Burroughs Corporation Ⓑ

BUSINESS MACHINES GROUP
SMALL SYSTEMS PLANT

B1700 Network Definition Language

PRODUCT SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
A	8/27/75	Original issue	S.M.B.	J Wale 9-8-75

GOL 1968 12-69

TABLE OF CONTENTS:

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

GENERAL

This document defines the Network Definition Language (NDL) for the
B1700.   It is intended to serve as both a formal specification of
the language and as a programmer's guide. The reader should be
somewhat familiar with 1) Burroughs Data Communication procedures
and hardware, and 2) B1700 NDL software system architecture.

RELATED PUBLICATIONS

    B1700 Data Communications Functional Description
    B1700 NDL/LIBRARY (P.S. 2212 5215)
    B1700 Message Control System (MCS) Manual
    B1700 NDL Reference Manual (# 1073715)
    DC/AUDIT Technical Documentation and
       Operating Instructions
    B1700 Data Comm Audit (P.S. 2212 5421)
    B1700 RJE/Terminal(NDL) (P.S. 2212 5249)

The functional description should be read before the NDL Reference
Manual,   since it describes the overall B1700 NDL system. The MCS
manual will be of value to anyone who needs the additional features
of message control system in their Datacomm System.

THE STRUCTURE OF NDL

NDL is a higher-order language for Data Communications. It provides
a simple means of generating a B1700 Network Controller (NC). If a
network is changed, the system can be quickly and easily
regenerated by recompiling. A network is defined by specifying in
NDL source code the network attributes: The physical devices in the
network (lines, terminals, adapters, etc.), the line discipline to
be used ("REQUESTS"), the order and priority of line use
("CONTROLS"), and the grouping of stations into files.

An NDL program consists of the following sequence of sections:

SECTION	FUNCTION
DECLARATION	Constant, audit file, and MCS signal character definition
REQUEST	Code for line disciplines for each type of remote device ("TERMINAL")
CONTROL	Code to decide the order and priority of line use.
TERMINAL	Lists the attributes for each type of remote device
STATION	Lists the attributes for every remote device in the network.
LINE	Lists the attributes for each line in the network
FILE	Groups stations into logical files

The NDL compiler translates the source into two outputs:

1)   The Network Information File (NIF), a file of tables
     describing the physical and logical attributes of the
     network.

2)   The Network Controller Codefile. REQUESTS and CONTROLs
     are compiled into code and incorporated as subroutines
     in the codefile. The NIF contains the initialization

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

values for the controller's line, station, terminal,
and file tables.

CODING AIDS

NDL source language preparation can be simplified by using the
coding aids provided by the compiler. Common datacomm control
characters have been predefined. For example, the symbols "ETX" and
"SIX" are recognized by the NDL compiler to signify the hexadecimal
strings 4"03" and 4"02" respectively (See DECLARATION SECTION).
Most users should find it unnecessary to write their own REQUESTs
and CONTROLs--the source language library ($LIBRARY) contains
REQUESTs and CONTROLs to handle common line disciplines and line
driving techniques. Default definitions (See TERMINAL SECTION) may
be used in the terminal, station, line, and file sections to reduce
the coding required.

SYNTAX CONVENTIONS

This section describes the Backus-Naur Form (BNF) to define the
syntax of the Network Definition Language.

METALINGUISTIC SYMBOLS

The following metalinguistic symbols will be used in this document:

< >   Left and right broken brackets are used to contain one or
           more digits and/or letters representing a metalinguistic
           variable whose definition is given by a metalinguistic
           formula.

::=   The symbol "::=" means "IS DEFINED AS". The metalinguistic
           variable to the left of this symbol is defined by the
           metalinguistic formula on its right.

| The symbol "|" means "OR". It separates alternative
           definitions of a metalinguistic variable.

METALINGUISTIC FORMULAS

Metalinguistic symbols are used in forming a metalinguistic formula. A metalinguistic formula is a rule which will produce an allowable sequence of characters and/or symbols. These formulas are used to define the syntax of the B1700 Network Definition Language. The syntax, combined with the semantics contained in this product specification, defines the B1700 Network Definition Language.

Any mark or symbol in a metalinguistic formula which is not one of the metalinguistic symbols, is equivalent to itself. The juxtaposition of the metalinguistic variables and/or symbols in a metalinguistic formula denotes the juxtaposition of those elements in the construct indicated.

An example of a metalinguistic formula is:

        <IDENTIFIER> ::= <LETTER> | <IDENTIFIER> <LETTER> |
                        <IDENTIFIER> <DIGIT>

This metalinguistic formula is read:

    An Identifier is defined as a letter, or an identifier
    followed by a letter, or an identifier followed by a digit.

The metalinguistic formula above defines a recursive relationship by which a construct called an Identifier may be formed. That is, evaluation of the formula shows that an Identifier begins with a letter. The letter may stand alone, or may be followed by any sequence of letters and digits.

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

BASIC ELEMENTS OF THE LANGUAGE

CHARACTER SET

The NDL character set is drawn from EBCDIC graphics, but any EBCDIC
character is valid in an <EBCDIC STRING>. The Datacomm System is
internally EBCDIC. Where appropriate all characters are translated,
as for example, when transmitting to a NON-EBCDIC device. These
subsets of EBCDIC characters will be used in the following BNF:

```
<EBCDIC CHARACTER>::=          <LETTER>
                            I  <DIGIT>
                            I  <SPECIAL CHARACTER>
                            I  <SLASH>
                            I  <VERTICAL BAR>
                            I  <BLANK>


<LETTER>::=                    AIBICI...IXIYIZ

<DIGIT>::=                     0I1I...I8I9

<SPECIAL CHARACTER>::=         -I<SLASH>I[I]I:I.I$I,I<I>I*I%Ia
                              I(I)I+I=I<VERTICAL BAR>I"I<BLANK>

<SLASH>::=                     /

<VERTICAL BAR>::=              I

<BLANK>:=                      (ONE GRAPHIC SPACE - HEX 40)
```

BASIC COMPONENTS

The primitives of the NDL language.

```
<BASIC COMPONENT>::=          <IDENTIFIER>
                            | <INTEGER>
                            | <STRING>
                            | <OPERATOR>
                            | <LOGICAL VALUE>
                            | <BRACKETS>
                            | <SEPARATOR>
                            | <ASSIGNER>
                            | <VARIABLE>
```

IDENTIFIERS

```
<IDENTIFIER>::=               <LETTER>
                            | <IDENTIFIER> <LETTER>
                            | <IDENTIFIER> <DIGIT>
```

RESTRICTIONS: A) There is no maximum on Identifier Length, but
                  only the first 10 characters will be used by the
                  compiler.

             B) Identifier must be unique (i.e.Two REQUESTs may
                 not have the same Identifier).

INTEGERS

```
<INTEGER>::=                  <DIGIT>
                            | <INTEGER> <DIGIT>
```

Integers are loaded as 24-bit binary numbers.

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

STRINGS

SYNTAX:

<STRING>::=   <HEXADECIMAL STRING>
                                        I <EBCDIC STRING>

<HEXADECIMAL STRING>::=   4"<HEX CONCATENATION>"

<HEX CONCATENATION>::=   <HEX CHARACTER> <HEX CHARACTER>
                                        I <HEX CONCATENATION> <HEX CHARACTER>
                                          <HEX CHARACTER>

NOTE - HEX CHARACTERS ARE ADDED BY TWOS.

<HEX CHARACTER>::=   <DIGIT> IAIBICIDIEIF

<EBCDIC STRING>::=   "<EBCDIC CONCATENATION>"

<EBCDIC CONCATENATION>::=   <EBCDIC CHARACTER>
                                        I <EBCDIC CHARACTER> <EBCDIC
                                          CONCATENATION>

SEMANTICS:

Any EBCDIC character may appear in an EBCDIC string. An internal
quote is represented by two consecutive quotes. For example: """"
represents an EBCDIC string of one single quote. The maximum length
of a Hexadecimal string is 40 bits, or 10 Hexadecimal digits.
Maximum length of a EBCDIC string is 40 characters.

