C 00C0
0C10     8100 C0C0 DB81 C0C0 0C16 C000 0C01 00C0 FC7F 0000 1388 00C0 EC00 C0C0 002C 00C0
0C20     0C00 C000 006C C0C0 DC12 C000 DBBA C0C0 CC33 C000 000C 00C0 F87F C0C0 0C0C 00C0
0C30     0C00 C0C0 000C C0C0 0C00 C000 0C01 C0C0 CC33 C000 006C 00C0 0400 C0C0 0C0F 00C0

PAGE NUMBER: 2/3

ADDR     D----S--- D----S--- D----S--- D----S--- D----S--- D----S--- D----S--- D----S---
0C00     0C00 C000 0001 C0C0 FF00 C000 00FF C0C0 0C00 C000 000C 00C0 0C00 C0C0 0C0C 00C0
0C10     0C00 C000 0C0C C0C0 D963 C000 0C0C C0C0 0C00 C000 000C 00C0 0C00 C0C0 0C0C 00C0
0C20     0C0C C0C0 006C C0C0 0C00 C000 DBBA C0C0 0C00 C000 000C 00C0 0C00 C0C0 0C0C 00C0
 **
```

Figure 11-1. Data Comm Processor Scratch Pad Memory Dump

maintained in page 0, not in page 2. Bits and their purposes are:

Bit	Purpose
7	SECOND STOP BIT
6	STANDBY RATE
5	RATE
4	NEW SYNCHRONOUS
3	DATA MODE
2	DATA TERMINAL READY
1	ORIGINATE
0	REQUEST TO SEND

Byte 5, LINE-Q-HEAD

This byte contains the LINE-NO of the highest priority line currently in the line queue. If there are no lines in the line queue, this field contains 1's.

Byte 6, FRWD-LNK

This field contains the LINE-NO of the next lower priority line currently in the line queue. If it contains 1's, then the present line is the lowest priority in the queue. If FRWD-LNK of page 0 contains hex 80, it points to the auxiliary co-line. Relevent information is contained in pages 2 and 3. (FRWKD-LNK in page 2 always contains LINE-NO of the next lower priority line, or 1's).

Byte 7, BKWD-LNK

This field contains the LINE-NO of the next highest priority line currently in the line queue. If this is the highest priority line, the field contains 1's.

Byte 8 and 9, TIMEOUT-L and TIMEOUT-M

This field contains the timeout value associated with the currently executing NDL receive instruction.

Bytes 10 and 11, TIMER-L and TIMER-M

This field contains a work area used by the manager timer routine.

Byte 12 and 13, TRANSLATE-L and TRANSLATE-M

This field contains the absolute D-word address of the translation table associated with the active station on this line or co-line.

PAGES 0/2	BYTE	PAGES 1/3
N-PTR-L	0	S-PTR-L
N-PTR-M	1	S-PTR-M
LINE NO	2	COMMUNICATE-L
ID	3	COMMUNICATE-M
DS-DESC	4	STATION-TAB-L
LINE-Q-HEAD	5	STATION-TAB-M
FRWD-LNK	6	LINE-TAB-L
BKWD-LNK	7	LINE-TAB-M
TIMEOUT-L	8	MESSAGE-HDR-L
TIMEOUT-M	9	MESSAGE-HDR-M
TIMER-L	10	TERM-TAB-L
TIMER-M	11	TERM-TAB-M
TRANSLATE-L	12	TEXT-SIZE-L
TRANSLATE-M	13	TEXT-SIZE-M
CRC-L/BCC	14	BUFFER-SIZE-L
CRC-M	15	BUFFER-SIZE-M
CHIP-FREQ	16	CUR-BUF-L
DDP-DESC	17	CUR-BUF-M
PARITY-MASK	18	CUR-ADDR-L
SYNC-CHAR	19	CUR-ADDR-M
TIMER2-L	20	BUF-CHAR
TIMER2-M	21	IN-CHAR
CONTINUE-L	22	ACTIVE-STATION
CONTINUE-M	23	LINE-CHAR
WORK1	24	SPM-TEMP-1
WORK2	25	SPM-TEMP-2
WORK3	26	SPM-TEMP-3
WORK4	27	SPM-TEMP-4
BIU-CHAR-4	28	BIU-CHAR-0
BIU-CHAR-5	29	BIU-CHAR-1
BIU-CHAR-6	30	BIU-CHAR-2
BIU-CHAR-7	31	BIU-CHAR-3

Figure 11-2. Scratch Pad Memory Layout

Bytes 14 and 15, CRC-L/BCC and CRC-M

This field contains a work area used in calculating the BCC or CRC on this line or co-line.

Byte 16, CHIP FREQ

This field indicates the line's priority. It is maintained only in page 0, not in page 2.

Byte 17, DDP-DESC

This field contains the current copy of the DDP descriptor used by hardware. DDP-DESC is maintained only in page 0, not in page 2.

Byte 18, PARITY MASK

This field contains a mask used in stripping the parity bit from incoming characters.

Byte 19, SYNC CHARACTER

This field contains the sync character associated with the line or co-line.

Bytes 20 and 21, TIMER2-L and TIMER2-M

This field contains a work area used by the manager gross-timer routine. It is only used in page 0, not in page 2.

Bytes 22 and 23, CONTINUE-L and CONTINUE-M

This field contains an absolute D-word address used in connection with an NDL continue or receive (continue) instruction.

Bytes 24 - 27, WORK1, WORK2, WORK3, WORK4

These field are used as work areas by the S-OP microstrings.

Byte 28, BIU-CHAR-4

This field contains a set of flags used by DCP internal routines. Bits and flags are:

Bit	Flag
7	AUX-ACTIVE
6	WAIT-FLAG
5	AUX-FLAG
4	IRF
3	XMT-MODE
2	RCV-MODE
1	TIMER-ACTIVE
0	BUFFER-FLAG

Byte 29, BIU-CHAR-5

This field contains a set of flags used by DCP internal routines. Bits and flags are:

Bit	Flag
7	VERTICAL
6	HORIZONTAL
5	NO-TRANSLATE
	4 BITS
	3 FULL DUPLEX
	2 TRANSPARENT
	1 CASE-SHIFT
	0 BCC/CRC-FLAG

Byte 30, BIU-CHAR-6

This field contains a set of flags used by DCP internal routines. Bits and flags are:

Bit	Flag
7	INPUT-FLAG
6	LN-CONTROL-FLAG
5	RESERVE
4	MOD-128
3	SYNCS
2	HORIZONTAL-ODD
1	CRC-1
0	SYNC/ASYNC

Byte 31, BIU-CHAR-7

Bit	Flag
7	NORESPONSE
6	SPACE-AVAIL
5	TIMER2-ACTIVE
4	STA-NRY-PENDING
3	LN-NRY-PENDING
2	GEN-PURPOSE-C
1	GEN-PURPOSE-B
0	GEN-PURPOSE-A

The following describes the contents of page 1 of scratch pad memory. Page 3 uses the same mnemonics but is for the full-duplex auxiliary line.

Byte 0 and 1, S-PTR-L and S-PTR-M

This field contains the absolute D-word address of the NDL S-OP in execution on this line or co-line.

Bytes 2 and 3, COMMUNICATE-L and COMMUNICATE-M

This field contains the absolute D-word address of the DC-LIT-REGISTERS.

Byte 4 and 5, STATION-TAB-L and STATION-TAB-M

This field contains the absolute D-word address of the station table associated with the active station on this line or co-line.

Bytes 6 and 7, LINE-TAB-L and LINE-TAB-M

This field contains the absolute D-word address of the line table associated with this line or co-line.

Bytes 8 and 9, MESSAGE-HDR-L and MESSAGE-HDR-M

This field contains the absolute D-word address of the first buffer of the message space currently being processed.

Bytes 10 and 11, TERM-TAB-L and TERM-TAB-M

This field has the absolute D-word address of the terminal table associated with the active station on this line or co-line.

Bytes 12 and 13, TEXT-SIZE-L and TEXT-SIZE-M

This field contains a working value used by the buffer storage routines.

Bytes 14 and 15, BUFFER-SIZE-L and BUFFER-SIZE-M

This field contains a working value used by the buffer storage routines.

Bytes 16 and 17, CUR-BUF-L and CUR-BUF-M

This field has the absolute D-word address of the DC buffer currently in use.

Bytes 18 and 19, CUR-ADDR-L and CUR-ADDR-M

This field contains the absolute D-word of the last used buffer data location.

Byte 20, BUF-CHAR

This field contains a work area used by the buffer storage routine.

Byte 21, IN-CHAR

This field is equivalent to the NDL character register.

Byte 22, ACTIVE STATION

This field contains the line relative station number of the station currently active on this line or co-line.

Byte 23, LINE-CHAR

This field contains a copy of the last character exactly as it appeared on the line (LCHAR).

Bytes 24 - 27, SPM-TEMP-1, SPM-TEMP-2, SPM-TEMP-3, SPM-TEMP-4

Each field is used as a work area by routines common to both the S-OP and host-control sections of DCPP firmware.

Bytes 28, 29, and 30, BIU-CHAR-0, BIU-CHAR-1, and BIU-CHAR-2

Each field is used as a work area, mostly by the host control routines.

Byte 31, BIU-CHAR3

This field contains a set of flags used by DCP internal routines. Bits and flags are:

Bit	Flag
7	RESERVED
6	VERTICAL-EVEN
5	RESERVED
4	RESERVED
3	RESERVED
2	RESERVED
1	SHIFT1
0	SHIFT0

SECTION 12

CP 9500 IMPLEMENTATION

INTRODUCTION

This section describes the implementation of CMS Data Communications on the CP 9500. This implementation conforms to all specifications described in Sections 1 through 10. This section describes the method by which they are implemented, and those features unique to the CP 9500.

The following information is intended as instructions for those involved in design and implementation of networks containing CP 9500 and for the interest of those wishing to develop an in-depth knowledge of CP 9500 Data Communications. Although much of the information contained within this section has no direct application for the programmer, a basic understanding will promote efficient system design and utilization of the CP 9500 in the data communication environment.

The content of this section assumes a prior knowledge of CMS data communications; therefore, before continuing, the reader should be fully acquainted with the information contained in Sections 1 through 10. Information relating to CMS will not, in general, be repeated in this section. The only exception to this is when restatement of information is considered useful for the purpose of clarification.

This section is organized in two major parts. The first part deals with the steps required to prepare the CP 9500 for data communications execution. The second part deals with interface between the various components during execution.

SYSTEM OVERVIEW

The CP 9500 is a multi-processor system with each processor being dedicated to a specific function. One such function is data communication. The Data Communications Processor (DCP) is dedicated to this function. Either single or multiple DCPs are supported by the Master Control Program. In the case of multiple DCPs, each DCP is assigned control of a subset of the total data communications network.

Another function, to which a processor is dedicated, is that of executing applications programs. This processor is known as the Task Processor (TP). Data communications programs, written in either MPLII or COBOL, run on the TP.

TPs and DCPs execute asynchronously, the overall co odination being performed by the MCP. The CP 9500 has one processor, dedicated to the execution of the MCP, known as the OS processor. The following are the major components of the CP 9500 Data Communications Subsystem (DCS):

1. Data Comm Loader.
2. Data Comm Activity.
3. DCP Firmware.
4. Data Comm Buffer Memory.
5. DCS Tables.
6. DCS Queues.

Data Comm Loader (DCL)

The DCL is the function of MCP which is responsible for:

1. Loading DCP firmware into the DCP's local memory.
2. Creating tables within the OS processors memory for use by the DCA.
3. Loading the required NDL tables into the DCP's local memory.
4. Formatting the preassigned memory space, known as data comm buffer memory, for use by the DCA and DCP firmware.

Data Comm Activity (DCA)

The CP 9500's MCP is comprised of interdependent modules, known as activities. The DCA is the activity responsible for providing the interface between:

1. The data comm user programs and their supporting MCS.
2. All data comm programs (including MCS) and the DCPs.

Within these interfaces the DCA must:

1. Validate CMS communicates.
2. Initiate DCP functions.
3. Handle results of DCP execution.
4. Perform housekeeping functions such as table maintenance.

Data Comm Processors (DCPs)

Each DCP executes under control of the firmware file generated by the NDL post compiler (NPC). All DCPs execute asynchronously to one another.

The DCP provides the interface between the DCA and that subset of network devices which it controls. The DCP must control the physical interface with each data communications line during the transmission and reception of messages. These messages are transmitted and received, character by character, according to the NDL defined protocol.

Data Comm Buffer Memory

This memory is allocated for the use of the DCS. It consists of the buffers used to hold data comm messages. Once generated, a message remains in buffer's memory until successfully transferred to its destination. During the transfer, the message may be linked to a number of different queues. Pointers required to administer these queues, which may be accessed by both DCA and DCP, must remain memory resident. These pointers reside in buffer memory.

DCS Tables

The DCS tables are divided into two categories. Those created by the DCL and only used by the DCA. When resident in memory, they are located in the OS processor's memory.

The NDL tables, created by the NDL compiler, define the characteristics of the data communications network. Of the NDL tables, only line and station tables reside in memory. These are loaded, by the DCL, into the DCP's local memory. Information contained within all other NDL tables is accessed directly from disk, as required.

DCS Queues

DCS queues are the major method of communication between the various modules of the DCS. Entries in all queues reside in buffer memory. Generally, each queue has pointers to the first and last entries in the queue. Pointers to queues manipulated by the DCP only reside in DCP memory, those manipulated by the DCA reside in OS memory, and those manipulated by both DCP and DCA reside in buffer memory.

IMPLEMENTING CP 9500 DATA COMM

Using the CMS Data Comm Subsystem on a CP 9500 system involves three stages:

1. Preparation.
2. Initialization.
3. Execution.

The following paragraphs describe these stages briefly. (Each stage is described in detail later in this section.)

Preparation

In the preparation stage, the user defines the physical resources needed by data comm, and starts up the CP 9500 system so that these resources are available. During the preparation stage, the following programs must be executed:

1. The CP 9500 configurer utility defines the physical resources that data comm needs (that is, the amount of buffer memory, number of DCPs).
2. The CP 9500 Network Definition Language (NDL) Post Compiler (NPC) generates the microcode files to be loaded into the DCPs.
3. The CP 9500 Warmstart Utility loads the CP 9500 system's firmware into its processors and starts execution. Warmstart also reserves the physical resources specified by the configurer utility in the SYSCONFIG file.
4. The data comm subsystem is not loaded until a Message Control System (MCS) program has been initiated. (NOTE: The actual loading of the MCS does not take place until after the data comm load module finishes.)

Initialization

Initiating a Message Control System (MCS) program causes the Master Control Program (MCP) to call upon the DCL module of its DCA to performs the following functions:

1. Load each DCP microcode file to its physical DCP.
2. Initialize data comm tables in reserved data comm memory.
3. Return control to the MCP; if the DCL succeeded in loading data comm, the MCS that was initiated is now actually loaded.

MCP activities assist DCL in initializing data comm.

Execution

When initialization is complete, the MCS controls the data comm subsystem by initiating the transfer of messages through the data comm interfaces provided by the MCP.

The DCA module is a major component in the control of message transfer. DCA is a resident MCP activity that controls the data comm portion of memory and provides the interface between the DCP(s) and either a message control system or non-MCS data comm programs.

DCP's provide the interface between the terminal device and DCA. Each DCP is controlled by resident microcode whose function it is to accept and deliver messages within the network using non-interpretive line control and request procedures.

SYSTEM CONFIGURATION

In order to implement CMS data comm on the CP 9500, the following components are required:

1. Hardware.
2. Firmware
3. Software.

Hardware

Hardware consists of:

1. OS Processor.
2. Data Storage and Maintenance Processor (DS&M).
3. One DCP.
4. One TP.

In addition to the previous hardware, the DCP must contain at least one Data Comm Interface (DCI) adapter. The DS&M must control at least one disk device. All other peripherals are optional and depend on application requirements.

Firmware

The firmware that controls a CP 9500 system consists of the following components (see figure 12-1):

1. The Master Control Program (MCP) resides in the OS processor. Its responsibilities are:

   a. Controlling of all peripherals, except data comm and disk devices.
   b. Interfacing with the DSCP, which controls disk devices.

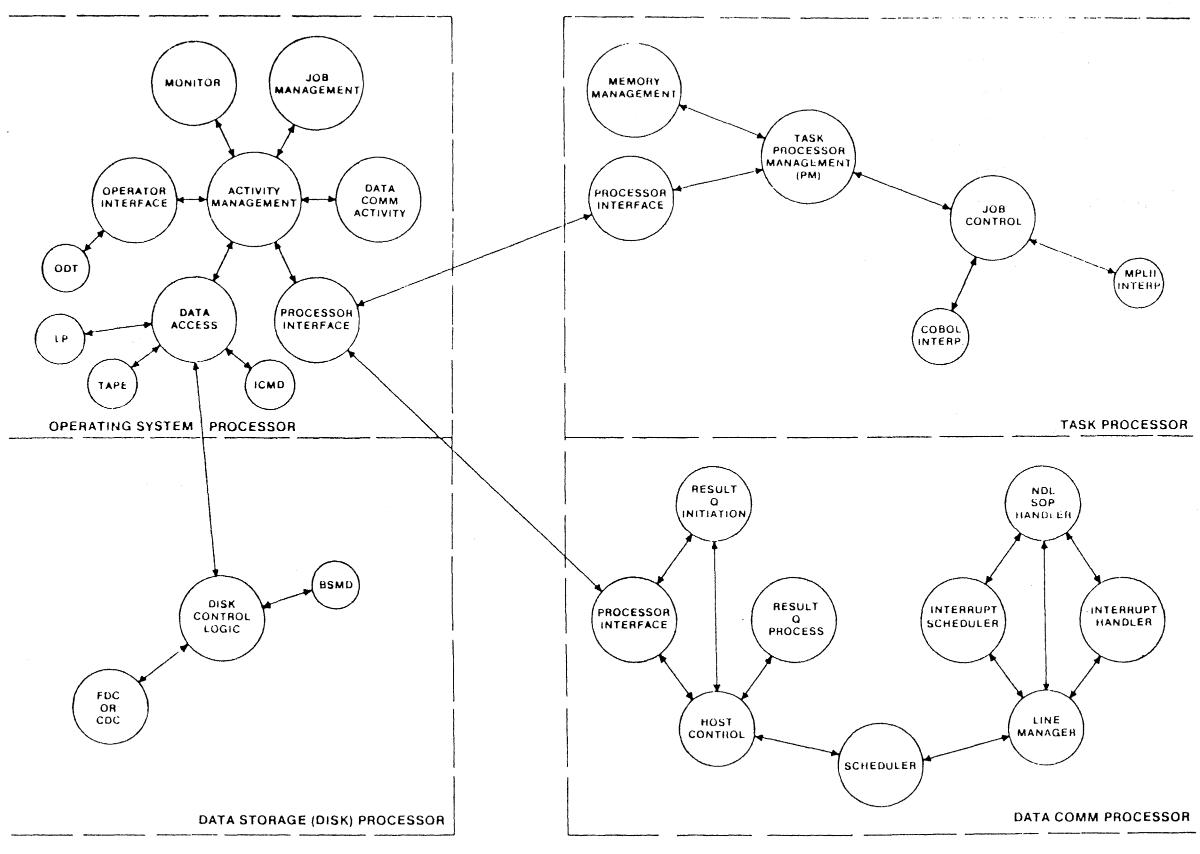


Figure 12-1. CP 9500 Firmware

c. Assigning user jobs for execution by interpreter control programs.
d. Serving as the hub of all communications between processors. The OS processor can communicate with all other processors on the system, while it is the only processor with which any one of the others can communicate.

2. Each task processor is controlled by a copy of the interpreter control program. Each ICP operates independently of other ICPs on the system, and can only communicate with the MCP. All I/O required by user jobs is performed by communicates issued between the ICP and the MCP.
3. Each ICP supports either the COBOL or the MPLII interpreter, or both. At least one ICP must support MPLII.
4. Each active DCP is controlled by a DCP firmware file generated by the CP 9500 NDL post compiler.
5. Physical control of disk devices is managed by the Data Storage Control Program (DSCP), which resides in the Data Storage and Maintenance (DS&M) Processor.

Software

Certain programs and files must exist before CP 9500 data comm may be initiated. An MCS program is required together with an NDLSYS defining the communications network. The NDLSYS is used at load time to locate the firmware file for each DCP. Data comm application programs are optional and dependent on system design.

CP 9500 Unique Features

This implementation supports the multiple MCS facility. Currently this facility is not available in CMS. Therefore, although many OS tables are designed around this facility, the interface within CMSNDL is not yet present and the facility may not be used.

CP 9500 Preparation

In order to prepare for the data comm execution, the following processes must be undertaken:

1. Create/modify SYSCONFIG entering desired data comm parameters. This requires execution of the configurer utility. (See CMS SOG, form 2007258.)
2. Warmstart the CP 9500.
3. Compile the NDL source file describing the desired data comm network.
4. Execute NPC to produce the DCP firmware files.
5. Execute the MCS program.

NOTE

The above is not necessarily in the exact order in which the steps must be performed. However, step 1 must preceed 2, 3 must preceed 4, and all must preceed 5.

The following paragraphs describe each step in detail. (Refer to figure 12-2 for system states during the various stages of preparation.)

SYSCONFIG

This file is required by WARMSTART and contains certain fields pertaining to CP 9500 data comm. These fields may be modified using the configurer utility. For complete instructions on configurer execution, refer to the CMS Software Operations Guide. The following describes the SYSCONFIG fields relevant to data communications.

Data Comm Buffer Memory Size

Warmstart reserves an area of memory known as OS buffer memory. This field specifies the amount of area to be reserved for data comm use. This amount must be sufficient in size to accommodate the buffers declared in NDL. The formula for computing this requirement is discussed later in the section.

DCP/TP Assignment

In NDL, the programmer refers to DCPs via the logical DCP number (0-n). SYSCONFIG allows assignment of physical processor (processor bus address) to logical DCP number. The assignment of DCPs is related to TP assignment. The following possibilities exist:

1. The user assigns only TPs. Any processor not assigned remains unassigned and cannot be used in this situation. The user may not require DCPs and may assign them as TPs.
2. The user prefers to leave the assignment of processors to DEFAULT. In this case, all processors with DCIs are assigned as DCPs; all others as TPs. DCP logical-to-physical relationship is produced by allocating the ascending order of the logical DCP number to the descending order of the bus address. For example: DCP bus address 7 becomes logical DCP 0, and DCP bus address 6 becomes logical DCP 1.

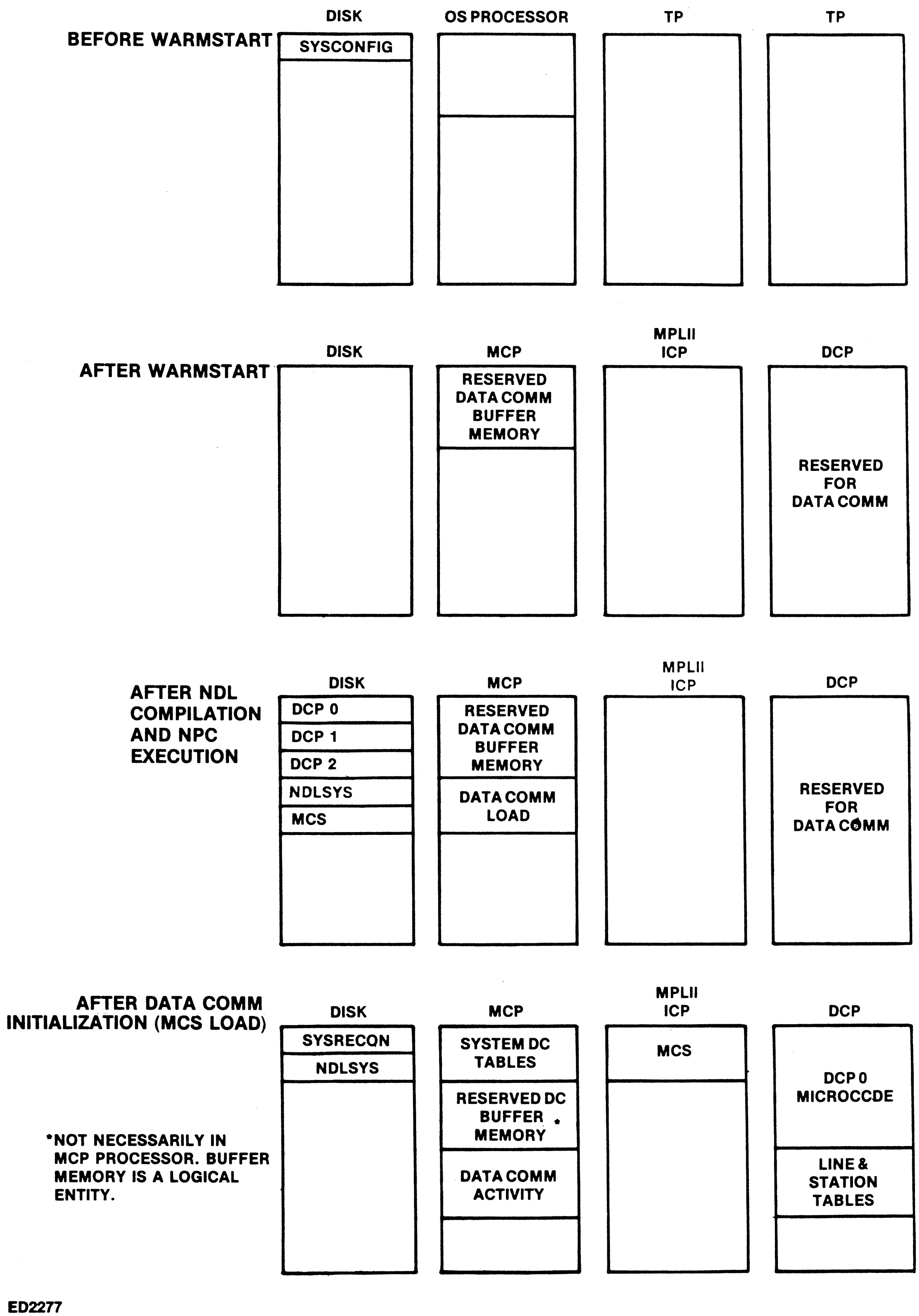


Figure 12-2. CP 9500 Preparation and Initialization

Warmstart

Warmstart uses the parameters in SYSCONFIG to reserve data comm resources. If SYSCONFIG is changed, the CP 9500 must be warmstarted in order to invoke these changes. Once invoked, all parameters remain in effect until the next warmstart.

If an error is encountered during warmstart (which prevents the SYSCONFIG parameters from being used) the DEFAULT processor assignment as described is actioned. Such errors include the following:

1. The processor at bus address N does not have a DCI or does not exist.
2. No TP was assigned.

After warmstart, all assigned DCPs are reserved exclusively for data comm use; that is, they cannot be dynamically loaded with ICP firmware.

NDL Compilation

The major requirement for this phase is related to the fact that NDL on the CP 9500 is non-interpretive. Therefore, a DCP firmware file must be defined for each required DCP. The definition of each file is used by NPC in generating that file. This subject is covered in detail in the CMS Network Definition Language Reference Manual.

NPC Execution

This process generates the required DCP firmware files to be loaded at data initiation. This subject is covered in detail in the CMS Network Definition Language Reference Manual. (Form 1090925)

CP 9500 DATA COMM INITIATION

The MCP module responsible for data initiation is the DCL. As stated previously, MCP is divided into modules, known as activities. There is a considerable degree of interaction between these activities. DCL interfaces with the following activities:

1. Job Management (JM).
2. Data Access (DA).
3. Monitor (MN).

The following paragraphs describe how and when this interaction takes place.

DCL Job Management Interface

MCS Load

Job management is responsible for loading all CMS programs, including data comm programs. Data comm is initiated upon execution of an MCS. Job management recognizes that the requested program is an MCS. After having determined that the MCS can be accommodated, JM invokes DCL with wait. The MCS name, task-id, and Message Reference Area (MRA) size are passed as parameters to DCL.

After DCL performs its functions, it returns control to JM which continues the MCS load.

Non-MCS Data Comm Program Loads

DCL is required only when initiating the DCS. When loading non-MCS data comm programs, JM invokes an action of the DCA (DC-JOB-LOG). This action is invoked after the program has been loaded but before it begins execution. The purpose of this action is to record the fact that this task-id is now valid. The task-id is passed as a parameter to DC-LOG-JOB.

Job management is responsible for invoking the DC-EOJ when either an MCS or non-MCS data comm program terminates.

DCL Data Access Interface

DCL uses data access for all disk-to-memory or memory-to-disk transfers required during the load process. Also data access is called to allocate the required portion of buffer memory for data comm use.

DCL Monitor Interface

Monitor is used by DCL to obtain information on the processor assignments made at warmstart. Monitor provides DCL with the physical-to-logical DCP relationships and also the local memory size of each DCP.

DCL PROCESS

At the most general level, DCL performs the following functions.

1. Loads the DCP firmware files and NDL tables into their assigned DCP memories.
2. Creates tables in OS processor memory for use by the DCA.

3. Initializes pointers in buffer memory for use by DCA and DCP.
4. Formats the remainder of DC buffer memory into buffers according to NDL specified parameters.

DCL then starts each DCP via a CLEAR command followed by an UNFREEZE. (These commands relate to the processor interface control hardware.)

DCL invokes monitor to mark each DCP as being in one of the following states:

1. Loading.
2. Running.
3. Available.

DCL determines whether the buffer space provided by DA buffer allocation is sufficient for the data comm subsystem to function. DCL declares a successful load if sufficient memory space is available for both the reserved data comm pointer area, and the minimum number of buffers requested by the NDL programmer. If the space is insufficient, DCL aborts the load and issues an error message to the operator indicating the failure.

Data Comm Load Input

DCL requires the following input files:

1. NDLSYS.
2. DCP Firmware File(s).

Succeeding paragraphs describe these files.

NDLSYS File

NDLSYS contains all tables and microcode file names needed to load and operate the data comm subsystem. Once opened by DCL, NDLSYS remains open until the MCS goes to EOJ. Until closed, none other than the current NDLSYS file may be used. DCL stores this file's FIB in the DCA buffer memory tables.

DCL uses the NDLSYS file to:

1. Determine whether the MCS is allowed with the particular NDLSYS.
2. Construct a copy of the NDLSYS file's Data Segment Table (DST) in DCA memory. (NOTE: Data segments in NDLSYS are numbered beginning at 1; DCA routines consider the first segment to be segment 0. The DCA's copy of the DST is adjusted to compensate.)
3. Reference the list of DCP firmware files named in NDLSYS to be loaded into the physical DCPs according to assignments reported by monitor.
4. Load the line and station tables into the DCP's memories.

DCP Firmware Files

NPC generates one microcode file for each DCP specified by the NDL programmer in the DCP terminal statement. DCP microcode files consist of sections of code which DCL loads into the low order of DCP memory, and tables and other data variables which are loaded into the high order area. NPC embeds a date stamp, random number, code length, and data size pointers in the first sector of each code file. DCL uses the date stamp and random number to match each file with the NDLSYS file used to create it.

The DCP firmware files are placed in DCP memory directly from disk via DA. Tables destined for the DCP are loaded from MCP memory to DCP memory.

The following describes DCP firmware file attributes:

FILETYPE:	@17@
RECORD:	180 BYTES
BLOCK:	180 BYTES
FILE NAME:	As per DCP Terminal Statement in NDLSYS.
SINGLE AREA:	TRUE
FILE SIZE:	Maximum address of the generated microcode.

Data Comm Load Output

DCL produces the following output:

1. Initialized DCP memories.
2. DCA tables.
3. Buffer memory space for pointers.
4. SYSRECON file.

DCP Memory

The following depicts the DCP layout:

@0000@	RESERVED MEMORY
@0160@	MICROCODE
	TABLES
@FFF0@	

Data Comm Activity Tables

DCL initializes the following tables in MCP memory for DCA's use:

ABSOLUTE	DC–DATA
VIRTUAL	FILE INFORMATION BLOCK
	DCP INFORMATION BLOCK
	MCS INFORMATION BLOCK
	MCS NAME BLOCK
	LINE INFORMATION BLOCK
	STATION INFORMATION BLOCK
	SUBNET INFORMATION BLOCK
	TASK INFORMATION BLOCK
	MESSAGE REFERENCE AREA

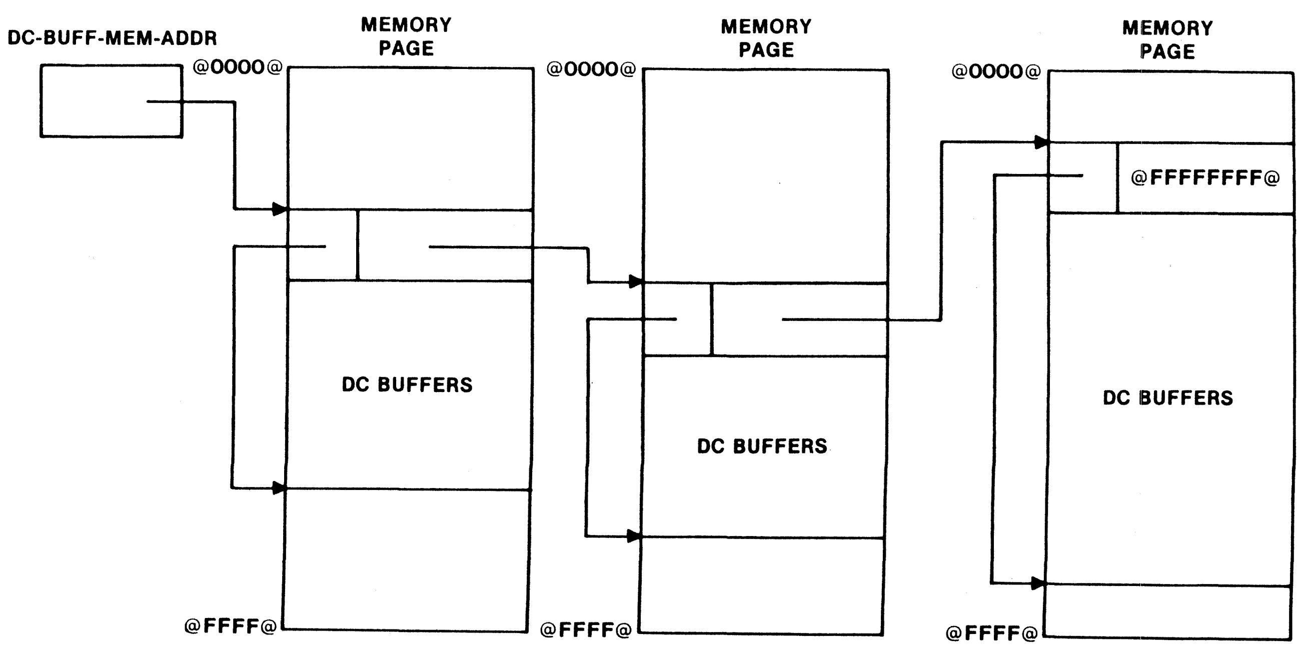
Buffer Memory

Buffer memory space is contiguous within a page. If more than one page has been specified, the pages are linked together. A pointer to the first page of the data comm buffer area is stored in ABSOLUTE-DATA area DC-DATA (in the DC-BUFF-MEM-ADDR field) of DCA memory. The first two bytes of the buffer area contain the length of this memory page. The next four consecutive bytes contain the link address to the next page (if one exists). The link address of the last page of buffer memory contains a "1" in each bit. (See figure 12-3.)

The data comm pointer area can range from a minimum of 31 bytes to a maximum of approximately 2,700 bytes. This variability is dependent on the number of subnet queues used by a given system. The breakdown of the DC-POINTER-AREA use is as follows:

				Minimum	Maximum
ABP	=	12 bytes	=>	12 Bytes	12 Bytes
RESULT–Q	=	10 Bytes	=>	10 Bytes	10 Bytes
REQUEST-Q	=	9 Bytes Times Highest Physical DCP =	=>	9 Bytes	81 Bytes
SUBNET–Q	=	10 Bytes Times	=>	0 Bytes	2550 Bytes
TOTAL				31 Bytes	2653 Bytes

The size of data comm buffers is defined in NDL. However, the CP 9500 uses four-byte rather than two-byte buffer links. In order for the text capacity of the data comm buffers to equal that of other CMS systems, DCL creates buffers four bytes larger than specified in NDL. The buffer size variable in absolute memory reflects this modified size. Depending on the amount of textual data to be contained within a message, multiple buffers may be used to accommodate a message.



ED2278

Figure 12-3. DC Buffer Page Linkage

SYSRECON File

DCL copies all station and line tables from the NDLSYS file into the SYSRECON file. During execution, the data comm subsystem references SYSRECON for the network configuration. If the configuration should change, SYSRECON is altered.

Figure 12-4 depicts SYSRECON file contents.

RANDOM NUMBER	2 BYTES
NDLSYS DATE	6 6 BYTES
TEMP/PERM FLAG	1 BYTES
LINE SEG PTR	2 BYTES
LINE SEG LENGTH	2 BYTES
LINE DISPLACEMENT PTR	2 BYTES
LINE DISPLACEMENT LENGTH	2 BYTES
STATION SEG PTR	2 BYTES
STATION SEG LENGTH	2 BYTES
STATION DISPLACEMENT PTR	2 BYTES
STATION DISPLACEMENT LENGTH	2 BYTES
RESERVED	155 BYTES
LINE SEGMENT	
LINE DISPLACEMENT SEG	
STATION SEGMENT	
STATION DISPLACEMENT SEG	

Figure 12-4. SYSRECON File Contents

At initial load time, DCL builds a disk file containing all NDL defined lines and stations, effectively making a copy of the line table and station table segments from the NDLSYS file. This SYSRECON file is initially OPENed, then CLOSED, then OPENed again. This is to guarantee that in the event of a need to recover from a system failure, the file has been entered into the disk directory.

The RANDOM NUMBER from NDLSYS is placed at the beginning of this file. A byte to indicate whether this file is to be saved when the MCS goes to EOJ is also included. This byte is set when this SYSRECON file is to be saved. At MCS EOJ time, the file is either PURGED or CLOSED with LOCK, depending on this indicator. This implementation allows the system to recover up to the last system table configuration. It may be necessary to do this when:

1. The MCS has made a change to the system which it wants to save. (Example: without having to recompile NDL.) In this case, subsequent LOAD should reflect this change.
2. When a DCP table memory parity error occurs and the MCS wishes to recover by reloading tables.
3. When the system has been halted because of a hardware failure and it is necessary to RESTART.

At initial load time, if a SYSRECON file already exists on the system drive, and if it matches the NDLSYS file, the DCL loads from the NDLSYS or SYSRECON file depending on the RECOVERY indicator. If RECOVERY is not required, a new SYSRECON file is created. If, however, the existing SYSRECON file does not match the NDLSYS file, DCL purges the existing file, loads the NDLSYS file, and creates a new SYSRECON file. If no SYSRECON file exists at initial load time, DCL simply loads the NDLSYS file, and creates a new SYRECON file.

Data Comm Load - Flow of Control

DCL flow of control proceeds as follows:

1. Open NDLSYS, read data, and build NDL table.
2. Create DCP and DCP-conversion tables, and determine which DCPs are on the system.
3. Create MCS-ID, MCS name, and MCS tables in the DCA.
4. Allocate buffer memory space for queue pointers, and initialize the Available Buffer Poll (ABP).
5. Build the LLN-conversion table in DCA memory, read the line tables from disk, transfer the line table to the DCP, and build the line data area.
6. Build the LSN-conversion and LSN-information tables in DCA memory, read the station tables from disk, and transfer the station tables to the DCP.
7. Build the subnet information table and the user jobs table in the DCA.
8. Remove the old SYSRECON file (if one exists), open a new SYSRECON file, fill in the file directory, transfer line and station tables from NDLSYS to SYSRECON, close the file lock, and open again.
9. Build the MRA link block and place a pointer in the MCS table.
10. Start the DCP by issuing a CLEAR and UNFREEZE for each DCP and inform the monitor.

The following paragraphs describe the individual DCL procedures.

LOAD-ACTION

Store parameters passed from JM. Check MCS-LOADED (declared in MIDL) to see if this is the first MCS on the system. If not, call VALIDATE-MCS and SET-UP-MRA. If it is the first MCS, call TEST-FOR-RESTART. If this is a restart, execute RESTART-PROC (permanent) reconfiguration. If not, execute OPEN-NDLSYS, BUILD-NDL-TABLE, BUILD-DCP-TABLES, CREATE-MCS-TABLES, VALIDATE-MCS, FORMAT-BUFF-MEM, SEND-DCP-FILES, SET-UP-MRA, and START-DCP. If the load should fail, call DEALLOCATE-LB, and set up the FCM with the appropriate event number.

OPEN-NDLSYS

Allocate a 19-byte workblock and fill it with "NDLSYS------." Set pack ID to 000000 and invoke OPEN-SYSFILE with WAIT. When control returns, store FIBID. Allocate and freeze a 182-byte buffer workblock. Read-in the program parameter block. Check the priority class and store the data segment table length and address, and the date of compilation.

BUILD-NDL-TABLE

Read NDL data segment table from disk. Store line, line displacement, station, station displacement, preset, MCS line, MCS file, and MCS name pointers used by DCL. Allocate the NDL data linked block and store the disk address/length of the modem, terminal, file, extended station, extended terminal, station name, file name, and DCP terminal format B tables. Read the NDL preset area from disk. Put modem and terminal counts, station table size, and NDL complete date into the NDL data linked block. Multiply the buffersize by two and store the result in an absolute variable. Put DCP limit, station, subnet, and line counts into ABSOLUTE-DATA variables. Store the minimum buffer count locally to DCL.

BUILD-DCP-TABLES

Create/initialize the DCP conversion and DCP linked blocks to binary ones, and initialize file name to blanks. Invoke monitor to determine the numbers of all processors that are physically potential DCPs. Determine whether NDL has a load file for each DCP. If so, fill the DCP table with the file name and physical DCP number. Put the relative DCP number in the DCP conversion table.

CREATE-MCS-TABLES

Initialize the MCS-ID table in ABSOLUTE-DATA to binary ones. Get a linked block for the MCS-NAME table and read it in from disk. Get a linked block for the MCS-TABLE and initialize it.

VALIDATE-MCS

Check the MCS-COUNT in MCS-NAME table; if it is @FF@, then any MCS name is valid, but only one MCS is allowed. Enter the job name, passed from JM, in the MCS-NAME table. Assume that there are always MCS-COUNT (non-zero) names in the table. If the MCS-COUNT is not binary ones, compare the MCS-NAME passed from job management with each entry in the MCS-NAME table. For each match, check the entry in the MCS-ID table to see if this relative MCS number is already being used. If not, this MCS is valid; enter the relative MCS number into the MCS-ID table.

FORMAT-BUFF-MEM

Calculate the minimum amount of space needed by data comm in unattached memory. Compare the minimum requirement with the amount available (specified by SYSCONFIG). If the available memory is equal to or greater than the amount required, continue the load; otherwise, return an ERROR result indicating that the DC-LOAD has been aborted. In buffer memory, create DCP-COUNT request queues, SUBNET-COUNT subnet queues, and a result queue. Fill in pointers in DC-POINTER-AREA memory. Use the remaining space for the available buffer poll. Fill in the first four bytes of each buffer link; the rest of the buffer is undefined. Fill in the HEAD, TAIL, and COUNT pointers in DC-POINTER-AREA of memory.

SEND-DCP-FILES

Test each DCP to see if it is valid. Call OPEN-DCP-FILE and store NPC-DATA-SIZE. Calculate the amount of DCP memory needed. If there is enough physical memory, and if the compiled random numbers of the NDL and DCP files match, call LOAD-DCP-FILE. When finished, check to see if the DCP has been loaded. If not, generate a message to Operator Interface (OI); call monitor to mark the DCP "DEAD"; delete its entry in the DCP tables.

OPEN-DCP-FILE

Allocate a 19-byte workblock. Invoke OPEN-SYSFILE and store the FIBID. Read/store the first sector of the DCP file (the file pointers).

LOAD-DCP-FILE

Using DA, load the code into DCP memory, then close the file. Place the MCP processor number, this processor's number, a READ and a RWL word, pointers to the request, result, and available buffer poll queues, and the starting address of the table into DCP memory.

SEND-LINE-TABLE

Allocate a linked block for the LLN conversion table of size (LINE COUNT * bytes per LLN entry). Freeze buffer workblock and read into it the line displacement list from the NDLSYS file. Get a workblock, freeze it, and read in the NDL segment containing the logical line states. Allocate a 720-byte workblock, freeze it, and read the first four sectors of line tables. For each line table, calculate its length, check that its logical processor number is valid, and that its physical line number is in range. If valid, update the port number within the line table (adjustment for PI in port #0), calculate the physical processor number, and place it into the LLN conversion block. Place the line table into DCP memory. Put the line table's address in the LLN conversion table. Put the line data field in DCP memory, and adjust its internal addresses. Insert the MCS logical line station into the line data field, and place the data field's address into the DCP code. If the line is full-duplex, place a second line data field into DCP memory. If the DCP is invalid and if unloaded into the processor field of the LLN conversion table, enter @FFFE@. When a partial line table remains at the end of a workblock, move the partial table to the beginning followed by enough new sectors to fill the workblock. Continue processing line tables until finished. Then, free the line table workblock and thaw the buffer block. Send a dummy line table to each DCP for port #0. Prefix the dummy line table with a line data field and set the pointers.

SEND-STATION-TAB

Allocate linked blocks to hold the LSN conversion and the LSN information tables. Freeze the buffer block and read in the first sector of the station displacement list. Allocate and freeze a 254-byte workblock. Read the first sector of station tables into the workblock. For each station table, initialize its entry in the LSN information and the LSN conversion tables, and determine if the station can be loaded. If the station can be loaded, calculate the table size. Transfer MCS information from the station table to the LSN conversion table. Send the station table to DCP memory. Change the queue pointers to binary ones, take the two's complement of active transmit and initiate receive delays, and fill-in its address in the line table and the LSN conversion table. If the station cannot be loaded, enter @FFFF@ (if unattached) or @FFFE@ (if unloaded) into this station's LSN conversion processor field. Enter the station's SYSRECON disk address into the station table address field in the LSN conversion table. If the bottom of the displacement list is reached, read in a new sector. If a partial station is left at the end of the workblock, move the partial station to the beginning of the workblock and read in a new sector of station tables after it. Continue processing until all stations are handled, at which time both the station workblock and buffer block are free.

SET-UP-MCS-TAB

Allocate and zero-out the linked block for the user job table. Read the MCS file information from the NDLSYS file, create the subnet table, and insert the MCS data.

SET-UP-SYSRECON-FILE

Get a workblock to hold the SYSRECON file directory. Calculate the length desired for the SYSRECON file, and open a file that size. For each segment, enter the segment address and length in the directory, then transfer the segment from NDLSYS. When complete, write the directory to disk and close, then re-open the file, storing the FIBID into the NDL table.

SET-UP-MRA

Create MRA linked block of size specified at DCLOAD invoke time; zero it out. The MRA size parameter is not tested to see if it is too large. The MCS table is updated with the MRA block-ID and the MCS task0ID.

START-DCP

Scan the DCP table for successfully loaded DCPs. Clear, unfreeze, and mark as "RUNNING" each successfully loaded DCP.

DEALLOCATE-LB

If the load fails, determine which MCP linked blocks have been allocated, and free them.

DATA COMM EXECUTION

When DCL process is completed, the CP 9500 is ready to begin executing data comm functions. Their functions are requested by data comm applications (MCS and non-MCS) and performed by the DCA and DCP firmware. Figure 12-5 shows the message flow as controlled by the DCA. Figure 12-6 shows the distribution of data comm functions and locations of tables and queues.

The following constitute the major processes involved in data comm execution:

1. Data Comm Interfaces (MCP Activities).
2. Data Comm Activity.
3. Data Comm Processors.
4. Buffer Management.

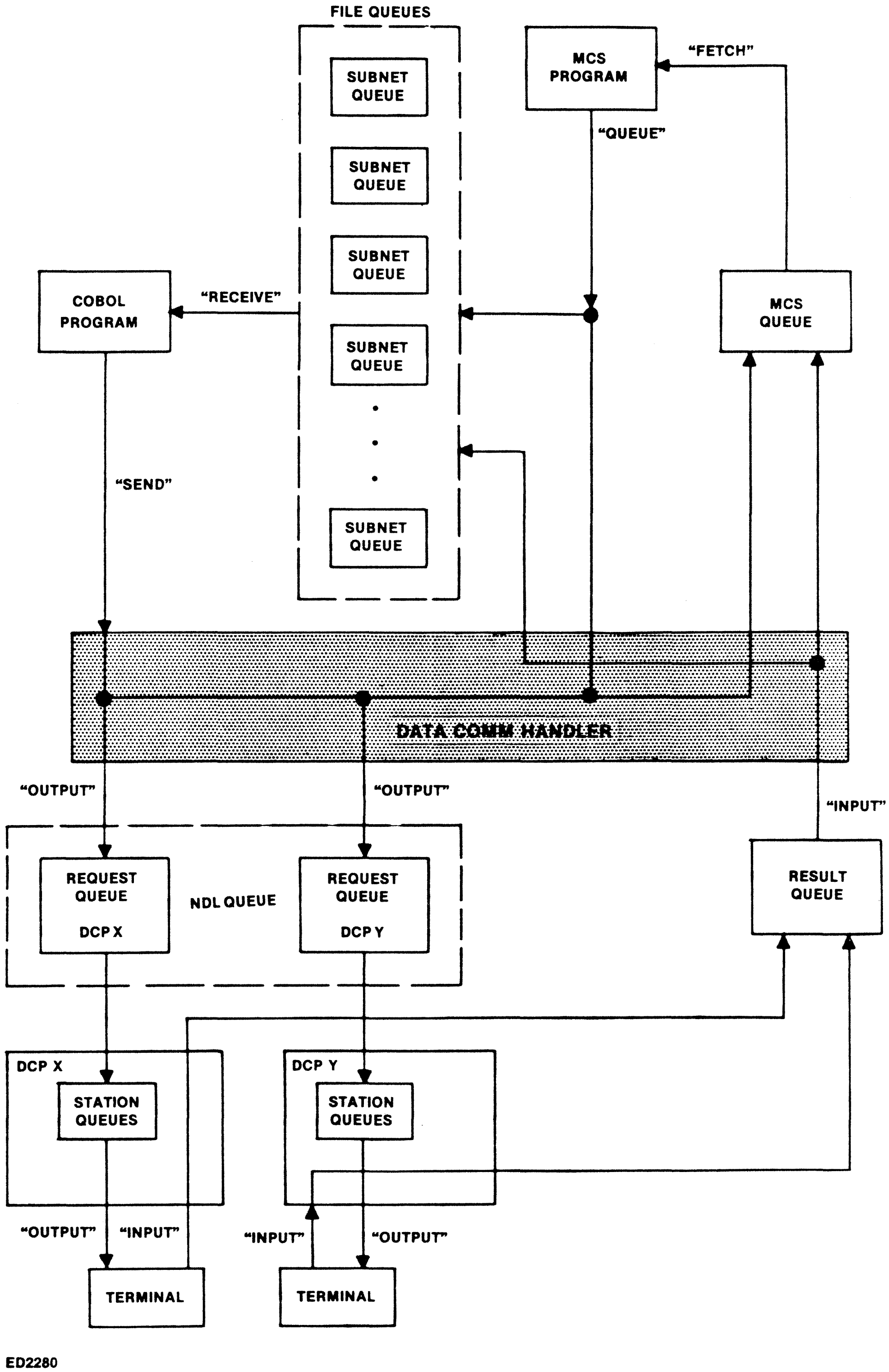
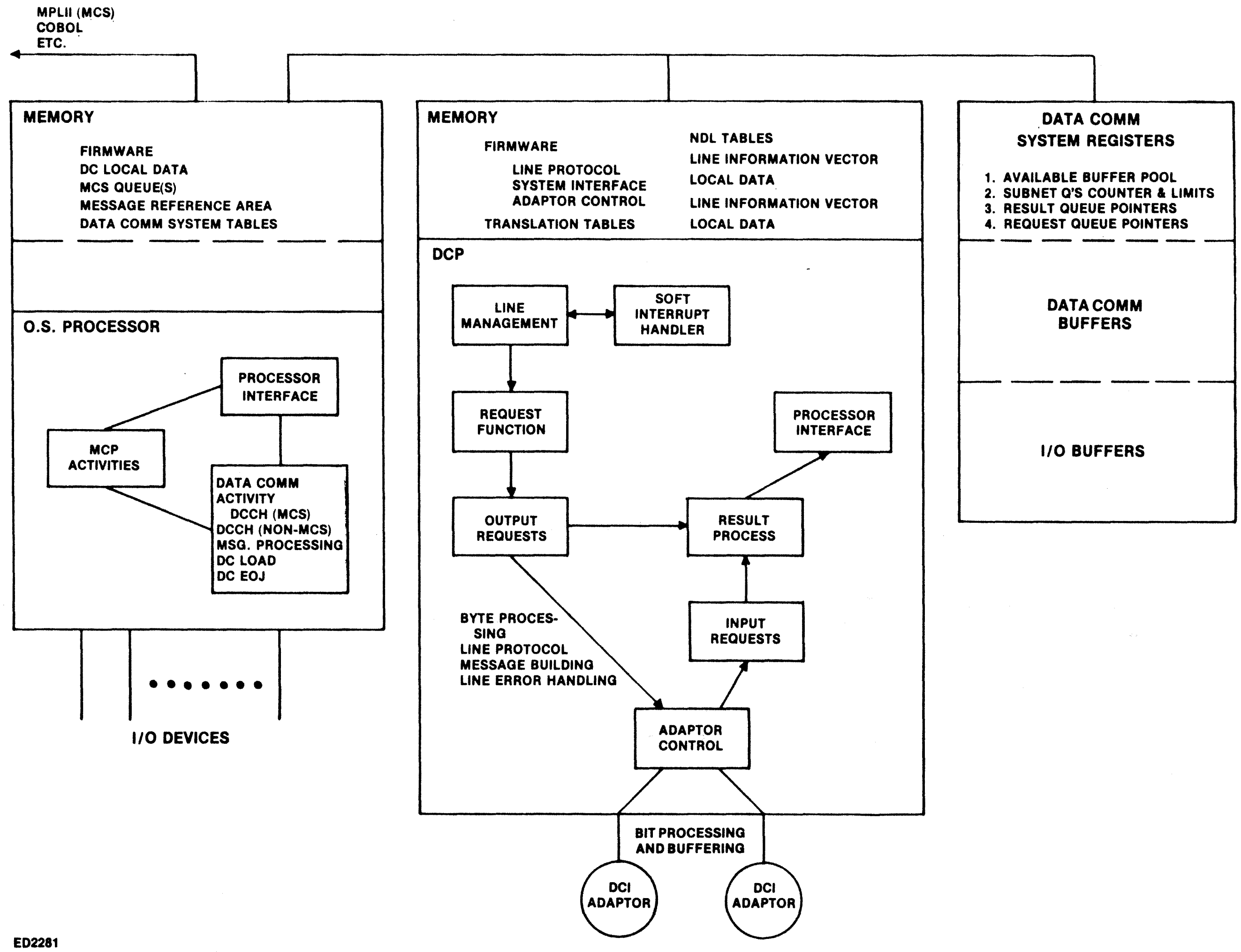


Figure 12-5. Message/Communicate Flows



ED2281

Figure 12-6. CP 9500 Data Comm Subsystem

Data Comm Interfaces

The DCA interfaces with the following MCP activities:

1. Processor Interface.
2. Data Access.
3. Monitor.

Processor Interface

Processor Interface (PI) uses a mailbox technique to deliver requests/results from one processor to another. PI code exists on all types of processors.

The PI code for DCP is generated by NPC. PI is not the major method by which the DCP communicates with the OS processor; therefore, DCP PI code is much simplier than that of other processors.

Data Access

Data access is used, by DCA, for file access, that is, NDLSYS and SYSRECON.

Monitor

If the firmware executing within the DCP detects an error, it performs a register dump to the save state area in DCP reserved memory, then freezes the processor. The monitor activity periodically tests the error status of each DCP. When a DCP is frozen, the monitor performs a dump of DCP memory and reports the DCP error to the operator.

Data Comm Activity

The following paragraphs describe the MCP's data comm activity. DCA controls all message flow within the CP 9500. All CMS communicates described in Sections 1 through 9 are validated and performed by the actions of the DCA.

The DCA requires access to all NDL tables (both in memory and on disk) and all queues. The DCA tables provide this access. A very small area of absolute data (always memory resident) exists for the

Field Name	Length Bytes	Description
Reload header count	1	Count of line marks returned by DCP
MCS loaded	2	Number of MCS programs loaded
DCP limit	1	Highest numbered logical DCP
Buffer size	2	NDL buffer +4
Station count	2	NDL total stations
User DC log	4	Each bit on represents a task-ID (0-31) currently executing
Subnet count	1	NDL total lines
Line count	1	NDL total lines
MCS ID table	32	Converts task-id to relative MCS number
Buffer memory address	4	Pointer to the first page of buffer memory
Buffer queue address	4	The start of the queue pointer in buffer memory
EOJ action ID	1	The action to be invoked when a data comm program terminates

Figure 12-7. DCA Absolute Data

DCA (see figure 12-7). The majority of DCA data exists in the form of linked blocks. Each linked block is a data segment and subject to MCP virtual memory handling. An action of the DCA wanting to access a linked block must call an activity management routine in order to locate the linked block. The following describes each linked block used by the DCA.

User Jobs

This linked block contains an entry for each task runnable on the system. This table is 32 entries in length. The USER-DC-LOG in absolute memory identifies which of these tasks are currently active.

Subnet Info

There is one entry for each subnet declared in the NDLSYS file. If no subnets are declared, this linked block is not created. The length of this table is stored in the SUBNET-COUNT field in the ABSOLUTE-DATA area.

MCS Table

This table contains an entry for each MCS runnable in the system. It is indexed by relative MCS number and contains information regarding that MCS. The length of this table varies and depends on the number of MCSs declared in NDL.

LSN, LLN Conversion

The LSN conversion table contains the station table addresses. The LLN conversion table contains the line table addresses. Each address is a four-byte field consisting of the following:

PROCESSOR	2 BYTE
ADDRESS	2 BYTE

When the table resides in memory, the address is an absolute address of the table's location. Unattached stations and lines and those stations associated with an unloaded DCP, however, reside on disk rather than in memory. For these, the processor and address fields have different meanings. The address field is a word offset of the table location in the SYSRECON disk file. For stations, the address field is an offset into the station segment; for lines, the address field is an offset into the line segment. The processor field is set to @FFFF@ to indicate an unattached station, and is set to @FFFE@ to indicate an unloaded line or station.

DCP Conversion

This table contains an entry for each possible DCP. It is indexed by the physical processor number and is used to convert to logical DCP number.

NDL Data

This linked block contains information stored from the NDLSYS file at load time. This area is 47 bytes in length.

MCS Name

This table is copied at load time from the NDLSYS file. It contains the count and names of all the MCSs able to run with this NDLSYS file. The table is also present for a single MCS system, in which case it is 20 bytes of @FF@.

MREF Area

One MREF area linked block is created at each MCS load time, and deallocated with that MCS's EOJ. The size of this area is passed to DCL by job management from the Program Parameter Block (PPB).

The message reference area for each MCS is set to all binary ones at MCS load time. When a message pointer is released from the message reference area, the PROC/PAGE fields of the address are set to @FFFF@. Thus, a null message reference is one in which at least the first byte is @FF@. Figure 12-8 shows the relationship between the various MCS linked blocks and absolute data.

DCA Initiation

When an ICP issues a request for a communicate to be performed, PI-SEND is INTERRUPTed with a mailbox containing the Communicate Parameter Area (CPA) for the requested communicate. PI-SEND interrupts PI-RECEIVE on the MCP. PI-RECEIVE fills its communicate area with the contents of the mailbox passed, and invokes the action passed to it by PI-SEND. For a DCA action, this ID is a NULL-ID. When INVOKE sees the null ACTION-ID, it decodes the communicate's verb and completes the invocation of the verb action specified. This procedure minimizes the action time devoted to a possible invalid verb type in the MCP.

Action Level Interfaces Within DCA

MCP Task State - Suspend, Reinstate

Frequently, DCA actions must wait until some external event occurs (RECEIVE a message), or until a system resource deficiency has been resolved (MESSAGE SPACE AVAILABLE, COUNT/LIMIT, and so on). To ensure that no two DCA actions either try to resolve or check for reso-

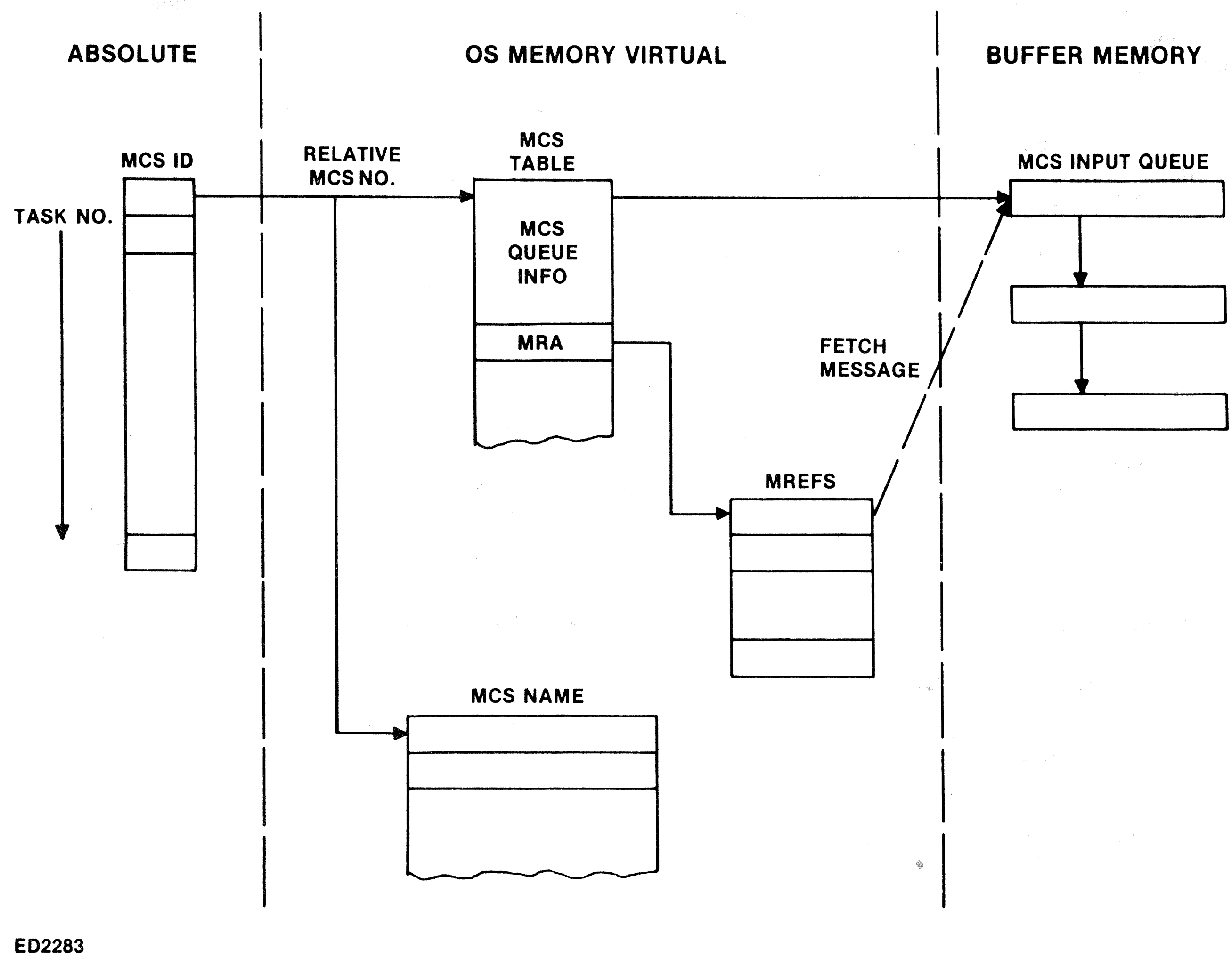


Figure 12-8. MCS Tables

lution of a waited condition, DCA enqueues actions to one of the following queues, each of which reflects a pending state:

DC-WTG-INPUT-Q	=>	For actions concerned with the presence of a message on the MCS queue
DC-WTG-COUNT-Q	=>	For actions concerned with the number of messages a task may send a station
DC-WTG-SPACE-Q	=>	For actions concerned with the presence of message space within the DCA
DC-WTG-RCV-Q	=>	For actions concerned with the presence of a message on a Subnet Queue
DC-WTG-OUTPUT-Q	=>	For actions concerned with the number of messages a task may send to the MCS
DC-LOAD-EOJ-Q	=>	To ensure that the LOAD and EOJ do not run simultaneously
DC-SYSRECON-Q	=>	To prevent any LINE ACCESS while system reconfiguration is in progress

If the condition in question requires the DCA action to wait, the DCA action sets a task suspension flag for the task that initiated the action. The action then yields the processor, anticipating another DCA action that will resolve the condition for which it is WAITed.

When an ICP directs DCA to restore a system resource, or receives notification that the required external event has occurred, the DCA action:

1. Enqueues itself to the queue corresponding to the condition it must resolve, and
2. Checks for any task WAITed on that condition.

If a task has been waited, the DCA action then:

1. Changes the state of the flag(s), indicating resolution of the condition causing the wait.
2. Determines the particular action suspended while executing the previously waited task.
3. Reinstates that action.

Figure 12-9 shows the pending states within the DCA and the associated actions.

DCA Accesses to DCP Tables

DCA uses the LSN conversion table to determine station table addresses and an LLN conversion table to determine line table bases.

CP 9500 data comm maintains the station ATTACH and WAIT status in two 32-bit fields in the DCA tables (versus the station table as defined for B 776). The output routing indicator (run mode bits in B 776) is maintained as a byte field in the DCA tables.

Disallow Input, Disallow Output

If a task is suspended waiting input from a subnet queue and the MCS issues a DISALLOW INPUT for that task and subnet queue, the task is reinstated when the DISALLOW is done.

If a task is suspended waiting for an output count to decrement when a DISALLOW OUTPUT communicate is issued, the task is reinstated when the DISALLOW is done.

Route Output

If output routing is changed while a SEND communicate is suspended waiting for message space, the message is sent to the previous routing after space is obtained.

Station Routing

If routing for a station is changed from a subnet to the MCS (ROUTE INPUT, DETACH STA, RELEASE STA), a task waiting input from that subnet is not reinstated even if this was the last station routed to the subnet. The MCS must either detach the task from the subnet (DISALLOW INPUT) or route another station's input to that subnet (ROUTE INPUT).

State	Set By	Suspends	Reset By	Reinstates
WAITING OUTPUT	SEND	TASK	CONTINUE.TASK SET.OUTPUT.LIMIT	TASK-ID FROM CPA
WAITING SPACE	SEND COMMON-HDR ENABLES/ DISABLE ATTACH-Q	TASK	RELEASE BUFFS CLEAR FETCH.MSG GET.MSG.SPACE DISALLOW.OUTPUT	ALL TASKS WAITING (TASK-ID)
WAITING ATTACH	ATTACH-Q ATTACH-STA	TASK	ALLOW/DISALLOW INPUT/OUTPUT	TASK-ID
WAITING INPUT	FETCH-MSG	TASK-ID FROM MCS TABLE	Q-TO-MCS QUEUE ROUTE.INPUT ROUTE.OUTPUT RESULT FUNCTION	TASK-ID FROM MCS TABLE
WAITING COUNT	SEND	TASK	SET.QUEUE.LIMIT RESULT FUNCTION DISALLOW.OUTPUT	ALL TASKS WAITING TASK-ID
WAITING INPUT	RECEIVE	TASK	Q-TO-SUBNET QUEUE ROUTE.INPUT DISALLOW.INPUT	ALL TASKS WAITING TASK#

Figure 12-9. Pending States

Queue Count/Limit Maintenance

Unprocessed Input Count/Input

Limit Set:	Initialized to 2 by the NDL compiler SET INPUT LIMIT communicate.
Count Incremented:	By the RESULT function when an input message is queued to an MCS queue and by ROUTE.INPUT when rerouting messages from subnet to a new subnet/MCS queue.
Count Decremented:	CONTINUE STATION communicate and by ROUTE.INPUT when rerouting input messages from MCS to a subnet queue.
Location:	Station table.
Checked:	By DCP during execution of GETSPACE function.

Subnet Queue Count/Limit

Limit Set:	Initialized to 2 by the NDL compiler SET QUEUE LIMIT communicate.
Count Incremented:	When the result function queues to a subnet queue. When an MCS reroutes messages to a subnet queue. When an MCS queues a message to the subnet queue.
Count Decremented:	Take from subnet queue (RECEIVE, DEQUEUE). When an MCS reroutes from a subnet queue.
Location:	Buffer memory.
Checked:	By DCP during execution of GETSPACE function.

Station Queue Count/Limit

Limit Set:	Initialized by 2 by the NDL compiler SET QUEUE LIMIT communicate.
Count Incremented:	When an output message is queued to NDL by either an MCS or data comm user job.
Count Decremented:	RESULT function processes an output message.
Location:	Station table.
Checked:	Task sent to a station.

Task Output Count/Limit

Limit Set:	Initialized to 2 by the NDL compiler SET OUTPUT LIMIT communicate.
Count Incremented:	DC user job SEND to MCS.
Count Decremented:	CONTINUE TASK Communicate When an output message is rerouted from the MCS Queue to a station queue via the ROUTE.OUTPUT communicate.
Location:	User Jobs Table (DCA)
Checked:	Task sent to MCS.

DCA/DCP Communication

There are two main areas of communication when the DCP interfaces with the DCA:

1. Message Communication.
2. NDL Table Access.

Message Communication

Queues provide the message interface between the DCA and the DCP. These queues are the request queue for messages to the DCP, and the result queue for messages to the DCA. These queues are briefly described and are thoroughly discussed under "Buffer Management."

Request Queue

Request queue messages are located in buffer memory. There is a request queue for each DCP. To link message addresses to a particular DCP's request queue, DCA proceeds as follows:

1. Lock the queue.
2. Bottom-link the message to any existing messages.
3. Change the top and bottom queue pointers if no previous messages were present.
4. Unlock the request queue.

The lock word and the head and tail of the request queue reside in remote memory.

The messages passed to the DCP are those defined for the CMS data comm subsystem with the addition of a DCA/DCP marker type (type 26) message.

When the DCA transfers a message to a DCP request queue, certain fields are initialized for use by the DCP. The message type determines the message header fields to be initialized. For station type functions:

1. Output.
2. Priority output.
3. Enable input.
4. Disable input.
5. Make station ready/not ready.

The following fields are initialized:

1. LLN - Used by DCP to find the base of the line information area.
2. LSN Most Byte - Initialized to the RSN from the station table. The DCP does not have access to the LSN conversion table (in OS memory); therefore, the RSN provides an easier method to access the station table (in DCP memory). Once the station table has been located, the DCP replaces the LSN field.
3. RESULT - Set to zero.
4. RESERVED - Set to the relative MCS number of the issuing MCS.

For the line oriented functions:

1. Make line ready/not ready.
2. Dialout.
3. LLN.
4. RESULT.
5. RESERVED.

are initialized in the manner described above.

Result Queue

There is a single result queue for all DCPs.

Result queue messages are located in buffer memory. To place the addresses of messages sent to the DCA into the result queue, the DCP proceeds as follows:

1. Lock the queue.
2. Insert this DCP's ID in the result queue's processor-ID field.
3. Bottom-link the messages to any existing messages.
4. Change the pointer at the top of queue if no messages were present.
5. Unlock the link.
6. Notify DCA that a message is on the result queue by issuing an interrupt (using processor interface).

The lock word, the head and tail, and the processor-ID of the queue are located in buffer memory.

All message headers placed in the result queue contain an LLN and an LSN in the appropriate fields.

Available Buffer Pool Queue

The available buffer pool queue maintains available buffer space. Both the DCP and DCA use and return space from this queue as needed. The ABP queue is located in buffer memory together with its lock word, head, tail, and buffer count.

NDL Table Accessing

DCL places all line and station table addresses in DCA memory, and updates them during reconfiguration. The DCP maintains these tables. When a CMS interrogate requests DCA to retrieve table information from a DCP, DCA reads the data item directly from DCP memory.

DATA COMM PROCESSORS (DCPS)

When loaded by DCL, each DCP is responsible for:

1. Host Control.
2. Message/Buffer Handling.
3. Line Management.
4. DCP Table Maintenance.
5. NDL S-Op Handling.
6. RCV/XMIT Character Handling.
7. Subroutines supporting S-Ops, Manager, and Host Control.

The logical flow of a DCP, on the most general level, consists of a line manager which constantly rotates control from one line to the next (see figure 12-10).

As the figures show, a round-robin and a top-down scheme are used for line switching. The co ordination of these two schemes are described in detail under "Line Management".

Line rotation begins when DCL starts-up the DCP and continues until either the DCP fails or the MCS goes to EOJ.

Control changes from one line to the next only after the currently executing line discipline has performed all actions required and/or allowed by its NDL specifications.

During the rotation cycle, line manager treats host control as a line. Host control handles all communicates from DCA via the request queue. In a DCP handling <n> lines, host control is given control after line <n>. When host control has finished, control is passed to line 1.

Host Control

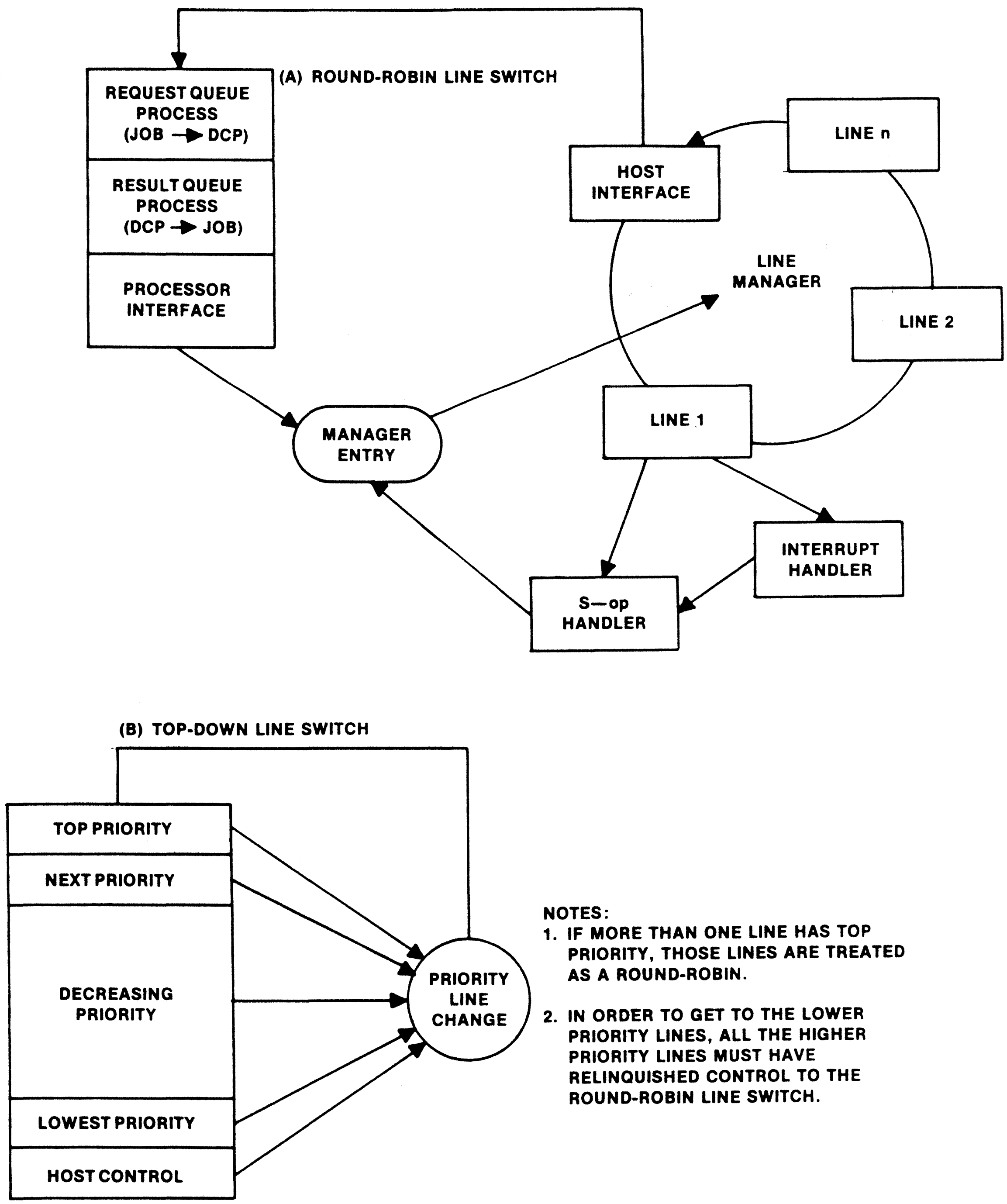
Each time host control is entered, a test is performed to determine if host control can run; that is, host control can function only every <n> times that it receives control. If host control can run, it performs the following functions in the order given:

1. A single message is dequeued from the request queue, if the queue is not empty.
   a. If this is an output, priority output, or enable/disable input message, it is placed on the appropriate station queue.
   b. Any other message types (for example, make line ready) are actioned immediately by the DCP.
2. Messages on the pseudo-result queue in DCP memory, if any have accumulated, are enqueued to the result queue proper. If the result queue was previously empty, an interrupt is sent to the result function in DCA by PI.

NOTE
Each time host control executes, only one of the above functions is performed.

Execution In An Idled System

In an environment where no lines are active (ready) the only active process is host control. A more detailed description of the functions of host control is provided in the paragraphs that follow.



NOTES:

1. IF MORE THAN ONE LINE HAS TOP PRIORITY, THOSE LINES ARE TREATED AS A ROUND-ROBIN.
2. IN ORDER TO GET TO THE LOWER PRIORITY LINES, ALL THE HIGHER PRIORITY LINES MUST HAVE RELINQUISHED CONTROL TO THE ROUND-ROBIN LINE SWITCH.

ED2285

Figure 12-10. DCP Logical Flow (Multi-Line)

DCP Queue Accessing

Figure 12-11 shows the pointers used by the DCP to access the various system queues. These pointers are either direct or indirect. Indirect pointers are initialized by the DCL. Direct pointers are used for queues solely maintained by the DCP. They are initialized by the DCP the first time it links an item into the queue. The only exception to this is the subnet queue. This indirect pointer is initialized by the route input function when a particular station's input is routed directly into a subnet queue. The DCP never accesses items within a subnet queue; it merely uses the subnet table to examine queue count and limit fields.

Request Function

The request function is responsible for the following:

1. Delinking messages from the request queue.
2. Decoding LLN and setting up L (multi-line only).
3. Decoding type and jumping to line relative function.
4. Resolving DCP result queue.
5. Executing PI.

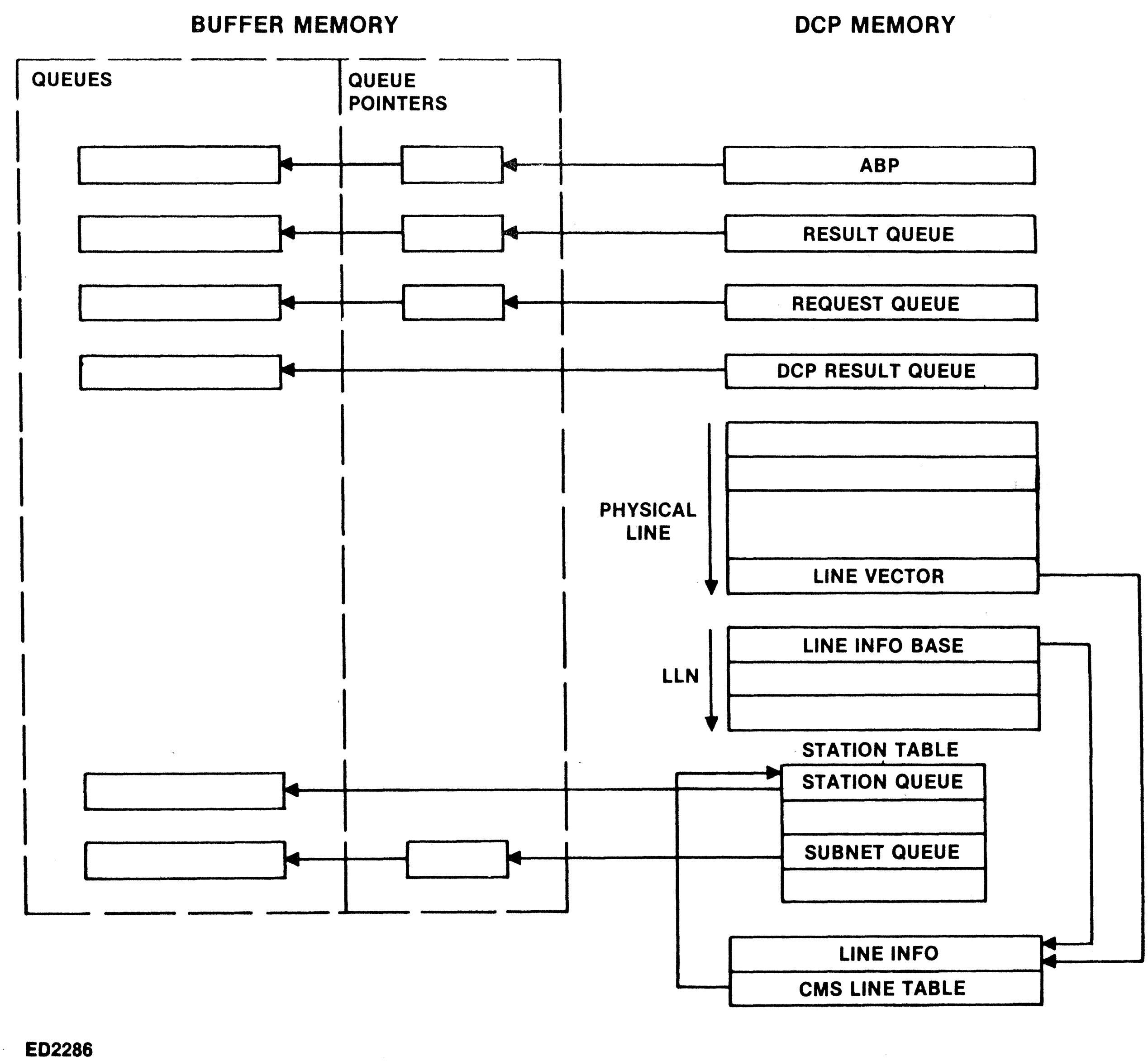


Figure 12-11. DCP Table and Queue Access

Request Queue Delinking

In the single-line case, the request function examines the line relative FUNCTION-IN-PROGRESS bit. If set, the request function cannot remove any messages from the request queue and thus goes to LABEL.RESOLVE.RES.Q. In the multi-line system, the request function retrieves the LLN from the message header and sets up L to point to the data comm line's line information area. The request function then examines the FUNCTION-IN-PROGRESS bit and if reset, delinks the message from the request queue. Otherwise, this function unlocks the request queue and goes to LABEL.RESOLVE.RES.Q. Before returning to the DCP local memory, the request function reads various fields in the message header and saves them in machine registers for use in later operations.

LLN Decoding and L Set-Up

The request function uses the LLN from the message header to index into the line information base address table (multi-line only). The address of the indicated line's line information area base address is placed into L.

Type Decode

The request function uses the message header type to index into a branch table to find the address of the code to perform the required function for this type. The types are as follows:

Input
Output
Priority Output
Enable Input
Disable Input
Make Station Ready
Make Station Not Ready
Make Line Ready
Make Line Not Ready
Dialout
Make Line Not Ready Immediate
Recover
Deallocate
Reconfigure/Reload Marker

If any type other than those indicated is placed on the result queue, the request function branches to the reconfigure/reload marker code to respond.

DCP Result Queue

The RESOLVE RESULT QUEUE routine is executed whenever there is no action required on the request queue. The DCP result queue eliminates the possibility of being locked out of the DCP/DCA result queue. Any NPC function that calls LINK.RESULT.QUEUE causes its message to be linked to a pseudo result queue, whose pointers are located in reserved memory. The request function examines the DCP result queue for entries and if it is non-empty, the request function attempts to link these messages to the DCP/DCA result queue. If the DCP/DCA result queue is locked, the request function gives control to the DCP PI code. The pseudo-result queue requires additional overhead but does not impact system throughput.

PI

The major function of PI within the DCP is to cause the DCA result function to be invoked when the system result queue becomes non null, that is, when the DCP adds a message to a previously empty queue. The DCP performs this function by creating and sending a mail box to the OS processor. PI's other function with the DCP is to receive ownership of this mail box when MCP returns it.

Line and Station Relative Functions

The following are line relative functions. When encountered by the host control, they are actioned immediately.

Make line ready/not ready/immediate not ready
Dialout
Recover
Deallocate

Make station ready/not ready are considered to be line relative because it would be impossible to action them as station relative functions. A station relative function can only be actioned for a "ready" station.

Output
Priority output
Enable/disable input

are station relative and are queued to the relevant station queue by host control. Output and priority output are actioned by the NDL transmit request, enable/disable input by line management.

Discarding Message Space

The DCP discards message space by setting the message header type of the message space to 27 and linking the message to the result queue. The DCA result function decodes the message header type and returns the space to the available buffer pool. By this method the result function may cause those tions waiting on space to be re instated.

Message Header Transfers from DCP

For input messages or function results, the DCP places the LSN or LLN into the correct portion of the header. Function headers taken from the request queue are similarly adjusted (replacing LLN or LSN) by the DCP request queue handler before functions are placed into a station queue. Therefore, when a recall is performed, the DCA result function need not read the station table for the LSN of each message in the station queue.

The transmission numbers in the header are decimal numbers found in the first three digits of the transmission number field. This field is initialized by the DCP and converted to ASCII by the result function for those headers in which the numbers have been stored.

When the DCP performs a recall, it places the station queue head pointer into the OPTIONS/EVENTS field of the recall header. The DCA result function places the station queue messages in the MCS queue after the recall result has been serviced from the result queue. Each header from the station queue is given a result field of "RECALLED" and if the message is an output message, the station queue count is decremented for that message. If a recalled output message is from a DCP that is now dead, the station queue count is not decremented. If, at a later time, the DCP recovers, there may be a problem with inconsistent counts. The DCP cannot clear the station queue count when a recall is performed because there is no READ W/LOCK for the count field; the DCA normally maintains that count.

The DCP queue pointers are four-byte fields (station queue, hold queue). The queue pointers accessed by the DCA are four-byte fields (MCS queue, subnet queue, request queue, result queue, ABP). The DCA code provides for four-byte buffer links.

Handling Message Buffers

The buffer space for all messages is located in buffer memory(s). This section describes how message space is obtained, characters are fetched and stored, and control is given to the DCCH.

GETSPACE

The GETSPACE S-Op may be used explicitly (GETSPACE) or implicitly (RECEIVE TEXT, INITIALIZE TEXT, STORE). Each GETSPACE, whether implicit or explicit, obtains space in the remote memory for the DCP's use. The following procedure is used:

1. Either a TERMINATE S-Op or a GETSPACE S-Op obtains the necessary number of buffers, as follows:
   a. If the space is obtained in order to store text, the subroutine to perform the GETSPACE S-Op is entered with two parameters:
      1) The number of buffers required by the terminal type that executes the statement is passed in a register.
      2) The number of bytes (MAXINPUT) required by the terminal type (1's complement) is passed in LNE.MSG.HDR.MSG.LEN.
   b. If a TERMINATE obtains space to report a condition, only one buffer is required.
2. If the Available Buffer Pool (ABP) is locked (that is, in use by DCCH or another DCP) upon entry, the state of the line is saved and the line is paused. When the line regains control, it again attempts to lock the ABP.
3. The ABP is locked by accessing the ABP Read-With-Lock (RWL) word. (All available space is in the ABP.) The DCP's processor-ID is inserted into the ABP as an aid to recovery in case the particular DCP fails. After locking the ABP, the DCP checks whether there are enough buffers in the ABP to perform the GETSPACE. If not, the ABP is unlocked and the GETSPACE is aborted via the appropriate action.

   NOTE
   Three buffers are reserved in the ABP to be used for single-buffer GETSPACEs and are not accessible by the normal S-Op-type GETSPACEs.

   a. If there is not enough space when a terminate GETSPACE occurs, the ABP is unlocked; the line is paused; and when the line regains control, it tries again to obtain the space.
   b. If the ABP has enough space, the DCP updates the count of available buffers, delinks the quantity that it needs, and then unlocks the ABP.
4. Once space has been obtained, the following fields in the message header are initialized:
   a. Address (processor #/line #) with the values from the address in the line table.
   b. Logical Station Number (LSN) with the LSN of the active station.

c. Each of the following is initialized to zero:

Tally 0
Tally 1
Tally 2
Toggles
MCS Data

d. RESULT/TYPE is initialized to @0001@.

e. TASK/MCS flag is initialized to zero.

5. All DCP variables that are required to perform text storing are also initialized (see section on "Character Storing" for a list of these variables.)

Character Fetching

Individual characters are fetched from output-type message buffers. The fetch can result from an explicit FETCH statement, or implicitly through a TRANSMIT TEXT statement. Either way, the fetch is performed by a common subroutine; this subroutine is responsible for returning the next sequential character in the buffer, both in the B0 register and in the CHAR register (which is a field maintained by the NDL virtual machine).

First, the routine makes sure that text is still available in the buffer (if there is none, a value of @FF@ is returned in the B1 register; otherwise, @00@ is returned in B1).

The routine also checks whether or not the pointers to the buffers need to be updated to move into the next buffer.

Character Storing

Characters are stored individually into input-type message buffers. The store can be a result of an explicit STORE instruction, or implicitly through a RECEIVE TEXT statement. In either case, a subroutine performs the actual store. This routine is entered with the character to be stored in the B0 register or in the CHAR register. If the amount of available text space is exhausted, the subroutine is responsible for returning an indicator to the caller.

A successful store is indicated by returning 0 in the B1 register, while a value of @FF@ in B1 indicates an unsuccessful store.

Variables associated with storing of textual characters are listed below with a description of their use as it applies. These variables are located in the line info area.

LNE.BUFFER.SIZE (Two bytes)

This contains the 2's complement of the buffer-size. It is incremented each time a character is stored/fetched. If an overflow is encountered while incrementing, the current buffer is full. The variable takes on special meanings in the following cases:

1. If NO-SPACE-AVAILABLE, the LNE.BUFFER.SIZE has a value of @FFFF@.
2. On the first and last buffers of the message, the buffer-size is the 2's complement of the actual space available in that buffer for text (possibly less than the declared buffer size).

LNE.BUFFER.COUNT (Four bytes)

This contains the absolute address in the message buffer of where the next character is to be stored/fetched. It is only valid if LNE.BUFFER.SIZE is not equal to @FFFF@. The variable is incremented with each character stored/fetched.

LNE.THIS.BUFFER.SIZE (Two bytes)

This contains the true (that is, uncomplemented) value of the amount of text space available in this buffer. It is only valid if LNE.BUFFER.SIZE is not @FFFF@. It normally contains the same value as BUFFER.SIZE defined in the NDLSYS with the exception of: 1) the first buffer, at which point it contains the value of BUFFER.SIZE minus MESSAGE.HEADER.SIZE; and 2) the last buffer, at which point it contains the amount of textual character space available to equal MAX-INPUT(LNE.TEXT.SIZE FALSE).

LNE.FLAGS.2 (SPACE.AVAIL)

A flag indicating whether or not space is available.

LNE.TEXT.SIZE (Two bytes)

Contains the 1's complement of the amount of characters that can be accumulated (MAX.INPUT). As characters are stored, the value is incremented until overflow occurs, indicating END.OF.BUFFER. The value is only incremented when buffer boundaries are crossed (for example, LNE.BUFFER.SIZE overflows and space is available).

LNE.CURRENT.BUFFER (Four bytes)

Contains the base address of the current buffer.

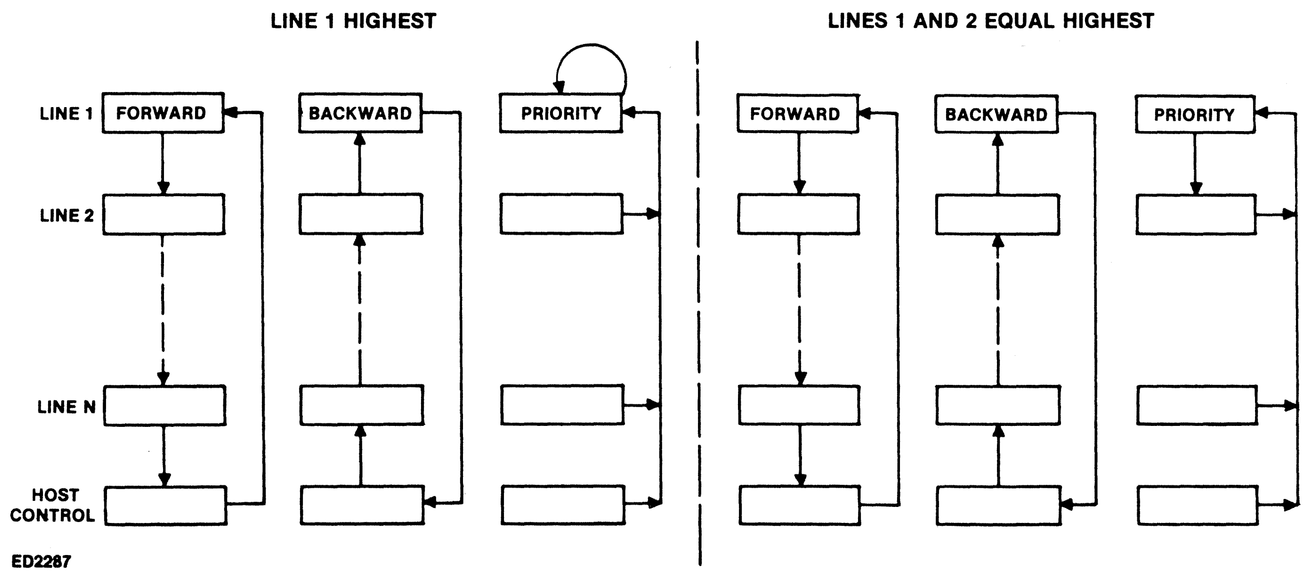


Figure 12-12. Line Linkage

Transferring Space Ownership to DCA

Message space ownership is always transferred by the DCP to the DCA. Regardless of the final destination of the space, it is placed onto the result queue and passed to the DCA. The DCA is then responsible for further routing of the space or returning it to the available buffer pool.

Line Management

Line switching is accomplished by one of two techniques: 1) a top down line change; and 2) a round robin line change. Linked lists, which connect ready lines, control both schemes. When a line is made ready, host control links the line into the round robin queue. After determining where it should fall in the top down queue, host control links the line into that queue as well.

The top down scheme is used when the line is relinguishing control but must regain control in time to service the next interrupt (for example, transmitting or receiving). Control is passed to the line referenced by the PRIORITY.POINTER field. The following factors affect a line's placement in the top down queue:

1. Priority - When a line is made ready, it is inserted in the top down queue according to the priority code in the station table for relative station 0.
2. Speed - Higher-speed lines have a higher priority and are at the top of the line queue.

NOTE

Each line in the top down queue has a pointer to the highest-priority line except when several lines all have the same priority and no ready line has a higher priority. In this case, a round robin scheme is used (at the highest priority only). (See figure 12-12.)

The round robin scheme is used when the line is relinguishing control at a time when it is not particularly busy (for example, pause or delay statements). Control is passed to the line referenced by the NEXT.POINTER field. The round robin line queue consists of forward and backward pointers.

The primary and auxiliary sides of a full-duplex line maintain their own line information area; therefore, each side is one entry in the line queue.

Host control is linked to the bottom of the round robin queue. Therefore, when no line is busy (using top down switching), host control is entered after the last line and before the first line. Because host control requires fewer variables than a data comm line, its variables are placed in a special area of reserved memory. Variables are positioned so line manager can treat host control like any other line.

At DCP initialization time, the host control function is linked to itself. As lines are made ready, the lines are linked into both queues. When a line goes not ready, it is delinked from both queues.

As a full-duplex line is made ready, the primary is linked into both queues, but the auxiliary remains inactive until the primary executes a FORK instruction. A subsequent IDLE by the auxiliary causes it to be delinked.

Single-Line Manager Schemes

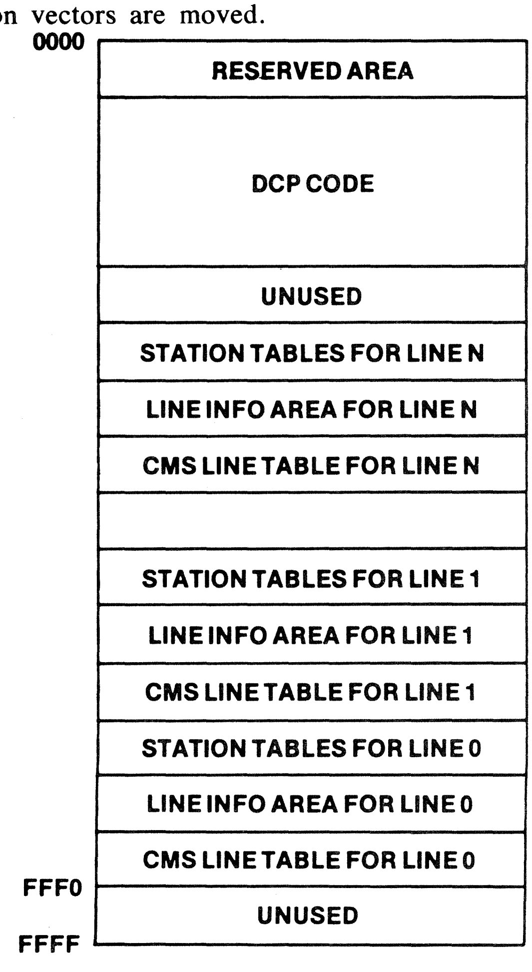
In a single-line manager, control passes back and forth between host control and the one ready line on that DCP. Interrupt handling routines return control directly to host control.

NOTE

The presence of one full-duplex line in a DCP causes NPC to generate a multi-line manager for that DCP.

DCP Table Maintenance

Figure 12-13 shows DCP memory following DCL execution. Tables are loaded starting at the high end of memory. Enough space is reserved above each line table/line information to accommodate MAX-STATION station tables. Station vectors within the CMS line table are initialized at data comm load time and point to the area reserved for the station table. Because of reconfiguration, relative station numbers may alter as stations and are attached/detached to/from a line. As station tables are accessed indirectly via the station vector, the station table should not be moved during reconfiguration. Only the station vectors are moved.



ED2288

Figure 12-13. DCP Memory

The top 15 bytes of memory are never used. By convention, the DCA always accesses remote memory with the interface control enabled. Access to address @FFF8@ through @FFFF@ have a special meaning in this mode; @FFF0@ is chosen as a convenient upper limit for DCP memory.

Station Table

The station table in DCP memory is similar to the CMS-defined station table. The size of the station table is the same for all stations on a system. Consequently, any station that makes use of the extended tallies causes all station tables in that system to have space allocated for the extended tallies. DCL loads all station tables at MCS start-time.

The station table is usually referenced by using the K-register. This register contains the base address of the active station for the currently executing line.

When the active station changes, it is validated. Then, the routine that changes the station number updates the K-register to point to the new station table. If an INVALID STATION occurs (STATION is greater than or equal to MAX.STATIONS), the pointers are set to a dummy station table in reserved memory. Figure 12-14 shows the layout of the station table as loaded into DCP memory.

Line Table

The DCP requires more line-relative data than is available in the CMS-defined line table. Therefore, a line information area prefixes the CMS-defined line table. Each CMS-defined line table consists of 16 bytes plus four bytes for each possible station that can be attached to the line. The maximum number of attached stations cannot be greater than MAX.ENTRIES.

The line table and associated line information area are always referenced using the L-register. The L-register points to the base of the line information area of the currently executing line. The line table for that line is appended to the end of the line information area. The L-register is replaced with the address of the base of the line area information area when the manager switches lines. The address of the line area is located in the previous line's line information area.

The sequence to change the L-register to point to the next line appears as:

M1 ← L + NEXT.LINE.POINTER %previous line
L ← I1 %new line

Offset		
0	LLN	RSN
2	END CHARACTER	LINE DELETE CHARACTER
4	BACKSPACE CHARACTER	WRU CHARACTER
6	CONTROL CHARACTER	STATION FREQUENCY
8	XMT ADDRESS-2	XMT ADDRESS-1
10	RUN MODE BITS	XMT ADDRESS-3
12	RCV ADDRESS-2	RCV ADDRESS-1
14	RESERVED	RCV ADDRESS-3
16	RCV TRANSMISSION NO.	SAVE Q HEAD PAGE
18	XMT TRANSMISSION NO.	SAVE Q HEAD ADDRESS
20	L S N	
22	UNPROCESSED INPUT LIMIT	UNPROCESSED INPUT COUNT
24	ORIGINAL RETRY	RETRY
26	TALLY (1)	TALLY (0)
28	TALLY (2)	TOGGLES (7 - 0)
30	OPTIONS	EVENTS (BYTE 1)
32	EVENTS (BYTES 2-3)	
34	INITIATE RCV DELAY	
36	ACTIVE XMT DELAY	
38	OUTPUT SAVE Q TAIL PAGE	
40	OUTPUT SAVE Q TAIL ADDRESS	
42	STATION Q LIMIT	STATION Q COUNT
44	RESERVED	RESERVED
46	SUBNET QUEUE PAGE	
48	SUBNET QUEUE ADDRESS	
50	MCS ID	LINE PRIORITY CODE
52	T Y P E	
54	S P E E D	
56	MODEM	TERMINAL
58	STA Q HEAD PAGE	
60	STA Q HEAD ADDRESS	
62	STA Q TAIL PAGE	
64	STA Q TAIL ADDRESS	
66	TALLY (4,3,6,5,8,7)	
72	TALLY (10,9,12,11,14,13)	
78	TALLY (16,15,18,17)	
82	OUTPUT SAVE Q COUNT	INPUT SAVE Q COUNT
84	INPUT SAVE_Q HEAD PAGE	
86	INPUT SAVE_Q HEAD ADDRESS	
88	INPUT SAVE_Q TAIL PAGE	
90	INPUT SAVE_Q TAIL ADDRESS	

ED2289

Figure 12-14. Station Table

The NPC sets a bit on a file basis which indicates the possibility of full-duplex on this file; that is, FD.POSSIBLE := G.TERB (*FULL.DUPLEX.TYPE) and G.LINEB (*LINE.FDX). This means that there is at least one full-duplex line on the DCP and at least one full-duplex terminal on the file to activate full-duplex logic.

Each full-duplex line requires the existence of line information areas for each half of the line but only one line table. To accommodate this requirement, the line information areas and line tables for the full-duplex operation are aligned as follows:

1. The line table is appended to the primary's line information area; that is, as in the half-duplex case.
2. The line information area for the auxiliary exists in memory, but cannot be conveniently located to make a direct line table access.
3. Each line information area contains two fields:

   a. LNE.PRI.PTR - Points to the base of the primary's line information area in full-duplex. Points to the base of its own line information area in half-duplex.
   b. LNE.CO.LINE.PTR = Points to the base of the co-line's line information area in full-duplex. Contains a value of null (@FFFF@) in half-duplex.

When L is not guaranteed to be addressing a primary or a half-duplex line, any access to the line table must be indirect. The indirect access is accomplished by using the LNE.PRI.PTR plus <desired offset> as an index into memory.

If FD.POSSIBLE = FALSE, all lines are half-duplex. A macro generation of the code required to access the full-duplex line table is invoked by calling:

INDIRECT.ACCESS (L.DISP, LIT.NM)

Translation Table Space Allocation

The space allocated for translation tables is a maximum of 512 bytes per table. If two or more terminal types require the same translation table, only one copy of the table is required. The memory allocation calculation is shown below.

NT *512 = maximum size (in bytes) for translation tables.

Where NT equals the number of terminals requiring different translation tables.

To fill translate table space, NPC:

1. Splits the tables, retrieved from NDLSYS, into a RCV table and an XMT table for each unique terminal type requiring translation.
2. Creates two separate tables from the CMS translate table.

The above translation table arrangement enables increased speed when translating characters, and simplifies loading the translation table into the line adaptors which have translation capability.

Since NPC places the translate tables into the code file, it knows the absolute address of the placement. Consequently, any references to translate tables are direct.

Terminal and Modem Tables

NPC generates in-line code for the DCP from information located in the terminal and modem tables. This eliminates the need to maintain these DCP tables.

NDL S-Op Handling

The following conventions are observed when handling NDL S-Ops:

1. Each S-Op in the control and request sets has been converted to microcode by the NDL Post Compiler (NPC).
2. Each S-instruction's microcode is treated as an independent unit. When S-instruction microcode is entered, the values contained in a specific register (other than J, K, L) are unknown. The S-instruction code may pass and receive values to or from subroutines. Subsequently, however, the S-instruction has no access to information that had been in the machine registers.
3. The K and L-registers are loaded with their respective values at line switch time (MULTI-.LINE). Thus, all execution on that line can make use of the values in K and L-register. Whenever the active station is altered, the K-register is altered to contain the new station table address. When host control is activated (MULTI.LINE), the L-register points to the base of its work area. This work area is similar to a line table. Because the single-line mode would require restoration of K before returning to the line, the K-register is not used by host control.

Register Conventions

The NDL process uses certain conventions to index into the tables located in DCP memory and to execute the S-Ops. Succeeding paragraphs describe these mechanisms.

L	The L-register (two bytes) contains the address of the line table information area of the line currently being actioned.
K	The K-register (two bytes) contains the address of the station table for the current active station of the line being actioned.
J	The J-register contains a value of zero.
M1	The M1 register is used to point to the area of memory being referenced. It is not maintained either from routine to routine or from S-Op to S-Op. Consequently, the executing routine must set-up this register.
M2	The M2 register is used to point to the area of memory being referenced. It is maintained neither from routine to routine nor from S-Op to S-Op. Consequently, the executing routine must set-up this register.
B1	The B1 register tests bits; namely, the following: 1. System flags (space available, line control, output, and so on.) 2. System status. 3. Toggles. This register is also a work register when needed.
B0	The B0 register is a work register; for example, some byte-variable S-Ops load the byte variable in question into B0.
MXA	This register is used for paging to remote memory modules.
MXB	This register is used for paging to remote memory modules.
MAX	The routine that alters this register must always restore its contents to the value of the local DCP memory page.
WR,B32	These are general-purpose work registers (each is two bytes long).

RCV/XMIT Character Handling (Interrupt Handling)

All interrupts from lines are "soft" interrupts. The presence of an interrupt is not detected unless the NDL discipline allows its processing during the current control pass.

This discussion will cover the relationships between S-Ops and the XMIT/RCV interrupt handlers in very general terms. Note that all character handling managers are "tuned" per terminal. (Refer to figure 12-15 for general interrupt handling.)

TRANSMIT

In the NDL program a TRANSMIT S-Op is encountered. (It is assumed that for this discussion the adapter has been previously set-up for the transmit by an initiate transmit.) Assume that a TRANSMIT CHAR S-Op has been encountered:

S-Op Code

1. Some preliminary set-up.
2. If auxiliary of a full duplex line, abort.
3. Call pre-manager transmit.
4. If break, then go to break addr.

Pre-Manager Code

1. Store character to be transmitted in LNE.IN.CHAR.
2. Translate, if pertinent.
3. Generate vertical parity, if pertinent.
4. Generate horizontal parity.
5. Store translated character with parity to L-CHAR.
6. Store MANAGER.XMIT in LNE.FUNCTION.
7. Give up control to top down manager.

(The DCP is now free to execute code on behalf of other lines.)

MANAGER.XMIT Code

1. Check transmit exception (read primary status from adapter).
2. Handle any exceptions:
   a. DSR/ = abort.
   b. CTS/ = abort.
   c. Break = wait for end, return to S-Op.
4. If XMIT.REQ (that is, adapter is ready for character), then:
   a. Write L-CHAR to adapter.
   b. Return to S-Op.
   c. ELSE, give up control to round robin manager.

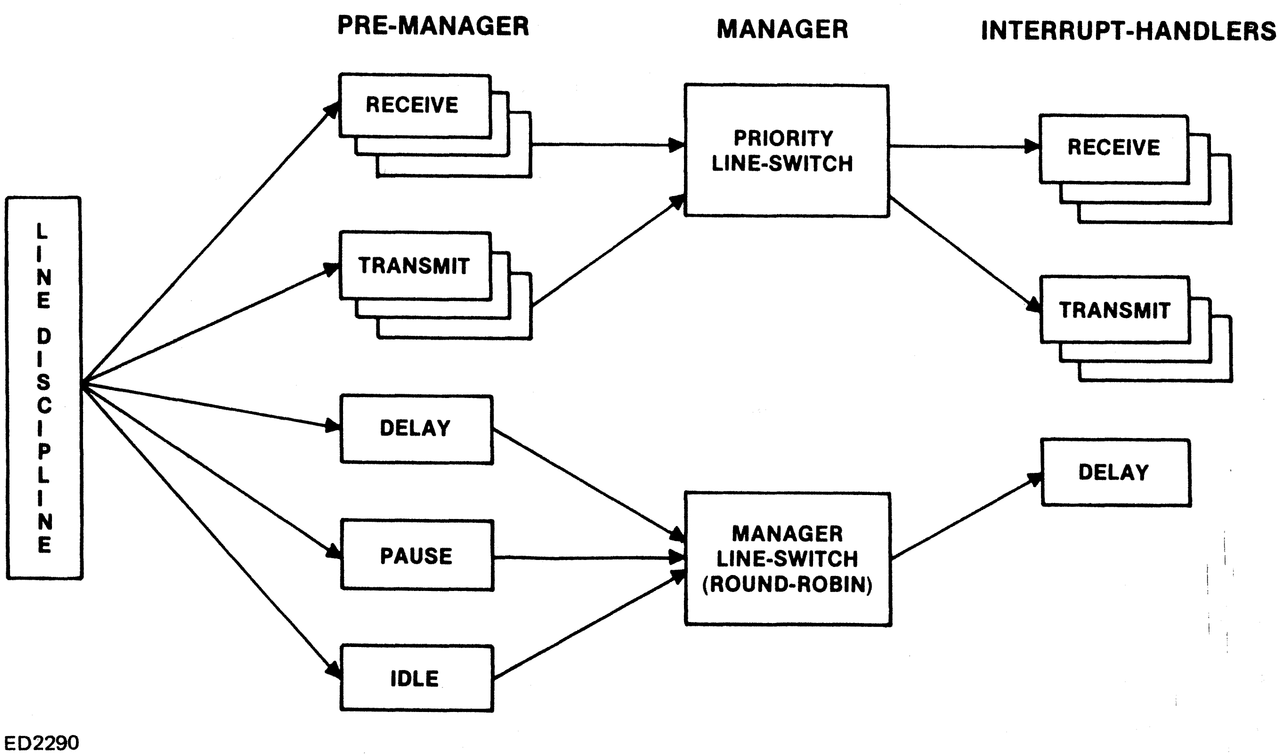


Figure 12-15. General Interrupt Handling

RECEIVE

In the NDL program a RECEIVE CHAR is encountered (assume that the adapter has been properly initialized via initiate receive).

S-Op Code

1. If primary of a full duplex line, generate timeout error and go to timeout branch of error switch.
2. Set-up timeout value
3. Call pre-manager receive.
4. If receive error, take appropriate branch of error switch.
5. If a search character is received, take appropriate branch.

Pre-Manager Code

1. Arm timer, if specified by the receive S-Op.
2. Store MANAGER.RECV.CHAR in LNE-.FUNCTION.
3. Give up control to top down manager.

(The DCP is now free to execute code on behalf of the lines.)

Manager Code

4. If RCV.EXCEPTION (contained in primary status on adapter) then:
   a. Set-up interface for error switch.
   b. Return to S-Op.
5. If RCV.REQUEST (that is, there is a character ready on adapter), then:
   a. Read character.
   b. Store character in L-CHAR.
   c. Sum horizontal parity.
   d. Strip vertical parity, if pertinent.
   e. Translate character.
   f. Store translated character in IN-HAR.
   g. Return to S-Op.
6. ELSE (that is, no character is ready on adapter):
   a. If timeout has expired:
      1) Set-up interface for error switch.
      2) Return to S-Op.
   b. If timer is still running:
      1) Give up control to round robin manager.

Subroutines Supporting S-Ops, Managers, Host Control

The NDL Post Compiler (NPC) generates subroutines on an as-needed basis. That is, if a subroutine is not needed in a code file, it is not present. Subroutines are used when the code is common to all terminals on the system, and is used frequently. Subroutines are called via the NBDS "hard call" micro instructions and exited via the "hard return." If a subroutine must yield before its function is complete, the subroutine is responsible for saving the return pointer and doing a "soft return" on exit.

BUFFER MANAGEMENT

Subsystem Queues

Host communication between the DCA, the DCP(s), and whatever MCS and user programs are present takes place through queues. The queueing process passes information and parameters between two or more logically separated routines in the DCA. Queueing causes the logical passing of data, buffers, and so on, without physically moving the data.

Each queue contains two addresses, and is stored in a reserved area of memory known and accessed by the modules associated with it. Each address indicates the physical location of a message. The first address in the queue is the "head"; it points to the next message to be removed from the queue for processing. The second address is the "tail;" it indicates the location of the last message associated with the queue.

The link mechanism connects the first and last (head and tail) messages with those between them.

Queue Linking Mechanism