OPERATORS

SYNTAX:

```
<OPERATOR>::=              <UNARY OPERATOR>
                        I <BINARY OPERATOR>

<UNARY OPERATOR>::=         + I - I NOT

<BINARY OPERATOR>::=        <ARITHMETIC OPERATOR>
                        I <LOGICAL OPERATOR>
                        I <RELATIONAL OPERATOR>

<ARITHMETIC OPERATOR>::=    + I - I * I <SLASH>

<LOGICAL OPERATOR>::=       AND I  OR

<RELATIONAL OPERATOR>::=    EQ I NE I GT I GE I LT I LE I = I > I <
```

SEMANTICS:

Operators are used in expressions (See EXPRESSIONS). Binary
operators require two operands, as in "10 * 3", unary operators
only one, as in "NOT BREAK". The alphabetic relational operators
are abbreviations:

```
                    EQ        Equal
                    NE        Not Equal
                    GT        Greater Than
                    GE        Greater Than or Equal
                    LT        Less Than
                    LE        Less Than or Equal
```

LOGICAL VALUES

SYNTAX:

<LOGICAL VALUE>::=   TRUE I FALSE

SEMANTICS:

TRUE is equivalent to the integer 1, and FALSE to 0. For example the
expression "BREAK EQ TRUE" could have been "BREAK EQ 1".

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

BRACKETS

SYNTAX:

<BRACKET>::=   (I) I [I]

SEMANTICS:

Parentheses normally signify a modification or variant form of the preceding symbol, for example "TRAN (TRANSMIT)" and "TRAN (RECEIVE)". True brackets are used for subscripts of TOG and TALLY.

SEPARATORS

SYNTAX:

<SEPARATOR>::=   , I . I : I <SPACE>

<SPACE>::=   <BLANK> I <SPACE> <BLANK>

SEMANTICS:

Note that one blank or any number of blanks may be used to separate basic components. These statements are equivalent:

1.   MYUSE=INPUT,OUTPUT.

2.   MYUSE = INPUT, OUTPUT.

3.   MYUSE
       =
       INPUT
       ,
       OUTPUT
       .

ASSIGNER

SYNTAX:

<ASSIGNER>::=   := | <VERTICAL BAR>

SEMANTICS:

The <ASSIGNER> is used in assignment statements in the REQUEST and
CONTROL sections. Example:

    TOG[0] :=   TRUE.

VARIABLES

SYNTAX:

```
<VARIABLE>::=                <SYSTEM STATUS VARIABLE>
                          |  <ERROR FLAG>
                          |  <USER TOGGLE>
                          |  <USER TALLY>
                          |  <STRING FUNCTION>
```

SEMANTICS:

NDL provides no means of declaring data areas like COBOL's
"WORKING-STORAGE" or UPL/SDL's "DECLARE". In the REQUEST and
CONTROL sections, the NDL programmer must use a set of predefined
identifiers to test or set the value of system items. All of the
variables may appear in an expression in either (or in some cases,
both) REQUESTs or CONTROLs. About half of the variables may have
their value changed explicitly in a REQUEST or CONTROL; that is,
they may appear on the left side of an <ASSIGNER> in an assignment
statement. The remainder are read only. See the following table as
to where a specific variable can be referenced and whether it can
be assigned.

Except for "TIME", system status variables refer to items in the
Network Controller's line, station, or terminal tables. "TIME" is
an MCP communicate.

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212   5223

VARIABLE:	ASSIGNABLE (YES=*)	SIZE IN BITS:
SYSTEM STATUS VARIABLES:		
CHAR.CHARACTER	*	8
FREQUENCY (INPUT)		8
FREQUENCY (OUTPUT)		8
IODESC		144 + 144 = 288
INPUTATTACHED		1
OUTPUTATTACHED		1
LENGTH (INPUT)		12
LENGTH (OUTPUT)		12
LINE		8
LINE (CONTROL KEY)		5
LINE (QUEUED)		1
MAXSTATIONS		10
RESULTDESC		24 + 24 = 48
RETRY	*	8
STATION	*(CONTROL)	10
STATION (ENABLED)		1
STATION (MYUSE)		2
STATION (QUEUED)		1
STATION (READY)		1
STATION (TYPE)		2

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

VARIABLE:	ASSIGNABLE: (YES=*)	SIZE IN BITS:
SYSTEM STATUS VARIABLES (CONT.)		
TERMINALTYPE		6
TIME		20
TRAN (RECEIVE)	*	8,16,24
TRAN (TRANSMIT)	*	8,16,24
ERROR FLAGS:		
ADDERR	*	1
BREAK	*	1
ACCESSERR	*	1
ENDFBUFFER	*	1
EXCEPTION	*	1
FORMATERR	*	1
LOSSOFCARRIER	*	1
LOSSOFDSR	*	1
PARITY	*	1
TIMEOUT	*	1
TRANERR	*	1
USER TOGGLES:		
TOG [<INDEX>]	*	1
LINE (TOG [<INDEX>])	*	1

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

VARIABLE:	ASSIGNABLE: (YES=*)	SIZE IN BITS:
USER TALLIES:		
TALLY [<INDEX>]	*	8
LINE (TALLY [<INDEX>])	*	8
TIME (TALLY)	*19	
STRING FUNCTIONS:		
CONVERT		
DECIMAL		

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

SYSTEM STATUS VARIABLES

SYNTAX:

```
<SYSTEM STATUS VARIABLE>::=   CHAR
                            I CHARACTER
                            I FREQUENCY (INPUT)
                            I FREQUENCY (OUTPUT)
                            I IODESC
                            I INPUTATTACHED
                            I OUTPUTATTACHED
                            I LENGTH (INPUT)
                            I LENGTH (OUTPUT)
                            I LINE
                            I LINE (CONTROL KEY)
                            I LINE (QUEUED)
                            I MAXSTATIONS
                            I RESULTDESC
                            I RETRY
                            I STATION
                            I STATION (ENABLED)
                            I STATION (LINE)
                            I STATION (MYUSE)
                            I STATION (QUEUED)
                            I STATION (READY)
                            I STATION (TYPE)
                            I TERMINALTYPE
                            I TIME
                            I TRAN (RECEIVE)
                            I TRAN (TRANSMIT)
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

SEMANTICS:

CHAR, CHARACTER

Both names refer to the same thing, the character register, a
one-byte field in the line table. Assignable, it is intended to be
used in REQUESTs while processing a message character by character.
The "FETCH" statement loads the register. For example:

```
        DO ECHOLOOP FOREVER.
            FETCH CHAR.
            TRANSMIT CHAR.
            IF CHAR EQ ETX THEN UNDO.
        END ECHOLOOP.
```

FREQUENCY (INPUT), FREQUENCY (OUTPUT)

These eight-bit read-only items in the station table are accessible
only by CONTROLs and are declared by the user in the <FREQUENCY
STATEMENT> in the STATION section.

IODESC

288-bit read-only field in line table. Accessible only in REQUESTS.
IODESC contains the two 144-bit I/O descriptors used for I/O on the
current line. Useful in debugging, probably in <AUDIT STATEMENT>.

INPUTATTACHED

One-bit flag in line table. Not assignable, accessible only in
REQUESTs. INPUTATTACHED is set true if there is a valid input
message buffer into which data may be received, false if no input
message buffer is attached to the line.

OUTPUTATTACHED

One-bit flag in line table. Not assignable, accessible only in
REQUESTs. OUTPUTATTACHED is set true if there is a valid output
message buffer attached to the line for transmission, false if
there is no output message buffer attached.

LENGTH (INPUT), LENGTH (OUTPUT)

Two 12-bit read-only fields in the line table. Accessible only in
REQUESTs. The length in bytes of the input or output text
respectively. It does not include the header length or ETX.

LINE

Eight-bit item in line table which stores the current line-number.
read-only, accessible by both REQUESTs and CONTROLs.

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

LINE (CONTROL KEY)

Five-bit, read-only item in line table. This item is accessible only
in CONTROLs, and it should be used to drive the "FLOW OF CONTROL"
in a CONTROL (See CONTROL SECTION), probably as a CASE STATEMENT
INDEX. The value of LINE (CONTROL KEY) gives the reason that the NC
entered the CONTROL. To some extent, the value dictates the action
that the CONTROL should take.

   The Values:

   0.   Line is not busy. I/O may be initiated.

   1.   Change in status of a station or stations on
        this line. This occurs in the following cases:

        A)   First time through CONTROL.
        B)   Remote file open/close involving a station on
             this line.
        C)   NC EOJ initiated - all stations disabled for
             input.
        D)   TERMINATE   ERROR has caused a station to go not
             ready.

        The CONTROL should rebuild the poll list, if any (See
        Initiate Autopoll), or retest the stations on the
        line. If no stations are eligible for I/O, the
        CONTROL should initiate idle.

   2.   An output message has been queued for this line, but
        the line is now in a read with no timeout or waiting
        for ring. CONTROL must resolve the conflict by:

        a)   INITIATE CANCEL which kills the read, or
        b)   CONTINUE   which leaves the read in progress and
             the output message in the queue.

   3.   A ring complete has been received. Used for switched
        lines only.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

4. The last REQUEST on this line did a TERMINATE OUTPUT(RETURN). The CONTROL must restart it by eventually executing a CONTINUE. The CONTROL may at this point change the value of STATION before returning in some disciplines.

5.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

LINE (QJEUED)

One-bit flag in line table, read-only, accessible to CONTROLs and REQUESTs. True if a message is queued for output for any station on this line.

MAXSTATIONS

The maximum number of stations on the current line, including passive stations. read-only, accessible only in CONTROLs. Its value is specified by the <MAXSTATIONS STATEMENT> in the line section. Normally used to test the current value of "STATION" to see if it should oe reset to one:

        IF STATION EQ MAXSTATIONS
        THEN STATION:= 1.
        ELSE STATION:=STATION+1.

RESULTDESC

48-bit read-only field accessible in REQUESTs. It consists of the 24-bit result descriptors in the two I/O descriptors kept for each line in the line table. RESULTDESC references sub-fields of "IODESC". Useful in debugging a datacomm system (as in "AUDIT TRACEFIEL (RESULTDESC).")

RETRY

Eight-bit TALLY intended to be used to count receive or transmission retries in REQUESTs. After each unsuccessful attempt, subtract one from RETRY. When RETRY equals zero, a TERMINATE ERROR would be appropriate. Assignable and accessible in both REQUESTs and CONTROLs. The initial value of retry is set by the <RETRY STATEMENT> in the station section. It may be reset to this value at run-time oy INITIALIZE RETRY in REQUESTs. (It is never implicitly reinitialized by the NC).

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

STATION

10-bit field in line table, assignable by CONTROLs, read-only by REQUESTs. It contains the current station number on this line. STATION must be greater than or equal to one, and it may never be greater than MAXSTATIONS. A fatal run error will occur if STATION exceeds these bounds, so the NDL programmer should always test the value of station before incrementing it. (See MAXSTATIONS, also CONTROL section). STATION numbers are determined by the order of stations in the <LINE STATION STATEMENT> in the line section. Note that STATION refers only to the line-relative station number. It is not the same as the "logical station number" which appears in the 27-byte message header. "Logical station number" is unique to each station in the system, regardless of which line it is on, and is determined by the order in which stations are declared in the station section. Neither is STATION the same as the "file-relative station number" which is used as a key by user programs in doing I/O on remote files. Neither "logical station number" nor "file-relative station number" are referenceable within NDL.

STATION (MYUSE)

2-bit, read-only item in the station table, accessible only in CONTROLs. Its value is assigned by the <MYUSE STATEMENT> in the station section, as follows:

1.   Station is input only.
2.   Station is output only.
3.   Station is used for input and output.

The default value is 3.

STATION (ENABLED)

One-bit, read-only flag in station table. Accessible by CONTROLs. "STATION (ENABLED)" must be true if I/O is to be initiated by the CONTROL for this station (See CONTROL INITIATE STATEMENT). Set in the station section by <ENABLEINPUT STATEMENT>, but can be overridden by the MCS.

STATION(LINE)

Eight-bit, read-only entry in the station table which indicates the
line index of the line currently associated with the station in a
switched-line environment. Set when a line CONTROL does an initiate
(input, output, input-output, output,input) and reset to zero on an
initiate disconnect, terminate disconnect, or terminate release
station. STATION(LINE) should be used in a CONTROL to determine if
a station is assigned to another line or the current line or, if
zero, unassigned.

STATION (QUEUED)

One bit, read-only. Accessible by both REQUESTs and CONTROLs. True
if current station has at least one message in its output queue. If
false, INITIATE OUTPUT is invalid in CONTROLs for this station.

STATION (READY)

One-bit, read-only flag in station table. Accessible by CONTROLs.
Must be true in order to initiate I/O in CONTROLs (See IF
STATEMENT). Station can be marked not ready by MCS. If an MCS is
present, TERMINATE ERROR marks a station not ready.

STATION (TYPE)

Two-bit read-only field accessible in both REQUESTs and CONTROLs. It
represents the master/slave type of the remote device currently
indexed by STATION.

    Possible values:

    1.   Slave station
    2.   Controller station
    3.   Normal station

See the <CONTROLLER STATEMENT> in the station section for
information on setting this variable. STATION (TYPE) should be used

in building poll strings (See <POLL STATEMENT>) to take advantage of group poll.

TERMINA_TYPE

Six-bit read-only field in terminal table. Value is set by the TERMINA_ TYPE statement in the terminal section.

TIME

20-bit field giving the time of day in tenths of a second since midnight. Invoking TIME at run-time loads the value directly from the system clock. Used to load TIME (TALLY) for message headers:

        TIME (TALLY) := Time.

TRAN (RECEIVE)

8, 16, or 24 bit field kept in the station table to hold the transmission number of input messages. Assignable and accessible in REQUESTs. Normally loaded from the internal message header by RECEIVE TRAN. After RECEIVE TRAN, TRANERR will be set if this transmission number equals the last transmission number. Length is set in terminal section by <TRANSMISSION NUMBER STATEMENT>. Set to zero by "INITIALIZE TRAN (RECEIVE)".

TRAN (TRANSMIT)

Output message transmission number. TRAN(TRANSMIT) can be assigned
directly in REQUESTs (TRAN(TRANSMIT):= Decimal (TALLY[0], 3).) or
set by the NC by specifying INITIALIZE TRAN(TRANSMIT). At BOJ time,
TRAN(TRANSMIT) is set to binary zeroes, so it should be reset
before sending it. Normally it should be given a value different
from the value of the last output transmission number, either by
the REQUEST or by the MCS. Not referenceable in CONTROLs.

ERROR FLAGS

SYNTAX:

<ERROR FLAG>::=   <RECEIVE ERROR FLAG>
                            I <TRANSMIT ERROR FLAG>
                            I <ITEM ERROR FLAG>

<RECEIVE ERROR FLAG>::=   PARITY
                            I ACCESSERR
                            I TIMEOUT
                            I BREAK
                            I LOSSOFCARRIER
                            I EXCEPTION

<TRANSMIT ERROR FLAG>::=   BREAK
                            I LOSSOFDSR
                            I TIMEOUT
                            I EXCEPTION

<ITEM ERROR FLAG>::=   ADDERR
                            I TRANERR
                            I FORMATERR
                            I ENDOFBUFFER

SEMANTICS:

All error flags are one-bit fields, assignable and accessible in
REQUESTs. ENDOFBUFFER is the only error flag assignable and
accessible in CONTROLs too. All error flags are reset by the NC
before initiating I/O. Each is automatically set on the occurrence
of the appropriate error state. In input REQUESTs the error flags
are moved to the read-error field in the 27-byte internal message
header.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

ADDERR

Set on input from a station when an error in the station address is
detected. The size and value of the station address as defined in
the   <TERMINAL ADDRESS STATEMENT> and <STATION ADDRESS STATEMENT>
allow the Network Controller to check the correct value against
that which is received.

ACCESSERR

Set if the MLC/SLC was not granted memory access before the next
character was received on the line, destroying the previous
character.

BREAK

A BREAK signal was received from the remote station during output.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

ENDOFBUFFER

Set on 4 conditions:

1.   In an input REQUEST, the number of characters
      stored   in   the   buffer   exceeds   the   maximum
      specified by the <MAXINPUT STATEMENT> in the
      terminal description.

2.   In an output REQUEST, the transmit I/O reached
      the   end of the message without encountering a
      terminating character (e.g. ACK, NAK, ENQ, ETB,
      EOT, ETX for TC500).

3.   In input REQUEST, the number of FETCHed
      characters exceeded the number actually in the
      buffer.

4.   In a CONTROL, poll list too long.

EXCEPTION

Set true if parity, ACCESSERR, ENDOFBUFFER, TIMEOUT, BREAK,
LOSSOFCARRIER, or LOSSOFDSR is set because of an initiated I/O.

FORMATERR

Set if an NDL REQUEST uses the RECEIVE <string> option to check
input   data, and the characters received do not correspond to the
<string> specified.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

LOSSOFCARRIER

Set if hardware, specifically the data set, reported LOSSOFCARRIER
frequency.

LOSSOFDSR

Set if hardware reports that data set ready was dropped during an
I/O.

PARITY

Set when parity error or BCC error is detected while receiving a
message.

TIMEOUT

A character was not received (on input) or the first character was
not transmitted within the time specified by the adapter.

TRANERR

After "RECEIVE TRAN(RECEIVE)", the transmission number received was
equal to the number in the last message from the same station.

USER TOGGLES

SYNTAX:

<USER TOGGLE>::=   TOG[<TOG INDEX>]
                          | LINE(TOG [TOG INDEX])

<TOG INDEX>::=   <INTEGER> | <VARIABLE>

SEMANTICS:

TOG

A variable number of one-bit fields in the station table can be
reserved for programmer use. They are assignable and accessible by
both REQUESTs and CONTROLs, but are more useful to REQUESTs since
there is one set of TOGs for each station. They must be set by
either an <ASSIGNMENT STATEMENT> or by an <INITIALIZE STATEMENT>.
TOGs are useful as local data or for communicating with the MCS.
TOG[0-7] are automatically loaded in the 27-byte internal header of
any message queued for the MCS or user program (including error
messages: e.g. "TERMINATE INPUT" or "TERMINATE ERROR"). TOGGLES can
be reloaded from the header of an output message by INITIALIZE TOG
[<INDEX>].

LINE(TOG)

A variable number of assignable one-bit fields in the line table can
be reserved for programmer use. Other than location, line TOGGLES
are similar to station TOGGLES.

RESTRICTIONS:
----

1) A "MAX" statement in the declaration section
must preceed any TOGGLE indexed by a variable.

2) No more than the absolute maximum of 100 line
TOGGLES and 100 station TOGGLES may be used.

3) TOGGLE indices start at 0 and proceed
sequentially.

4) In the event a "MAX" statement is not used the
number of TOGGLES will be determined by the
largest index used.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

USER TALLIES

SYNTAX:

```
<USER TALLY>::=            TALLY [<TALLY INDEX>]
                        | LINE (TALLY [<TALLY INDEX>])
                        | TIME(TALLY)

<TALLY INDEX>::=          <INTEGER> | <VARIABLE>
```

SEMANTICS:

TALLY

A variable number of eight-bit fields in the station table which can
be reserved for programmer use. Assignable and accessible in both
REQUESTs and CONTROLs. They must be set by either an <ASSIGNMENT
STATEMENT> or an <INITIALIZE STATEMENT>. Useful as local data or
MCS   communication (TALLY[0-2] will be transported in the 27-byte
internal header).

LINE(TALLY)

A variable number of eight-bit fields in the line table which can be
reserved for programmer use. Assignable and accessible to both
REQUESTs and CONTROLs.

TIME(TALLY)

20-bit field in station table, assignable and accessible only in
REQUESTs. Normally assigned the value of "TIME" in input REQUESTs.
Moved by the NC to the internal message header on a "RECEIVE TEXT".

TIME(TALLY) is subject to the same restrictions as user TALLIES.

STRING FUNCTIONS

SYNTAX:

```
<STRING FUNCTION>::=        <DECIMAL FUNCTION>
                         I <CONVERT FUNCTION>

<DECIMAL FUNCTION>::=       DECIMAL (<EXPRESSION>,<INTEGER>)

<CONVERT FUNCTION>::=       CONVERT (<EXPRESSION>)
                         I CONVERT (<EXPRESSION>,1)
```

SEMANTICS:

The <DECIMAL FUNCTION> is identical to the SDL/UPL "DECIMAL". It converts the value of the <EXPRESSION>, treated as a binary number, to an EBCDIC string of decimal digits of length <integer>. For example, DECIMAL (4"FF", 3) is converted to the string "255", or 4"F2F5F5". DECIMAL returns a maximum of eight characters even if the value of <integer> is greater. If the value of the <expression> exceeds 24 bits only the low-order 24 bits is used.

The <CONVERT FUNCTION> converts binary strings of any length to character strings. If ",1" is specified then one character is generated for each bit ("0" for binary 0, "1" for binary 1). The default is 4 bits to one character (Hexadecimal conversion). Leading zeroes will be supplied to fill out leftmost Hexadecimal positions, if necessary. For example:

1) CONVERT (255,1) = "00000000000000011111111"

2) CONVERT (255) = "0000FF"

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

EXPRESSIONS

SYNTAX:

```
<EXPRESSION>::=               <TERM>
                            | <TERM> <BINARY OPERATOR> <EXPRESSION>
                            | <EXPRESSION> <BINARY OPERATOR> <TERM>

<TERM>::=                     <PRIMARY>
                            | <UNARY OPERATOR> <PRIMARY>

<PRIMARY>::=                  (<EXPRESSION>)
                            | <VARIABLE>
                            | <CONSTANT IDENTIFIER>
                            | <STRING>
                            | <INTEGER>
                            | <LOGICAL VALUE>

<CONSTANT IDENTIFIER>::=      (SEE CONSTANT DECLARATION)
```

SEMANTICS:

Expressions are used in the REQUEST and CONTROL sections in various
statements.   Note that no distinction is made between logical and
arithmetic expressions. Since REQUEST and CONTROL code is executed
on the SDL-Machine, expressions like

        (TRUE + 10 NE DISCONNECT) * TALLY[0]

are syntactically legal and will execute without run-time errors.

OPERATOR PRECEDENCE:

If there are no parentheses to guide evaluation of an expression (as
in "TRJE + 10 NE DISCONNECT"), operator precedence determines the
order of evaluation. Two rules are used: (1) equal precedence
operators are evaluated from left to right, and (2) higher
precedence operators are evaluated first. From highest to lowest,
the precedence order is:

> unary +, unary -(equal precedence)
>
> *, <SLASH>
>
> binary +, binary-
>
> LT, LE, EQ, NE, GT, GE, =, <, >
>
> NOT
>
> AND
>
> OR

For example "TRUE + 10 NE DISCONNECT * 3" is evaluated as if it had
been written "(TRUE + 10) NE (DISCONNECT * 3)"

EXPRESSION VALUE LENGTH:

If no binary operator was used, the length of the value left on the
value stack after expression evaluation is equal to the length of
the primary (i.e. 24 bits for <integer>, 1 bit for "TRUE", etc).
Logical binary operators (AND, OR) return values whose length is
equal to the longer of the two operands; arithmetic operators
return 24-bit values.

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

SECTIONS OF AN NDL PROGRAM

SYNTAX:

<NETWORK DESCRIPTION>::=   <DECLARATION SECTION>
                               <REQUEST SECTION>
                               <LINE CONTROL SECTION>
                               <TERMINAL SECTION>
                               <STATION SECTION>
                               <LINE SECTION>
                               <FILE SECTION>

SEMANTICS:

The sections of an NDL program must appear in the above order. All
sections are required.

DECLARATION SECTION

SYNTAX:

<DECLARATION SECTION>::=   DECLARATION: <DECLARATION LIST>

<DECLARATION LIST>::=   <DECLARATION>
                              I <DECLARATION LIST> <DECLARATION>
                              I <EMPTY>.

<DECLARATION>::=   <SIGNAL DECLARATION>
                              I <AUDITFILE DECLARATION>
                              I <CONSTANT DECLARATION>
                              I <NIF NAME DECLARATION>
                              I <MAX TALLY/TOG DECLARATION>

SEMANTICS:

The declaration section provides a means of specifying global
definitions for the user's Network Controller.

SIGNAL DECLARATION

SYNTAX:

<SIGNAL DECLARATION>::=   SIGNAL = "<SIGNAL CHARACTER STRING>".

<SIGNAL
CHARACTER STRING>::=   <EBCDIC CHARACTER>
                              I <EBCDIC CHARACTER> <EBCDIC CHARACTER>
                              I <EBCDIC CHARACTER> <EBCDIC CHARACTER>
                                <EBCDIC CHARACTER>

SEMANTICS:

Signal defines up to three one-character prefixes to be used in
Station-To-Station MCS communication. Any messages that have text
beginning with one of the signal characters are queued
automatically by the NC for the MCS. This is true even if the MCS
has chosen not to participate in user program-remote file I/O, and
this station is in the remote file.

AUDITFILE DECLARATION

SYNTAX:

```
<AUDITFILE DECLARATION>::=     AUDITFILE <AUDITFILE IDENTIFIER>
                              (<FILE ATTRIBUTE LIST>).

<FILE ATTRIBUTE LIST>::=       <FILE ATTRIBUTE>
                            I  <FILE ATTRIBUTE>, <FILE ATTRIBUTE LIST>

<FILE ATTRIBUTE>::=            <LABEL PART>
                            I <DEVICE PART>
                            I <BUFFERS PART>
                            I <VARIABLE PART>
                            I <SAVE PART>
                            I <RECORDS PART>
                            I <AREAS PART>
                            I <EMPTY>

A.   <LABEL PART>::=           LABEL = <FILE NAME>

     <FILE NAME>::=            <FAMILY-ID>
                            I <FAMILY-ID> <SLASH> <FILE-ID>
                            I <FAMILY-ID> <SLASH> <FILE-ID> <SLASH>
                            I <PACK-ID> <SLASH> <FAMILY-ID>
                              <SLASH> <FILE-ID>

     <FILE-ID>, <FAMILY-
     ID>, <PACK-ID>::=         <EBCDIC STRING>

     NOTE - 10 CHARACTER MAXIMUM ON EACH ID.
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

B.   <DEVICE PART>::=   DEVICE = <DEVICE SPECIFIER>

      <DEVICE SPECIFIER>::=   DISK   (ANY DISK)
                                I DISKFILE   (HEAD PER TRACK)
                                I DISKPACK   (CENTURY-TYPE)
                                I DISKCARTRIDGE   (CAELUS-TYPE)
                                I TAPE   (ANY TAPE)
                                I TAPE7   (7-TRACK)
                                I TAPE9   (9-TRACK)
                                I TAPEPE   (PHASE-ENCLOSED)
                                I PRINTER
                                I PRINTER OR BACKUP
                                I PRINTER BACKUP

C.   <BUFFERS PART>::=   BUFFERS = <INTEGER>

D.   <VARIABLE PART>::=   VARIABLE
      NOTE - VARIABLE-LENGTH RECORDS

E.   <SAVE PART>::=   SAVE = <INTEGER>
      NOTE - NUMBER DAYS TO SAVE FILE

F.   <RECORDS PART>::=   RECORDS = <RECORD SPECIFIER>

      <RECORD SPECIFIER>::=   <RECORD LENGTH>
                                I <RECORD LENGTH> <SLASH> <BLOCK LENGTH>

      <RECORD LENGTH>::=   <INTEGER>
      <BLOCK LENGTH>::=   <INTEGER>
      NOTE - LENGTHS ARE IN BYTES.

G.   <AREAS PART>::=   AREAS = <NUMBER AREAS>
                                  <SLASH> <BLOCKS PER AREA>

      <NUMBER AREAS>::=   <INTEGER>
      <BLOCKS PER AREA>::=   <INTEGER>
      NOTE - DISK FILES ONLY

SEMANTICS:

Auditfiles are used in <AUDIT STATEMENT>s in the REQUEST section. A
maximum of three auditfiles may be declared. Any auditfile may be
referenced in any REQUEST. As far as the MCP is concerned,
auditfiles are simple sequential files opened output by the user's
NC. As such they may be modified at BOJ time by means of label
equation ("?FILE" cards). The internal file name is the <AUDITFILE
IDENTIFIER>. Default file attributes are device = disk, label =
<AUDITFILE IDENTIFIER>, buffers = 1 (unbuffered I/O is impossible
under MCP control), fixed-length records, save = 30 days. If device
type is disk then records = 180/1, areas = 40/100 (4000 record
maximum). if device type is TAPE, RECORDS = 180/1. If device type
is PRINTER, RECORDS = 132/1. The lock bit in the FPB is set to
ensure "CLOSE WITH LOCK" if the NC is DS'ed.

EXAMPLES:

        AUDITFILE AUDITDISK (DEVICE = DISKCARTRIDGE,
                        LABEL = "PACKNAME"/"MYFAMILY"/"MYFILE",
                        BUFFERS = 2, VARIABLE, SAVE = 7,
                        RECORDS = 180/10, AREAS = 1/1000).

        AUDITFILE DEFAULT ( ).

        AUDITFILE AUDITTAPE (DEVICE = TAPE, LABEL = "AUDITFILE"/"A",
                        RECORDS = 180/1).

        AUDITFILE PRNTR (DEVICE = PRINTER).

CONSTANT DECLARATION

SYNTAX:

<CONSTANT DECLARATION>::=   CONSTANT <CONSTANT DEFINITION LIST>.

<CONSTANT DEFINITION
LIST>::=   <CONSTANT DEFINITION>
                               | <CONSTANT DEFINITION>, <CONSTANT
                                 DEFINITION LIST>

<CONSTANT DEFINITION>::=   <CONSTANT IDENTIFIER>=<STRING>

<CONSTANT IDENTIFIER>::=   <IDENTIFIER>

SEMANTICS:

Constant declaration provides a means of equating identifiers with
strings. Subsequent references, occurrences of the identifier cause
the compiler to substitute the equated string. This is useful for
frequently used strings such as control characters, or common
concatenations of control characters (i.e. CONSTANT CRLF= 4
"0D250000" for "CARRIAGE RETURN-LINEFEED). Common control
characters have been defined within the compiler by default to the
following Hexadecimal values:

ACK	=	2E	EOT	=	37	NUL	=	00
BEL	=	2F	ESC	=	27	POL	=	97
CAN	=	18	ETB	=	26	RS	=	1E
CR	=	0D	ETX	=	03	SI	=	0F
DC1	=	11	FF	=	0C	SEL	=	98
DC2	=	12	FS	=	1C	SO	=	0E
DC3	=	13	FSL	=	A2	SOH	=	01
DC4	=	3C	GS	=	1D	STX	=	02
DEL	=	07	HT	=	05	SYN	=	32
DLE	=	10	LF	=	25	US	=	1F
ENQ	=	2D	NAK	=	30	VT	=	0B
FS	=	1C						

These identifiers may be redefined to other strings, but may not be
used as non-constant identifiers.

EXAMPLES OF CONSTANTS:

   I. Valid

```
CONSTANT NUL = 4"00",
     BAD1 = "ERROR",
     RETRANSMIT  = "PLEASE SEND AGAIN".
CONSTANT QUOTE = """".
```

  II. Invalid

```
CONSTANT 12KANGAROO = % IDENTIFIER MUST BEGIN WITH LETTER.
               12.   % not a string.
CONSTANT CLEAR      = % RESERVED WORD MAY NOT
                      % BE USED AS CONSTANT
          4"000".  = % MALFORMED STRING.
```

NIF NAME DECLARATION

SYNTAX:

<NIF NAME DECLARATION>::=   NIF = <FILE NAME>.

SEMANTICS:

The name specified is substututed in the NIF FPB and in the object
file's PPB for the default name "NDL"/"NIF".

MAX TALLY/TOG DECLARATION

SYNTAX:

```
MAX TALLY/TOG
DECLARATION>::=              MAX TALLY [ <INTEGER> ] .
                          I MAX LINE ( TALLY [ <INTEGER> ] ) .
                          I MAX TOG [ <INTEGER> ] .
                          I MAX LINE ( TOG [ <INTEGER> ] ) .
```

SEMANTICS:

The integer specifies the largest allowable TALLY or TOGGLE and
limits the TALLY or TOGGLE section of the line or station table.