Each data buffer begins with a single link address, which indicates the location of the message's next buffer. In the last buffer of a message, the link is null.

Following the buffer link, the first buffer of each message contains a message link. This indicates the first buffer of the next message.

The head address allows access to the first message; the first message's first buffer contains a link permitting access to the second, and so on.

Generally, a CP 9500 Data Comm Subsystem queue is used in the same order as it was built, (that is, on a FIFO basis). The following algorithms are used to maintain CP 9500 DCS queues:

1. Queue Linking (figure 12-16). When a message is added to the bottom of a queue, the queue's tail address is retrieved. It is replaced by the address of the new message. The old tail is used to update the message link address in the message which was previously the last; this message's previously null link address is set to point to the new end message.

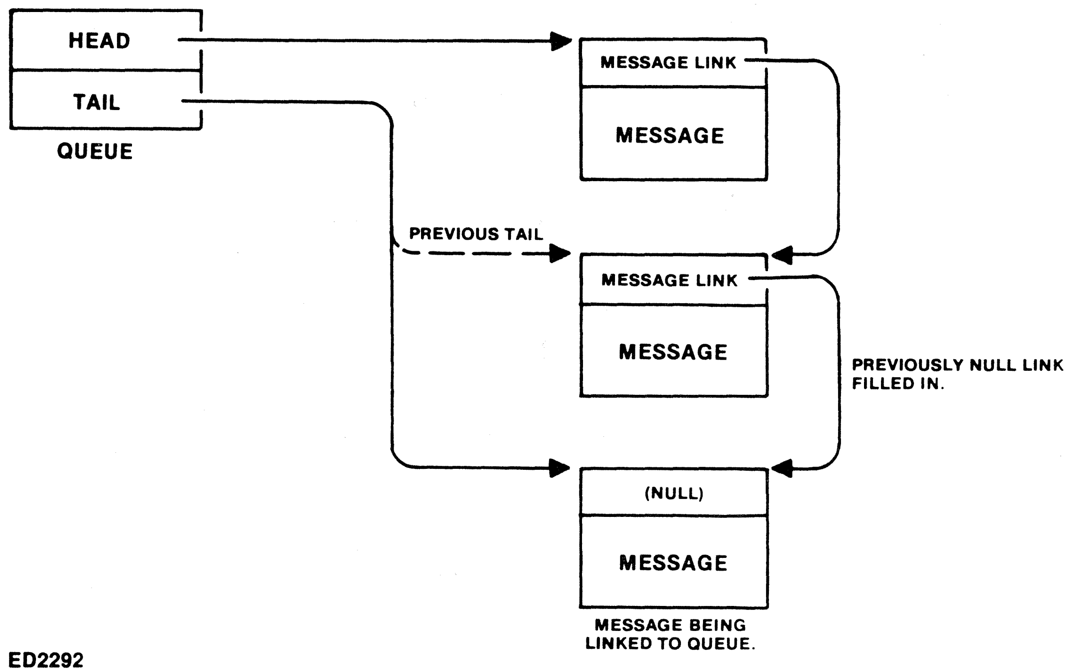


Figure 12-16. Queue Linking

2. Top Queueing (figure 12-17). In certain cases, such as when a high-priority communication contains data affecting communications queued earlier, top queueing is used. The old head address is retrieved, and replaced by the address of the new message. The old head address is then used to link the new beginning message to the former head message, which is now second.

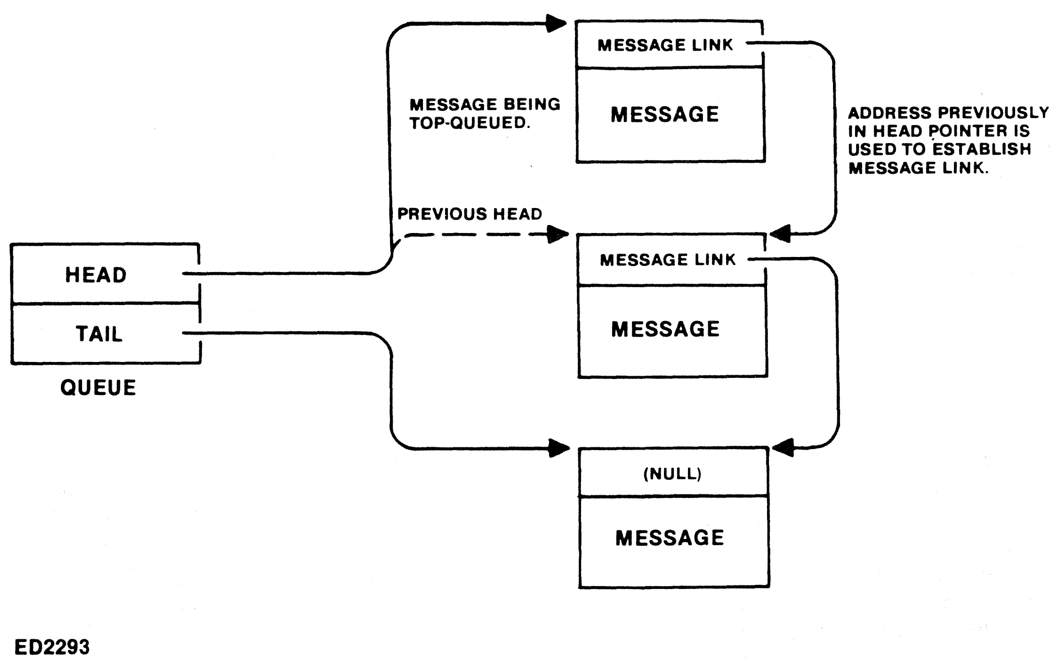
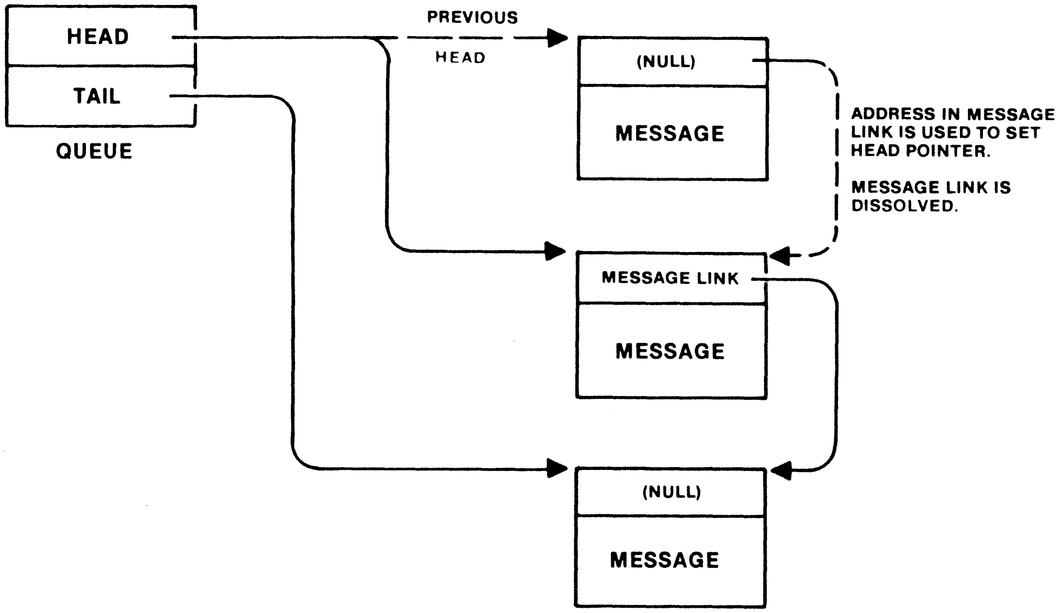


Figure 12-17. Top Queueing



ED2294

Figure 12-18. Queue Delinking

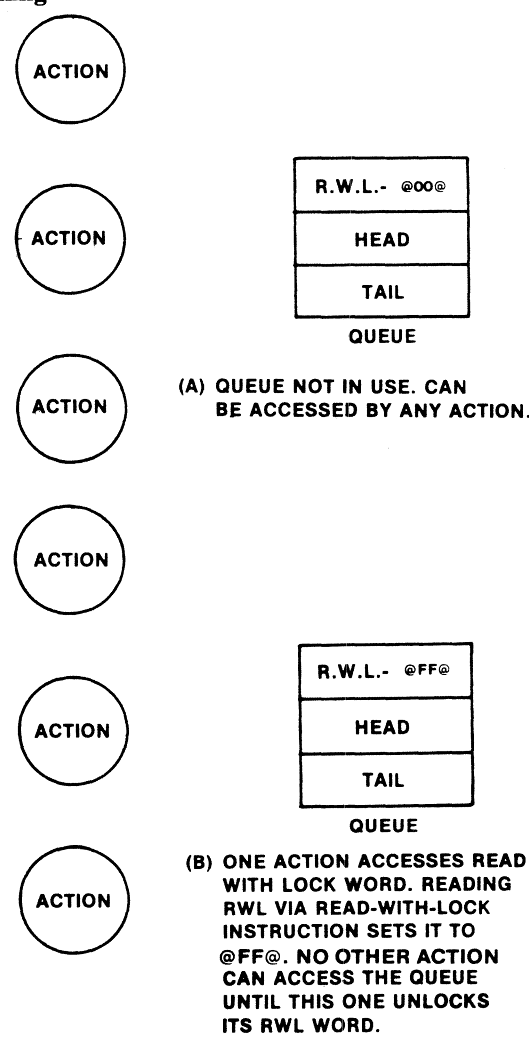
3. Queue Delinking (figure 12-18). The head address indicates the next message to be removed from the queue for processing, so it is retrieved. The message link address in the message being removed is used to replace the head address. The head has been shifted to indicate the message following the one removed.

4. Queue Lockout (figure 12-19). Queues in a multiprocessor environment must have their integrity protected. The request, result, and ABP queues each use a Read With Lock (RWL) Word.

NOTE

Any action or processor accessing the RWL words must have all interrupts disabled; it is then considered "muted."

The RWL word is subjected to the RWL hardware instruction; this reads the value, then replaces it with binary ones (@FF@) in the same clock cycle. If the value ready is @FF@, another action is using the queue, access is not allowed. If the value read is @00@, the queue can be accessed by this action only. (Any other action finds a value of @00@ to the RWL word just before terminating.)



ED2295

Figure 12-19. Queue Locking

Queue Pointers in Buffer Memory

The request, result, ABP, and subnet queues are stored in a contiguous area of reserved memory, (DC-DATA-AREA). These queues are described in the following paragraphs. Each description refers to figure 12-20 which indicates the format and size of the DC-POINTER-AREA. A description of MCS queues is also provided, but no MCS queue resides in the DC-POINTER-AREA.

Request Queues

Each DCP present on the CP 9500 has a request queue associated with it. The request queue contains messages from the DCA for that particular DCP.

Space for eight request queues is reserved in the contiguous queue storage area (DC-POINTER-AREA) in reserved memory. This corresponds to the maximum number of DCPs allowed on the CP 9500. Each of the request queues contains a RWL word, (one byte), a four-byte head address, and a four-byte tail address. Each request queue occupies nine bytes.

Result Queue

There is one result queue in the data comm subsystem. It is used by the DCPs to send messages to the DCA. Since a DCP must lock the result queue to link a message into it, the PROC-ID byte is set to identify which processor is using the queue. This aids in recovery if the DCP fails before releasing the result queue.

The DCA does not concern itself with setting the PROC-ID byte. The DCA resides in the operating system processor. Recovery from an MCP failure is assumed impossible.

The result queue occupies ten bytes: one for the RWL word, one for the PROC-ID, four for the head address, and four for the tail.

Available Buffer Pool (ABP)

The ABP controls the use of message space by the DCPs. When a DCP gets space for a message, it de-links the space from the ABP. Message space is deallocated (relinked to the ABP) only by the DCA.

Note that the ABP also has a RWL word, and a PROC-ID byte. Any action of processor accessing the RWL word must be muted (all interrupts disabled). The PROC-ID byte identifies the processor using the ABP; the ABP is still locked if the DCP fails.

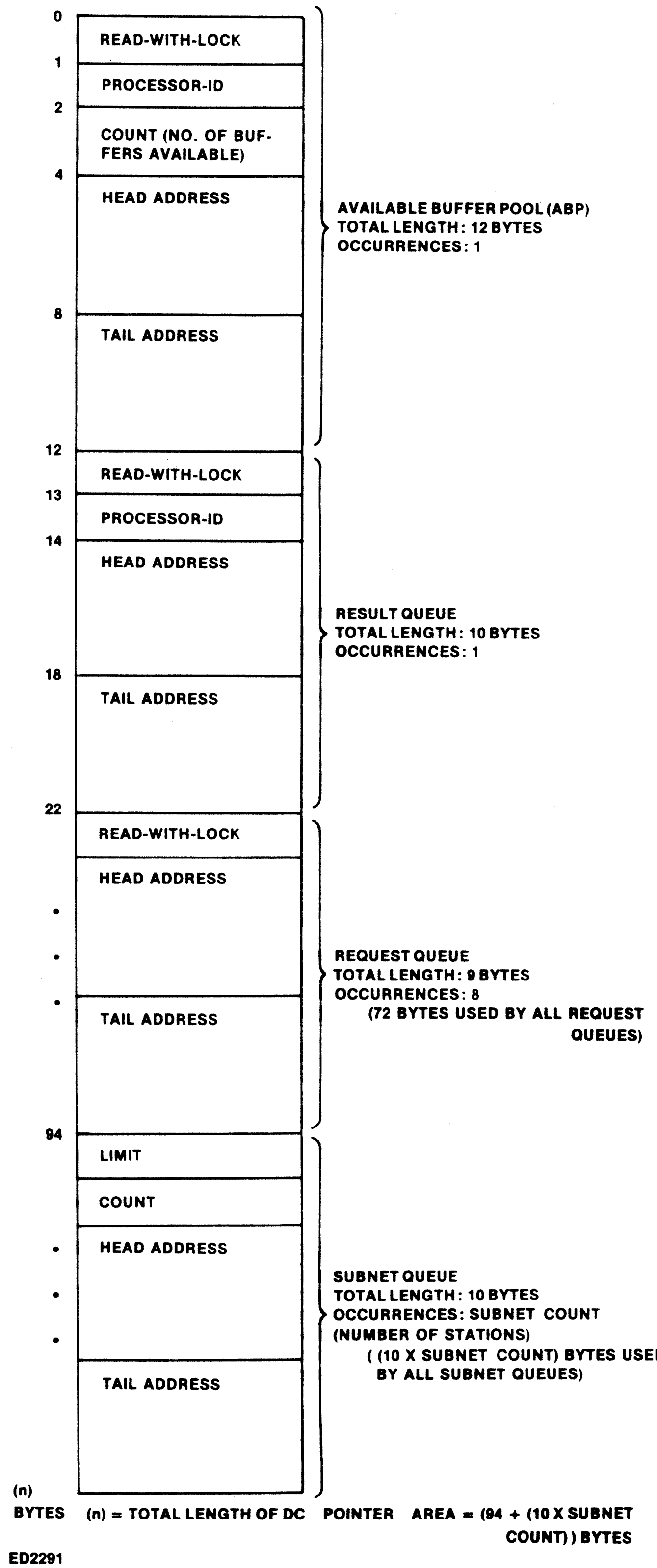


Figure 12-20. Queue Pointers in Reserved Buffer Memory

A DCP does not unlock the ABP until it has all of the space needed for a given message. Space is only available as buffers; all buffers are the same size. This BUFFER-SIZE is generated by NDL, which records it in the NDLSYS file as a count of two-byte words. The Data Comm Loader (DCL) converts the NDLSYS file's BUFFER-SIZE as follows:

1. DCL doubles the word-count given by the NDLSYS file to obtain a byte-count.

2. Since four bytes are used for each message link or buffer link in the CP 9500, while two bytes are used in the NDLSYS file, BUFFER-SIZE is incremented by four.

The converted BUFFER-SIZE, which includes both buffer and message links, is then recorded in DCA absolute memory.

Data Comm Buffer Format

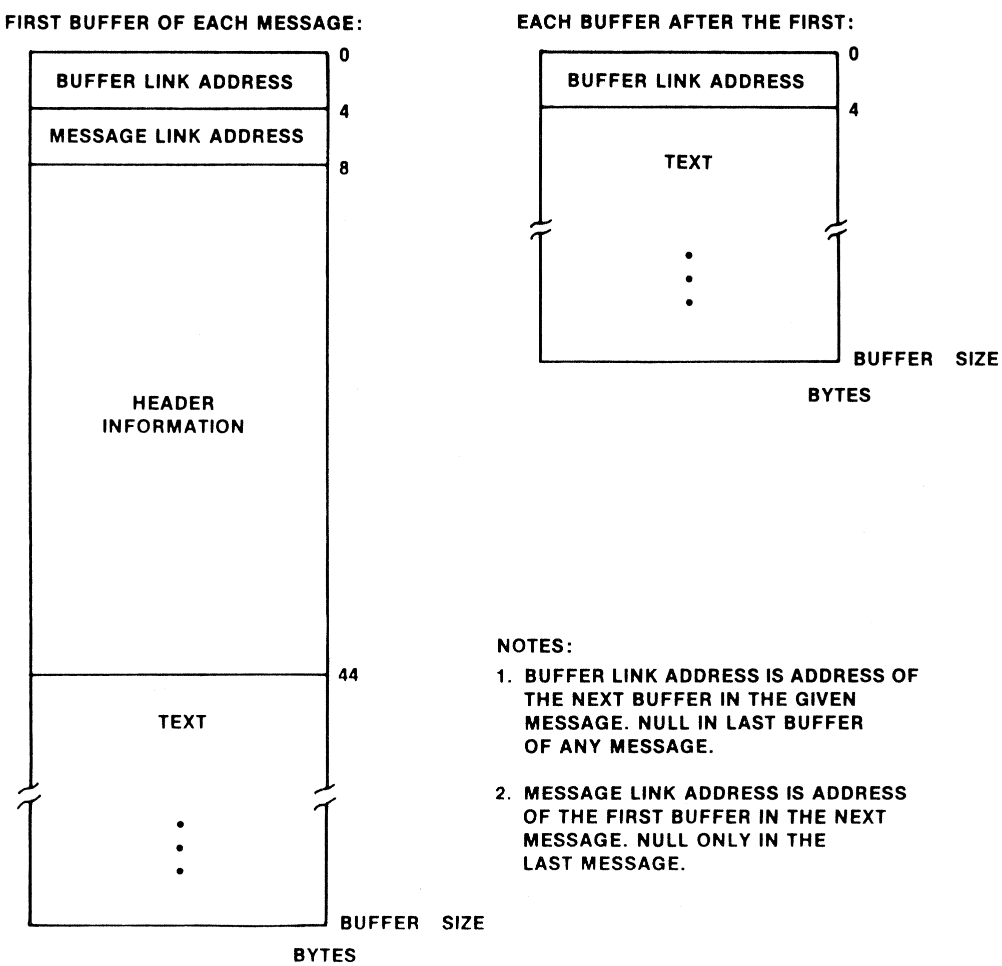
When a data comm buffer is empty (linked into the ABP), the only use for its link word is to maintain the ABP's integrity. Once the buffer is in a message (delinked from the ABP), there are two kinds of links within the message format.

The first is a link to the next buffer of the particular message. If a given buffer is last in a message, this link is null (@FFFF@).

In the first buffer of each message, the buffer link is followed by a message link. This indicates the first buffer of the next message. (See figure 12-22.) If this is the last message, the link is null (@FFFF@).

Each message's initial buffer has 36 bytes of header information following the message link.

Text occupies the remainder of each buffer after the link and/or header information is installed. (See figure 12-21.)



ED2296

Figure 12-21. Formats of Data Comm Buffers

MCS Queues

All messages sent to a given MCS program are placed on its MCS queue. This queue is maintained by the MCS to which it belongs. It does not reside in the DC-POINTER-AREA. In a multi-MCS environment, each MCS program has its own CMS queue.

Subnet Queues

Subnet queues are data comm files providing chronologically ordered messages from data comm stations (terminals) for processing by user data comm tasks.

All linking of messages to any subnet queue(s) must be performed by DCA. However, there are two ways this can occur. An MCS program, deriving its input solely from its own MCS queue, may issue a message to the DCA, directing the DCA to place a specific message on a subnet queue. Also, the DCA may route messages directly from the result queue to some subnet queue(s), bypassing the MCS program(s).

Reconfiguration

To redefine a station or line, that line must be in a "not ready" state. To make this determination, DCA creates a header (type 26) and links it to the appropriate DCP's request queue. In processing the header (type 26) the DCP returns a result of 0 if the line is in the required state; otherwise a result of 7 is returned (unable to initiate).

Station tables only remain in DCP memory while attached to a line. Whenever a station is redefined, the changes must be made to the memory copy and the copy in SYSRECON.

Data Comm Reload

To reload a DCP, that DCP must be in an "idle" state (all lines not ready). To make this determination, a header (type 26) is created by the DCA for each line defined on the DCP being reloaded and placed in the appropriate DCP request queue. When the DCP services this header, it returns a result containing an indication of whether that line on the DCP is in a "not ready" state. If all the lines on that DCP are not ready, RELOAD is permitted. A message result of 7 (unable to initiate) indicates the line is ready. A result of 0 (complete and successful) indicates the line is not ready.

Processor Interface (PI) code is embedded within each DCP codefile. At RELOAD time a number of locations must be saved and re initialized after the code overlay. Included in these locations are many of the PI variables.

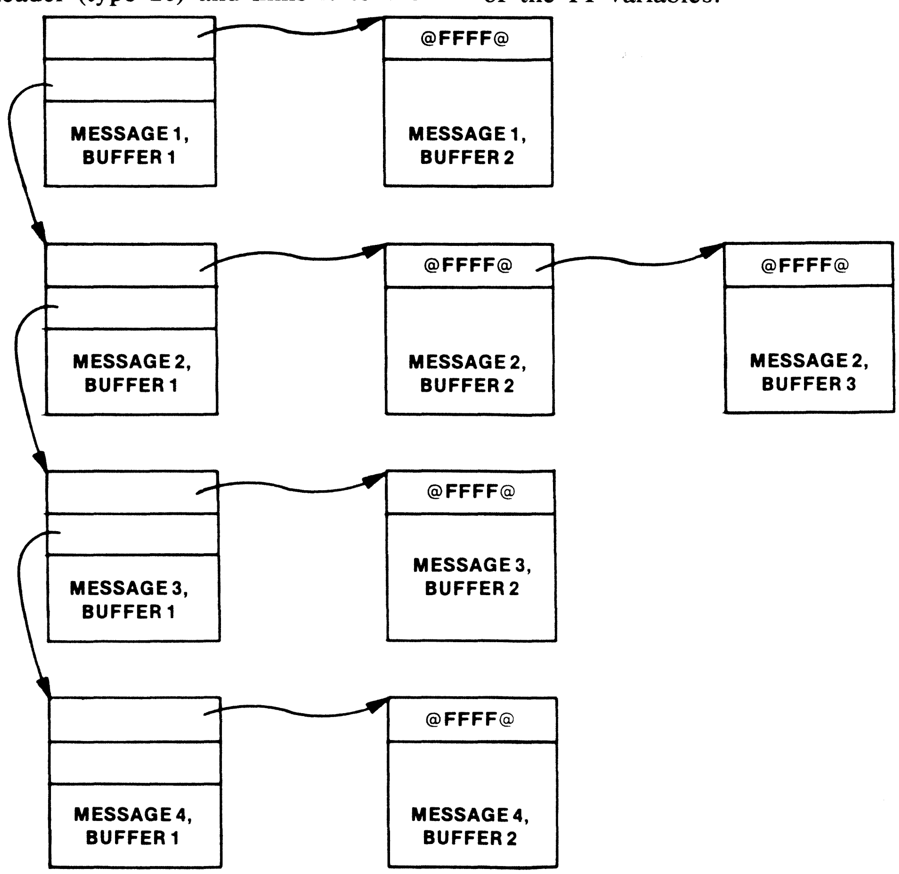


Figure 12-22. Data Comm Buffer/Message Link Mechanism

APPENDIX A DATA COMMUNICATIONS INITIATION AND TERMINATION

GENERAL

This appendix describes the initiation and orderly termination of the data communications subsystem and what is involved in these operations.

DATA COMMUNICATIONS INITIATION/TERMINATION

Initiation

When the SCL handler recognizes a request to load/execute a program that involves data comm, the following occurs:

1. If the program is an MCS and there currently is no MCS within the system, the data comm system is loaded and initialized from the program file NDLSYS, prior to the execution of the requested program (MCS). In the event that an MCS currently exists within the system, the load/execute of the requested program is aborted and the error message, LOAD FAILURE MCS ALREADY PRESENT, is displayed on the SPO.
2. If the program is not an MCS, and an MCS exists within the system, a normal load/execute of the program is performed.

Termination

When the system recognizes that a task is terminating or is to be terminated and that this task involves data comm, the following occurs:

1. If the program is an MCS, control is given to the data comm subsystem which checks tasks waited by the data comm system, and for those tasks, sets the status key in the CD area equal to 91, causing control to be returned to the task. The data comm system is removed and the indicator(s) utilized by the master communicate handler and the SCL handler, to indicate the presence of the data comm system is set/reset. The MCP can then remove the MCS.
   a. User tasks can continue or go to end-of-job (EOJ) at their discretion.
   b. Any future requests for access to the data comm system are refused. A value of 91 is set in the Status Key field of the CD area and control is immediately returned to the task. As before, the task can continue or go to end-of-job.
2. If the program is not an MCS, and the data comm system is present, control is given to the data comm system which will:
   Detach the task from subnet queues
   Detach the task from stations
   Send a message of type TASK DETACH to the MCS
   Return to the operating system for normal EOJ/DS.

If the data comm system is not present, the normal EOJ/DS is performed.

APPENDIX B

DATA COMMUNICATIONS COMMUNICATES

INTRODUCTION

This appendix describes the subset of CMS communicates relevant to data communications. This subset is known collectively as the Class D communicates.

A communicate is the process by which an S-program requests the MCP to perform a function on its behalf. Generally, these functions may be requested by multiple S-programs and manipulate data not directly accessible by the S-program. Having these functions within the MCP eliminates the need for duplication of code and also insures that the integrity of the data is maintained. The interface to the MCP is provided by the S-programs interpreter via the communicate S-Op. The format of this S-Op may vary for different languages, but because the MCP interface is common to all languages, the interpreter must present the parameters for the communicate in a fixed format. The data area used to pass parameters to the MCP is known as the communicate parameter area (CPA). The general format of the CPA is as follows:

1. Verb. Defines the type of action to be performed.
2. Adverb. Qualifies the verb and defines the specific actions.
3. Object. Describes the entity on which the action is to be performed.

The class D communicates consist of verb values @30@ through @3F@. Because of the large number of data communications functions required, the verb of a class D communicate is used to specify a general type of function; the adverb defines the actual function.

This appendix is arranged in two parts. The first defines all the class D verbs and the meaning of each adverb value within a given verb. The second defines the CPA layout of each verb/adverb pair.

Sections 7 and 9 describe COBOL and MPLII user data communications functions. Within this appendix one set of CPA layouts exist for user data comm; this being equally applicable to both COBOL and MPLII. As stated previously, the interface to a communicate is common; it is the joint responsibility of the language compiler and interpreter to provide the correct interface.

The following are the class D verb values.

Verb	Description
@30@	MCS control communicates.
@31@	MCS interrogates.
@32@	MCS redefinition.
@33@	User data comm.
@34@	MCS DCP oriented communicates.

Verb-Adverb CPA Values

Verb = 30

Communicate	Adverb
QUEUE	00
QUEUE.DEPTH	01
SET.INPUT.LIMIT	02
SET.QUEUE.LIMIT	03
EXCHANGE.REFERENCE	04
FETCH.MESSAGE	05
GET.MESSAGE.SPACE	06
RELEASE.MESSAGE.SPACE	07
READ.HEADER	08
WRITE.HEADER	09
READ.TEXT	0A
WRITE.TEXT	0B
COPY.TEXT	0C
CONTINUE.STATION	0D
CONTINUE.TASK	0E
ROUTE.INPUT	0F
ROUTE.OUTPUT	10
ALLOW.INPUT	11
DISALLOW.INPUT	12
ALLOW.OUTPUT	13
DISALLOW.OUTPUT	14
SET.OUTPUT.LIMIT	15

Verb = 31

Communicate	Adverb
LINE.COUNT	00
STATION.COUNT	01
MODEM.COUNT	02
TERMINAL.COUNT	03
SUBNET.COUNT	04
LINE.NUMBER	05
STATION.NUMBER	06
QUEUE.NUMBER	07
LINE.DESCRIPTION	08
STATION.DESCRIPTION	09
MODEM.DESCRIPTION	0A
TERMINAL.DESCRIPTION	0B
SUBNET.DESCRIPTION	0C
LINE.STATIONS	0D
SUBNET.STATIONS	0E
LINE.STATUS	0F
STATION.STATUS	10
TASK.NAME	11
TASK.NUMBER	12
RECALL	13
CLEAR	14
SUBNET.STATUS	15
TASK.STATUS	16

Verb = 32

Communicate	Adverb
REDEFINE.LINE	00
REDEFINE.STATION	01

Verb = 33

Communicate	Adverb
ENABLE.INPUT	00
DISABLE.INPUT	01
ENABLE.OUTPUT	02
DISABLE.OUTPUT	03
RECEIVE	04
SEND	05
ACCEPT	06

Verb = 34

Communicate	Adverb
DCP.RELOAD	00
DCP.PROGRAM.NAMES	01
DCP.PROGRAM.COUNT	02
DCP.DESCRIPTION	03
DCP.PROGRAM.TERMINALS	04
DCP.PROCESSORS	05

CPA Layouts

The following CPA layouts are divided in two categories:

1. Communicates which may only be invoked by an MCS program.
2. Communicates which may only be invoked by a user data comm program.

Within each category the CPA layouts are arranged in alphabetical order of function name.

MCS CPA Layouts

All functions set the most significant eight bits of FETCHVALUE equal to @00@ and the remaining sixteen bits to the "functional result." The "functional result" is defined as follows:

1. @0000@ = complete and successful.
2. All other values = CMS event number defining the error encountered.

ALLOW.INPUT

ALLOW.INPUT (<queue number>, <task number> <error option>);

Byte	Value	Meaning
0	30	Verb
1	11	Adverb = ALLOW.INPUT
2	*	Filter
3	*	Queue Number
4	*	Task Number

ALLOW.OUTPUT

ALLOW.OUTPUT (<station number>, <task number> <error option>);

Byte	Value	Meaning
0	30	Verb
1	13	Adverb = ALLOW.OUTPUT
2-3	*	Station Number
4	*	Task Number

CLEAR

CLEAR (<queue reference> <error option>);

Byte	Value	Meaning
0	31	Verb
1	14	Adverb = CLEAR
2-3	*	Queue Reference

CONTINUE.STATION

CONTINUE.STATION (<station number> <error option>);

Byte	Value	Meaning
0	30	Verb
1	0D	Adverb = CONTINUE.STATION
2-3	*	Station Number

CONTINUE.TASK

CONTINUE.TASK (<task number> <error option>);

Byte	Value	Meaning
0	30	Verb
0E	OE	Adverb = CONTINUE.TASK
2	*	Task Number

COPY.TEXT

COPY.TEXT (<message variable>, <starting byte>, <byte length> <message variable>, <starting byte> <error option>);

Byte	Value	Meaning
0	30	Verb
1	0C	Adverb = COPY.TEXT
2-3	*	Index to Message Reference
4-5	*	Starting Byte Within Text Area
6-7	*	Index to Message Reference
8-9	*	Starting Byte Within Text Area
10-11	*	Length = NUMBER OF BYTES

DCP.DESCRIPTION

DCP.DESCRIPTION (<DCP number>, <variable> <error option>);

Byte	Value	Meaning
0	34	Verb
1	03	Adverb = DCP.DESCRIPTION
2	*	DCP Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

DCP.PROCESSORS

DCP.PROCESSORS

Byte	Value	Meaning
0	34	Verb
1	05	Adverb = DCP.PROCESSORS

DCP.PROGRAM.COUNT

DCP.PROGRAM.COUNT (<DCP number>)

Byte	Value	Meaning
0	34	Verb
1	02	Adverb = DCP.PROGRAM.COUNT
2	*	DCP Number

DCP.PROGRAM.NAMES

DCP.PROGRAM.NAMES (<variable>);

Byte	Value	Meaning
0	34	Verb
1	01	Adverb = DCP.PROGRAM.NAMES
2	*	Filler
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

DCP.PROGRAM.TERMINALS

DCP.PROGRAM.TERMINALS (<DCP number>, <variable>, <program name> <error option>);

Byte	Value	Meaning
0	34	Verb
1	04	Adverb = DCP.PROGRAM.TERMINALS
2	*	DCP Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable
8	*	Filler
9	*	Segment Number of Program Name
10-11	*	Offset of Program Name
12-13	*	Size of Program Name

DCP.RELOAD

DCP.RELOAD (<DCP number>, <program name> <error option>);

Byte	Value	Meaning
0	34	Verb
1	00	Adverb = DCP.RELOAD
2	*	DCP Number
3	*	Segment Number of Program Name
4-5	*	Offset of Program Name
6-7	*	Size of Program Name

DEQUEUE

See FETCH.MESSAGE.

DISALLOW.INPUT

DISALLOW.INPUT (<queue number>, <task number> <error option>);

Byte	Value	Meaning
0	30	Verb
1	12	Adverb = DISALLOW.INPUT
2	*	Filler
3	*	Queue Number
4	*	Task Number

DISALLOW.OUTPUT

DISALLOW.OUTPUT (<station number>, <task number> <error option>);

Byte	Value	Meaning
0	30	Verb
1	14	Adverb = DISALLOW.OUTPUT
2-3	*	Station Number
4	*	Task Number

EXCHANGE.MESSAGE

EXCHANGE.REFERENCE (<message variable>, <message variable>);

Byte	Value	Meaning
0	30	Verb
1	04	Adverb = EXCHANGE.REFERENCE
2-3	*	Index to Message Reference
4-5	*	Index to Message Reference

FETCH.MESSAGE AND DEQUEUE

FETCH.MESSAGE (<message variable>, <queue reference> <wait option>);

Byte	Value	Meaning
0	30	Verb
1	05	Adverb = FETCH.MESSAGE/ DQUEUE
2-3	*	Index to Message Reference
4-5	*	Queue Reference
6	*	Wait Option
		00 = WAIT
		01 = DON'T WAIT

GET.MESSAGE.SPACE

GET.MESSAGE.SPACE (<message variable>, <byte length>);

Byte	Value	Meaning
0	30	Verb
1	06	Adverb = GET.MESSAGE.SPACE
2-3	*	Index to Message Reference
4-5	*	Length = NUMBER OF BYTES

LINE.COUNT

LINE.COUNT

Byte	Value	Meaning
0	31	Verb
1	00	Adverb = LINE.COUNT

LINE.DESCRIPTION

LINE.DESCRIPTION (<line number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	08	Adverb = LINE.DESCRIPTION
2	*	Line Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

LINE.NUMBER

LINE.NUMBER (<line address>)

Byte	Value	Meaning
0	31	Verb
1	05	Adverb = LINE.NUMBER
2-3	*	Line Address

LINE.STATIONS

LINE.STATIONS (<line number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	0D	Adverb = LINE.STATIONS
2	*	Logical Line Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

LINE.STATUS

LINE.STATUS (<line number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	0F	Adverb = LINE.STATUS
2	*	Logical Line Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

MODEM.COUNT

MODEM.COUNT

Byte	Value	Meaning
0	31	Verb
1	02	Adverb = MODEM.COUNT

MODEM.DESCRIPTION

MODEM.DESCRIPTION (<modem number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb

Byte	Value	Meaning
1	0A	Adverb = MODEM.DESCRIPTION
2	*	Modem Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

QUEUE

QUEUE (<message variable>, <queue reference> <error option>);

Byte	Value	Meaning
0	30	Verb
1	00	Adverb = QUEUE
2-3	*	Index to Message Reference
4-5	*	Queue Reference

QUEUE.DEPTH

QUEUE.DEPTH (<queue reference>)

Byte	Value	Meaning
0	30	Verb
1	01	Adverb = QUEUE.DEPTH
2-3	*	Queue Reference

QUEUE.NUMBER

QUEUE.NUMBER (<queue name>)

Byte	Value	Meaning
0	31	Verb
1	07	Adverb = QUEUE.NUMBER
2	*	Filler
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

READ.HEADER

READ.HEADER (<message variable>, <variable> <error option>);

Byte	Value	Meaning
0	30	Verb
1	08	Adverb = READ.HEADER
2-3	*	Index to Message Reference
4	*	Filler
5	*	Segment Number of Variable
6-7	*	Offset of Variable
8-9	*	Size of Variable

READ.TEXT

READ.TEXT (<message variable>, <starting byte> <byte length>, <variable> <error option>);

Byte	Value	Meaning
0	30	Verb
1	0A	Adverb = READ.TEXT

Byte	Value	Meaning
2-3	*	Index to Message Reference
4-5	*	Starting Byte Within Text Area
6-7	*	Length = NUMBER OF BYTES
8	*	Filler
9	*	Segment Number of Variable
10-11	*	Offset of Variable

RECALL

RECALL (<queue reference> <error option>);

Byte	Value	Meaning
0	31	Verb
1	13	Adverb = RECALL
2-3	*	Queue Reference

REDEFINE.LINE

REDEFINE.LINE (<line number>, <variable> <error option>);

Byte	Value	Meaning
0	32	Verb
1	00	Adverb = REDEFINE.LINE
2	*	Logical Line Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

REDEFINE.STATION

REDEFINE.STATIONS (<station number>, <variable> <error option>);

Byte	Value	meaning
0	32	Verb
1	01	Adverb = REDEFINE.STATION
2-3	*	Logical Station Number
4	*	Filler
5	*	Segment Number of Variable
6-7	*	Offset of Variable
8-9	*	Size of Variable

RELEASE.MESSAGE.SPACE

RELEASE.MESSAGE.SPACE (<message variable>);

Byte	Value	Meaning
0	30	Verb
1	07	Adverb = RELEASE.MESSAGE.SPACE
2-3	*	Index to Message Reference

ROUTE.INPUT

ROUTE.INPUT (<station number>, <queue reference> <reroute> <error option>);

Byte	Value	Meaning
0	30	Verb
1	0F	Adverb = ROUTE.INPUT
2-3	*	Station Number
4-5	*	Queue Reference
6	*	Reroute Option 00 = REROUTE 01 = DON'T REROUTE

ROUTE.OUTPUT

ROUTE.OUTPUT (<station number>, <queue reference> <error option>);

Byte	Value	Meaning
1	10	Adverb = ROUTE.OUTPUT
2-3	*	Station Number
4-5	*	Queue Reference

SET.INPUT.LIMIT

SET.INPUT.LIMIT (<station number>, <limit> <error option>);

Byte	Value	Meaning
0	30	Verb
1	02	Adverb = SET.INPUT.LIMIT
2-3	*	Station Number
4	*	Limit

SET.OUTPUT.LIMIT

SET.OUTPUT.LIMIT (<task number>, <limit> <error option>);

Byte	Value	Meaning
0	30	Verb
1	15	Adverb = SET.OUTPUT.LIMIT
2	*	Filler
3	*	Task Number
4	*	Limit

SET.QUEUE.LIMIT

SET.QUEUE.LIMIT (<queue reference>, <limit> <error option>);

Byte	Value	Meaning
0	30	Verb
1	03	Adverb = SET.QUEUE.LIMIT
2-3	*	Queue Reference
4	*	Limit

STATION.COUNT

STATION.COUNT

Byte	Value	Meaning
0	31	Verb
1	01	Adverb = STATION.COUNT

STATION.DESCRIPTION

STATION.DESCRIPTION (<station number>, <variable> <eror option>);

Byte	Value	Meaning
0	31	Verb
1	09	Adverb = STATION.DESCRIPTION
2-3	*	Station Number
4	*	Filler
5	*	Segment Number of Variable
6-7	*	Offset of Variable
8-9	*	Size of Variable

STATION.NUMBER

STATION.NUMBER (<station name>)

Byte	Value	Meaning
0	31	Verb
1	06	Adverb = STATION.NUMBER
2	*	Filler
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

STATION.STATUS

STATION.STATUS (<station number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	10	Adverb = STATION.STATUS
2-3	*	Logical Station Number
4	*	Filler
5	*	Segment Number of Variable
6-7	*	Offset of Variable
8-9	*	Size of Variable

SUBNET.COUNT

SUBNET.COUNT

Byte	Value	Meaning
0	31	Verb
1	04	Adverb = SUBNET.COUNT

SUBNET.DESCRIPTION

SUBNET.DESCRIPTION (<queue number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	0C	Adverb = SUBNET.DESCRIPTION
2	*	Queue Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

SUBNET.STATIONS

SUBNET.STATIONS (<queue number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	0E	Adverb = SUBNET.STATIONS
2	*	Queue Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

SUBNET.STATUS

SUBNET.STATUS (<queue number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	15	Adverb = SUBNET.STATUS
2	*	Queue Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

TASK.NAME

TASK.NAME (<task number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	11	Adverb = TASK.NAME
2	*	Task Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

TASK.NUMBER

TASK.NUMBER (<task name>)

Byte	Value	Meaning
0	31	Verb
1	12	Adverb = TASK.NUMBER
2	*	Filler
3	*	Segment Number of Variable
4-5	*	Offset of VAriable
6-7	*	Size of Variable

TASK.STATUS

TASK.STATUS (<task number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	16	Adverb = TASK.STATUS
2	*	Task Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

TERMINAL.COUNT

TERMINAL.COUNT

Byte	Value	Meaning
0	31	Verb
1	03	Adverb = TERMINAL.COUNT

TERMINAL.DESCRIPTION

TERMINAL.DESCRIPTION (<terminal number>, <variable> <error option>);

Byte	Value	Meaning
0	31	Verb
1	0B	Adverb = TERMINAL.DESCRIPTION
2	*	Terminal Number
3	*	Segment Number of Variable
4-5	*	Offset of Variable
6-7	*	Size of Variable

WRITE.HEADER

WRITE.HEADER (<message variable>, <variable> <error option>);

Byte	Value	Meaning
0	30	Verb
1	09	Adverb = WRITE.HEADER
2-3	*	Index to Message Reference
4	*	Filler
5	*	Segment Number of Variable
6-7	*	Offset of Variable
8-9	*	Size of Variable

WRITE.TEXT

WRITE.TEXT (<message variable>, <starting byte>, <byte length>, <variable><error option>);

Byte	Value	Meaning
0	30	Verb
1	0B	Adverb = WRITE.TEXT
2-3	*	Index to Message Reference
4-5	*	Starting Byte within Text Area
6-7	*	Length = NUMBER OF BYTES
8	*	Filler
9	*	Segment
10-11	*	Offset of Variable

ACCEPT

Byte	Value	Meaning
0	33	Verb
1	06	Adverb = ACCEPT
2	*	Filler
3	*	Segment Number of Input CD Area
4-5	*	Offset of Input CD Area

ENABLE INPUT

Byte	Value	Meaning
0	33	Verb
1	00	Adverb = ENABLE INPUT
2	*	Filler
3	*	Segment Number of Input CD Area
4-5	*	Offset of Input CD Area
6	*	Filler
7	*	Segment Number of KEY
8-9	*	Offset of KEY
10-11	*	Size of KEY

DISABLE INPUT

Byte	Value	Meaning
0	33	Verb
1	01	Adverb = DISABLE INPUT
2	*	Filler
3	*	Segment Number of Input CD Area
4-5	*	Offset of Input CD Area
6	*	Filler
7	*	Segment Number of KEY
8-9	*	Offset of KEY
10-11	*	Size of KEY

ENABLE OUTPUT

Byte	Value	Meaning
0	33	Verb
1	02	Adverb = ENABLE OUTPUT
2	*	Filler
3	*	Segment Number of Output CD Area
4-5	*	Offset of Output CD Area
6	*	Filler
7	*	Segment Number of KEY
8-9	*	Offset of KEY
10-11	*	Size of KEY

DISABLE OUTPUT

Byte	Value	Meaning
0	33	Verb
1	03	Adverb = DISABLE OUTPUT
2	*	Filler
3	*	Segment Number of Output CD Area
4-5	*	Offset of Output CD Area
6	*	Filler
7	*	Segment Number of KEY
8-9	*	Offset of KEY
10-11	*	Size of KEY

RECEIVE

Byte	Value	Meaning
0	33	Verb
1	04	Adverb = RECEIVE
	*	Adverb 2 = WAIT IF NO MESSAGE = 00; = DO NOT WAIT IF NO MESSAGE = 01
3	*	Segment Number of Input CD Area
4-5	*	Offset of Input CD Area
6	*	Filler
7	*	Segment Number of Data Area
8-9	*	Offset of Data Area
10-11	*	Size of Data Area

SEND

Byte	Value	Meaning
0	33	Verb
1	05	Adverb = SEND
2	*	Adverb 2: BIT 7 (MSB) = 0 = WAIT 1 = NOWAIT BITS 6-1 = 0 BIT 0 (LSB) = 0 = EGI 1 = EMI
3-4	*	Skip Control
5	*	Segment Number of Output CD Area
6-7	*	Offset of Output CD Area
8	*	Filler
9	*	Segment Number of Data Area
10-11	*	Offset of Data Area
12-12	*	Size of Data Area

NOTE

* WAIT is used only by MPLII.

APPENDIX C
SAMPLE CMS
DATA COMMUNICATION PROGRAMS

The following describes a model data comm system consisting of an MCS, functionally equivalent COBOL, MPLII, and RPG programs, and an NDL program. Each program is illustrated by means of a functional description followed by the program listing. This system is not intended to be used in a production environment; it is merely an example of the possible use of DC subsystem facilities.

By including these sample programs, the interface between the various levels of the DC subsystem is illustrated.

THE MODEL MCS

Functional Description

The model MCS is a slightly expanded version of the MCS published as an example in the MPLII Reference Manual. It has the following characteristics:

1. When started, the MCS participates for both input and output for all stations defined in the network.
2. Each line is made ready.
3. On receiving an input message, the MCS returns it to the sending station if no control character was used, or interprets the message as a command if a control character was used.
4. DC commands may be input from the system console.
5. If a user DC task attempts input from a subnet or output to a station, the MCS allows such input/output after having performed the appropriate ROUTE.INPUT or ROUTE.OUTPUT operations. (That is, the MCS becomes non-participating for those stations with which the user task communicates.)
6. ENABLE.INPUT, ENABLE.OUTPUT, DISABLE.INPUT, and DISABLE.OUTPUT messages have no effect on the MCS.
7. More than one DC task is supported.
8. Any one task may use one subnet at a time; that subnet must not have more than ten stations in it. (Note: This is not a system restriction but one peculiar to this MCS.)
9. When a task goes to end-of-job, the MCS again participates for the stations with which the task was communicating.

Detailed Description

Identifiers

User-defined names are defined as they are encountered during the discussion of the functions of the MCS. However, there are a number of universally-used identifiers which are described here (see seq. 1600-16400).

Note the use of defines for various message types (seq. 1800 -2200); also BEGIN is defined as [D0;[, CH as [CHARACTER], TRUE as [[@FFFF@], and FALSE as [@0000@]. Defining identifiers for queue references is also useful (seq. 2400 -2700). MSG is a message reference and MSG.HDR is a data structure into which message headers are placed (seq. 3500 - 5500). TEXT is used to contain message text of command messages.

Director

The main driver of the MCS is near the end of the program starting at seq. 68800. The algorithm used is:

1. Initialize.
2. Take the next message from the MCS queue.
3. Log the message.
4. If the message does not have "complete and successful" in the result field, then analyze the result and go to step 6.
5. Perform action routines as determined by message type.
6. Return message space to DC buffer if still in use. (That is, if pointed to by MSG.)
7. Stop if commanded by the operator.
8. Go to step 2.

Each step in this simple logic flow will be described in more detail.

Initialize Routine

Name: INITIALIZE. Seq: 66000-68700.

This routine performs the following:

1. Displays program version number.
2. Space-fills print buffer (used for logging).
3. Routes input and output messages to the MCS queue for all stations.

4. Sets the MCS queue limit to five messages.
5. Makes all lines ready.

NOTE

1. MAX.STAS and MAX.LINES contain the highest valid logical station number and logical line number respectively. These are found (seq. 6700 - 6800) using built-in functions.

2. The LINE.PENDING flag is used to prevent multiple "make line ready" messages being queued to the network controller, thus saving message space.

3. The basic technique of creating a message is demonstrated at seq. 67900 -68300.

Take Message from MCS Queue

This is done at seq. 69900 -70000.

The FETCH.MESSAGE built-in procedure points MSG to the next message in the MCS queue, and delinks that message from the MCS queue.

The READ.HEADER built-in procedure copies the message header pointed to by MSG into MSG.HDR.

NOTE

FETCH.MESSAGE has a NOWAIT option which allows the MCS to continue executing, even if no messages were in the MCS queue. (To check this, compare MSG with the null value using the NULL built-in function.) This feature must be used with acumen as, if used carelessly, the MCS can get into a processor-bound loop. This means, on the B 80, that user tasks do not get any processor time because the MCS has a higher priority. However, if used in conjunction with conditional I/O, for example, it can increase the efficiency of the MCS. The motto here is: BE CAREFUL.

Log the Message

This is done at seq. 70100 -70300.

It is always useful to write debug-code into a program from the start.

Here, compile-time and run-time options set or reset the debug code. If the user dollar - option DEBUG (seq.300) is set, the debug code is compiled into the program. When the MCS has been debugged, this code can be excluded from the final version merely by resetting this option for the final compile.

The procedure LOGGIT (seq. 64600 - 65600) prints logical station number, message type, and up to the first 1200 characters of message text for all messages taken from the MCS queue. This can be done only if the flag PRINT.EM is true. (PRINT.EM is set or reset from the SPO by SD and ED commands.)

Process Non-Zero Results

This is done by calling DO.RESULT (seq. 47200 - 49300) at line 70600.

A non-zero result implies that there is a special circumstance associated with this message.

The procedure DO.RESULT is not complete in that is handles only three of the ten defined results. A production MCS should be coded to handle all possible results. The three handled are:

1. Line not ready.
2. Station not ready.
3. Control character or WRU character received.

The first two of these asks the operator to ready the line or station as required. If a control/WRU is received, the message is handled by the DO.INPUT.MSG routine which is described later.

NOTE

The control/WRU received result indicates that the associated message is an input message consisting of either the WRU character or the station's control character followed by the message text. (The actual case can be determined from the events field in the message header.) The MCS programmer and the NDL programmer must agree on the following:

1. Will control characters be recognized?
2. Will they be passed to the MCS as part of the text?

In a "real" MCS, this procedure would be more comprehensive. It would perform more error handling (analyzing the events field and perhaps logging the specific error on disk or the SPO).

Perform Action Routines

The code for this is at seq. 70600 - 74900.

A switch is made depending on the value of the message type field (MSG.TYPE), and a different action is taken depending on the type of message. The only message types which are handled here are:

1 Input (from remote device)
6 Station has been made ready
0 Line has been made ready
15 DC input from SPO
18 Request (from MCP) to attach a task to a subnet
19 Request (from MCP) to attach a task to a station
25 Task has gone to end-of-job

It is obvious that a "real" MCS would handle the majority, if not all, of the possible message types.

Each action routine is now described in turn.

Input Message (1)

An input message is handled by DO.INPUT.MSG (seq. 37100-38900) which is called at seq. 70800. (It may also be called from DO.RESULT at seq. 4890.)

DO.INPUT.MSG performs the following:

1. Decrements the unprocessed input message count for the station which sent the message (CONTINUE.STATION).
2. If the control character received flag is not set, it sends the message back to the station; otherwise,
3. Sets SPO.MSG to false (indicating that the message was not from the SPO); sets TELL.SPO to false (indicating that any reply to this command is to be returned to the sending station). Depending on the command, TELL.SPO may be reset to true in the DO.DC.INPUT procedure; calls DO.DC.INPUT (seq. 49700 - 57500) to handle this message as a command text.

NOTE

1. For every input message which appears on the MCS input queue, the MCS must issue a CONTINUE.STATION command to acknowledge receipt of the message. If this is not done, the network controller is prevented from obtaining DC message space for a station when the number of unacknowledged messages (which have been placed on the MCS queue from that station) exceed its input limit. The input limit for a station is set using the SET.INPUT.LIMIT statement, the default being two.

2. The most efficient way of re-routing a message is to change the required fields in the message header and put it on the network controller queue (or subnet queue if sending it to a task). (See seq. 37800 - 38100.)

3. By setting the retry field in the message header to @FF@, the MCS is indicating that the retry count is to be handled completely by the network controller. Any other value would set the retry count for the corresponding station to MSG.RETRY. This facility is B 80 implementation dependent. However, if required it may be emulated on other CMS systems by inclusion of the following code between (PTO). Seq 40001500 - 40001600 of REQUEST UPOLLED and seq 70000500 - 70000600 of REQUEST SELECTIT. (See sample NDL program.)

IF RETRY = 255 THEN INITIALIZE RETRY.

Make Station Ready (6)

A message of this type appears in the MCS queue (with result = 0) as a confirmation that a station has been made ready. This means that the MCS must have previously queued a make station ready message to the network controller.

This action routine is coded at seq. 71300 - 71900. The actions taken are:

1. Set STA.PENDING flag for this station to 0, indicating action complete.

2. Inform the operator (at the SPO) that the station is ready.

Make Line Ready (8)

A message of this type appears in the MCS queue (with result = 0) as a confirmation that a line has been made ready. This means that the MCS must have previously queued a make line ready message to the network controller.

This action routine is coded at seq. 72100 - 72700. The actions taken are:

1. Set LINE.PENDING flag for this line to 0, indicating action complete.
2. Inform the operator that the line is ready.

Operator Input (15)

A message of this type appears in the MCS queue as a result of the operator inputting a DC message at the SPO.

The action routine (DO.DC.INPUT) is called at seq. 73700 after setting TELL.SPO and SPO.MSG to true. (This is necessary to distinguish type 15 messages from type 1 messages when control-flag = true.)

Procedure DO.DC.INPUT (seq. 49700 - 57500) analyzes messages as command strings and performs actions accordingly. The available commands are:

END	:terminates the MCS
RS <n>	:readies station <n>
RL <n>	:readies line <n>
TO <n> <text>	:sends <text> to station <n>
TO SPO <text>	:sends <text> to SPO
SS <n> <text>	:see TO
SS SPO <text>	:see TO
SS	:start debug print
ED	:end debug print
QM <n> <text>	:queue a message with <text> on subnet queue <n>
ZIP <text>	:pass <text> to SCL/loader
WRU	:return version message
WM	:see WRU

DO.DC.INPUT calls two routines (LOOP.UP and SCAN) to handle the logical analysis of the command string. As the functions are not directly related to data comm, they will not be described in detail, but a brief description is given for completeness.

LOOK.UP (seq. 27500 - 30000) performs a linear search through the VERB.TABLE (see seq. 8700 - 9700 and 77900 - 79000) looking for a match between the current token and the name of the verbs in VERB.TABLE.

SCAN (seq. 16900 - 27100) uses SOURCE as input and TOKEN as output (seq. 10100 - 13800). After a call on SCAN, TOKEN contains the next identifier, and the number of special character from SOURCE. T.SIZE contains the binary equivalent of TOKEN if it is a number. WHICH.VERB identifies the verb.

The logic of DO.DC.INPUT proceeds as follows:

1. Copy up to 255 characters of the message text into SOURCE.
2. Set-up variables for SCAN.
3. If the message came from a remote station, skip over the first token as it is the special control character. (This assumes that the control character for any station is not a space character and is non alpha-numeric, and that the network controller passes the control character to the MCS as part of the message text. This need not be the case, but it is used here as a convention.)
4. Search for the verb.
5. Take action appropriate to the verb found.

The logic for each verb is fairly straightforward. The major points are noted below:

END	The quit flag is set, and this is used to stop the main loop (seq. 52000 and 75100).
RS	The READY.STATION procedure (seq. 32900 -35100) sets STA.PENDING flag to 1. (This prevents multiple make station ready messages existing for a station at any one time.) It then constructs a make station ready message and queues it to the network controller queue. Note that GET.MESSAGE.SPACE is used to get a new message header with no associated message text. This means that the next space occupied by the RS message is returned to the DC message pool. Although trivial in this case, the technique can save space when used in a read MCS. This procedure has no effect if STA.PENDING is 1 on entry.
RL	Uses READY.LINE (seq 35500 - 36700) and is very similar to RS.
TD/SS	The main point of interest in this command is the use of procedure TELL (seq. 30400 -35500). This procedure sends the message TALE (in its entirety if the second parameter is either missing or not of type fixed, or the first TALE.SIZE characters otherwise) to either the SPO or a remote station. The logic proceeds as follows: 1) calculate how many characters are to be sent; 2) if the message is for the SPO then display it; otherwise: 3) compare the number of characters to be sent +1 (for a form feed character) with the

size of the text area of the message pointed to by MSG. If the current text area is too small, the current message is released and gerted with the required text size; and 4) construct the message (placing a form feed character at the beginning) and queue it to the network controller queue.

SD	Opens the printer file and writes heading (seq. 39300 -40700). A real system would use conditional I/O. (Make sure files have enough buffers.)
ED	Closes printer file. Same comment as for SD.
QM	Constructs an input message with text as for command and places this message on a subnet queue.
ZIP	Passes message to SCL/loader. The error option in ZIP should be used as an invalid SCL string following ZIP, causing the MCS to be aborted (DS/DP) by the MCP.
WRU/WM	Uses TELL to send version message (seq. 1200).

If LOOK.UP cannot find the verb, or a verb which is restricted to SPO, use is entered from a remote device and the last entry in the "case" is performed.

Attach Task to Subnet Queue (18)

A message of this type is placed on the MCS queue as part of the processing of the first input data communication request from a task which refers to this subnet queue. The message is processed by DO.ATTACH.SUBN (seq. 57900 -61700), and the logic performed is:

1. Determine the number of stations on the subnet and their logical station numbers (seq. 58600 -59400).

2. Store these in the table TASK.STATIONS (see comments at seq. 14600 - 15400) unless the table slot for this task is already in use, in which case access to the requested subnet is denied.

3. Route the input for each station in the subnet to the task. (Note that the MCS must explicitly route each station in the subnet family.)
4. Allow the task to access the subnet queue.

Attach Task to Station (19)

A message of this type is placed on the MCS queue by the MCP as part of the processing of the first output data communications request from a task which refers to this station. The message is processed by DO.ATTACH.STA (seq. 63900 - 64200) which routes output from the task to the network controller queue and allows the task to communicate with the station.

Detach Task (25)

A message of this type is placed on the MCS queue by the MCP when a data comm task goes to end-of-job. The action taken is a design feature of the MCS. In this case, DO.TASK.DETACH (seq. 62100 - 63500) is called to re-route input messages for the stations which were attached to the task back to the MCS queue. If this re-routing was not performed, then any input messages from these stations would pile-up on the subnet queue until the unprocessed input count exceeded the queue limit of the subnet queue. (Note that the task no longer exists and it is unnecessary to re-route output messages.)

Return Message Space

This is performed at 75000. It is done so that message space is returned to the pool of available DC message space as soon as possible. If, for instance, a message appears in the MCS queue which is ignored, (for example: enable-input) associated message space would become free.

Stop

If the quit flag is set to 1 (seq. 52000), the MCS prints "MCS HALTED" and goes to end-of-job (seq. 75100 - 75400).

NOTE
DO NOT USE THIS MCS IN A PRODUCTION ENVIRONMENT

MODEL.MCS does not perform any serious error-handling. Should an error occur it is aborted (DS/DP) by the MCP. This is not desirable in any data comm system.

It is hoped that the above narrative is of help to potential MCS authors.

SAMPLE MCS PROGRAM