RESTRICTIONS:

1)   No more than 100 TALLIES or TOGGLES may be
      declared, making the largest index 99.

2)   If the declared maximum is exceeded by an
      integer subscript, a warning will result and the
      maximum will be increased.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

REQUEST SECTION

SYNTAX:

```
<REQUEST SECTION>::=            <REQUEST LIST>

<REQUEST LIST>::=               <REQUEST> <REQUEST LIST>
                             I  <REQUEST>

<REQUEST>::=                    REQUEST <IDENTIFIER>:
                                <REQUEST STATEMENT LIST>

<REQUEST STATEMENT LIST>::=     <REQUEST STATEMENT>
                             I  <REQUEST STATEMENT>.
                                <REQUEST STATEMENT LIST>

<REQUEST STATEMENT>::=          <ASSIGNMENT STATEMENT>
                             I  <AUDIT STATEMENT>
                             I  <CASE STATEMENT>
                             I  <DISPLAY STATEMENT>
                             I  <DO STATEMENT>
                             I  <FETCH STATEMENT>
                             I  <FINISH STATEMENT>
                             I  <IF STATEMENT>
                             I  <INITIALIZE STATEMENT>
                             I  <INITIATE STATEMENT>
                             I  <NULL STATEMENT>
                             I  <RECEIVE STATEMENT>
                             I  <TERMINATE STATEMENT>
                             I  <TRANSMIT STATEMENT>
                             I  <UNDO STATEMENT>
                             I  <INCREMENT STATEMENT>
                             I  <DECREMENT STATEMENT>
```

SEMANTICS:

The REQUEST section is a series of line discipline routines (REQUESTs) which are used by the Network Controller in communicating with the various remote devices. A REQUEST must be defined for each terminal. If it is possible for a terminal to send input to the system and receive output from the system then both a receive and transmit REQUEST must be defined for that terminal. The REQUEST to be used for each of these conditions is specified by the terminal's <REQUEST STATEMENT> as discussed in the terminal section of this product specification.

REQUESTs are treated as I/O coroutines by the network CONTROLLer. For instance, when a line CONTROL desires to output a message to a TC500, the NC selects the correct transmit REQUEST from the TC500 terminal definition and gives it CONTROL. The TC500 transmit REQUEST executes the line discipline until the I/O is successful or until it detects a serious error, at which point the NC regains control. The REQUEST will also yield control to the NC while hardware I/O is in progress for the TC500 station. During this time the NC can process other lines or yield to the MCP, until the I/O is complete. Thus the NC can handle several datacomm I/O processes concurrently. Concurrency is transparent to the REQUEST, however. REQUESTs are coded as if I/O were instantaneous.

Four REQUEST statements are concerned with transfer of control -- "DO", "UNDO", "CASE", and "IF". NDL has no "GO TO". There is no means of declaring subroutines for a REQUEST, nor may the programmer declare local variables beyond those defined in the variables section.

REQUESTs are linked to specific terminals by referencing the REQUEST identifier in the <TERMINAL REQUEST STATEMENT> in the terminal section. REQUESTs are reentrant, so two stations using the same REQUEST may be handled concurrently on different lines.

ASSIGNMENT STATEMENT

SYNTAX:

<ASSIGNMENT STATEMENT>::=   <ASSIGNABLE VARIABLE>
                                 <ASSIGNER> <EXPRESSION>.

<ASSIGNABLE VARIABLE>::=   (SEE VARIABLES)

SEMANTICS:

The <ASSIGNMENT STATEMENT> is the means by which an NDL REQUEST
changes the value of an assignable variable. Examples:

        TALLY [0]:=TALLY [1] * 3.
        TOG [1]:=EXCEPTION OR LINE (TOG [0]).
        TIME (TALLY):=TIME.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

AUDIT STATEMENT

SYNTAX:

```
<AUDIT STATEMENT>::=          AUDIT <AUDITFILE IDENTIFIER>
                              (<AUDIT ELEMENT LIST>).

<AUDIT ELEMENT LIST>::=       <MESSAGE>
                            I <EXPRESSION LIST>
                            I <EXPRESSION LIST>, <MESSAGE>
                            I <MESSAGE>, <EXPRESSION LIST>
                            I <EXPRESSION LIST>, <MESSAGE>,
                              <EXPRESSION LIST>

<AUDITFILE IDENTIFIER>::=     (SEE <AUDITFILE DECLARATION
                              STATEMENT> IN DECLARATION SECTION)

<MESSAGE>::=                  BUFFER (<MESSAGE MODIFIER>)
                            I TEXT (<MESSAGE MODIFIER>)
                            I WORKAREA (<MESSAGE MODIFIER>)

<MESSAGE MODIFIER>::=         <IO>
                            I <IO>, <INTEGER>

<IO>::=                       INPUT
                            I OUTPUT

<EXPRESSION LIST>::=          <EXPRESSION>
                            I <EXPRESSION>, <EXPRESSION LIST>
```

SEMANTICS:

The audit statement is a write to a user-declared file. The data to
be written is the <AUDIT ELEMENT LIST>, formed by concatenating the
specified combination of <MESSAGE> and <EXPRESSION LIST>s. If the
concatenated string is shorter than the auditfile record length,
rightmost blanks will be added; if longer, rightmost truncation
will occur. See DECLARATION SECTION for the auditfile declaration
format.

<MESSAGE> breaks down as follows:

  1.   "TEXT" - message excluding the header

  2.   "BUFFER" - message including the entire 27-byte
        header.

  3.   "WORKAREA" - terminal header workarea, length as
        in <HEADER SIZE STATEMENT> in terminal section.

<INTEGER> option in the message modifier tells the NC how many
characters of the <MESSAGE> to audit. For example, this statement
writes the first 10 characters of the text to FILEA:

        AUDIT FILEA (TEXT (INPUT, 10)).

The default value for <INTEGER> is 0, indicating that the entire
message (up to the auditfile record size) is to be written.

<EXPRESSION LIST> may be included before or after or without
<MESSAGE>. The fields resulting from evaluation of each expression
are concatenated in the order specified. (See EXPRESSIONS for the
length of expression values). The string functions, "DECIMAL" and
"CONVERT", are useful here to change TALLIES, TOGGLES, and flags to
easily-printable format. This example results in a 36-character
string consisting of "TALLY 1 = nnn, INPUT = ttttttttt THATS ALL":

        AUDIT FILEA ("TALLY 1 =", DECIMAL (TALLY [1], 3), ", INPUT=",
        TEXT (INPUT, 10), "THATS ALL").

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

CASE STATEMENT

SYNTAX:

<CASE STATEMENT>::=   <CASE HEAD> <CASE BODY>

<CASE HEAD>::=   CASE <EXPRESSION>.

<CASE BODY>::=   <REQUEST STATEMENT LIST>
                             <CASE ENDING>

<CASE ENDING>::=   END CASE.

SEMANTICS:

The <EXPRESSION> serves as an index into the list of <REQUEST
STATEMENT>s, the statement selected is executed, and the others
ignored. Control is then transferred to the statement following the
<CASE ENDING> unless the statement is an "UNDO".

of <EXPRESSION> are from 0 through N-1.

The statements in the list may be any legal <REQUEST STATEMENT>
allowed in NDL. If the user wishes to execute nothing for a given
value of <EXPRESSION>, the <NULL STATEMENT> is appropriate.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

DECREMENT STATEMENT

SYNTAX:

<DECREMENT STATEMENT>::=   DECREMENT TRAN (<TRAN MODIFIER>).

<TRAN MODIFIER>::=   TRANSMIT
                           I RECEIVE

SEMANTICS:

The Decrement Statement decreases the value of an EBCDIC Decimal
encoded transmission number by 1. It is intended to be used in
systems where the MCS does not set output transmission numbers.

TRAN MODIFIER:

    TRAN (TRANSMIT) - OUTPUT TRANSMISSION NUMBER.

    TRAN (RECEIVE) - INPUT TRANSMISSION NUMBER.

Example: The transmission number of the last output message was
"380". DECREMENT TRAN (TRANSMIT) will decrease the value to "379"
(Hexadecimal F3F7F9). The field will be set to all "9"s on
underflow.

DISPLAY STATEMENT

SYNTAX:

<DISPLAY STATEMENT>::=   DISPLAY <EXPRESSION LIST>.