```
$CONTROL 300 DATA 2000                                                  00000100
$RESET CHECKS                                                           00000200
$SET DEBUGN                                                             00000300
$LIST                                                                   00000400
$NOWARNING                                                              00000500
$MCS 4                                                                  00000600
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00000605
%                                                                      %00000610
%                  PROPRIETARY PROGRAM MATERIAL                        %00000615
%                                                                      %00000620
%      THIS MATERIAL IS PROPRIETARY TO BURROUGHS CORPORATION  AND IS   %00000625
%  NOT TO BE REPRODUCED, USED OR DISCLOSED EXCEPT IN ACCORDANCE WITH   %00000630
%  PROGRAM LICENCE OR UPON WRITTEN AUTHORIZATION OF THE PATENT         %00000635
%  DIVISION OF BURROUGHS CORPORATION, DETROIT, MICHIGAN 48232.         %00000640
%                                                                      %00000645
%              COPYRIGHT (C) 1979 BURROUGHS CORPORATION                %00000650
%                                                                      %00000655
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00000660
    PROCEDURE OLDWEIRDHAROLD;                                           00000700
%                                                                       00000800
%   THIS SAMPLE MCS SHOULD BE READ IN CONJUNCTION WITH                  00000900
%   THE NOTES PROVIDED. (FILEID: DCMCSNOTES.)                           00000910
%                                                                       00000920
%%%%%%%%%%%% T H I S   I S   A   S A M P L E   O N L Y %%%%%%%%%%%%%%%%00000930
%                                                                       00000940
%   AND IS NOT SUITABLE FOR USE IN A PRODUCTION ENVIRONMENT.            00000950
%                                                                       00001000
    DEFINE VERSION                                                      00001100
         £"MODEL MCS MARK 3.0.2"£;                                      00001200
%                                                                       00001300
%                                                                       00001400
%                                                                       00001500
    DEFINE                                                              00001600
%     MESSAGE TYPES                                                     00001700
     INPUT          £1£,                                                00001800
     OUTPUT         £2£,                                                00001900
     ENABLE.INPUT   £4£,                                                00002000
     MAKE.STA.RDY   £6£,                                                00002100
     MAKE.LINE.RDY  £8£,                                                00002200
%     QUEUES                                                            00002300
    MCSQ            £@0000@£,                                           00002400
     NCQ            £@1000@£,                                           00002500
     SUBQ           £@2000@£,                                           00002600
    STAQ            £@3000@£,                                           00002700
%     MISCELLANEOUS                                                     00002800
     CH             £CHARACTER£,                                        00002900
    BEGIN           £DO;£,                                              00003000
    TRUE            £@FFFF@£,                                           00003100
    FALSE           £@0000@£,                                           00003200
     MAX.ERRS       £7£;                                                00003300
%                                                                       00003400
     DECLARE                                                            00003500
     MSG MESSAGE.REFERENCE,                                             00003600
     1 MSG.HDR                  CH(35),                                 00003700
       2 MSG.LINE               CH(1),                                  00003800
```