<EXPRESSION LIST>::=   <EXPRESSION>
                          | <EXPRESSION>, <EXPRESSION LIST>

SEMANTICS:

Display puts a character string out to the SPO via a communicate.

DO STATEMENT

SYNTAX:

`<DO STATEMENT>::=`	`<DO HEAD> <REQUEST STATEMENT LIST> <END PART>`
`<DO HEAD>::=`	`DO <IDENTIFIER PART> <FOREVER PART>.`
`<END PART>::=`	`END <INDENTIFIER PART>.`
`<IDENTIFIER PART>::=`	`<DO IDENTIFIER>`
	`I <EMPTY>`
`<FOREVER PART>::=`	`FOREVER`
	`I <EMPTY>`
`<DO IDENTIFIER>::=`	`<IDENTIFIER>`

SEMANTICS:

The DO Statement provides a means of grouping two or more REQUEST
statements (including other <DO STATEMENT>s) as one statement. This
is useful in <IF STATEMENT>s and <CASE $TATEMENT>s. DO Statements
may be nested. The DO-Identifier is optional. Example:

```
        IF TOG[1] THEN
        DO OUTER.
          DO INNER FOREVER.
            FETCH CHARACTER.
            IF CHAR EQ ETX THEN UNDO INNER.
          END INNER.
          TALLY [1] := TALLY [1] + 1.
        END OUTER.
```

FOREVER forces the <REQUEST STATEMENT LIST> to be repeated until an
UNDO or a TERMINATE is encountered.

RESTRICTIONS:

1. If a <DO IDENTIFIER> is included in the <DO HEAD>, it must also appear in the <END PART>.

2. If the <DO HEAD> does not include an identifier. The <END PART> must not contain one.

FETCH STATEMENT

SYNTAX:

<FETCH STATEMENT>::=   FETCH <POINTER ADVANCE PART>
                               <FETCH OBJECT>.

<POINTER ADVANCE PART>::=   0I1I<EMPTY>

<FETCH OBJECT>::=   CHAR
                             I CHARACTER
                             I <EMPTY>

SEMANTICS:

FETCH is only used in processing input. Before using FETCH, an
"INITIATE RECEIVE" should be executed. All <FETCH OBJECT>s have the
same meaning - FETCH always loads the character register with the
character   in the input message addressed by the message pointer.
The message pointer is advanced by 1 character after the FETCH if
the <POINTER ADVANCE PART> is 1 or <EMPTY>. A zero <POINTER ADVANCE
PART> leaves the message pointer unchanged, so a subsequent FETCH
or receive accesses the same character. N.B.: "ENDOFBUFFER" will be
set   if the FETCH is executed with the message pointer set beyond
the last character in the buffer.

        FINISH STATEMENT

SYNTAX:

<FINISH STATEMENT>::=   FINISH TRANSMIT <TRANSMIT
                               PART>-<TRANSMIT-RECEIVE PART>.

<TRANSMIT PART>::=   [<TRANSMIT LIST>]

<TRANSMIT LIST>::=   <TRANSMIT ATTRIBUTE>
                             I <TRANSMIT ATTRIBUTE>,<TRANSMIT LIST>

<TRANSMIT ATTRIBUTE>::=   <TRANSMIT BASIC ATTRIBUTE>
                             I NO <TRANSMIT BASIC ATTRIBUTE>

<TRANSMIT BASIC
ATTRIBUTE>::=   EOT I CR I TRANSLATE

<TRANSMIT-RECEIVE
PART>::=   INITIATE RECEIVE <RECEIVE PART>

<RECEIVE PART>::=   (SEE INITIATE RECEIVE).

SEMANTICS:

"FINISH TRANSMIT" causes transmission of the output message
constructed by a series of <TRANSMIT STATEMENT>s. "FINISH TRANSMIT
- INITIATE RECEIVE" links to the transmit I/O descriptor a receive
I/O descriptor which the hardware executes if the transmit
completes without error. (A Write-Flip-To-Read Operation).

The <TRANSMIT BASIC ATTRIBUTE>s set variants in the hardware I/O
descriptor; they are instructions to the B1700 Datacomm Hardware.

EOT : EOT is to be treated as a control character.

CR : CR is to be treated as a control character (teletype adapter only).

TRANSLATE : Translate EBCDIC to ASCII.

DEFAULTS : Translate, EOT, no CR

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

IF STATEMENT

SYNTAX:

<IF STATEMENT>::=   IF <EXPRESSION> THEN <REQUEST STATEMENT>
                            <ELSE PART>.

<ELSE PART>::=   ELSE <REQUEST STATEMENT>
                          I <EMPTY>

SEMANTICS:

The <EXPRESSION> is evaluated, and the statement following the
"THEN" is executed if the least significant bit of the value is
equal to one (true). If the bit is off (false) the "ELSE" statement
(if present) is executed. In all cases, control passes to the
statement following the IF statement. Any <REQUEST STATEMENT> is
legal after "THEN" and "ELSE" including <IF STATEMENT>s, <NULL
STATEMENT>s, and <DO STATEMENT>s. Example:

        IF CHAR NE ACK
        THEN DO. RECEIVE ADDRESS (RECEIVE).
                IF ADDERR
                THEN TERMINATE ERROR. % NO "ELSE"
            END.
        ELSE INITIATE RECEIVE.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

INCREMENT STATEMENT

SYNTAX:

```
<INCREMENT STATEMENT>::=      INCREMENT TRAN (<TRAN MODIFIER>)

<TRAN MODIFIER>::=            TRANSMIT
                           I RECEIVE
```

SEMANTICS:

The increment statement provides a means of changing the value of an
EBCDIC encoded transmission number by 1. It is intended to be used
with output transmission numbers in systems where the MCS does not
maintain the numbers.

TRAN MODIFIER:

     TRAN (TRANSMIT) - OUTPUT TRANSMISSION NUMBER

     TRAN (RECEIVE) - INPUT TRANSMISSION NUMBER

Example: The output transmission number of the last message was
"379".   "INCREMENT TRAN (TRANSMIT)." will bump the value to "380"
(in Hexadecimal notation: F3F8F0). The field will be set to
character zeroes on overflow.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

INITIALIZE STATEMENT

SYNTAX:

```
<INITIALIZE STATEMENT>::=     INITIALIZE <ITEM LIST>

<ITEM LIST>::=                <ITEM>
                           I  <ITEM>, <ITEM LIST>.

<ITEM>::=                     TEXT
                           I  TRAN (RECEIVE)
                           I  TRAN (TRANSMIT)
                           I  TALLY [<TALLY INDEX]
                           I  TOG [<TOG INDEX>]
                           I  RETRY
```

SEMANTICS:

"INITIALIZE TEXT" sets the input message pointer in the NC to the first byte of the input buffer (including the terminal header). It is done automatically by "INITIATE RECEIVE" and by a successful completion of autopoll.

"INITIALIZE TRAN (RECEIVE)" resets the station input transmission number to binary zeroes.

"INITIALIZE TRAN (TRANSMIT)" sets the station output transmission number from the output external message header if a message is attached, otherwise to character zeroes.

"INITIALIZE <TALLY/TOGGLE>" sets the listed TALLYS and TOGGLES from the internal 27-byte message header if there is one, else to zero.

"INITIALIZE RETRY" resets retry to its starting value (See <RETRY STATEMENT> in STATION section).

INITIATE STATEMENT

SYNTAX:

<INITIATE STATEMENT::=   INITIATE <INITIATE ACTION>

<INITIATE ACTION>::=   RECEIVE <RECEIVE PART>
                             I TRANSMIT

<RECEIVE PART>::=   <EMPTY>
                             I (<RECEIVE LIST>)

<RECEIVE LIST>::=   <RECEIVE ATTRIBUTE>
                             I <RECEIVE ATTRIBUTE>, <RECEIVE LIST>

<RECEIVE ATTRIBUTE>::=   <RECEIVE BASIC ATTRIBUTE>
                             I NO <RECEIVE BASIC ATTRIBUTE>

<RECEIVE BASIC
ATTRIBUTE>::=   CONTROL
                             I TRANSLATE
                             I TIMEOUT
                             I EOT
                             I CR
                             I PARITY

SEMANTICS:

The INITIATE RECEIVE form of the <INITIATE STATEMENT> commands the
adapter (i.e. .LINE) to accept input.