```
        2 MSG.RESULT          CH(1),                                   00003900
        2 MSG.TYPE            CH(1),                                   00004000
        2 MSG.TSK             CH(1),                                   00004100
        2 DUMMY               CH(1),                                   00004200
        2 MSG.STA             FIXED,                                   00004300
        2 DUMMY               CH(1),                                   00004400
        2 MSG.EVENTS          CH(3),                                   00004500
          3 DUMMY             CH(1),                                   00004600
          3 MSG.CC.RECVD     BIT(1),                                   00004700
          3 DUMMY             BIT(7),                                  00004800
          3 DUMMY             CH(1),                                   00004900
        2 MSG.SUBQ            FIXED,                                   00005000
        2 MSG.LNTH             FIXED,                                  00005100
        2 MSG.MAX.LNTH         FIXED,                                  00005200
        2 DUMMY                 CH(1),                                 00005300
        2 MSG.RETRY           CH(1),                                   00005400
        2 DUMMY               CH(16),                                  00005500
      TEXT                    CH(255),                                 00005600
      QUIT                    FIXED,                                   00005700
      LSN                     FIXED,                                   00005800
      LLN                     FIXED,                                   00005900
      MAX.SUBQS               FIXED,                                   00006000
      MAX.STAS                FIXED,                                   00006100
      MAX.LINES               FIXED;                                   00006200
 %                                                                     00006300
 %                                                                     00006400
 %                                                                     00006500
      MAX.SUBQS:=SUBNET.COUNT-1;                                       00006600
      MAX.STAS:=STATION.COUNT;                                         00006700
      MAX.LINES:=LINE.COUNT;                                           00006800
 %                                                                     00006900
 %                                                                     00007000
 %                                                                     00007100
      DECLARE                                                          00007200
      1 STA.TABLE(MAX.STAS)   CH(1),                                   00007300
        2 STA.PENDING         BIT(1),                                  00007400
        2 DUMMY               BIT(7),                                  00007500
 %                                                                     00007600
      1 LINE.TABLE(MAX.LINES) CH(1),                                   00007700
        2 LINE.PENDING        BIT(1),                                  00007800
        2 DUMMY               BIT(7);                                  00007900
     MAX.STAS:-1;                                                      00008000
     MAX.LINES:-1;                                                     00008100
 %                                                                     00008200
 %                                                                     00008300
 %   COMMAND HANDLING STUFF.                                           00008400
 %                                                                     00008500
 %                                                                     00008600
     DEFINE V.TAB.SIZE £077£; % 7 * NO. OF VERBS                       00008700
     SEGMENT VERB.SEG(V.TAB.SIZE);                                     00008800
     REMAP VERB.SEG:                                                   00008900
     1 VERB.TABLE (V.TAB.SIZE) CH(7),                                  00009000
          2 VERB CH(4),             %NAME OF COMMAND                   00009100
          2 VERB.NO  FIXED,         %WHICH.VERB - ALLOWS SYNONYMS      00009200
          2 VERB.SPO.ONLY CH(1);    %TRUE IF VERB RESTRICTED TO SPO USE 00009300
```

```
     DECLARE                                                           00009400
          WHICH.VERB FIXED,        %INDEX TO RECOGNISE CURRENT VERB    00009500
          SPO.MSG FIXED,           %TRUE IF CURRENT MESSAGE FROM SPO   00009600
          TELL.SPO FIXED;          %TRUE IF REPLY IS TO SPO            00009700
%                                                                      00009800
%                  SCANNER STUFF                                       00009900
%                                                                      00010000
     SEGMENT TIPE.SEG(128);                                            00010100
     REMAP TIPE.SEG: TIPE(128) CH(1);                                  00010200
     DECLARE                                                           00010300
     ALL.DONE       FIXED,         %TRUE IF NO MORE TO SCAN IN THIS REC. 00010400
     SOURCE         CH(255),       %TEXT CURRENTLY BEING SCANNED       00010500
     M.SIZE    FIXED,              %SIZE OF SOURCE                     00010600
     PTR            FIXED,         %OFFSET IN SOURCE OF NEXT CHAR.     00010700
     CHAR           CH(1),         %CHARACTER WHAT SCANNER IS LOOKING AT 00010800
     C.TIPE         FIXED,         %CODE INDICATING TYPE OF "CHAR" :   00010900
                                   %    0 = ALPHA                      00011000
                                   %    1 = NUMERIC                    00011100
                                   %    5 = SPECIAL CHARACTER          00011200
                                   %    6 = SPACE                      00011300
                                   %    9 = NON-GRAPHIC CHARACTER      00011400
     TOKEN          CH(0);         %POINTS TO CURRENT SYMBOL           00011500
     SEGMENT TOKEN.SEG(262);                                           00011600
     REMAP TOKEN.SEG:                                                  00011700
          T.TYPE FIXED,            % TOKEN TYPE                        00011800
          T.SIZE FIXED,            % TOKEN STRING LENGTH               00011900
          T.VALUE FIXED,           % TOKEN VALUE IF T.TYPE IS NUMBER   00012000
          T.STRING CH(255),        % TOKEN STRING                      00012100
          T.DUMMY CH(1);           %                                   00012200
     DEFINE                                                            00012300
          IDENTIFIER£0£,                                               00012400
          NUMBER    £1£,                                               00012500
          STRING    £2£,                                               00012600
          SPECIAL.CHAR£3£,                                             00012700
          TERMINATR £4£,                                               00012800
          BLANK     £5£,                                               00012900
          RETN £6£,                                                    00013000
          END.CHAR  £@FF@£,                                            00013100
          IDENTIFIER.TOKEN              £(T.TYPE=IDENTIFIER)£,         00013200
          NUMBER.TOKEN                  £(T.TYPE=NUMBER)£,             00013300
          STRING.TOKEN                  £(T.TYPE=STRING)£,             00013400
          SPECIAL.CHAR.TOKEN            £(T.TYPE=SPECIAL.CHAR)£,       00013500
          TERMINATOR.TOKEN              £(T.TYPE=TERMINATR)£,          00013600
          SPACE.TOKEN                   £(T.TYPE=BLANK)£,              00013700
          RETURN.TOKEN                  £(T.TYPE=RETN)£;               00013800
%                                                                      00013900
%    OTHER DECLARATIONS FOR USE OF THE GENERAL POPULACE                00014000
%                                                                      00014100
     DECLARE                                                           00014200
     1 TASK.STATIONS     CH(199),                                      00014300
          2 TSK.STN      FIXED;                                        00014400
%                                                                      00014500
%    TASK.STATIONS IS USED TO HOLD LSN OF EACH STATION WHICH           00014600
%    IS CAPABLE OF INPUTING TO A TASK. THERE IS A SLOT IN THIS         00014700
%    TABLE FOR UP TO 9 TASKS. EACH SLOT CONSISTS OF A NUMBER           00014800
```

```
%   (FIXED VALUE) INDICATING THE NUMBER OF STATIONS FOR THIS TASK       00014900
%   FOLLOWED BY A LIST OF LSN-S FOR THIS TASK. SINCE EACH SLOT          00015000
%   IS 11 NUMBERS LONG THERE IS A RESTRICTION OF 10 STATIONS            00015100
%   PER TASK. FURTHERMORE A TASK CAN ONLY COMMUNICATE WITH 1 SUBNET AT  00015200
%   ANY GIVEN TIME. THIS TABLE IS USED IN DO.ATTACH.SUBN AND            00015300
%   TASK.DETACH.                                                        00015400
%                                                                       00015500
    DECLARE                                                             00015600
    1 P.BUF CH(132),                                                    00015700
         2 P.STN CH(5),                                                 00015800
         2 DUMMY CH(1),                                                 00015900
         2 P.TYP CH(5),                                                 00016000
         2 DUMMY CH(1),                                                 00016100
         2 P.TXT CH(120),                                               00016200
    PRINT.EM FIXED;     %%%%% TRUE IF MESSAGES ARE TO BE PRINTED        00016300
    FILE P WORK.AREA P.BUF;                                             00016400
$PAGE                                                                   00016490
    FORWARD PROCEDURE DO.DC.INPUT;                                      00016500
%                                                                       00016600
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00016700
%                                                                       00016800
    PROCEDURE SCAN;                                                     00016900
%                                                                       00017000
%   I EXTRACT THE NEXT TOKEN FROM THE PARAMETER STRING                  00017100
%   AND PLACE IT IN T.STRING.  THE TYPE, LENGTH, AND (FOR NUMBERS)      00017200
%   THE BINARY EQUIVALENT ARE ALSO NOTED.                               00017300
%                                                                       00017400
    DEFINE COPY £ BEGIN           %BIG ON CORE - LOW ON TIME OVERHEAD   00017500
    IF ALL.DONE THEN                        %END OF INPUT               00017600
    BEGIN                                                               00017700
         T.TYPE:=TERMINATR;T.SIZE:=1;T.STRING:=END.CHAR;                00017800
    END; ELSE BEGIN                                                     00017900
    SUBSTR(T.STRING,T.SIZE,1):=SUBSTR(SOURCE,PTR,1); %COPY              00018000
    PTR:+1; T.SIZE:+1;                                                  00018100
    ALL.DONE:=(PTR>=M.SIZE);                                            00018200
    END;END£;                                                           00018300
    DEFINE                                                              00018400
    NEXT£                                                               00018500
         IF ALL.DONE THEN UNDO EXTRACT;                                 00018600
         CHAR:=SUBSTR(SOURCE,PTR,1);                                    00018700
         C.TIPE:=TIPE(CHAR)£,                                           00018800
    ALPHA £(C.TIPE=0)£,                                                 00018900
    NUMERIC £ (C.TIPE=1)£,                                              00019000
    DOT£(CHAR=".")£,                                                    00019100
    CONJUNCTION£( (CHAR="-"))£;                                         00019200
$IF CHECKS THEN                                                         00019300
    DISPLAY("START GET.TOKEN");                                         00019400
    DISPLAY((IF CONTROL.MODE THEN "TRUE" ELSE "FALSE"));                00019500
$END                                                                    00019600
    DO EXTRACT;          %TO PROVIDE COMMON RETURN POINT                00019700
    DO SKIP.BLANKS FOREVER;              %SKIP LEADING SPACES           00019800
         IF PTR>=M.SIZE OR ALL.DONE THEN                                00019900
         BEGIN ALL.DONE:=TRUE;COPY;UNDO EXTRACT;END;                    00020000
         NEXT;                                                          00020100
         IF CHAR /= " " THEN UNDO SKIP.BLANKS;                          00020200
```

```
          PTR:+1;                                                       00020300
      END SKIP.BLANKS;                                                  00020400
      T.SIZE:=[T.TYPE:=0];                                              00020500
      T.STRING:=" ";                                                    00020600
 %                                                                      00020700
 %    HAVING DONE ALL THAT BORING STUFF WE CAN NOW GO GET A TOKEN       00020800
 %                                                                      00020900
      IF ALPHA THEN                     %WE HAVE AN IDENTIFIER          00021000
      BEGIN                                                             00021100
          T.TYPE:=IDENTIFIER;                                           00021200
          DO FOREVER;                                                   00021300
              IF ALPHA THEN       %THIS IS TRUE MOST TIMES              00021400
              BEGIN COPY;NEXT;END;ELSE                                  00021500
 %             IF NOT (NUMERIC  OR CONJUNCTION)                         00021600
 %             THEN                                                     00021700
              UNDO EXTRACT;                                             00021800
 %             ELSE                                                     00021900
 %            BEGIN                                                     00022000
 %            COPY;                                                     00022100
 %            NEXT;                                                     00022200
 %            END;                                                      00022300
          END;                                                          00022400
      END;                                                              00022500
 %                                                                      00022600
      IF NUMERIC THEN                   %WE HAVE A NUMBER               00022700
      BEGIN                                                             00022800
          T.TYPE:=NUMBER;                                               00022900
          T.VALUE:=0;                                                   00023000
          DO FOREVER;                   %LOOK FOR THE REST              00023100
              IF NOT NUMERIC THEN       %THATS ALL                      00023200
              UNDO EXTRACT;                                             00023300
              COPY;                                                     00023400
              T.VALUE:=T.VALUE*10-"0"+CHAR; %BINARY CONVERT.            00023500
              NEXT;                                                     00023600
          END;                                                          00023700
      END;                                                              00023800
 %                                                                      00023900
      IF CHAR="""" THEN                 %WE HAVE A STRING               00024000
      BEGIN                             %TO HANDLE THE STRING DRIVEN THIN00024100
          T.TYPE:=STRING;                                               00024200
          PTR:+1;                       %SKIP OVER "                    00024300
          DO FOREVER;                                                   00024400
              NEXT;                                                     00024500
              IF CHAR="""" THEN         %WE MIGHT BE FINISHED           00024600
              BEGIN                                                     00024700
                  IF[PTR:+1]>=M.SIZE THEN UNDO EXTRACT;                 00024800
                  NEXT;                                                 00024900
                  IF CHAR /= """" THEN%WE ARE UNSTRUNG                  00025000
                  UNDO EXTRACT;                                         00025100
              END;                                                      00025200
 %            IF                                                        00025300
 %            CHAR=" " OR                                               00025400
 %            CHAR="/" OR                                               00025500
 %            CHAR="=" THEN NASTY.STRING:=TRUE;                         00025600
              COPY;                                                     00025700
```

```
            END;
        END;
%
%       LET US ASSUME  THAT WE HAVE A SPECIAL CHARACTER.
%       COPY CAN LOOK OUT FOR END OF LINE.
%
        T.TYPE:=SPECIAL.CHAR;
        COPY;
        END EXTRACT;
        SETNAME(TOKEN,SUBSTR(T.STRING,0,T.SIZE));
$IF CHECKS THEN
        DISPLAY("END OF GET.TOKEN");DISPLAY(TOKEN);
$END
        END SCAN;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
        PROCEDURE LOOK.UP;
%
%       I SEARCH THROUGH THE VERB.TABLE FOR A MATCH WITH THE NEXT
%       TOKEN IN SOURCE. WHICH.VERB WILL CONTAIN A VALUE TO INDICATE
%       THE VERB FOUND. IF A VERB IS NOT FOUND OR A VALID
%       VERB IS ENTERED FROM A REMOTE DEVICE BUT IS RESTRICTED TO
%       USE FROM THE SPO THEN WHICH.VERB WILL HAVE THE VALUE @FFFF@.
%
        DECLARE I FIXED;
        WHICH.VERB:=@FFFF@;             %TO INDICATE INVALID - MAY BE CHANGED
                                        %LATER IF WE GET A VALID VERB
        I:=0;
        SCAN;                           %TOKEN SHOULD NOW BE A VERB
        DO LOOK FOREVER;
            IF I >= V.TAB.SIZE THEN UNDO LOOK; %DID NOT GET A MATCH
            IF VERB(I)=TOKEN THEN       %FOUND A MATCH
            BEGIN
                IF NOT SPO.MSG AND VERB.SPO.ONLY(I)
                THEN;                   %LEFT AS INVALID
                ELSE WHICH.VERB:=VERB.NO(I);
                UNDO LOOK;              %STOP LOOKING IN EITHER CASE
            END;
            ELSE
            I:+7;
        END LOOK;
        END LOOK.UP;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
        PROCEDURE TELL(TALE,TALE.SIZE);
%
%       I WRITE THE MESSAGE POINTED AT BY TALE TO SPO OR
%       REMOTE DEVICE. IF CALLED WITH 1 PARAMETER THEN WRITE ALL OF TALE
%       OTHERWISE WRITE FIRST TALE.SIZE CHARACTERS.
%
        DECLARE HOW.MANY FIXED;         %WILL CONTAIN NO. OF CHARS TO WRITE
        DECLARE FF CH(1);               %WILL CONTAIN FORM FEED CHARACTER
        FF:="@0C@";                     %   -     JUST LIKE THAT
```

```
     HOW.MANY:=IF TYPE(TALE.SIZE)=2 THEN TALE.SIZE ELSE SIZE(TALE);      00031300
     IF TELL.SPO THEN                                                     00031400
          DISPLAY(SUBSTR(TALE,0,HOW.MANY));                               00031500
     ELSE                                                                 00031600
     BEGIN                                                                00031700
          %%% WE FIRST SEE IF THE CURRENT MSG HAS BIG ENOUGH              00031800
          %%% TEXT AREA TO TAKE THE TEXT WE WISH TO SEND.                 00031900
          %%% IF NOT THROW IT AWAY AND GET MORE SPACE.                    00032000
          IF MSG.MAX.LNTH < HOW.MANY + 1 THEN                             00032100
          BEGIN                                                           00032200
               RELEASE.MESSAGE.SPACE(MSG);                                00032300
               GET.MESSAGE.SPACE(MSG,HOW.MANY+1);                         00032400
          END;                                                            00032600
          MSG.RETRY:=@FF@;          %LET NC HANDLE IT.                    00032700
          MSG.TYPE:=OUTPUT;                                               00032800
          WRITE.TEXT(MSG,0,1,FF);  %INSERT FF                             00032900
          WRITE.TEXT(MSG,1,HOW.MANY,TALE);                                00033000
          MSG.LNTH:=HOW.MANY+1;     %GOT TO SAY EXPLICITLY                00033100
          WRITE.HEADER(MSG,MSG.HDR);                                      00033200
          QUEUE(MSG,NCQ);           %AFTER ALL THAT SURGERY WE WRITE IT   00033300
     END;                                                                 00033400
     END TELL;                                                            00033500
%                                                                         00033600
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00033700
%                                                                         00033800
      PROCEDURE READY.STATION(LSN);                                       00033900
      IF STA.PENDING(LSN) THEN;    %WAIT FOR GOOD RESULT                  00034000
      ELSE                                                                00034100
      DO;                                                                 00034200
         STA.PENDING(LSN):=1; %TO SAY WE'VE BEEN HERE                     00034300
         GET.MESSAGE.SPACE(MSG,0);                                        00034400
         READ.HEADER(MSG,MSG.HDR);                                        00034500
         MSG.TYPE:=MAKE.STA.RDY;                                          00034600
         MSG.STA:=LSN;                                                    00034700
         WRITE.HEADER(MSG,MSG.HDR);                                       00034800
         QUEUE(MSG,NCQ);                                                  00034900
      END;                                                                00035000
      END READY.STATION;                                                  00035100
%                                                                         00035200
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00035300
%                                                                         00035400
      PROCEDURE READY.LINE(LLN);                                          00035500
      IF LINE.PENDING THEN;   %WAIT FOR GOOD RESULT                       00035600
      ELSE                                                                00035700
      DO;                                                                 00035800
         LINE.PENDING(LLN):=1;      %TO SAY WE'VE BEEN HERE ALREADY       00035900
         GET.MESSAGE.SPACE(MSG,0);                                        00036000
         READ.HEADER(MSG,MSG.HDR);                                        00036100
         MSG.TYPE:=MAKE.LINE.RDY;                                         00036200
         MSG.LINE:=LLN;                                                   00036300
         WRITE.HEADER(MSG,MSG.HDR);                                       00036400
         QUEUE(MSG,NCQ);                                                  00036500
      END;                                                                00036600
      END READY.LINE;                                                     00036700
%                                                                         00036800
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00036900
%                                                                       00037000
      PROCEDURE DO.INPUT.MSG;                                           00037100
%                                                                       00037200
%   TAKE MSG FROM REMOTE DEVICE. RETURN OR ANALYSE COMMAND IF CC RECV.  00037300
%                                                                       00037400
    CONTINUE.STATION(MSG.STA);    %ALLOWS STATION TO KEEP SENDING IN MSG00037500
    IF NOT MSG.CC.RECVD THEN                                            00037600
    BEGIN                                                               00037700
     MSG.TYPE:=OUTPUT;                                                  00037800
     MSG.RETRY:=@FF@;                                                   00037900
     WRITE.HEADER(MSG,MSG.HDR);                                         00038000
     QUEUE(MSG,NCQ);                                                    00038100
    END;                                                                00038200
    ELSE                                                                00038300
    BEGIN                                                               00038400
    TELL.SPO:=FALSE;              %ASSUME REPLY TO STN - MAY BE CHANGED 00038500
    SPO.MSG:=FALSE;                                                     00038600
    DO.DC.INPUT;                                                        00038700
    END;                                                                00038800
     END DO.INPUT.MSG;                                                  00038900
%                                                                       00039000
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00039100
%                                                                       00039200
    PROCEDURE START.LOGGING;                                            00039300
%                                                                       00039400
%   OPEN PRINTER FILE AND SET LOGGING FLAG.                             00039500
%                                                                       00039600
    IF PRINT.EM THEN                                                    00039700
    TELL("CANNOT SD: ALREADY LOGGING.");                                00039800
    ELSE BEGIN                                                          00039900
    PRINT.EM:=TRUE;                                                     00040000
    OPEN(P);                                                            00040100
    P.STN:=" LSN";                                                      00040200
    P.TYP:="TYPE";                                                      00040300
    SUBSTR(P.BUF,60):="TEXT";                                           00040400
    WRITE(P);                                                           00040500
    END;                                                                00040600
    END START.LOGGING;                                                  00040700
%                                                                       00040800
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00040900
%                                                                       00041000
    PROCEDURE STOP.LOGGING;                                             00041100
%                                                                       00041200
%   CLOSE PRINT FILE AND RESET LOGGING FLAG.                            00041300
%                                                                       00041400
    IF NOT PRINT.EM THEN                                                00041500
    TELL("CANNOT ED: NOT LOGGING.");                                    00041600
    ELSE BEGIN                                                          00041700
    PRINT.EM:=FALSE;                                                    00041800
    CLOSE(P);                                                           00041900
    END;                                                                00042000
    END STOP.LOGGING;                                                   00042100
%                                                                       00042200
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00042300
```

```
%
    FUNCTION VALID.STA;
%
%   IF THE VALUE OF THE CURRENT TOKEN IS BIGGER THAN THE
%   NUMBER OF STATIONS WE HAVE THEN RETURN FALSE (AFTER
%   DISPLAYING A WARNING) OTHERWISE RETURN TRUE.
%
    IF T.VALUE > MAX.STAS  THEN
    BEGIN
        TELL.SPO:=SPO.MSG;       %RETURN TO SENDER
        TELL("REQUEST DENIED: INVALID STATION NUMBER");
        RETURN FIX(FALSE);
    END;
    ELSE RETURN FIX(TRUE);
    END VALID.STA;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    FUNCTION VALID.LINE;
%
%    I DO FOR LINES WHAT VALID.STA DOES FOR STATIONS.
%
    IF T.VALUE > MAX.LINES THEN
    BEGIN
        TELL.SPO:=SPO.MSG;       %RETURN TO SENDER
        TELL("REQUEST DENIED: INVALID LINE NUMBER");
        RETURN FIX(FALSE);
    END;
    ELSE RETURN FIX(TRUE);
    END VALID.LINE;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    FUNCTION VALID.SUBQ;
%
%   CHECK FOR VALID SUBNET NUMBERS
%
    IF T.VALUE > MAX.SUBQS THEN
    BEGIN
        TELL.SPO:=SPO.MSG;
        TELL("REQUEST DENIED: INVALID SUBNET NUMBER");
        RETURN FIX(FALSE);
    END;
    ELSE RETURN FIX(TRUE);
    END VALID.SUBQ;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
     PROCEDURE DO.RESULT;
%
%     NON-ZERO RESULTS ARE HANDLED HERE
%
     CASE MSG.RESULT;

```

```
          DO;   %LINE NOT RDY                                            00047800
            TEXT:="RL";                                                  00047900
            CONVERT(0,SUBSTR(TEXT,2,1),MSG.LINE);                        00048000
            DISPLAY(SUBSTR(TEXT,0,5));                                   0^048100
          END;                                                           00048200
          DO;   %STA NOT RDY                                             00048300
            TEXT:="RS";                                                  00048400
            CONVERT(0,SUBSTR(TEXT,2,1),MSG.STA);                         00048500
            DISPLAY(SUBSTR(TEXT,0,5));                                   00048600
          END;                                                           00048700
          DO;   %CNTL OR WRU                                             00048800
            DO.INPUT.MSG;                                                00048900
          END;                                                           00049000
          ;;;;;;                                                         00049100
        END CASE;                                                        00049200
        END DO.RESULT;                                                   00049300
%                                                                        00049400
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00049500
%                                                                        00049600
    PROCEDURE DO.DC.INPUT;                                               00049700
%                                                                        00049800
%   THIS PROCEDURE HANDLES COMMANDS FROM THE SPO AND FROM REMOTE         00049900
%   DEVICES.  LOOK.UP GETS THE VERB, SCAN GETS THE PARAMETERS            00050000
%   AND TELL SENDS THE REPLY (IF ANY).                                   00050100
%                                                                        00050200
%                                                                        00050300
%   FIRST GET THE TEXT AND SET UP INIT. CONDITIONS FOR SCAN.             00050400
%                                                                        00050500
    READ.TEXT(MSG,0,255,SOURCE);                                         00050600
    M.SIZE := IF MSG.LNTH <= 255 THEN MSG.LNTH ELSE 255;                 00050700
    PTR:=0;                                                              00050800
    ALL.DONE:=FALSE;                                                     00050900
    IF NOT SPO.MSG THEN SCAN;      %SKIP OVER CONTROL CHAR               00051000
                                   %ASSUMES THAT NC SENDS IT TO US.      00051100
%                                                                        00051200
%   GET THE VERB                                                         00051300
%                                                                        00051400
    LOOK.UP;                                                             00051500
%                                                                        00051600
%   HANDLE EACH VERB AS NECESSARY                                        00051700
%                                                                        00051800
    CASE WHICH.VERB;                                                     00051900
         QUIT:=1;                  % END                                 00052000
         BEGIN                     % RS                                  00052100
              SCAN;                                                      00052200
              IF NUMBER.TOKEN AND VALID.STA THEN                         00052300
              READY.STATION(T.VALUE);                                    00052400
         END;                                                            00052500
         BEGIN                     % RL                                  00052600
              SCAN;                                                      00052700
              IF NUMBER.TOKEN AND VALID.LINE THEN                        00052800
              READY.LINE(T.VALUE);                                       00052900
         END;                                                            00053000
         BEGIN                     % TO OR SS                            00053100
              SCAN;                                                      00053200
```

```
                IF TOKEN = "SPO" THEN TELL.SPO:=TRUE;                 00053300
                ELSE                                                  00053400
                BEGIN                                                 00053500
                    TELL.SPO:=FALSE;                                  00053600
                    IF NUMBER.TOKEN THEN                              00053700
                    BEGIN                                             00053800
                        IF VALID.STA THEN                             00053900
                        MSG.STA:=T.VALUE;                             00054000
                        ELSE UNDO(*);                                 00054100
                    END;                                              00054200
                    ELSE                                              00054300
                    BEGIN                                             00054400
                        TELL.SPO:=SPO.MSG;  %RETURN TO SENDER         00054500
                        TELL("REQUEST DENIED: NEEDS VALID DEST.");    00054600
                        UNDO(*);                                      00054700
                    END;                                              00054800
                END;                                                  00054900
                TELL(SUBSTR(SOURCE,PTR),(M.SIZE-PTR));                00055000
            END;                                                      00055100
            START.LOGGING;           % SD                             00055200
            STOP.LOGGING;            % ED                             00055300
        BEGIN                        % QM                             00055400
            SCAN;                                                     00055500
            IF NUMBER.TOKEN AND VALID.SUBQ THEN                       00055600
            BEGIN                                                     00055700
                MSG.TYPE:=INPUT;                                      00055800
                MSG.LNTH:=M.SIZE-PTR;                                 00055900
                WRITE.TEXT(MSG,0,MSG.LNTH,SUBSTR(SOURCE,PTR));        00056000
                WRITE.HEADER(MSG,MSG.HDR);                            00056100
                QUEUE(MSG,(SUBQ+T.VALUE));                            00056200
            END;                                                      00056300
            ELSE TELL("REQUEST DENIED: NEEDS VALID SUBQ");            00056400
        END;                                                          00056500
        BEGIN                        % ZIP                            00056600
            ZIP(2,SUBSTR(SOURCE,PTR,(M.SIZE-PTR)));                   00056700
            TELL("MESSAGE ZIPPED.");                                  00056800
        END;                                                          00056900
        BEGIN                        % WRU OR WM                      00057000
            TELL(VERSION);                                            00057100
        END;                                                          00057200
            TELL("CANNOT RECOGNISE COMMAND");  %ALL OTHER CASES       00057300
        END CASE;                                                     00057400
        END DO.DC.INPUT;                                              00057500
    %                                                                 00057600
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00057700
    %                                                                 00057800
         PROCEDURE DO.ATTACH.SUBN;                                    00057900
    %                                                                 00058000
    %   I MAKE MCS NON-PARTICIPATING FOR EACH STATION IN              00058100
    %   THE SUBNET THAT THE TASK WANTS TO RECEIVE FROM.               00058200
    %                                                                 00058300
        DECLARE                                                       00058400
        1 SUB.DESC          CH(14),                                   00058500
            2 SUB.NAME      CH(12),                                   00058600
            2 SUB.SIZE      FIXED,                                    00058700
```

```
    I                       FIXED;
    SUBNET.DESCRIPTION(MSG.SUBQ,SUB.DESC);
    SUB.SIZE:+SUB.SIZE;
    DECLARE
    1 SUB.STAS            CH(SUB.SIZE),  %STATIONS ON THIS SUBNET
         2 SUB.STN        FIXED;
    SUBNET.STATIONS(MSG.SUBQ,SUB.STAS);
    IF TSK.STN((MSG.TSK-1)*22)/=0 THEN %ALREADY ATTACHED TO A Q
    BEGIN
         DISPLAY("CANNOT ATTACH: TASK ALREADY HAS SUBQ");
         DISALLOW.INPUT(MSG.SUBQ,MSG.TSK);
    END;
    ELSE
    BEGIN
         TSK.STN((MSG.TSK-1)*22):=SUB.SIZE;   %NO. OF STNS*2
         I:=0;
         DO ROUTER FOREVER;
              IF I >= SUB.SIZE THEN UNDO ROUTER;
              IF I >= 20 THEN
              BEGIN
                   DISPLAY("CANNOT ROUTE MORE THAN 10 STNS PER Q");
                   UNDO ROUTER;
              END;
              ROUTE.INPUT(SUB.STN(I),(SUBQ+MSG.SUBQ));
              TSK.STN((MSG.TSK-1)*22+I+2):=SUB.STN(I);
              I:+2;
         END ROUTER;
          ALLOW.INPUT(MSG.SUBQ,MSG.TSK);
    END;
     END DO.ATTACH.SUBN;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    PROCEDURE DO.TASK.DETACH;
%
%   I MAKE MCS PARTICIPATE FOR THOSE STATIONS IN SUBQ OF
%   THE TASK WHICH HAS JUST GONE TO EOJ.
%
    DECLARE(I,J)FIXED;
    J:=TSK.STN((MSG.TSK-1)*22);
    I:=0;
    DO FOREVER;
         IF I>=J OR I>=20 THEN UNDO;
         ROUTE.INPUT(TSK.STN((MSG.TSK-1)*22+I+2),MCSQ);
         I:+2;
    END;
    TSK.STN((MSG.TSK-1)*22):=0;
    END DO.TASK.DETACH;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
     PROCEDURE DO.ATTACH.STA;
    ROUTE.OUTPUT(MSG.STA,NCQ);
     ALLOW.OUTPUT(MSG.STA,MSG.TSK);
     END DO.ATTACH.STA;
```

```
%                                                                        00064300
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00064400
%                                                                        00064500
$IF DEBUGN THEN                                                          00064600
     PROCEDURE LOGGIT;                                                   00064700
%     DEBUG CODE GOES HERE                                               00064800
    P.BUF:="";                                                           00064900
    IF NOT PRINT.EM THEN RETURN;  %NO LOGGING                            00065000
    CONVERT(0,P.STN,MSG.STA);                                            00065100
    CONVERT(0,P.TYP,MSG.TYPE);                                           00065200
    READ.TEXT(MSG,0,120,P.TXT);                                          00065300
    WRITE(P);                                                            00065400
     END  LOGGIT;                                                        00065500
$END                                                                     00065600
%                                                                        00065700
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00065800
%                                                                        00065900
     PROCEDURE INITIALIZE;                                               00066000
%                                                                        00066100
%                                                                        00066200
%   MAKE MCS PARTICIPATING.                                              00066300
%   READY ALL THE LINES.                                                 00066400
%   SET THE MCS Q LIMIT TO 5.                                            00066500
%   PUT SPACES IN THE PRINT BUFFER (FOR LOGGING);                        00066600
%                                                                        00066700
    DISPLAY(VERSION);                                                    00066800
    P.BUF:="";                                                           00066900
     LSN:=0;                                                             00067000
     DO FOREVER;                                                         00067100
        ROUTE.INPUT(LSN,MCSQ);                                           00067200
        ROUTE.OUTPUT(LSN,MCSQ);                                          00067300
        IF [LSN:+1] > MAX.STAS THEN UNDO;                                00067400
     END;                                                                00067500
     SET.QUEUE.LIMIT((MCSQ),5);                                          00067600
     LLN:=0;                                                             00067700
     DO FOREVER;                                                         00067800
         LINE.PENDING(LLN):=1;                                           00067810
        GET.MESSAGE.SPACE(MSG,0);                                        00067900
        READ.HEADER(MSG,MSG.HDR);                                        00068000
        MSG.TYPE:=MAKE.LINE.RDY;                                         00068100
        MSG.LINE:=LLN;                                                   00068200
        WRITE.HEADER(MSG,MSG.HDR);                                       00068300
        QUEUE(MSG,NCQ);                                                  00068400
        IF [LLN:+1]>MAX.LINES THEN UNDO;                                 00068500
     END;                                                                00068600
     END INITIALIZE;                                                     00068700
$PAGE                                                                    00068710
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00068800
%                                                                       %00068900
%    SSS  TTTTT   A    RRR   TTTTT    H  H  EEEE  RRR    EEEE           %00069000
%   S       T    A A   R  R    T      H  H  E     R  R   E              %00069100
%    SS     T    AAA   RRR     T      HHHH  EEE   RRR    EEE            %00069200
%      S    T   A   A  R R     T      H  H  E     R R    E              %00069300
%   SSS     T   A   A  R  R    T      H  H  EEEE  R  R   EEEE 0         %00069400
%                                                                       %00069500
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00069600
     INITIALIZE;                                                        00069700
     DO MAIN.LOOP FOREVER;                                              00069800
        FETCH.MESSAGE(MSG);                                             00069900
        READ.HEADER(MSG,MSG.HDR);                                       00070000
$IF DEBUGN THEN                                                         00070100
        LOGGIT;                                                         00070200
$END                                                                    00070300
        IF MSG.RESULT /= 0 THEN DO.RESULT;                              00070400
        ELSE                                                            00070500
        CASE MSG.TYPE;                                                  00070600
         ;                         % 0                                  00070700
         DO.INPUT.MSG;             % 1                                  00070800
         ;                         % 2                                  00070900
         ;                         % 3                                  00071000
         ;                         % 4                                  00071100
         ;                         % 5                                  00071200
        DO;                        % 6                                  00071300
              LSN:=MSG.STA;                                             00071400
              STA.PENDING(LSN):=0;                                      00071500
              TEXT:="SR";                                               00071600
              CONVERT(0,SUBSTR(TEXT,2,1),LSN);                          00071700
              DISPLAY(SUBSTR(TEXT,0,5));                                00071800
        END;                                                            00071900
         ;                         % 7                                  00072000
        DO;                        % 8                                  00072100
              LLN:=MSG.LINE;                                            00072200
              LINE.PENDING(LLN):=0;                                     00072300
              TEXT:="LR";                                               00072400
              CONVERT(0,SUBSTR(TEXT,2,1),LLN);                          00072500
              DISPLAY(SUBSTR(TEXT,0,5));                                00072600
        END;                                                            00072700
         ;                         % 9                                  00072800
         ;                         % 10                                 00072900
         ;                         % 11                                 00073000
         ;                         % 12                                 00073100
         ;                         % 13                                 00073200
         ;                         % 14                                 00073300
        DO;                                                             00073400
           TELL.SPO:=TRUE;        %ASSUME REPLY TO SPO - MAY BE CHANGED 00073500
           SPO.MSG:=TRUE;                                               00073600
           DO.DC.INPUT;            % 15                                 00073700
        END;                                                            00073800
         ;                         % 16                                 00073900
         ;                         % 17                                 00074000
         DO.ATTACH.SUBN;           % 18                                 00074100
         DO.ATTACH.STA;            % 19                                 00074200
         ;                         % 20                                 00074300
         ;                         % 21                                 00074400
         ;                         % 22                                 00074500
         ;                         % 23                                 00074600
         ;                         % 24                                 00074700
         DO.TASK.DETACH;           % 25                                 00074800
        END CASE;                                                       00074900
        RELEASE.MESSAGE.SPACE(MSG);                                     00075000
```

```
         IF QUIT THEN UNDO;   %SET BY OPERATOR                          00075100
      END MAIN.LOOP;                                                    00075200
    DISPLAY("MCS HALTED");                                              00075300
      STOP;                                                             00075400
      END OLDWEIRDHAROLD;                                               00075500
      FINI;                                                             00075600
$PAGE                                                                   00075610
    FILE.DEFAULT(P):=TYPE2;                                             00075700
    NO.BUFFERS(P):=6;                                                   00075800
    NO.LABEL(P):=1;                                                     00075900
    FILL TIPE.SEG WITH                                                  00076000
    9*,9*,9*,9*,9*,9*,9*,9*,9*,9*,                                      00076100
    9*,9*,9*,9*,9*,9*,9*,9*,9*,9*,                                      00076200
    9*,9*,9*,9*,9*,9*,9*,9*,9*,9*,                                      00076300
    9*,9*,                                                              00076400
    6*,                                                                 00076500
    5*,5*,5*,5*,5*,5*,5*,5*,5*,5*,                                      00076600
    5*,5*,5*,5*,5*,                                                     00076700
    1*,1*,1*,1*,1*,1*,1*,1*,1*,1*,                                      00076800
    5*,5*,5*,5*,5*,5*,5*,                                               00076900
    0*,0*,0*,0*,0*,0*,0*,0*,0*,0*,                                      00077000
    0*,0*,0*,0*,0*,0*,0*,0*,0*,0*,                                      00077100
    0*,0*,0*,0*,0*,0*,                                                  00077200
    5*,5*,5*,5*,5*,5*,                                                  00077300
    0*,0*,0*,0*,0*,0*,0*,0*,0*,0*,                                      00077400
    0*,0*,0*,0*,0*,0*,0*,0*,0*,0*,                                      00077500
    0*,0*,0*,0*,0*,0*,                                                  00077600
    5*,5*,5*,5*,5*;                                                     00077700
                                                                        00077800
    FILL VERB.SEG WITH                                                  00077900
    "END @0000FF@",            %     STOP MCS                           00078000
    "RS  @00001FF@",           %     READY STATION                      00078100
    "RL  @00002FF@",           %     READY LINE                         00078200
    "TO  @0003000@",           %     SEND A MESSAGE                     00078300
    "SS  @0003000@",           %     SAME AS TO                         00078400
    "SD  @0004000@",           %     START DEBUG PRINT                  00078500
    "ED  @0005000@",           %     END DEBUG PRINT                    00078600
    "QM  @0006000@",           %     QUEUE MESSAGE                      00078700
    "ZIP @0007000@",           %   ZIP MESSAGE TO MCP                   00078800
    "WRU @0008000@",           %   WHO ARE YOU                          00078900
    "WM  @0008000@";           %   WHAT MCS - SAME AS WRU               00079000
```

SAMPLE DATA COMM TASKS

Three sample data comm tasks are included in this appendix. These tasks are functionally equivalent, each being coded in a different language: COBOL, MPLII, and RPG. Because of the nature of RPG, it is not practical to describe the tasks in terms of line numbers; much of the task is transparent to the user.

Functional Description

Upon initiation, the DC task opens a printer file (used for logging messages) and obtains the name of the subnet queue from which it obtains messages.

MPLII

The symbolic queue name is accepted from the SPO.

COBOL

The symbolic queue name is taken from the initiating message if present; otherwise, it is accepted from the SPO. The DC task then:

1. Receives a message.
2. Logs the contents of the input CD.
3. Prints the message text.

4. Echoes "good" messages to the originating station.
5. Logs the output CD if a message was echoed.
6. Reports any errors on the SPO.
7. Returns to step 1.

Detailed Description

Program Logic

The logic of the program proceeds as follows: (see MPLII 29400 - 30200, COBOL 23800 - 28200.)

1. Open the printer file.
2. Find the symbolic queue name to be used for DC input.
3. Turn messages around until end-of-job.
4. Close the printer file.
5. Stop.

GET.QUEUE.NAME

(See MPLII 10500 - 11200, COBOL 10900 - 11600.)

This routine results in the symbolic queue name to be used to be placed in SYMBOLIC.QUEUE. In the COBOL version, there is code to check if the INITIAL CD has been filled form the EX or ZIP of the task. If the SYMBOLIC-QUEUE field is blank, then the task waits on an ACCEPT. In the MPLII version, there are no provisions for handling an initiating message. (There is, of source, no reason why this could not be done.)

Turnaround

(See MPLII 27600 - 28800, COBOL 26200 - 27100.)

The logic proceeds as follows:

1. Take the next message from the (input) queue.
2. If the message is good, send it to the originating station.
3. Print the contents of the input CD and the message received.
4. If the input message is good (implying that it has been sent back), print the contents of the output CD.
5. Report any errors found in this transaction.

NOTE
If steps 2 and 3 are interchanged, the program takes longer to turn a message around. Coded as it is, the program causes the message to be sent to the remote device in parallel with the printing of the input CD and message. The reason for this is that the printing operation requires many communicates, each causing the program to be short-waited if the printer buffers are full. The SEND, however, is only a single communicate and the program regains control before the physical data comm transfer is complete (allowing the print communicates to be issued).

Program readability has been improved by the use of DEFINES in MPLII (seq. 8400 - 1010) and condition names in COBOL (seq. 8800 - 9100 and 9800 - 10300). However, a large amount of S-code is generated for DC constructs in MPLII and the result is that the DEFINES used in this example cause the program to be much larger than if FUNCTION(s) and PROCEDURE(s) had been used to encode DC constructs.

GET.MESSAGE

(See MPLII 11300-12600, COBOL 11700-13300.)

The logic proceeds as follows:

1. Space fill the area used to contain the next message.
2. Set-up the required fields in the input CD.
3. Take the next message from the subnet queue.
4. Set the EOJ flag to true if the first three characters of the message are END.

XMIT

(See MPLII 17800 - 19000, COBOL 16000 - 17000).

The logic proceeds as follows:

1. Set-up the output CD.
2. Send the message.
3. Set a flag (OUTPUT.STATUS.VALID) to true.

NOTE
This flag is reset in DISP.ERRORS.OUT after displaying the output status. The flag is used to prevent the same output status being analyzed twice (as would otherwise occur after an error which would have prevented the echoing of a message).

LOG.IN.CD

(See MPLII 13600 - 17700, COBOL 14100 - 16800.)

The logic proceeds as follows:

1. Space fill the print buffer.
2. Place the contents of the input CD into the print buffer.
3. Write the print buffer.
4. If the message is good, print the message text.

NOTE
In the MPLII program, the sub-queue fields in the print buffer are set to spaces and printed in order to preserve the same print format as the COBOL program.

LOG.OUT.CD

(See MPLII 19100 - 20300, COBOL 18000 - 19000.)

This procedure copies the contents of the output CD to the print buffer and then writes this buffer.

ANALYZE.ERRORS (See MPLII 26800 - 27500, COBOL 25400 - 21600.)

This procedure displays errors messages on the SPO and waits for an operator reply. If no errors were encountered during this transaction, this procedure is a no-op.

The actual work is done by DISP.ERRORS.IN (MPLII 20400-22700, COBOL 19100-21600) and DISP.ERRORS.OUT (MPLII 22800-26700, COBOL 21700-25300). If the operator replies with an END in the next turnaround cycle, the program goes to end of job.

SAMPLE COBOL PROGRAM

```
000100$ LINE-CODE OPTCODE
000105**********************************************************************
000110*                                                                    *
000115*                    PROPRIETARY PROGRAM MATERIAL                    *
000120*                                                                    *
000125*     THIS MATERIAL  IS  PROPRIETARY  TO  BURROUGHS  CORPORATION *
000130* AND  IS NOT  TO  BE REPRODUCED, USED  OR  DISCLOSED  EXCEPT IN *
000135* ACCORDANCE WITH PROGRAM LICENCE  OR UPON WRITTEN AUTHORIZATION *
000140* OF  THE  PATENT DIVISION  OF  BURROUGHS CORPORATION,  DETROIT, *
000145* MICHIGAN 48232.                                                    *
000150*                                                                    *
000155*            COPYRIGHT (C) 1979 BURROUGHS CORPORATION               *
000160*                                                                    *
000165**********************************************************************
000200 IDENTIFICATION DIVISION.
000300 ENVIRONMENT DIVISION.
000400 INPUT-OUTPUT SECTION.
000500 FILE-CONTROL.
000600     SELECT LOG ASSIGN TO PRINTER.
000700 DATA DIVISION.
000800 FILE SECTION.
000900 FD LOG.
001000 01 PRINT-LINE PIC X(120).
001100*---THE ABOVE LINE IS ONLY FOR SPACE FILLING.
001200 01 LP-IN.
001300     02 LP-SYMBOLIC-QUEUE PIC X(12).
001400     02 FILLER PIC X.
001500     02 LP-SUB-Q-1 PIC X(12).
001600     02 FILLER PIC X.
```

```
001700          02 LP-SUB-Q-2 PIC X(12).
001800          02 FILLER PIC X.
001900          02 LP-SUB-Q-3 PIC X(12).
002000          02 FILLER PIC X.
002100          02 LP-MESSAGE-DATE PIC 99/99/99.
002200          02 FILLER PIC X.
002300          02 LP-MESSAGE-TIME PIC 99/99/99/99.
002400          02 FILLER PIC X.
002500          02 LP-SYMBOLIC-SOURCE PIC X(12).
002600          02 FILLER PIC X.
002700          02 LP-TEXT-LENGTH-IN PIC ZZZZ9.
002800          02 FILLER PIC X.
002900          02 LP-END-KEY PIC 9.
003000          02 FILLER PIC X.
003100          02 LP-STATUS-KEY-IN PIC 99.
003200          02 FILLER PIC X.
003300          02 LP-MESSAGE-COUNT PIC ZZZZZZ9.
003400*         THE ABOVE RECORD ALLOWS LOGGING THE INPUT CD.
003500 01 BUFFER-LINE PIC X(80).
003600*         THIS LINE IS TO DISPLAY RECEIVED TEXT.
003700 01 LP-OUT.
003800          02 LP-DEST-COUNT PIC ZZZZ9.
003900          02 FILLER PIC X.
004000          02 LP-TEXT-LENGTH-OUT PIC ZZZZ9.
004100          02 FILLER PIC X.
004200          02 LP-STATUS-KEY-OUT PIC 99.
004300          02 FILLER PIC X.
004400          02 LP-ERROR-KEY-OUT PIC 9.
004500          02 FILLER PIC X.
004600          02 LP-SYMBOLIC-DESTINATION PIC X(12).
004700*         THE ABOVE RECORD ALLOWS LOGGING THE OUTPUT CD.
004800 WORKING-STORAGE SECTION.
004900 01 DC-BUFFER.
005000          02 DC-SLOT PIC X(80) OCCURS 24 TIMES.
005100*         DC BUFFER FORMATTED AS TO SCREEN.
005200          02 DC-BUF-REDEF REDEFINES DC-SLOT.
005300              03 DC-MESSAGE PIC XXX.
005400*             TO ALLOW ACCESS TO THE "END" MESSAGE.
005500              03 DC-REST PIC X(1917).
005600 01 COMM-ERROR.
005700          02 TYPE-FIELD PIC X(15).
005800          02 ERROR-FIELD PIC 99.
005900          02 COMMENT-FIELD PIC X(45).
006000 77 PRINT-LINES PIC 99 VALUE 0.
006100 77 SPARE-CHARACTERS PIC 99 VALUE 0.
006200 77 LINE-POINTER PIC 99 VALUE 0.
006300 77 END-FLAG PIC X VALUE "R".
006400 88 EOJ VALUE "S".
006500 77 OUT-STATUS-VALID PIC X VALUE "F".
006600 88 OUTPUT-STATUS-VALID VALUE "T".
006700 COMMUNICATION SECTION.
006800 CD  INPUT-CD FOR INITIAL INPUT.
006900*
007000*         NOTE "INITIAL" CLAUSE.
007100*         IF AN INITIATING MESSAGE IS INCLUDED IN THE "EX" OR "ZIP"
```

```
007200*    OF THIS TASK, THEN THIS CD AREA WILL BE OVERWRITTEN WITH THE
007300*    FIRST 87 CHARACTERS OF THE TEXT. THE PROGRAM DOES *NOT* SPACE
007400*    FILL THE SUB-QUEUE FIELDS. MCP WILL RETURN STATUS 20 (ACCESS
007500*    DENIED) IF SUB-QUEUE FIELDS ARE NON-SPACE.
007600*
007700 01 IN-CD.
007800     02 SYMBOLIC-QUEUE PIC X(12). % APPROPRIATE NDL "FILE" NAME.
007900     02 SUB-Q-1 PIC X(12) VALUE "            ".
008000     02 SUB-Q-2 PIC X(12) VALUE "            ".
008100     02 SUB-Q-3 PIC X(12) VALUE "            ".
008200     02 MESSAGE-DATE PIC 9(6).
008300     02 MESSAGE-TIME PIC 9(8).
008400     02 SYMBOLIC-SOURCE PIC X(12).
008500     02 TEXT-LENGTH-IN PIC 9(4).
008600     02 END-KEY PIC 9.
008700     02 STATUS-KEY-IN PIC 99.
008800 88 GOOD-INPUT-STATUS VALUE 00.
008900 88 UNKNOWN-INPUT VALUE 20.
009000 88 MCS-MISSING VALUE 91.
009100 88 KNOWN-INPUT-ERRORS VALUES 20 91.
009200     02 MESSAGE-COUNT PIC 9(6).
009300 CD OUTPUT-CD FOR OUTPUT.
009400 01 OUT-CD.
009500     02 DESTINATION-COUNT PIC 9(4) VALUE 1.
009600     02 TEXT-LENGTH-OUT PIC 9(4).
009700     02 STATUS-KEY-OUT PIC 99.
009800 88 GOOD-OUTPUT-STATUS VALUE 00.
009900 88 UNKNOWN-OUTPUT VALUE 20.
010000 88 BAD-DESTINATION-COUNT VALUE 30.
010100 88 BAD-TEXT-LENGTH VALUE 50.
010200 88 DCSS-MISSING VALUE 91.
010300 88 KNOWN-OUTPUT-ERRORS VALUES 20 30 50 91.
010400     02 ERROR-KEY PIC 9.
010500     02 SYMBOLIC-DESTINATION PIC X(12).
010600 PROCEDURE DIVISION.
010700 MAIN.
010800     GO TO START-OF-PROGRAM. % MPL-LIKE PROGRAM LAYOUT
010900 GET-QUEUE-NAME.
011000*
011100*    GETS SUBNETQUEUE NAME FROM OPERATOR (IF NOT IN INIT MESSAGE).
011200*    THIS NAME MUST BE DEFINED IN THE NDL PROGRAM FILE.
011300*
011400     IF SYMBOLIC-QUEUE IS EQUAL TO SPACES % = NO INIT MESSAGE
011500         THEN DISPLAY "TYPE INPUT-QUEUE NAME..."
011600         ACCEPT SYMBOLIC-QUEUE.
011700 GET-MESSAGE.
011800*
011900*    SPACE FILL DATA-COMM BUFFER, TAKE NEXT MESSAGE FROM SUBNET Q,
012000*    SET EOJ FLAG IF END RECEIVED.
012100*
012200     MOVE SPACES TO
012300         SYMBOLIC-SOURCE
012400         END-KEY
012500         STATUS-KEY-IN.
012600     MOVE 0 TO
```

```
012700           MESSAGE-DATE
012800           MESSAGE-TIME
012900           TEXT-LENGTH-IN
013000           MESSAGE-COUNT.
013100       MOVE SPACES TO DC-BUFFER.
013200       RECEIVE INPUT-CD MESSAGE INTO DC-BUFFER.
013300       IF DC-MESSAGE = "END" MOVE "S" TO END-FLAG.
013400 WRITE-LINES.
013500*------------------------------------------------------------------
013600*      WRITE INFORMATION FROM LAST MESSAGE RECEIVED TO PRINTER.
013700*------------------------------------------------------------------
013800       MOVE SPACES TO PRINT-LINE.
013900       WRITE BUFFER-LINE FROM DC-SLOT(LINE-POINTER) AFTER 1.
014000       ADD 1 TO LINE-POINTER.
014100 LOG-IN-CD.
014200*------------------------------------------------------------------
014300*      WRITE CONTENTS OF CURRENR INPUT CD TO PRINTER.
014400*------------------------------------------------------------------
014500       MOVE SPACES TO PRINT-LINE.
014600       MOVE SYMBOLIC-QUEUE TO LP-SYMBOLIC-QUEUE.
014700       MOVE SUB-Q-1 TO LP-SUB-Q-1.
014800       MOVE SUB-Q-2 TO LP-SUB-Q-2.
014900       MOVE SUB-Q-3 TO LP-SUB-Q-3.
015000       MOVE MESSAGE-DATE TO LP-MESSAGE-DATE.
015100       MOVE MESSAGE-TIME TO LP-MESSAGE-TIME.
015200       MOVE SYMBOLIC-SOURCE TO LP-SYMBOLIC-SOURCE.
015300       MOVE TEXT-LENGTH-IN TO LP-TEXT-LENGTH-IN.
015400       MOVE END-KEY TO LP-END-KEY.
015500       MOVE STATUS-KEY-IN TO LP-STATUS-KEY-IN.              Text-lenght-in
015600       ACCEPT INPUT-CD MESSAGE COUNT.                       -------------- =
015700       MOVE MESSAGE-COUNT TO LP-MESSAGE-COUNT.                    80
015800       WRITE LP-IN AFTER 2.                                  = Print-line
015900       IF NOT GOOD-INPUT-STATUS                              Rest =
016000           GO TO LOG-IN-CD-END. = EXIT                       Spare-Charaters.
016100       DIVIDE 80 INTO TEXT-LENGTH-IN GIVING PRINT-LINES
016200           REMAINDER SPARE-CHARACTERS.
016300       MOVE 1 TO LINE-POINTER.
016400       PERFORM WRITE-LINES UNTIL LINE-POINTER > PRINT-LINES.
016500       IF SPARE-CHARACTERS IS NOT = 0
016600           PERFORM WRITE-LINES.
016700 LOG-IN-CD-END.
016800       EXIT.
016900 XMIT.
017000*------------------------------------------------------------------
017100*      SEND CURRENT MESSAGE BACK TO ORIGINATOR.
017200*      MARK OUTPUT STATUS AS NOT HAVING BEEN ANALYSED.
017300*------------------------------------------------------------------
017400       MOVE TEXT-LENGTH-IN TO TEXT-LENGTH-OUT.
017500       MOVE SYMBOLIC-SOURCE TO SYMBOLIC-DESTINATION.
017600       MOVE SPACES TO STATUS-KEY-OUT
017700           ERROR-KEY.
017800       SEND OUTPUT-CD FROM DC-BUFFER WITH EMI.
017900       MOVE "T" TO OUT-STATUS-VALID.
018000 LOG-OUT-CD.
018100*------------------------------------------------------------------
```

```
018200*    WRITE THE CONTENTS OF THE CURRENT OUTPUT CD TO PRINTER.
018300*-----------------------------------------------------------------------
018400     MOVE SPACES TO PRINT-LINE.
018500     MOVE DESTINATION-COUNT TO LP-DEST-COUNT.
018600     MOVE TEXT-LENGTH-OUT TO LP-TEXT-LENGTH-OUT.
018700     MOVE STATUS-KEY-OUT TO LP-STATUS-KEY-OUT.
018800     MOVE ERROR-KEY TO LP-ERROR-KEY-OUT.
018900     MOVE SYMBOLIC-DESTINATION TO LP-SYMBOLIC-DESTINATION.
019000     WRITE LP-OUT AFTER 1.
019100 DISP-ERRORS-IN.
019200*-----------------------------------------------------------------------
019300*    REPORT TO OPERATOR ON ERRORS FROM LAST INPUT FROM DATA-COMM.
019400*-----------------------------------------------------------------------
019500     MOVE "RECEIVE  ERROR " TO TYPE-FIELD.
019600     MOVE STATUS-KEY-IN TO ERROR-FIELD.
019700     IF UNKNOWN-INPUT
019800         MOVE " (QUEUE UNKNOWN OR ACCESS DENIED)"
019900         TO COMMENT-FIELD.
020000     IF MCS-MISSING
020100         MOVE " (MCS/DCSS NOT PRESENT)"
020200         TO COMMENT-FIELD.
020300     IF NOT KNOWN-INPUT-ERRORS
020400         MOVE " (UNKNOWN ERROR)"
020500         TO COMMENT-FIELD.
020600     DISPLAY COMM-ERROR.
020700     IF UNKNOWN-INPUT
020800         MOVE SPACES TO SYMBOLIC-QUEUE
020900         PERFORM GET-QUEUE-NAME.
021000     IF MCS-MISSING
021100         DISPLAY "INITIATE A SUITABLE MCS THEN ""AX"" THIS TASK"
021200         ACCEPT DC-MESSAGE.
021300     IF NOT KNOWN-INPUT-ERRORS
021400         DISPLAY "PROGRAM ERROR - ""DP"" THIS TASK"
021500         ACCEPT DC-MESSAGE
021600         STOP RUN.
021700 DISP-ERRORS-OUT.
021800*-----------------------------------------------------------------------
021900*    REPORT TO OPERATOR ON ERRORS FROM LAST OUTPUT TO DATA-COMM.
022000*-----------------------------------------------------------------------
022100     MOVE "TRANSMIT ERROR" TO TYPE-FIELD.
022200     MOVE STATUS-KEY-OUT TO ERROR-FIELD.
022300     IF UNKNOWN-OUTPUT
022400         MOVE " (STATION UNKNOWN OR ACCESS DENIED)"
022500         TO COMMENT-FIELD.
E  022600     IF BAD-DESTINATION-COUNT      MOVE "(BAD DESTINATION COUNT)
022800         TO COMMENT-FIELD.
022900     IF BAD-TEXT-LENGTH
023000         MOVE " (REQUIRED TEXT-LENGTH > DC-BUFFER SIZE)"
023100         TO COMMENT-FIELD.
023200     IF DCSS-MISSING
023300         MOVE " (MCS/DCSS NOT PRESENT)"
023400         TO COMMENT-FIELD.
023500     IF NOT KNOWN-OUTPUT-ERRORS
023600         MOVE " (UNKNOWN-ERROR)"
023700         TO COMMENT-FIELD.
```

```
023800     DISPLAY COMM-ERROR.
023900     IF UNKNOWN-OUTPUT
024000         DISPLAY "CORRECT, THEN ""AX"" THIS TASK"
024100         ACCEPT DC-MESSAGE.
024200     IF BAD-DESTINATION-COUNT OR NOT KNOWN-OUTPUT-ERRORS
024300         DISPLAY "PROGRAM ERROR - ""DP"" THIS TASK"
024400         ACCEPT DC-MESSAGE
024500         STOP RUN.
024600     IF BAD-TEXT-LENGTH
024700         DISPLAY "STATION IS NOT TD830 - SELECT ANOTHER QUEUE"
024800         MOVE SPACES TO SYMBOLIC-QUEUE
024900         PERFORM GET-QUEUE-NAME.
025000     IF DCSS-MISSING
025100         DISPLAY "INITIATE A SUITABLE MCS THEN ""AX"" THIS TASK"
025200         ACCEPT DC-MESSAGE.
025300     MOVE "F" TO OUT-STATUS-VALID.
025400 ANALYSE-ERRORS.
025500*
025600*    REPRT ERRORS IF ANY.
025700*
025800     IF NOT GOOD-INPUT-STATUS
025900         PERFORM DISP-ERRORS-IN.
026000     IF NOT GOOD-OUTPUT-STATUS AND OUTPUT-STATUS-VALID
026100         PERFORM DISP-ERRORS-OUT.
026200 TURNAROUND.
026300*
026400*    TAKE NEXT MESSAGE AND SEND IT BACK FROM WHENCE IT CAME
026500*    IF ERROR FREE.
026600*
026700     PERFORM GET-MESSAGE.
026800     IF GOOD-INPUT-STATUS PERFORM XMIT. % XMIT *BEFORE* LOG ALLOWS
026900     PERFORM LOG-IN-CD THRU LOG-IN-CD-END. % PRINTER AND OUTPUT-DC
027000     IF GOOD-INPUT-STATUS PERFORM LOG-OUT-CD. % TO INTERLEAVE
027100     PERFORM ANALYSE-ERRORS.
027200 START-OF-PROGRAM.
027300*****
027400*****
027500******* START OF PROGRAM *****************************************
027600*****
027700*****
027800     OPEN OUTPUT LOG.
027900     PERFORM GET-QUEUE-NAME.
028000     PERFORM TURNAROUND UNTIL EOJ.
028100     CLOSE LOG RELEASE.
028200     STOP RUN.
```

SAMPLE RPG PROGRAM

```
00001F*****************************************************************
00002F*                                                               *
00003F*                  PROPRIETARY PROGRAM MATERIAL                 *
00004F*                                                               *
00005F*     THIS MATERIAL  IS  PROPRIETARY  TO  BURROUGHS  CORPORATION *
00006F* AND  IS NOT  TO  BE REPRODUCED, USED  OR  DISCLOSED  EXCEPT IN *
00007F* ACCORDANCE WITH PROGRAM LICENCE  OR UPON WRITTEN AUTHORIZATION *
00008F* OF  THE  PATENT DIVISION  OF  BURROUGHS CORPORATION,  DETROIT, *
00009F* MICHIGAN 48232.                                               *
00010F*                                                               *
00011F*           COPYRIGHT (C) 1979 BURROUGHS CORPORATION            *
00012F*                                                               *
00013F*****************************************************************
00015FKEY     IP.    80  80            KEYBORD
00020F*
00030F*
00040F*     FILEO IS THE SYMBOLIC QUEUE NAME . THIS NAME MUST BE DEFINED IN THE
00050F*     FILE SECTION OF THE NDLSYS FILE TO BE USED.
00060F*     THIS FILE IS FURTHER DESCRIBED ON THE T-SPEC.
00070F*     THE FILE IS DEFINED AS COMBINED DEMAND. THIS MEANS THAT A MESSAGE
00080F*     CAN BE RECEIVED AND  TRANSMITTED IN THE SAME CYCLE.
00090F*
00100F*
00110FFILEO   CD    132 132            DATACOM
00120FOUTPUT  O     132 132            PRINTER
00130T*
00140T*
00150T*     THE DATA COMMUNICATION SPECIFICATION (T-SPEC) FURTHER DESCRIBES THE
00160T*     FILE DEFINED ON THE F-SPEC.
00170T*     THE ENTRY OF T IN COLUMN 16 DEFINES THAT THE FILE CAN TRANSMIT AND
00180T*     RECEIVE (NOTE - AN ENTRY OF R WOULD MEAN THE SAME WHEN THE FILE IS
00190T*     COMBINED).
00200T*     THE ENTRY OF S IN COLUMN 40 DEFINES THAT THE FIELD NAME IN COLUMNS
00210T*     41-47 WILL CONTAIN THE STATION NAME. THE STATION NAME IS THE
00220T*     SYMBOLIC SOURCE, AND THE STATION NAME MUST BE DEFINED IN THE STATION
00230T*     SECTION OF THE NDLSYS FILE TO BE USED.
00240T*     THE ENTRY OF 01 IN COLUMNS 53-54 IS THE ERROR INDICATOR. THIS
00250T*     INDICATOR WILL BE SET ON WHEN ANY OF THE ERRORS LISTED ON THE
00260T*     O-SPECS HAS OCCURED.
00270T*     THE ENTRY OF S IN COLUMNS 63 DEFINES THAT THE FIELD NAME IN COLUMNS
00280T*     64-70 WILL CONTAIN THE MESSAGE LENGTH OF ANY MESSAGES RECEIVED OR
00290T*     TRANSMITTED.
00300T*
00310T*
00320TFILEO    T                       SSTAT        01        SMESS
00330I*
00340I*
00350I*     THE COMBINED DATA COMMUNICATIONS FILE MUST BE FURTHER DESCRIBED ON
00360I*     AN I-SPEC.
00370I*
00380I*
00390IFILEO   NS  05
00400I                                        1  80 ALL
00410C*
00420C*
```

```
00430C*     THE FIELD NAMES STAT AND MESS MUST BE DEFINED IN THE C-SPECS.
00440C*     STAT MUST BE DEFINED AS ALPHANUMERIC AND MESS MUST BE DEFINED AS
00450C*     NUMERIC.
00460C*
00470C*
00480C                        MOVE '        'STAT    8
00490C                        Z-ADD0        MESS    70
00500C*
00510C*
00520C*     THE OPCODE 'READ' IS USED TO RECEIVE MESSAGES FROM AN INPUT OR
00530C*     COBINED DEMAND DATA COMMUNICATIONS FILE.
00540C*     THE INDICATOR NORMALLY SPECIFIED IN COLUMNS 58-59 OF THE C-SPEC IS
00550C*     IGNORED WHEN THE FILE NAME SPECIFIED IN COLUMNS 33-42 IS A DATA
00560C*     COMMUNICATIONS FILE.
00570C*     AFTER THE RECEIVE STAT WILL CONTAIN THE NAME OF THE STATION
00580C*     WHICH TRANSMITTED THE MESSAGE.
00590C*     AFTER THE RECEIVE MESS WILL CONTAIN THE MESSAGE LENGTH OF THE
00600C*     MESSAGE.
00610C*     IF THE RECEIVE FAILED THEN THE ERROR INDICATOR 01 WILL BE SET ON.
00620C*     THE INDICATOR 06 IS USED TO INDICATE A RECEIVE ERROR.
00630C*
00640C*
00650C                        SETOF                     0106
00660C                        READ FILEO
00670C   01                   SETON                     06
00680C*
00690C*
00700C*     WHEN THE MESSAGE CONTAINS 'END', THEN THE PROGRAM GOES TO END OF JOB
00710C*
00720C*
00730C  N01      'END'        COMP ALL                      LR
00740C  N01                   MOVE MESS     MES     70
00750C*
00760C*
00770C*     THE MESSAGE LENGTH OF THE MESSAGE TO BE TRANSMITTED BACK MUST BE
00780C*     MOVED INTO MESS, OTHERWISE THE MESSAGE LENGTH OF THE RECEIVED
00790C*     MESSAGE WILL BE USED.
00800C*
00810C*     NOTE - 80 CHARACTERS ARE EXPECTED TO BE RECEIVED (THE FIELD 'ALL' IS
00820C*     80 BYTES LONG). IF THE OPERATOR TRANSMITS 80 CHARACTERS THE FIRST
00830C*     TIME AND THEN 70 CHARACTERS THE SECOND TIME THEN THE SECOND MESSAGE
00840C*     WILL CONTAIN THE LAST 10 CHARACTERS OF THE FIRST MESSAGE. THE BUFFER
00850C*     CANNOT BE CLEARED, THEREFORE AN ERROR MESSAGE SHOULD BE DISPLAYED IF
00860C*     THE MESSAGE TRANSMITTED IS LESS THAN 80 CHARACTERS LONG. THE
00870C*     OPERATOR SHOULD BE DIRECTED TO RE-TRANSMIT 80 CHARACTERS.
00880C*     THE ALTERNATIVE IS TO TAKE ACCOUNT OF THE MESSAGE LENGTH
00890C*     PROGRAMMATICALLY AND PROCESS THE MESSAGE ACCORDINGLY.
00900C*
00910C*
00920C  N01                   Z-ADD132      MESS
00930C  N01      COUNT        ADD  1        COUNT   50
00940C*
00950O*
00960O*     THE COMBINED DATA COMMUNICATIONS FILE MUST BE FURTHER DESCRIBED ON
00970O*     AN O-SPEC.
```

```
009800*
009900*     IF THE ERROR INDICATOR IS SET OFF THEN THE RECEIVED MESSAGE IS
010000*     ECHOED BACK TO THE STATION  THAT TRANSMITTED THE MESSAGE.
010100*
010200*
010300FILEO   D       N01    05
010400*
010500*
010600*     @OC@ IS 'CLEAR AND HOME' - THE SCREEN IS CLEARED OF THE LAST MESSAGE
010700*
010800*
010900                                         1 @OC@
011000                        ALL       81
011100                                  69 ' MESSAGE '
011200                        COUNT     95
011300                                 119 ' RECEIVED FROM STATION '
011400                        STAT     127
011500     D       N01    05
011600                                  24 ' WITH MESSAGE LENGTH OF '
011700                        MES       31
011800                                  55 ' AND SENT TO STATION '
011900                        STAT      63
012000                                  87 ' WITH MESSAGE LENGTH OF '
012100                        MESS      94
012200*
012300*
012400*     IF THE ERROR INDICATOR IS SET ON THEN AN ERROR LIST IS PRINTED.
012500*     THESE ERRORS CANNOT BE DETECTED BY THE MCS.
012600*     THE RPG PROGRAM WILL ONLY SET ON AN ERROR INDICATOR AND THEREFORE
012700*     CANNOT  DETERMINE WHICH OF THE POSSIBLE ERRORS HAVE ACTUALLY
012800*     OCCURED. ALL THE POSSIBLE ERRORS ARE THEREFORE LISTED AND THE
012900*     OPERATOR IS DIRECTED TO INVESTIGATE FURTHER.
013000*
013100*
013200*     RECEIVE ERROR.
013300*
013400*
013500OUTPUT  D 11     06
013600                                  24 '*** ERROR ***   EITHER -'
013700                                  48 'FILE ON F-SPEC DOES NOT '
013800                                  72 'CORRESPOND TO FILE SECTI'
013900                                  96 'ON OF NDLSYS, OR THERE I'
014000                                 120 'S NO MCS EXECUTING --- I'
014100                                 130 'NVESTIGATE'
014200*
014300*
014400*     TRANSMIT ERROR.
014500*
014600*
014700OUTPUT  D 11     01
014800                                  24 '*** ERROR ***   EITHER -'
014900                                  48 ' STATION NAME ON T-SPEC '
015000                                  72 'DOES NOT CORRESPOND TO S'
015100                                  96 'TATION SECTION OF NDLSYS'
015200                                 120 ', OR MESSAGE LENGTH EXCE'
```

```
015300                                   123 'EDS'
015400      D 11      01
015500                                    24 'RECORD LENGTH ON F-SPEC,'
015600                                    48 ' OR THERE IS NO MCS EXEC'
015700                                    69 'UTING --- INVESTIGATE'
015800*
015900*
016000*    IF THE ERROR INDICATOR IS SET OFF THEN THE RECEIVED MESSAGE IS
016100*    PRINTED ON THE LINE PRINTER.
016200*
016300*
016400      D 2      N01 05
016500                          ALL       80
016600                                    90 'MESSAGE '
016700                          COUNT     95
016800                                   119 ' RECEIVED FROM STATION '
016900                          STAT     127
017000      D 1      N01 05
017100                                    24 ' WITH MESSAGE LENGTH OF '
017200                          MES       31
017300                                    55 ' AND SENT TO STATION '
017400                          STAT      63
017500                                    87 ' WITH MESSAGE LENGTH OF '
017600                          MESS      94
```

```
$CONTROL 200 DATA 1500                                                   00001000
$FORMAT                                                                  00001100
$NOLIST                                                                  00001200
$DATACOM                                                                 00001300
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00001305
%                                                                       %00001310
%                PROPRIETARY PROGRAM MATERIAL                           %00001315
%                                                                       %00001320
%     THIS MATERIAL IS PROPRIETARY TO BURROUGHS CORPORATION  AND IS     %00001325
%  NOT TO BE REPRODUCED, USED OR DISCLOSED EXCEPT IN ACCORDANCE WITH    %00001330
%  PROGRAM LICENCE OR UPON WRITTEN AUTHORIZATION OF THE PATENT          %00001335
%  DIVISION OF BURROUGHS CORPORATION, DETROIT, MICHIGAN 48232.          %00001340
%                                                                       %00001345
%             COPYRIGHT (C) 1979 BURROUGHS CORPORATION                  %00001350
%                                                                       %00001355
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00001360
   PROCEDURE OUTER;                                                      00001400
   DEFINE CH £CHARACTER£;                                                00001500
   DECLARE PRINT.LINE CH(120);                                           00001600
   FILE LOG WORK.AREA PRINT.LINE;                                        00001700
%                                                                        00001800
%  PRINT BUFFER STUFF.                                                   00001900
%                                                                        00002000
   REMAP PRINT.LINE:                                                     00002100
   01 LP.IN,                                                             00002200
        02 LP.SYMBOLIC.QUEUE     CH(12),                                 00002300
        02 DUMMY                 CH(1),                                  00002400
        02 LP.SUB.Q.1            CH(12),                                 00002500
        02 DUMMY                 CH(1),                                  00002600
        02 LP.SUB.Q.2            CH(12),                                 00002700
        02 DUMMY                 CH(1),                                  00002800
        02 LP.SUB.Q.3            CH(12),                                 00002900
        02 DUMMY                 CH(1),                                  00003000
        02 LP.MESSAGE.DATE       CH(8),                                  00003100
        02 DUMMY                 CH(1),                                  00003200
        02 LP.MESSAGE.TIME       CH(11),                                 00003300
        02 DUMMY                 CH(1),                                  00003400
        02 LP.SYMBOLIC.SOURCE    CH(12),                                 00003500
        02 DUMMY                 CH(1),                                  00003600
        02 LP.TEXT.LENGTH.IN     CH(5),                                  00003700
        02 DUMMY                 CH(1),                                  00003800
        02 LP.END.KEY            CH(1),                                  00003900
        02 DUMMY                 CH(1),                                  00004000
        02 LP.STATUS.KEY.IN      CH(2),                                  00004100
        02 DUMMY                 CH(1),                                  00004200
        02 LP.MESSAGE.COUNT      CH(7);                                  00004300
   REMAP PRINT.LINE:                                                     00004400
   BUFFER.LINE                   CH(80);                                 00004500
   REMAP PRINT.LINE:                                                     00004600
   01 LP.OUT                     CH(80),                                 00004700
        02 LP.DEST.COUNT         CH(5),                                  00004800
        02 DUMMY                 CH(1),                                  00004900
        02 LP.TEXT.LENGTH.OUT    CH(5),                                  00005000
        02 DUMMY                 CH(1),                                  00005100
        02 LP.STATUS.KEY.OUT     CH(2),                                  00005200
```

```
         02 DUMMY                  CH(1),                                00005300
         02 LP.ERROR.KEY.OUT       CH(1),                                00005400
         02 DUMMY                  CH(1),                                00005500
         02 LP.SYMBOLIC.DESTINATION  CH(12);                             00005600
%                                                                        00005700
%   DATA-COMM BUFFER STUFF                                               00005800
%                                                                        00005900
    SEGMENT DC.BUFFER(1920);                                             00006000
    REMAP DC.BUFFER:                                                     00006100
    DC.SLOT                        CH(80);                               00006200
    REMAP DC.BUFFER:                                                     00006300
    DC.MESSAGE                     CH(3),                                00006400
    DC.REST (1917)                 CH(1);                                00006500
%                                                                        00006600
%   OTHER GLOBAL GOODIES                                                 00006700
%                                                                        00006800
    DECLARE                                                              00006900
    01 COMM.ERROR,                                                       00007000
         02 TYPE.FIELD             CH(15),                               00007100
         02 ERROR.FIELD            CH(2),                                00007200
         02 COMMENT.FIELD          CH(45),                               00007300
    PRINT.LINES                    FIXED,                                00007400
    LINE.POINTER                   FIXED,                                00007500
    SYMBOLIC.QUEUE                 CH(12),                               00007600
    END.FLAG                       CH(1);                                00007700
    DEFINE EOJ £(END.FLAG="S")£;                                         00007800
    END.FLAG:="R";                                                       00007900
    DECLARE                                                              00008000
    OUT.STATUS.VALID               CH(1);                                00008100
    DEFINE OUTPUT.STATUS.VALID £(OUT.STATUS.VALID="T")£;                 00008200
    OUT.STATUS.VALID:="F";                                               00008300
%                                                                        00008400
%   DEFINES FOR ERROR HANDLING.... DO NOT DO THIS IN REAL LIFE.          00008500
%   AS IT GENERATES A LOT OF S-CODE.... USED HERE AS ILLUSTRATION ONLY.  00008600
%                                                                        00008700
    DEFINE                                                               00008800
    GOOD.INPUT.STATUS         £(DC.INPUT.STATUS=0)£,                     00008900
    UNKNOWN.INPUT             £(DC.INPUT.STATUS=20)£,                    00009000
    MCS.MISSING               £(DC.INPUT.STATUS=91)£,                    00009100
    KNOWN.INPUT.ERRORS        £(UNKNOWN.INPUT OR MCS.MISSING)£,          00009200
    GOOD.OUTPUT.STATUS        £(DC.OUTPUT.STATUS=0)£,                    00009300
    UNKNOWN.OUTPUT            £(DC.OUTPUT.STATUS=20)£,                   00009400
    BAD.DESTINATION.COUNT     £(DC.OUTPUT.STATUS=30)£,                   00009500
    BAD.TEXT.LENGTH           £(DC.OUTPUT.STATUS=50)£,                   00009600
    DCSS.MISSING              £(DC.OUTPUT.STATUS=91)£,                   00009700
    KNOWN.OUTPUT.ERRORS       £(UNKNOWN.OUTPUT OR                        00009800
                                BAD.DESTINATION.COUNT OR                 00009900
                                BAD.TEXT.LENGTH OR                       00010000
                                DCSS.MISSING)£;                          00010100
%                                                                        00010200
%                                                                        00010300
%                                                                        00010400
    PROCEDURE GET.QUEUE.NAME;                                            00010500
%                                                                        00010600
%   GETS SUBNET QUEUE NAME FROM OPERATOR. THIS NAME MUST BE              00010700
```

```
%   DEFINED IN THE NDL PROGRAM FILE SECTION.                        00010800
%                                                                   00010900
    DISPLAY("TYPE INPUT QUEUE NAME...");                            00011000
    ACCEPT(SYMBOLIC.QUEUE);                                         00011100
    END GET.QUEUE.NAME;                                             00011200
    PROCEDURE GET.MESSAGE;                                          00011300
%                                                                   00011400
%   SPACE-FILL DATA-COMM BUFFER, TAKE NEXT MESSAGE FROM SUBNET Q,   00011500
%   SET EOJ FLAG IF END RECEIVED.                                   00011600
%                                                                   00011700
    DECLARE I FIXED;                                                00011800
    DO MOVE.SPACES.TO.DC.BUFFER FOREVER;                            00011900
         DC.SLOT(I*80):="";                                         00012000
         IF [I:+1] >= 24 THEN UNDO;                                 00012100
    END MOVE.SPACES.TO.DC.BUFFER;                                   00012200
    DC.RECEIVE(SYMBOLIC.QUEUE,DC.SLOT,1920);                        00012300
    IF DC.MESSAGE = "END" THEN END.FLAG:="S";                       00012400
    END GET.MESSAGE;                                                00012600
    PROCEDURE WRITE.LINES;                                          00012700
%                                                                   00012800
%   WRITE INFORMATION FROM LAST MESSAGE RECEIVED TO PRINTER         00012900
%                                                                   00013000
    PRINT.LINE:="";                                                 00013100
    BUFFER.LINE:=DC.SLOT(LINE.POINTER);                             00013200
    WRITE(LOG,BEFORE,LINE);                                         00013300
    LINE.POINTER:+80;                                               00013400
    END WRITE.LINES;                                                00013500
    PROCEDURE LOG.IN.CD;                                            00013600
%                                                                   00013700
%   WRITE CONTENTS OF CURRENT INPUT CD TO PRINTER                   00013800
%                                                                   00013900
    DECLARE                                                         00014000
    TEMP                         CH(8),                             00014100
    F.TEMP                        FIXED;                            00014200
    PRINT.LINE:="";                                                 00014300
    LP.SYMBOLIC.QUEUE:=SYMBOLIC.QUEUE;                              00014400
    DO MOVE.DATE;                                                   00014500
         TEMP:=DC.DATE;                                             00014600
         IF TEMP = "" THEN LP.MESSAGE.DATE:="00/00/00";             00014610
         ELSE                                                       00014620
         DO;                                                        00014630
         SUBSTR(LP.MESSAGE.DATE,0,2):=SUBSTR(TEMP,0,2);             00014700
         SUBSTR(LP.MESSAGE.DATE,2,1):="/";                          00014800
         SUBSTR(LP.MESSAGE.DATE,3,2):=SUBSTR(TEMP,2,2);             00014900
         SUBSTR(LP.MESSAGE.DATE,5,1):="/";                          00015000
         SUBSTR(LP.MESSAGE.DATE,6,2):=SUBSTR(TEMP,4,2);             00015100
         END;                                                       00015110
    END MOVE.DATE;                                                  00015200
    DO MOVE.TIME;                                                   00015300
    TEMP:=DC.TIME;                                                  00015400
         IF TEMP = "" THEN LP.MESSAGE.TIME:="00/00/00/00/";         00015410
         ELSE                                                       00015420
         DO;                                                        00015430
         SUBSTR(LP.MESSAGE.TIME,0,2):=SUBSTR(TEMP,0,2);             00015500
         SUBSTR(LP.MESSAGE.TIME,2,1):="/";                          00015600
```

```
          SUBSTR(LP.MESSAGE.TIME,3,2):=SUBSTR(TEMP,2,2);                00015700
          SUBSTR(LP.MESSAGE.TIME,5,1):="/";                             00015800
          SUBSTR(LP.MESSAGE.TIME,6,2):=SUBSTR(TEMP,4,2);                00015900
          SUBSTR(LP.MESSAGE.TIME,8,1):="/";                             00016000
          SUBSTR(LP.MESSAGE.TIME,9,2):=SUBSTR(TEMP,6,2);                00016100
          END;                                                          00016110
     END MOVE.TIME;                                                     00016200
     LP.SYMBOLIC.SOURCE:=DC.ORIGIN;                                     00016300
     CONVERT(1,LP.TEXT.LENGTH.IN,DC.TEXTLENGTH);                        00016400
     CONVERT(1,LP.END.KEY,DC.ENDKEY);                                   00016500
     CONVERT(0,LP.STATUS.KEY.IN,DC.INPUT.STATUS);                       00016600
     DC.ACCEPT(SYMBOLIC.QUEUE,F.TEMP);                                  00016700
     CONVERT(1,LP.MESSAGE.COUNT,F.TEMP);                                00016800
     WRITE(LOG,BEFORE,LINE(2));                                         00016900
     IF NOT GOOD.INPUT.STATUS THEN RETURN;                              00017000
     LINE.POINTER:=0;                                                   00017100
     F.TEMP:=DC.TEXTLENGTH;                                             00017200
     DO FOREVER;                                                        00017300
          IF LINE.POINTER >= F.TEMP THEN UNDO;                          00017400
          WRITE.LINES;                                                  00017500
     END;                                                               00017600
     END LOG.IN.CD;                                                     00017700
     PROCEDURE XMIT;                                                    00017800
%                                                                       00017900
%    SEND CURRENT MESSAGE BACK TO ORIGINATOR,                           00018000
%    MARK OUTPUT STATUS AS NOT HAVING BEEN ANALYSED.                    00018100
%                                                                       00018200
     DECLARE                                                            00018300
     SYMBOLIC.DESTINATION          CH(12),                              00018400
     TEXT.LENGTH.OUT               FIXED;                               00018500
     SYMBOLIC.DESTINATION:=DC.ORIGIN;                                   00018600
     TEXT.LENGTH.OUT:=DC.TEXTLENGTH;                                    00018700
     DC.SEND(SYMBOLIC.DESTINATION,DC.SLOT,TEXT.LENGTH.OUT,EMI);         00018800
     OUT.STATUS.VALID:="T";                                             00018900
     END XMIT;                                                          00019000
     PROCEDURE LOG.OUT.CD;                                              00019100
%                                                                       00019200
%    WRITE THE CONTENTS OF THE CURRENT OUTPUT CD TO PRINTER             00019300
%    SYMBOLIC SUB.QS AND DEST. COUNT SET BY COMPILER                    00019400
%                                                                       00019500
     PRINT.LINE:="";                                                    00019600
     LP.DEST.COUNT:="    1";                                            00019700
     CONVERT(1,LP.TEXT.LENGTH.OUT,DC.TEXTLENGTH);                       00019800
     CONVERT(0,LP.STATUS.KEY.OUT,DC.OUTPUT.STATUS);                     00019900
     CONVERT(1,LP.ERROR.KEY.OUT,DC.ERRORKEY);                           00020000
     LP.SYMBOLIC.DESTINATION:=DC.ORIGIN;                                00020100
     WRITE(LOG,BEFORE,LINE);                                            00020200
     END LOG.OUT.CD;                                                    00020300
     PROCEDURE DISP.ERRORS.IN;                                          00020400
%                                                                       00020500
%    REPORT TO OPERATOR ON ERRORS FROM LAST INPUT FROM DATA-COMM        00020600
%                                                                       00020700
     TYPE.FIELD:="RECEIVE ERROR";                                       00020800
     CONVERT(0,ERROR.FIELD,DC.INPUT.STATUS);                            00020900
     IF UNKNOWN.INPUT THEN                                              00021000
```

```
          COMMENT.FIELD:=" (QUEUE UNKNOWN OR ACCESS DENIED)";                00021100
     IF MCS.MISSING THEN                                                      00021200
          COMMENT.FIELD:=" (MCS/DCSS NOT PRESENT)";                          00021300
     IF NOT KNOWN.INPUT.ERRORS THEN                                           00021400
          COMMENT.FIELD:=" (UNKNOWN ERROR)";                                 00021500
     DISPLAY(COMM.ERROR);                                                     00021600
     IF UNKNOWN.INPUT THEN                                                    00021700
     DO;                                                                      00021800
          SYMBOLIC.QUEUE:="";                                                 00021900
          GET.QUEUE.NAME;                                                     00022000
     END;                                                                     00022100
     IF MCS.MISSING THEN                                                      00022200
     DO;                                                                      00022300
          DISPLAY("INITIATE A SUITABLE MCS THEN ""AX"" THIS TASK");           00022400
          ACCEPT(DC.MESSAGE);                                                 00022500
     END;                                                                     00022600
     END DISP.ERRORS.IN;                                                      00022700
     PROCEDURE DISP.ERRORS.OUT;                                               00022800
%                                                                             00022900
%    REPORT TO OPERATOR ON ERRORS FROM LAST OUTPUT TO DATA-COMM               00023000
%                                                                             00023100
     TYPE.FIELD:="TRANSMIT ERROR";                                            00023200
     CONVERT(0,ERROR.FIELD,DC.OUTPUT.STATUS);                                 00023300
     IF UNKNOWN.OUTPUT THEN                                                   00023400
          COMMENT.FIELD:=" (STATION UNKNOWN OR ACCESS DENIED)";              00023500
     IF BAD.DESTINATION.COUNT THEN                                            00023600
          COMMENT.FIELD:=" (INVALI DESTINATION COUNT)";                      00023700
     IF BAD.TEXT.LENGTH THEN                                                  00023800
          COMMENT.FIELD:=" (REQUIRED TEXT LENGTH > DC.BUFFER SIZE)";         00023900
     IF DCSS.MISSING THEN                                                     00024000
          COMMENT.FIELD:=" (MCS/DCSS NOT PRESENT)";                          00024100
     IF NOT KNOWN.OUTPUT.ERRORS THEN                                          00024200
          COMMENT.FIELD:=" (UNKNOWN ERROR)";                                 00024300
     DISPLAY(COMM.ERROR);                                                     00024400
     IF UNKNOWN.OUTPUT THEN                                                   00024500
     DO;                                                                      00024600
          DISPLAY("CORRECT. THEN ""AX"" THIS TASK");                          00024700
     END;                                                                     00024800
     IF BAD.DESTINATION.COUNT THEN                                            00024900
     DO;                                                                      00025000
          DISPLAY("PROGRAM ERROR - ""DP"" THIS TASK");                        00025100
          ACCEPT(DC.MESSAGE);                                                 00025200
          STOP;                                                               00025300
     END;                                                                     00025400
     IF BAD.TEXT.LENGTH THEN                                                  00025500
     DO;                                                                      00025600
          DISPLAY("STATION IS NOT TD830 - SELECT ANOTHER QUEUE");             00025700
          SYMBOLIC.QUEUE:="";                                                 00025800
          GET.QUEUE.NAME;                                                     00025900
     END;                                                                     00026000
     IF DCSS.MISSING THEN                                                     00026100
     DO;                                                                      00026200
          DISPLAY("INITIATE A SUITABLE MCS THEN ""AX"" THIS TASK");           00026300
          ACCEPT(DC.MESSAGE);                                                 00026400
     END;                                                                     00026500
```

```
         OUT.STATUS.VALID:="F";                                                00026600
         END DISP.ERRORS.OUT;                                                  00026700
         PROCEDURE ANALYSE.ERRORS;                                             00026800
    %                                                                          00026900
    %    REPORT ERRORS IF ANY                                                  00027000
    %                                                                          00027100
         IF NOT GOOD.INPUT.STATUS THEN DISP.ERRORS.IN;                         00027200
         IF NOT GOOD.OUTPUT.STATUS AND OUTPUT.STATUS.VALID THEN                00027300
              DISP.ERRORS.OUT;                                                 00027400
         END ANALYSE.ERRORS;                                                   00027500
         PROCEDURE TURNAROUND;                                                 00027600
    %                                                                          00027700
    %    TAKE NEXT MESSAGE AND SEND IT BACK FROM WHENCE IT CAME IF ERROR FREE00027800
    %                                                                          00027900
         GET.MESSAGE;                                                          00028000
         IF GOOD.INPUT.STATUS THEN XMIT;                                       00028010
         LOG.IN.CD;                                                            00028100
         IF GOOD.INPUT.STATUS THEN LOG.OUT.CD;                                 00028200
         ANALYSE.ERRORS;                                                       00028700
         END TURNAROUND;                                                       00028800
    %%%                                                                        00028900
    %%%                                                                        00029000
    %%%%%     START.OF.PROGRAM                                                 00029100
    %%%                                                                        00029200
    %%%                                                                        00029300
         OPEN(LOG);                                                            00029400
         GET.QUEUE.NAME;                                                       00029500
         DO FOREVER;                                                           00029600
              IF EOJ THEN UNDO;                                                00029800
              TURNAROUND;                                                      00029900
         END;                                                                  00030000
         CLOSE(LOG);                                                           00030100
         STOP;                                                                 00030200
         END OUTER;                                                            00030300
         FINI;                                                                 00030400
         FILE.DEFAULT(LOG):=TYPE2;                                             00030500
         RECORD(LOG):=120;                                                     00030600
         BUFFER(LOG):=120;                                                     00030700
```

SAMPLE NDL PROGRAM

The sample NDL program provides control of two lines, both using Burroughs asynchronous poll/select line discipline. One line provides "host poll/select" (that is, the channel polls and selects remote terminals to solicit input and route output), and the other line provides "terminal poll/select" (that is, the channel is polled and selected by a remote host to control message transfer).

The poll/select line discipline is a multipoint procedure. The central host system solicits input from (polls). Each remote station in turn uses the following poll message:

EOT AD1 AD2 POL ENQ

where EOT, POL, and ENQ are predefined line control characters, and AD1 and AD2 are address characters identifying one of the remote stations. The remote station replies with EOT if no message is available (whereupon the polling system proceeds to the next station on that line), or sends the message using the following format:

SOH AD1 AD2 STX <text> ETX BCC

where SOH, STX, and ETX are line control characters, and BCC is a block check character computed from the <text> portion of the message. (The host system calculates its own BCC when receiving the message for comparison with the transmitted BCC to validate the message transfer.) If the message is received correctly, the polling system sends ACK, and the remote station completes the transaction with EOT. If the message is received incorrectly, the polling system sends NAK, whereupon a subsequent poll is needed to retry the message.

The host system directs output to (selects) a remote station using the following select message;

EOT AD1 AD2 SEL ENQ

where EOT, SEL, and ENQ are line control characters and AD1 and AD2 are address characters identifying one of the remote stations. The remote station replies ACK or NAK, depending on its ability to receive the message at that time. If the remote station sends ACK, the central system sends the message in the format described for a response to a poll. The remote station then sends ACK or NAK as determined by the BCC computation. The transaction is then complete, and is retried by a subsequent select if the message was NAK'ed. If the remote station was unable to receive the message (that is, it sent NAK to the select message), then a subsequent select is needed to retry the message.

An alternative method of selecting may be used if the central system is confident that the remote station is capable of receiving a message. This method is known as fast select, and omits the initial ACK or NAK response to the select message. The formatted text message follows immediately after the fast select message as follows:

EOT AD1 AD2 FSL SOH AD1 AD2 STX <text> ETX BCC

where FSL is the fast select control character. In all other respects, the transaction is identical to a normal select.

NOTE

To improve readability in the following description, a dot(.) has been included in references to sequence numbers. For example, seq. 7001.0300 refers to sequence number 70010300.

The Implementation

The sample NDL program provides host poll/select on physical channel 5 and terminal poll/select on physical channel 6. (The B 80 was a one-digit channel number for line address; see CMS NDL Manual.) The line descriptions are at seq. 8700.xxxx and 8800.xxxx. The host poll/select line is described first, starting from the line description, and then the differences required for the terminal poll/select line are noted. The host poll/select line is identified to the system as logical line number zero (LLNO) since it is the first line declared in the program.

The Host Line

The following describes the NDL code for the host poll/select line.

Line Section

The line address is defined at sequence 8700.0100.

The stations on the line are defined at seq. 45900. All stations on the line must be defined as terminals using the same control sets, and have the same communication hardware (sync. or async.). All stations on a line must communicate at the same speed.

Station Section

Seq. 8200.xxxx - 8500.xxxxx describe the stations defined for LLNO. Each station description is identical (except for the address characters); therefore, only STATIONO (seq. 8200.xxxx) is described. The use of a DEFAULT station reduces the source file size (see the NDL Reference Manual).

STATIONO is identified to the system as logical station number 0 (LSNO) since it has the lowest identifier in alphabetic order of station identifiers.

Seq. 8200.0100 defines the @ character (HEX 40) as the control character for the station. The control character is detected by an input request RECEIVE statement specifying CONTROL (seq. 6000.2900). If the character is detected by the RECEIVE, the message is routed to the MCS unconditionally.

ENABLEINPUT is set true for this station; otherwise, the station is not polled. ENABLEINPUT is set false for an output-only station, unless the MCS is designed to explicitly set terminals ENABLEINPUT as part of a network startup procedure.

The FREQUENCY statement presets a read-only value which the NDL programmer may use to control the frequency at which a station is polled. The control set POLL (seq. 3000.xxxx) uses this value to control the relative polling rate of each station.

The LOGIN statement resets bit 14 of the MCS data field in the message header. If LOGIN is set to true, then bit 14 of the MCS data field is set. The use of this flag is entirely at the discretion of the

MCS programmer (for example, to enable the MCS to enter a log-in routine for the terminal operator).

The mandatory statement MYUSE specifies the communication requirements of the station. If MYUSE is not output or INPUT,OUTPUT, then the system returns an error result to the MCS (UNABLE TO INITIATE) for output messages. Setting ENABLEINPUT to true causes a syntax error if MYUSE is OUTPUT only. This station is declared INPUT,OUTPUT to permit both polling and selecting of the station.

The RETRY statement specifies an initial value of 10, to which the run-time variable retry is set (by data comm load, terminate normal, terminate block, terminate error, and initialize retry). Note that, in an output request, the run-time value is set to the retry value in the message header of the output message, unless the message header specifies 255 (hex FF). In this case, the RETRY statement value is used. (This is B 80 implementation only.) The maintenance of retry counts and the declaring of errors is the responsibility of the NDL programmer through the control and request set logic. The value of RETRY must be determined empirically, since the configuration, line speed, and type of connection affect the integrity of messages. A value of zero should be avoided.

The WIDTH and WRAPAROUND statements define values which may be interrogated by the MCS, and have no effect on the NDL program.

The TERMINAL statement associates the station with a terminal (physical device) description. In this case, the description is of a TD 830 display terminal (seq. 8100.xxxx). Corresponding characteristics defined in the STATION and TERMINAL sections must be compatible.

The actual address characters which identify the station in message transfers (1A for this station) are defined in the ADDRESS statement. The number of characters must correspond to the associated terminal address statement.

The TYPE statement (seq. 8200.1200) selects parameters from a list provided in the TERMINAL TYPE statement (seq. 8100.1500). The statement is not required if the terminal TYPE statement defines only one set of parameters.

Terminal Section

Seq. 8100.xxxx describes the one terminal which has been associated with all the stations on LLNO. It is possible that the network contains physical devices having slightly different characteristics, in which case, suitable terminal descriptions can be added to the NDL program. These terminal descriptions can then be associated with the selected stations via the station TERMINAL statement. Since all stations on a line must be identical in certain characteristics (notably SPEED, TYPE, and CONTROL), a DEFAULT terminal defining the common characteristics can be used. Each terminal description then refers to the default terminal for its major characteristics, leaving only the variants to be described individually.

The ADDRESS statement specifies the number of characters which constitute the terminal address.

The SPEED statement declares a range of speeds from which the station SPEED statement must select one value.

The TURNAROUND statement assigns a 12 millisecond transmit delay to the procedures used for stations of this type, since the line is direct-connect and no modem values are available.

The TIMEOUT statement assigns a one second timeout value which is used when no explicit value is applied to CONTROL and REQUEST RECEIVE statements. The one second timeout is used, for example, at seq. 6000.2600 and 6000.3200.

The CONTROL statement associates the line control procedure POLL with stations of this type.

The REQUEST statement associates the receive and transmit requests POLLIT and SELECTIT with stations of this type.

The MAXINPUT statement (seq. 37500) defines the amount of buffer space required by GETSPACE and RECEIVE TEXT statements in input requests for stations of this type. A message with a text length greater than 1920 characters is rejected by the request POLLIT (see seq. 6000.3100; ENDOFBUFFER error action causes excess characters to be discarded).

The BLOCKED statement informs the DCSS that this device is not capable of sending or receiving blocked messages.

END, BACKSPACE, LINEDELETE, and WRU define the format control characters for this device. These characters must be specified if referenced in the associated CONTROL or REQUEST sets. No action is taken on receipt of these characters unless the CONTROL or REQUEST set references the identifier. For example, seq. 6000.3100 compares for the literal character ETX (a pre-defined constant). If the possibility exists that different terminals using the request POLLIT have alternative end-of-text characters, then the statement should be recoded:

RECEIVE TEXT [1, END, ENDOFBUFFER : 7]

(where [and] are left and right square brackets respectively) which achieve the same result using the END character defined for each terminal.

The TYPE statement defines the connection requirements for this terminal.

The BYTE statement declares the character size and parity requirements for this terminal. All stations on a line must have the same character size.

SCREEN, HOME, CLEAR, CARRIAGE, LINEFEED, and WRAPAROUND define values which may be interrogated by the MCS, and have no effect on the NDL program.

Control and Request

The line control POLL, receive request POLLIT, and transmit request SELECTIT are assigned to the stations on the line through the associated terminal TD 830. Line execution starts at seq. 3000.4200. Line control initiates the output request (SELECTIT) at seq. 3000.6100. The INITIATE is not performed (behaves as a no-op) unless a message is queued for the current station (that is, the station indicated by the current value of the NDL variable STATION). At the completion of the output request, line control restarts at sequence 3000.4200.

If no message for output is queued, line control tests (via FREQUENCY) whether the current station is due for polling (seq 3000.6800 - 3001.0100). The input requests POLLIT is initiated at seq. 3000.9800. Line control restarts at seq. 3000.4200 after completion of the input request; otherwise, a branch to seq. 3000.4700 is taken.

Seq. 3001.0600 initializes the station index to MAXSTATIONS when all stations on the line have been serviced, allowing the control set to handle different line configurations. Note that the system does not initialize STATION to any particular value; therefore, considering this, line control must be coded. Also, the occurrence of PAUSE and DELAY statements give processor time to interrupts from other lines.

Execution of the input request starts at seq. 6000.1100. The transmission of the poll message can be seen at seq. 6000.1200 - 6000.1300. The request identifies the receipt of a text message by detection of SOH at seq. 600.2000. An explicity GETSPACE is included to allow the use for an error recovery procedure. Seq. 6000.2800 ensures that the buffer pointer is set to the beginning of the buffer. Seq 6000.2900 provides for detection of the stations control character. Note that this character is not implicitly stored in the text buffer. This is performed by seq. 6000.3000. A successfully received message is passed to the MCP via the TERMINATE NORMAL statement at 6000.3900. Note that this initializes the run-time value of RETRY; whereas, the explicit INITIALIZE RETRY is required on detection of EOT prior to the TERMINATE NOINPUT (seq. 6000.4300). Seq. 6000.4900 - 6000.5100 maintain the retry count, and declare an error when the retry count is exhausted. Seq. 6000.4500 - 6000.4600 "flush" the line when errors are encountered (the exit from this infinite loop is via the TIMEOUT coded in error switch 1). Seq. 6000.4700 provides for the detection of continuous carrier, to prevent infinite flushing of the line.

Execution of the output request set starts at seq. 7000.0600. The select message sequence can be seen at seq. 7000.0700 - 7000.1200. Seq. 7000.2300 resets the buffer pointer to the start of the text buffer. Note that this is an entirely different buffer from the buffer in the receive request described earlier. A TERMINATE ENABLEINPUT at this point (initiating the receive request) causes all text buffer references to apply to the input buffer. Successful termination of the request (7000.3300) causes the output buffer space to be returned to the data comm buffer pool. The select sequence is discontinued in favor of the input request (7000.6500) if a NAK is received to the select, implying that the terminal is transmit ready and therefore unable to receive.

The Terminal Line

The coding for the line supporting terminal poll/select has the following differences from the previously described line code.

Line Section

The physical line address is channel 6.

The station on the line uses different CONTROL and REQUEST sets (declared via the associated terminal).

This line is modem connected via a modem (dataset) whose physical characteristics are defined in the MODEM TA713 description (seq. 8000.xxxx).

Station Section (8600.xxxx)

RETRY -The retry value is greater (100) since the host normally has responsibility for discontinuing the transaction.

TERMINAL -The station is associated with a terminal using the terminal poll/select control and request sets.

TYPE.MODEM -The station is connected to the line using a modem whose physical characteristics are defined in the MODEM TA713 declaration (seq. 8000.xxxx).

Terminal Section (8010.xxxx)

The line uses terminal poll/select control and request sets. Note that USELECTED is the receive request, and UPOLLED is the transmit request.

This is a modem connected terminal.

Modem Section (000.xxxx)

The transmit delay, receive delay, type, and speed of the modem used (in this case) at both ends of the line are defined.

Control and Request

The line control UPOLL, receive request USELECTED, and transmit request UPOLLED are assigned to the station on the line through the associated terminal TD830X4.

Line control execution starts at seq. 2000.2000

Line control validates the control message until the sequence is recognized as a poll, a select, or a fast select (2000.2300 - 2000.4600) whereupon the receive (2001.0400) or or transmit (2000.8500) request is initiated as appropriate. Note that the no-message-available EOT is transmitted by line control as a result of INITIATE REQUEST (seq. 200.9300) behaving as a no-op when the station is not queued. TOG[0] is used as a fast select indicator to skip the portions of code not required when using that protocol.

The input and output requests have no significant differences from the previously described request sets, except that the inverse side of the procedure is handled.

DCP Section

The values for BUFFER and BUFFERCOUNT are selected to give a total buffer allocation of 15 Kb.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00000100
%                                                                       %00000200
%                  PROPRIETARY PROGRAM MATERIAL                         %00000300
%                                                                       %00000400
%      THIS MATERIAL IS PROPRIETARY TO BURROUGHS CORPORATION  AND IS    %00000500
%  NOT TO BE REPRODUCED, USED OR DISCLOSED EXCEPT IN ACCORDANCE WITH    %00000600
%  PROGRAM LICENCE OR UPON WRITTEN AUTHORIZATION OF THE PATENT          %00000700
%  DIVISION OF BURROUGHS CORPORATION, DETROIT, MICHIGAN 48232.          %00000800
%                                                                       %00000900
%               COPYRIGHT (C) 1979 BURROUGHS CORPORATION                %00001000
%                                                                       %00001100
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%00001200
$SET LIST CODE NTCH                                                      00010000
CONSTANT·                                                                00010100
          EOT = 4"04",                                                   00010200
          SOH = 4"01",                                                   00010300
          STX = 4"02",                                                   00010400
          ACK = 4"06",                                                   00010500
          NAK = 4"15",                                                   00010600
          ETX  = 4"03",                % END OF TEXT                     00010700
          ENQ  = 4"05",                % ENQUIRE                         00010800
          BS   = 4"08",                % BACKSPACE                       00010900
          LF   = 4"0A",                % LINE FEED                       00011000
          CR   = 4"0D",                % CARRIAGE RETURN                 00011100
          DC4  = 4"14",                % DEVICE CONTROL 4                00011200
          DEL  = 4"7F",                % DELETE                          00011300
```

```
          FF   = 4"0C",                  %    FORMS FEED                 00011400
          POL  = 4"70",                  %    POLL CHARACTER             00011500
          SEL  = 4"71",                  %    SELECT CHARACTER           00011600
          FSL  = 4"73".                                                  00011700
%                                                                        00100000
%       NOTE THAT THESE REQUESTS DO NOT CONFORM EXACTLY                  00100100
%       TO ANY BURROUGHS STANDARD AND ALTHOUGH THESE REQUESTS            00100200
%       WILL FUNCTION CORRECTLY THEY SHOULD NOT BE REGARDED              00100300
%       AS THE ONLY OR EVEN THE "BEST" POSSIBLE SETS.                    00100400
%                                                                        00100500
CONTROL  UPOLL:                                                          20000000
%                                                                        20000100
%    **********************************************                      20000200
%          I AM A POLLED TERMINAL ......                                 20000300
%    *****************************************************               20000400
%                                                                        20000500
%          TOG[0] = INDICATOR FOR SELECT, FASTSELECT.                    20000600
%          TOG[1] = USED IN USELECTED FOR NOSPACE CONDITION.             20000700
%                                                                        20000800
%    **********************************************                      20000900
%                                                                        20001000
           ERROR[0] = TIMEOUT:1,% STANDARD ERROR MACRO                   20001100
                      STOPBIT:3,                                         20001200
                      BUFOVFL:3,                                         20001300
                      BREAK:3,                                           20001400
                      PARITY:4,                                          20001500
                      LOSSOFCARRIER:1.                                   20001600
%                                                                        20001700
%    **********************************************                      20001800
%                                                                        20001900
%                                                                        20002000
    1: TOG[0] = FALSE.                                                   20002100
       TOG[1] = FALSE.                                                   20002200
    2: INITIATE RECEIVE.                 %WAIT FOR SOMETHING TO          20002300
       RECEIVE (NULL)[0].                %APPEAR ON THE LINE             20002400
%                                                                        20002500
%              WE HAVE SEEN A CHARACTER                                  20002600
%                                                                        20002700
       IF CHAR = EOT                     %IF NOT,ITS NOT FOR US          20002800
%                                        %AND WE FALL THROUGH TO ERROR   20002900
%                                        %HANDLING AT LABEL 3            20003000
%                                                                        20003100
           THEN BEGIN                                                    20003200
                RECEIVE (25 MILLI) ADDRESS [ERROR[0], ADDERR:3].         20003300
%                                        %NOTE THAT AN ADDERR            20003400
%                                        %MERELY MEANS THAT THE          20003500
%                                        %POLL OR SELECT WAS NOT         20003600
%                                        %FOR US                         20003700
%                                                                        20003800
                RECEIVE (25 MILLI) [POL:20, SEL:30, FSL:10,              20003900
                                         ERROR[0]].                      20004000
%                                        %HERE WE USE NDLS BRANCHING     20004100
%                                        %ABILITY TO GO OFF TO THE       20004200
%                                        %APPROPRIATE ROUTINE IF         20004300
%                                        %A POL SEL OR FSL IS RECEIVED   20004400
```

```
%                                                                    20004500
              END.                                                   20004600
%                                                                    20004700
%                       WE DONT WANT THIS MESSAGE                    20004800
%                       SO LOOP ROUND UNTIL LINE                     20004900
%                       GOES IDLE.                                   20005000
%                                                                    20005100
%             WE ARE PROBABLY HERE BECAUSE THE CHAR WAS NOT AN       20005200
%             EOT.                                                   20005300
%                                                                    20005400
%                                                                    20005500
%                                                                    20005600
%                                                                    20005700
%                                                                    20005800
      3: RECEIVE (25 MILLI) [ERROR[0]].                              20005900
         GO TO 3.                         %TIMEOUT GETS US OUT OF LOOP   20006000
%                                         %TO LABEL 1. PARITY (IE JUNK ON 20006100
%                                         %LINE) TO LABEL 4          20006200
%                                                                    20006300
%             NOW WE LOOP WAITING FOR "MARK"                         20006400
%                                                                    20006500
      4: IF CHAR NEQ 4"FF" THEN GO TO 3.                             20006600
%                                                                    20006700
%             ALL QUIET LETS GO AND LISTEN AGAIN                     20006800
%                                                                    20006900
      5: GO TO 1.                                                    20007000
%      **********************************************               20007100
%           FAST SELECT ...........                                 20007200
%      **********************************************               20007300
      10: TOG[0] = TRUE.                  %FLAG FOR USELECTED        20007400
      11: INITIATE ENABLEINPUT.           %ENTER USELECTIT           20007500
%                                                                    20007600
%             WE ONLY GET HERE IF WE DIDNT ENTER USELECTED           20007700
%             THIS WILL ONLY HAPPEN IF THE STATION IS NOT READY      20007800
%             IN THIS CASE THE "STATION" IS OUR HOST.                20007900
%                                                                    20008000
         GO TO 3.                                                    20008100
%      **********************************************               20008200
%           POLLED .............                                    20008300
%      **********************************************               20008400
      20: RECEIVE (25 MILLI) [ ENQ:21,ERROR[0]].                     20008500
%                                                                    20008600
%             ANYTHING OTHER THAN AN ENQ IS AN ERROR SO WE WILL IGNORE20008700
%             THIS POL ALTHOUGH IT WAS FOR US. NOTE THAT THE POL/SEL 20008800
%             DISCIPLINE DOES NOT ALLOW US TO TELL THE HOST THAT     20008900
%             WE SAW HIS POL BUT DIDNT LIKE THE FORMAT.              20009000
%                                                                    20009100
         GO TO 3.                                                    20009200
      21: INITIATE REQUEST.               % WE WILL ENTER UPOLLED    20009300
%                                         %IF WE HAVE A MESSAGE QUEUED   20009400
%                                         %AND THE "STATION" IS READY    20009500
%                                                                    20009600
         INITIATE TRANSMIT.               %NOTHING TO SEND           20009700
         TRANSMIT EOT.                    %SO XMIT EOT               20009800
         FINISH TRANSMIT.                                            20009900
```

```
        GO TO 1.
%   *********************************************
%         SELECT .................
%   *********************************************
    30: RECEIVE (25 MILLI) [ENQ:11,ERROR[0]].
        GO TO 3.
CONTROL POLL:
%
%   I POLL THE TERMINALS ......
%
%
%    VARIABLES USED:-
%      LINE(QUEUED)           SET IF WE DID ANYTHING
%      LINE(TALLY[0])         NO OF TIMES WE HAVE ENTERED LINE
%                             CONTROL SINCE WE LAST CHANGED
%                             LINE(TALLY[1]).
%      LINE(TALLY[1])         CURRENT ACCEPTABLE FREQUENCY.
%      STATION(FREQUENCY)     READ ONLY VALUE
%      STATION(TALLY)         CURRENT FREQUENCY
%
%      ALL OF THESE ARE USED IN THE CODE TO TAKE NOTE OF AND ACT
%      UPON THE VARIOUS VALUES DECLARED BY THE USER IN THE STATION
%      FREQUENCY STATEMENTS.
%      THE MAIN AIM IS TO ALLOW THE USER TO SPECIFY HOW OFTEN STATIONS
%      ARE TO BE POLLED RELATIVE TO EACH OTHER. NOTE THAT SPECIFYING
%      A FREQUENCY DOES NOT DO ANYTHING OTHER THAN PLACE A VALUE
%      IN THE STATION TABLE WHICH IS AVAILABLE TO THE REQUEST/CONTROL
%      SET PROGRAMMER AS STATION(FREQUENCY).
%      A SIMPLE CONTROL SET  WITHOUT FREQUENCY HANDLING FOLLOWS;
%      THE COMMENTS IN THE ACTUAL CONTROL SET APPLY EQUALLY TO
%      THIS SIMPLER SET.
%
%              0: IF STATION >0 THEN
%                     BEGIN
%                         PAUSE.
%                         STATION = STATION - 1.
%                         INITIATE REQUEST.
%                         INITIATE ENABLEINPUT.
%                         GO TO 0.
%                     END.
%                 STATION = MAXSTATIONS.
%                 IDLE.
%
      LINE(TALLY[0]) = LINE(TALLY[0]) + 1.  % BUMP ENTRY COUNTER
      LINE(QUEUED) = TRUE.                 %WE DO WANT TO CHANGE FREQUENCY
%                                          %AT THE END OF THIS CYCLE
%                                          %THROUGH THE STATIONS
%
0:                                         %COME BACK HERE IF WE DO
%                                          %NOTHING FOR THE CURRENT
%                                          %STATION
%
      IF STATION > 0 THEN                  % ARE WE AT THE END OF CYCLE
%                                          %NOTE THAT DUE TO THE LOGIC
%                                          % OF THE LOOP TO LABEL 1 THIS
```

```
%                                     %WILL ONLY BE THE CASE WHEN THE 30005400
%                                     %LAST STATION WE HANDLED WAS    30005500
%                                     %STATION ZERO.                  30005600
     BEGIN                                                            30005700
1:                                                                    30005800
          PAUSE.                      %GIVE SOMEONE ELSE A CHANCE     30005900
          STATION = STATION - 1.      %SET UP NEXT STATION            30006000
          INITIATE REQUEST.           %WILL ENTER SELECTIT IF         30006100
%                                     %STATION IS QUEUED AND READY    30006200
%                                                                     30006300
%      NO OUTPUT FOR THIS TERMINAL.                                   30006400
%                                                                     30006500
% WE NOW DETERMINE IF THIS TERMINAL CAN BE USED FOR INPUT             30006600
%      AND IF SO WHETHER WE WANT TO POLL HIM.                         30006700
%                                                                     30006800
          IF STATION(ENABLED) THEN                                    30006900
          BEGIN                                                       30007000
%                                                                     30007100
%      IS HE AT THE RIGHT FREQUENCY YET?                              30007200
%                                                                     30007300
%                                                                     30007400
%      NOTE THAT THE LOWER THE FREQUENCY THE MORE OFTEN               30007500
%      WE WILL POLL THE STATION.                                      30007600
%                                                                     30007700
               IF STATION(TALLY) GT LINE(TALLY[1]) THEN               30007800
               BEGIN                                                  30007900
                    PAUSE.                                            30008000
                    STATION(TALLY) = STATION(TALLY) - LINE(TALLY[1]). 30008100
%                                     %MAYBE NEXT TIME                30008200
                    LINE(QUEUED) = TRUE.%WE HAD A CANDIDATE WHO DIDNT 30008300
%                                     %HAVE A LOW ENOUGH FREQUENCY    30008400
%                                     %SO WE HAD BETTER MAKE SURE     30008500
%                                     %THAT WE DONT COUNT HIM DOWN    30008600
%                                     %TOO QUICKLY AND UPSET THE      30008700
%                                     %RELATIVE FREQUENCIES BY        30008800
%                                     %COUNTING DOWN IN BIG STEPS     30008900
%                                     %                               30009000
               END                                                    30009100
               ELSE                                                   30009200
               BEGIN                                                  30009300
                    PAUSE.                                            30009400
                    STATION(TALLY) = STATION(FREQUENCY).%GO BACK TO   30009500
%                                     %REAL FREQUENCY TO GIVE OTHER   30009600
%                                     %STATIONS SOME KIND OF CHANCE   30009700
                    INITIATE ENABLEINPUT.  %ENTER POLLIT IF STATION   30009800
%                                     %IS READY VALID ENABLEINPUT     30009900
               END.                                                   30010000
          END.                                                        30010100
          GO TO 0.                    %DIDNT DO ANYTHING FOR THIS     30010200
%                                     %STATION - TRY THE NEXT         30010300
          END.                                                        30010400
     PAUSE.                           %GIVE  ANOTHER LINE A CHANCE    30010500
     STATION = MAXSTATIONS.           %END OF CYCLE                   30010600
     IF LINE(QUEUED) THEN             %WE CHANGE TALLY 1              30010700
     BEGIN                                                            30010800
```

```
          LINE(QUEUED) = FALSE.          % RESET OUR FLAG                 30010900
%                                        %NOTE THAT THE ONLY CYCLE THAT   30011000
%                                        %WILL NOT SET LINE(QUEUED)       30011100
%                                        %IS ONE IN WHICH NOBODY HAD      30011200
%                                        %AN OUTPUT QUEUED AND THERE      30011300
%                                        %WERE NO STATIONS READY FOR      30011400
%                                        %INPUT.IE NOBODY WHO FAILED      30011500
%                                        %BECAUSE OF A TOO HIGH FREQUENCY30011600
%                                        % IF WE DID POLL ANYONE WE       30011700
%                                        %REENTERED LINE CONTROL AT       30011800
%                                        %THE TOP WHERE WE SET            30011900
%                                        %LINE QUEUED                     30012000
          IF LINE(TALLY[0]) = 0 THEN     %WE HAVENT LEFT THE CONTROL      30012100
%                                        %SINCE WE LAST CHANGED TALLY 1   30012200
%                                        %SO WE MAKE SURE THAT WE DONT    30012300
%                                        %OFFEND OUR HIGH PRIORITY        30012400
%                                        %IE LOW FREQUENCY USERS BY       30012500
%                                        % IGNORING THEM. TO DO THIS      30012600
%                                        % WE MAKE THE QUALIFYING VALUE   30012700
%                                        %ONE AND COUNT STATIONS DOWN     30012800
%                                        %SLOWLY WHICH MEANS THAT THE     30012900
%                                        %FIRST STATION(S) GET LOTS OF    30013000
%                                        %EXTRA POLLS.                    30013100
          BEGIN                                                           30013200
               LINE(TALLY[1]) = 1.                                        30013300
               LINE(BUSY) = FALSE.       %ALLOW SYSTEM TO REENTER US      30013400
%                                        %FOR THIS LINE IF ANYTHING       30013500
%                                        %HAPPENS                         30013600
               DELAY(1 SEC).             %GO TO SLEEP                     30013700
               LINE(BUSY) = TRUE.        %NOBODY WANTED US SO ON WE GO    30013800
          END                                                             30013900
          ELSE                                                            30014000
          BEGIN                                                           30014100
               LINE(TALLY[1]) = LINE(TALLY[0]). %RAISE OR LOWER FREQ      30014200
               LINE(TALLY[0]) = 0.       %DEPENDING ON HOW LONG SINCE     30014300
%                                        %WE LAST DID (IE HOW LONG WE     30014400
%                                        %HAVE BEEN POLLING WITHOUT       30014500
%                                        %SELECTING ANYONE).              30014600
          END.                                                            30014700
          GO TO 1.                                                        30014800
     END.                                                                 30014900
IDLE.                                                                     30015000
%                                                                         30015100
%      WE ONLY IDLE THE LINE IF LINE(QUEUED) IS FALSE                     30015200
%    THIS WILL ONLY BE THE CASE IF THERE ARE NO OUTPUT MESSAGES           30015300
%    QUEUED FOR ANY OF THE STATIONS ON THIS LINE AND NONE OF              30015400
%    THE STATIONS ARE ENABLED INPUT AND OR READY.                         30015500
%      IN THIS CASE THE ONLY WAY ANYTHING WILL CHANGE IS WHEN THE MCS     30015600
%    DOES SOMETHING OR A MESSAGE IS QUEUED FOR ONE OF THE STATIONS.       30015700
%    IN EITHER CASE NDL.INTERP WILL START US UP AGAIN SO WE MAY AS        30015800
%    MARK THE LINE IDLE SINCE WE ARE DOING NOTHING EXCEPT WASTE           30015900
%    PROCESSOR RESOURCES.                                                 30016000
%                                                                         30016100
%        NOTE THAT THE FREQUENCY HANDLING CODE WILL RUN INEFFICIENTLY     30016200
%        IF THERE ARE NO ACTIVE STATIONS WITH LOW FREQUENCIES.            30016300
```

```
%       THIS WILL LEAD TO LOW POLLING RATES AS THE CODE WILL              30016400
%       GO ROUUND THE CYCLE <LOWEST DECLARED FREQUENCY> TIMES             30016500
%       BEFORE ACTUALLY ENTERING THE INPUT REQUEST. THIS NOT ONLY         30016600
%       AFFECTS THIS LINE BUT ALSO ANY OTHER ON THIS DCP SINCE THE        30016700
%       DCP IS INVOLVED IN TIME CONSUMING USELESS PROCESSING.             30016800
%           THE DEFAULT FREQUENCY IS ZERO (ALWAYS POLL) SO THIS WILL      30016900
%       ONLY BE A PROBLEM IF SOMEONE MAKES A MESS OF THE STATION          30017000
%       DECLARATIONS.                                                     30017100
%                                                                         30017200
REQUEST  UPOLLED:                                                         40000000
%                                                                         40000100
%    ***********************************************                      40000200
%          I WAS POLLED  .............                                    40000300
%    *****************************************************                40000400
%                                                                         40000500
           ERROR[0] = TIMEOUT:20,                                         40000600
                      STOPBIT:20,                                         40000700
                      BUFOVFL:20,                                         40000800
                      PARITY:20,                                          40000900
                      BREAK:20,                                           40001000
                      LOSSOFCARRIER:20.                                   40001100
%                                                                         40001200
%    ***********************************************                      40001300
%                                                                         40001400
%                                                                         40001500
      1: INITIATE TRANSMIT.                                               40001600
         TRANSMIT  SOH.                                                   40001700
         INITIALIZE BCC.                                                  40001800
         TRANSMIT ADDRESS.                                                40001900
         TRANSMIT TRAN.                                                   40002000
         TRANSMIT STX.                                                    40002100
         INITIALIZE TEXT.                                                 40002200
         TRANSMIT TEXT.                                                   40002300
         TRANSMIT ETX.                                                    40002400
         TRANSMIT BCC.                                                    40002500
         FINISH TRANSMIT.                                                 40002600
         INITIATE RECEIVE.                                                40002700
         RECEIVE (1 SEC) [ACK:10,NAK:12,ERROR[0]].                        40002800
         GO TO 20.                                                        40002900
%                                                                         40003000
     10: INCREMENT TRAN.                                                  40003100
         INITIATE TRANSMIT.                                               40003200
         TRANSMIT EOT.                                                    40003300
         FINISH TRANSMIT.                                                 40003400
         TERMINATE NORMAL.                                                40003500
%                                                                         40003600
     12: NAKFLAG = TRUE.                                                  40003700
     20: IF RETRY = 0  THEN TERMINATE ERROR.                              40003800
         RETRY = RETRY - 1.                                               40003900
         TERMINATE NOINPUT.                                               40004000
REQUEST  USELECTED:                                                       50000000
%                                                                         50000100
%    *****************************************************                50000200
%          I WAS SELECTED  .............                                  50000300
```

```
%    ****************************************************           50000400
%          TOG[0] = 0 = SELECT.                                    50000500
%                   1 = FAST SELECT.                               50000600
%          TOG[1] = 1 = NO SPACE, SO TERMINATE NOINPUT.            50000700
%                                                                  50000800
%    ****************************************************           50000900
%                                                                  50001000
%                                                                  50001100
           ERROR[0] = TIMEOUT:22,                                  50001200
                      STOPBIT:19,                                  50001300
                      BUFOVFL:19,                                  50001400
                      PARITY:21,                                   50001500
                      LOSSOFCARRIER:22.                            50001600
%                                                                  50001700
%    ****************************************************           50001800
%                                                                  50001900
%                                                                  50002000
       GETSPACE[23].                                               50002100
       IF TOG[0] THEN GO TO 10.        %FSL SO TEXT MSG FOLLOWS NOW 50002200
%                                                                  50002300
       INITIATE TRANSMIT.                                          50002400
       TRANSMIT ACK.                                               50002500
       FINISH TRANSMIT.                                            50002600
%                                                                  50002700
       INITIATE RECEIVE.                                           50002800
   10: RECEIVE (1 SEC) SOH [ERROR[0],FORMATERR:19].                50002900
       INITIALIZE BCC.                 %START BCC ACCUMULATION     50003000
       RECEIVE ADDRESS [ERROR[0], ADDERR:19].                      50003100
       RECEIVE TRAN [ERROR[0], TRANERR:NULL].                      50003200
       RECEIVE STX [ERROR[0], FORMATERR:19].                       50003300
       INITIALIZE TEXT.                %SET POINTER TO FRONT OF TEXT 50003400
       CONTROLFLAG=FALSE.                                          50003500
       RECEIVE[0,ETX:18,CONTROL].      %LOOK FOR CONTROL CHAR      50003600
       STORE[ENDOFBUFFER:19].                                      50003700
       RECEIVE TEXT [ERROR[0],ENDOFBUFFER:24,ETX].                 50003800
   18: RECEIVE BCC [ERROR[0], BCCERR:19].                          50003900
%                                                                  50004000
       INCREMENT TRAN.                                             50004100
       INITIATE TRANSMIT.                                          50004200
       TRANSMIT ACK.                                               50004300
       FINISH TRANSMIT.                                            50004400
       TERMINATE NORMAL.                                           50004500
%                                                                  50004600
%                                                                  50004700
   19: RECEIVE (25 MILLI) [0].                                     50004800
       GO TO 19.                                                   50004900
   21: IF CHAR NEQ 4"FF" THEN GO TO 19.                            50005000
   22: INITIATE TRANSMIT.                                          50005100
       TRANSMIT NAK.                                               50005200
       FINISH TRANSMIT.                                            50005300
       NAKFLAG = TRUE.                 %SAY THAT WE NAK'ED IT      50005400
       IF TOG[1] THEN TERMINATE NOINPUT.%RETRY GETSPACE FOREVER    50005500
       IF RETRY = 0 THEN TERMINATE ERROR.%STANDARD ERROR HANDLER   50005600
       RETRY = RETRY - 1.                                          50005700
       TERMINATE NOINPUT.                                          50005800
```

```
%                                                                      50005900
   23: TOG[1] = TRUE.                    %GETSPACE FAILURE             50006000
      GO TO 19.                                                        50006100
   24: RECEIVE CHAR[0,ETX:18].           %OVERFLOW - JUST DROP REMAINING 50006200
      GO TO 24.                          %CHARS BUT KEEP ON GOING      50006300
REQUEST POLLIT:                                                        60000000
     ERROR [1] = TIMEOUT:5,                                            60000100
                 STOPBIT:3,                                            60000200
                 BUFOVFL:3,                                            60000300
                 PARITY:4,                                             60000400
                 LOSSOFCARRIER:5.                                      60000500
     ERROR [2] = TIMEOUT:2,                                            60000600
                 STOPBIT:2,                                            60000700
                 BUFOVFL:2,                                            60000800
                 PARITY:2,                                             60000900
                 LOSSOFCARRIER:2.                                      60001000
     TOG[0] = FALSE.                                                   60001100
     INITIATE TRANSMIT.                                                60001200
     TRANSMIT EOT.                                                     60001300
     TRANSMIT ADDRESS.                                                 60001400
     TRANSMIT POL.                                                     60001500
     TRANSMIT ENQ.                                                     60001600
     FINISH TRANSMIT.                                                  60001700
     INITIATE RECEIVE.                                                 60001800
     RECEIVE (1 SEC) [1].                                              60001900
     IF CHAR = SOH THEN                                                60002000
     BEGIN                                                             60002100
          CONTROLFLAG = FALSE.                                         60002200
          INITIALIZE BCC.                                              60002300
          RECEIVE (1 SEC) ADDRESS [1,ADDERR:3]. %RIGHT STATION?        60002400
          GETSPACE [6].                  %OK -  GET SOME SPACE         60002500
          RECEIVE TRAN [1,TRANERR:NULL].                               60002600
          RECEIVE STX[1,FORMATERR:3].                                  60002700
          INITIALIZE TEXT.                                             60002800
          RECEIVE[1,ETX:1,CONTROL].                                    60002900
          STORE[ENDOFBUFFER:3].                                        60003000
          RECEIVE TEXT[1,ETX,ENDOFBUFFER:7].                           60003100
1:        RECEIVE BCC[1,BCCERR:3].                                     60003200
          INITIATE TRANSMIT.                                           60003300
          TRANSMIT ACK.                                                60003400
          FINISH TRANSMIT.                                             60003500
          INITIATE RECEIVE.                                            60003600
          RECEIVE (1 SEC) EOT [2,FORMATERR:NULL].                      60003700
2:        INCREMENT TRAN.                                              60003800
          TERMINATE NORMAL.                                            60003900
     END.                                                              60004000
     IF CHAR = EOT THEN                                                60004100
   BEGIN INITIALIZE RETRY.                                             60004200
     TERMINATE NOINPUT.                                                60004300
   END.                                                                60004400
3:   RECEIVE (25 MILLI) [1].                                           60004500
     GO TO 3.                                                          60004600
4:   IF CHAR NEQ 4"FF" THEN GO TO 3.                                   60004700
5:   IF TOG[0] THEN TERMINATE NOINPUT.                                 60004800
     IF RETRY = 0 THEN TERMINATE ERROR.                                60004900
```

```
        RETRY = RETRY - 1.                                               60005000
        TERMINATE NOINPUT.                                               60005100
    6:  STATION(TALLY) = 0.                 %NO SPACE SO MAKE SURE THAT WE 60005200
        TOG[0] = TRUE.                      %POLL HIM NEXT CYCLE AND DONT  60005300
        GO TO 3.                            %DECREMENT HIS RETRY COUNT     60005400
    7:  RECEIVE CHAR[1,ETX:1].                                           60005500
        GO TO 7.                                                         60005600
%                                                                        60005700
%           NOTICE THAT WE NEVER NAK A MESSAGE AS WE DON'T WANT HIM TO   60005800
%        REXMIT HIS MESSAGE IMMEDIATELY - PERHAPS SOME OTHER STATIONS    60005900
%        HAVE WORK TO DO AND WE DON'T WANT TO HANG THEM UP WHILE         60006000
%        WE SORT THIS TROUBLEMAKER OUT.                                  60006100
REQUEST SELECTIT:                                                        70000000
        ERROR [1] = TIMEOUT:4,                                           70000100
                    STOPBIT:2,                                           70000200
                    BUFOVFL:2,                                           70000300
                    PARITY:3,                                            70000400
                    LOSSOFCARRIER:4.                                     70000500
    1:  TOG[0] = FALSE.                                                  70000600
        INITIATE TRANSMIT.                                               70000700
        TRANSMIT EOT.                                                    70000800
        TRANSMIT ADDRESS.                                                70000900
        TRANSMIT SEL.                                                    70001000
        TRANSMIT ENQ.                                                    70001100
        FINISH TRANSMIT.                                                 70001200
        INITIATE RECEIVE.                                                70001300
        RECEIVE(1 SEC) [1].                                              70001400
        IF CHAR = ACK THEN                                               70001500
        BEGIN                                                            70001600
            INITIATE TRANSMIT.                                           70001700
            TRANSMIT SOH.                                                70001800
            INITIALIZE BCC.                                              70001900
            TRANSMIT ADDRESS.                                            70002000
            TRANSMIT TRAN.                                               70002100
            TRANSMIT STX.                                                70002200
            INITIALIZE TEXT.                                             70002300
            TRANSMIT TEXT.                                               70002400
            TRANSMIT ETX.                                                70002500
            TRANSMIT BCC.                                                70002600
            FINISH TRANSMIT.                                             70002700
            INITIATE RECEIVE.                                            70002800
            RECEIVE(1 SEC) [1].                                          70002900
            IF CHAR =  ACK THEN                                          70003000
            BEGIN                                                        70003100
                INCREMENT TRAN.                                          70003200
                TERMINATE NORMAL.                                        70003300
            END.                                                         70003400
              IF CHAR = NAK THEN                                         70003500
              BEGIN                                                      70003600
                 NAKFLAG = TRUE.                                         70003700
                 IF RETRY = 0 THEN TERMINATE ERROR.                      70003800
                 RETRY = RETRY-1.                                        70003900
                 GO TO 1.         %IE ASK HIM TO RE XMIT HIS MESSAGE     70004000
%                                 %WE MAY GO ROUND THIS LOOP <RETRY> TIMES70004100
%                                 % SO MAYBE THIS SHOULD BE REPLACED     70004200
```

```
%                                  %WITH A TERMINATE NOINPUT TO ALLOW      70004300
%                                  %US TO TALK TO OTHER STATIONS WHILE     70004400
%                                  %WE SORT THIS GUY OUT.                  70004500
                END.                                                       70004600
                GO TO 2.                                                   70004700
         END.                                                              70004800
         IF CHAR = NAK THEN                                                70004900
         BEGIN                                                             70005000
              NAKONSELECT = TRUE.                                          70005100
              INITIALIZE RETRY.    %WE WILL NEVER MARK HIM DOWN IF WE      70005200
%                                  %GET NAK ON SEL.THIS MIGHT NOT BE A     70005300
%                                  %GOOD PLAN. IF NOT THEN REMOVE THIS     70005400
%                                  %INITIALIZE RETRY AND HE WILL GET       70005500
%                                  %THE STANDARD RETRY LOGIC.              70005600
              TOG[0] = TRUE.                                               70005700
              GO TO 4.                                                     70005800
         END.                                                              70005900
2:       RECEIVE (25 MILLI) [1].                                           70006000
         GO TO 2.                                                          70006100
3:       IF CHAR NEQ 4"FF" THEN GO TO 2.                                   70006200
4:       IF RETRY = 0 THEN TERMINATE ERROR.                                70006300
         RETRY = RETRY - 1.                                                70006400
         IF TOG[0] THEN TERMINATE ENABLEINPUT. %HE NAK'ED US - PROBABLY    70006500
%                                          %HE IS IN XMIT SO WE WILL GO    70006600
%                                          %AND POLL HIM RIGHT NOW         70006700
         TERMINATE NOINPUT.                                                70006800
MODEM TA713:                                                               80000000
    TRANSMITDELAY = 256 MILLI. %THIS VALUE IS THE TRANSMIT DELAY           80000100
    NOISEDELAY = 50 MILLI.                                                 80000200
    TYPE = ASYNC.                                                          80000300
    SPEED = 1200.                                                          80000400
TERMINAL TD830X4:                                                          80100000
           ADDRESS = 2.                                                    80100100
           SPEED = 1200. %SPEED CONTROLLED SOLELY BY ADC                   80100200
           TURNAROUND = 12 MILLI.                                          80100300
           TIMEOUT = 1 SEC. %RQST WAITS THIS TIME BEFORE FURTHER ACTION    80100400
           CODE = ASC67.                                                   80100500
           PARITY = VERTICAL:EVEN,HORIZONTAL:EVEN.                         80100600
           CONTROL = UPOLL.                                                80100700
           REQUEST = USELECTED:RECEIVE,UPOLLED:TRANSMIT.                   80100800
           MAXINPUT = 1920.                                                80100900
           BLOCKED = FALSE.                                                80101000
           END = ETX.                                                      80101100
           BACKSPACE = BS.                                                 80101200
           LINEDELETE = DEL.                                               80101300
           WRU = ENQ.                                                      80101400
           TYPE = ASYNC(MODEM).                                            80101500
           BYTE = 7, PARITY.                                               80101600
           SCREEN = TRUE.                                                  80101700
           HOME = DC4.                                                     80101800
           CLEAR = FF.                                                     80101900
           CARRIAGE = CR.                                                  80102000
           LINEFEED = LF.                                                  80102100
           WIDTH = 32.                                                     80102200
           WRAPAROUND = TRUE.                                              80102300
```

```
TERMINAL TD830:                                                      81000000
            ADDRESS = 2.                                             81000100
            SPEED = 1200. % SPEED CONTROLLED SOLELY BY ADC.          81000200
            TURNAROUND = 12 MILLI. % THIS VALUE IS THE TRANSMIT DELAY.  81000300
            TIMEOUT = 1 SEC. %RQST WAITS THIS TIME BEFORE FURTHER ACTION.81000400
            CODE = ASC67.                                            81000500
            PARITY = VERTICAL:EVEN,HORIZONTAL:EVEN.                  81000600
            CONTROL = POLL.                                          81000700
            REQUEST = POLLIT:RECEIVE,SELECTIT:TRANSMIT.              81000800
            MAXINPUT = 1920.                                         81000900
            BLOCKED = FALSE.                                         81001000
            END = ETX.                                               81001100
            BACKSPACE = BS.                                          81001200
            LINEDELETE = DEL.                                        81001300
            WRU = ENQ.                                               81001400
            TYPE = ASYNC(DIRECT).                                    81001500
            BYTE = 7, PARITY.                                        81001600
            SCREEN = TRUE.                                           81001700
            HOME = DC4.                                              81001800
            CLEAR = FF.                                              81001900
            CARRIAGE = CR.                                           81002000
            LINEFEED = LF.                                           81002100
            WIDTH = 80.                                              81002200
            WRAPAROUND = TRUE.                                       81002300
STATION  STATION0:                                                   82000000
            CONTROL = 4"40".                                         82000100
            ENABLEINPUT = TRUE.                                      82000200
            FREQUENCY = 0.                                           82000300
            LOGIN = FALSE.                                           82000400
            MYUSE = INPUT,OUTPUT.                                    82000500
            RETRY = 10.                                              82000600
            WIDTH = 80.                                              82000700
            WRAPAROUND = TRUE.                                       82000800
            TERMINAL = TD830.                                        82000900
            ADDRESS = "1A". % CHANGE TO YOUR STATION'S ADDRESS.      82001000
            SPEED = 1200.                                            82001100
            TYPE = ASYNC(DIRECT).                                    82001200
STATION  STATION1:                                                   83000000
            CONTROL = 4"40".                                         83000100
            ENABLEINPUT = TRUE.                                      83000200
            FREQUENCY = 0.                                           83000300
            LOGIN = FALSE.                                           83000400
            MYUSE = INPUT,OUTPUT.                                    83000500
            RETRY = 10.                                              83000600
            WIDTH = 80.                                              83000700
            WRAPAROUND = TRUE.                                       83000800
            TERMINAL = TD830.                                        83000900
            ADDRESS = "1B". % CHANGE TO YOUR STATION'S ADDRESS.      83001000
            SPEED = 1200.                                            83001100
            TYPE = ASYNC(DIRECT).                                    83001200
STATION  STATION2:                                                   84000000
            CONTROL = 4"40".                                         84000100
            ENABLEINPUT = TRUE.                                      84000200
            FREQUENCY = 0.                                           84000300
            LOGIN = FALSE.                                           84000400
```

```
            MYUSE = INPUT,OUTPUT.                                          84000500
            RETRY = 10.                                                    84000600
            WIDTH = 80.                                                    84000700
            WRAPAROUND = TRUE.                                             84000800
            TERMINAL = TD830.                                              84000900
            ADDRESS = "1C". % CHANGE TO YOUR STATION'S ADDRESS.            84001000
            SPEED = 1200.                                                  84001100
            TYPE = ASYNC(DIRECT).                                          84001200
STATION  STATION3:                                                         85000000
            CONTROL = 4"40".                                               85000100
            ENABLEINPUT = TRUE.                                            85000200
            FREQUENCY = 0.                                                 85000300
            LOGIN = FALSE.                                                 85000400
            MYUSE = INPUT,OUTPUT.                                          85000500
            RETRY = 10.                                                    85000600
            WIDTH = 80.                                                    85000700
            WRAPAROUND = TRUE.                                             85000800
            TERMINAL = TD830.                                              85000900
            ADDRESS = "1D". % CHANGE TO YOUR STATION'S ADDRESS.            85001000
            SPEED = 1200.                                                  85001100
            TYPE = ASYNC(DIRECT).                                          85001200
STATION  STATION4:                                                         86000000
            CONTROL = 4"40".                                               86000100
            ENABLEINPUT = TRUE.                                            86000200
            FREQUENCY = 0.                                                 86000300
            LOGIN = FALSE.                                                 86000400
            MYUSE = INPUT,OUTPUT.                                          86000500
            RETRY = 100.                                                   86000600
            WIDTH = 32.                                                    86000700
            WRAPAROUND = TRUE.                                             86000800
            TERMINAL = TD830X4.                                            86000900
            ADDRESS = "1A". %CHANGE TO ADDRESS BY WHICH YOU'LL BE POLLED   86001000
            SPEED = 1200.                                                  86001100
            TYPE = ASYNC(MODEM).                                           86001200
            MODEM = TA713.                                                 86001300
LINE  LINE0:                                                               87000000
            ADDRESS = 5. % CHANGE TO YOUR PHYSICAL LINE ADDRESS.           87000100
            MAXSTATIONS = 4.                                               87000200
            STATION = STATION0, STATION1, STATION2, STATION3.              87000300
            TYPE = DIRECT.                                                 87000400
        LINE  LINE1:                                                       88000000
            ADDRESS = 6. %CHANGE TO YOUR PHYSICAL LINE ADDRESS             88000100
            MAXSTATIONS = 1.                                               88000200
            STATION = STATION4.                                            88000300
            TYPE = MODEM.                                                  88000400
            MODEM = TA713.                                                 88000500
DCP DCP880:                                                                89000000
          BUFFER      = 120.                                               89000100
          BUFFERCOUNT = 128.    %TO SAVE SPACE SOME OVERLAP                89000200
                                % OF BUFFERS IS ALLOWED FOR. IT IS         89000300
                                % EXPECTED THAT NOT ALL STATIONS WILL HAVE89000400
                                % ALL BUFFERS LOADED AT THE SAME TIME.     89000500
             MEMORY = 49152.                                               89000600
             LIMIT  = 128.                                                 89000700
FILE  FILE0:                                                               90000000
```

```
             FAMILY = STATION0, STATION1, STATION2, STATION3, STATION4.   90000100
FILE FILE1:                                                               90000200
          FAMILY = STATION0.                                              90000300
FILE FILE2:                                                               90000400
          FAMILY = STATION1.                                              90000500
FILE FILE3:                                                               90000600
          FAMILY = STATION2.                                              90000700
FILE FILE4:                                                               90000800
          FAMILY = STATION3.                                              90000900
FILE FILE5:                                                               90001000
          FAMILY = STATION4.                                              90001100
```

Documentation Evaluation Form

Title: CMS Data Communications Subsystem Reference Manual

Form No: 1090909

Date: July, 1980

Burroughs Corporation is interested in receiving your comments and suggestions regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments:

From:

Name

Title

Company

Address

Phone Number

Date

Remove form and mail to:

Documentation Dept, TIO - East
Burroughs Corporation
Box CB7
Malvern, PA 19355

2" BINDER

1½" BINDER

1" BINDER

Computer Management System (CMS)

Data Communications Subsystem

REFERENCE MANUAL

1090909

Printed in U.S.A.