INITIATE TRANSMIT sets the output message pointer to the first byte
in the output message buffer,

CONTROL implies that the data will be input into the line input
CONTROL workarea as opposed to the input buffer.

TRANSLATE attribute causes hardware translation of data received.

TIMEOUT attribute enables the receive timeout timer.

EOT causes the adapter to treat EOT as a CONTROL character.

CR (teletype adapter only) causes the adapter to treat CR as a
CONTROL character.

PARITY (teletype adapter only) causes parity checking of each
character. Parity checking on other adapters will not be affected
by this option.

DEFAULTS: TIMEOUT, TRANSLATE, EOT, NO CR, NO CONTROL, NO PARITY.

NUL_ STATEMENT

SYNTAX:

<NULL STATEMENT>::=   <EMPTY>.

SEMANTICS:

The NUL_ statement may be used in an IF or CASE where an executable
statement is required but a NO-OP is desired. Example:

```
    CASE TALLY[3].
        IF TOG [3] THEN. % NULL
        ELSE INITIATE INPUT.
        . % NULL
        DO.
            <REQUEST STATEMENT LIST>
        END.
    END CASE.
```

Note that the case contains three executable statements, an IF, a
NULL, and a DO. If TALLY [0] is equal to 1 entering the case, the
NULL path is taken and the case is exited immediately. If TALLY [0]
is equal to 0, the if path is taken. If the THEN branch is taken,
this NULL also results in exit from the case.

RECEIVE STATEMENT

SYNTAX:

```
<RECEIVE STATEMENT>::=        RECEIVE <ITEM LIST>.

<ITEM LIST>::=                <ITEM>
                            | <ITEM>, <ITEM LIST>

<ITEM>::=                     ADDRESS <ADDRESS QUALIFIER>
                            | TEXT
                            | <STRING>
                            | <CONSTANT>
                            | TRAN <TRAN QUALIFIER>

<ADDRESS QUALIFIER>::=        (RECEIVE)
                            | (TRANSMIT)
                            | (STATION)

<TRAN QUALIFIER>::=           (RECEIVE)
                            | (TRANSMIT)
                            | <EMPTY>
```

SEMANTICS:

The receive statement is used in input REQUESTs following an
"INITIATE RECEIVE". Starting with the current setting of the input
message pointer, each item in the <ITEM LIST> is tested against the
corresponding field in the input message. If they do not agree, the
appropriate <ERROR FLAG> is set. Items are as follows:

ADDRESS

ADDRESS (RECEIVE)/ ADDRESS (TRANSMIT) refer to the strings specified
in the station's <STATION ADDRESS STATEMENT> (See STATION Section).
If the address in the input message does not agree with the
specified address, "ADDERR" is set.

ADDRESS (STATION)

The address field in the message will be compared against the receive addresses of all stations on the current line. If no match is found, ADDERR is set. If a match is found, the station index is reset to the matching station (n.b. This has serious implications for REQUEST coding - all the variables that are station-specific are now different. For instance, TALLY [0-2] and TOG [0-7] reference entirely different fields after a successful RECEIVE ADDRESS (STATION)). ADDRESS (STATION) provides the REQUEST writer with a means of implementing group poll, contention, and RJE line disciplines, since this is the only NDL language construct that permits a REQUEST to change station index at run time and respond to any of a set of stations which may have originated the current message.

TEXT

The input message pointer is set to the last character in the message, on the assumption that a terminating character follows. If more characters were received than specified in the station's <MAXINPUT STATEMENT>, ENDOFBUFFER is set. Note: A run error will result if RECEIVE TEXT is executed after INITIATE RECEIVE (CONTROL).

STRING, CONSTANT

FORMATERR is set if the corresponding characters don't agree.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

TRAN

TRAN (RECEIVE) / TRAN (TRANSMIT) refer to the transmission numbers
of the last input/output message, respectively. If the characters
in the input message are equal to the referenced last transmission
number,   TRANERR is set.   Tran   <EMPTY>   is   equivalent   to
TRAN(RECEIVE).

Example: If a certain terminal uses this input format- SOH, ADDRESS,
TRANSMISSION NUMBER, STX, TEXT, ETX - then it may be checked with
this receive series:

        RECEIVE SOH, ADDRESS, STX, TEXT, ETX.
        IF FORMATERR THEN UNDO.
        RECEIVE ADDRESS (RECEIVE), TRAN (RECEIVE).
        IF ADDERR OR TRANERR THEN UNDO.
        RECEIVE STX, TEXT, ETX.
        IF FORMATERR OR ENDOFBUFFER THEN UNDO.

TERMINATE STATEMENT

SYNTAX:

```
<TERMINATE STATEMENT>::=       TERMINATE <TERMINATE OPTION>.

<TERMINATE OPTION>::=          DISCONNECT
                             I ECHO
                             I ERROR
                             I INPUT
                             I INPUT (RETURN)
                             I INPUT (RETURN, NO BUFFER)
                             I LOGICALICK
                             I NOINPUT
                             I OUTPUT
                             I RELEASE STATION
                             I OUTPUT (RETURN)
```

SEMANTICS:

TERMINATE is used to exit a REQUEST. Depending on the option
selected, the TERMINATE may cause buffer allocation/deallocation,
message queueing/dequeueing, changes in station status, and saving
of the current REQUEST. If state is saved (See "RETURN" options)
then the REQUEST can be restarted at the statement following the
terminate. If state is not saved (all other options) then the
terminate ends the REQUEST.

All REQJESTs must end with a non-state-saving terminate. If none is
supplied, no syntax error will result. However, should control fall
past the last statement in the REQUEST, the compiler automatically
generates a TERMINATE ERROR at the end.

TERMINATE OPTIONS:

DISCONNECT

Used in REQUESTs when the last I/O had an error indicating loss of connection such as loss of data set ready. A break I/O with the disconnect variant is issued. If stations are still enabled on the line, a new test/wait for ring is sent.

INPUT

Used in receive REQUESTs to signal completion of an input line discipline. The result index is set to show successful input. The 27-byte message header is appended to the input text, and the input message is queued for the MCS/user program. Any output message still attached is assumed to have been successfully transmitted, so the output buffer is de-allocated. The line may now be used for other I/O. The REQUEST ends.

INPUT (RETURN)

Used to signal successful reception of a block (not the last) of input. As in "TERMINATE INPUT" the message is completed and queued. Another input buffer is allocated. No action is taken on attached output messages. Control is returned to the next REQUEST statement.

INPUT (RETURN, NO BUFFER)

Signals successful reception of an input message, but not completion of the line discipline. Input message is completed and queued. No new buffer is allocated. No action is taken on attached output messages. Flow of control returns immediately to the next statement in sequence. This option is intended to be used to overlap processing of an input message by the MCS/user program with acknowledgement of reception by the REQUEST.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

LOGICALACK

The input REQUEST is interrupted but not ended and the input message
queued. If LOGICALACK is true for this station and an MCS present,
the input message will be queued for the MCS; otherwise the input
message is queued for the user program and the REQUEST continues.

LOGICALACK is a method of verifying receipt of messages through an
MCS. The Network Controller will not restart if an error is
detected as indicated by the message result:

      1. INVALID LSN IN RESPONSE
      2. LINE NOT WAITING FOR RESPONSE
      3. INCORRECT LSN IN RESPONSE

OUTPUT

Successful completion of an output line discipline. A completion
message is queued for the MCS if required. Output buffer and any
attached input buffer are de-allocated. The REQUEST ends.

OUTPUT (RETURN)

Signals successful transmission of a block of output, but not
completion of the line discipline. The old output buffer is
de-allocated, but no action is taken on any attached input buffer.
The state of the REQUEST is saved, and the line CONTROL is entered
(LINE(CONTROL KEY)=4). The CONTROL must restart the REQUEST with
a "CONTINUE". When the REQUEST is restarted, it begins to execute
the statement following "TERMINATE OUTPUT (RETURN)". The REQUEST
should test the value of "OUTPUTATTACHED" to determine if another
output message has been attached before trying to transmit.

ECHO

This option is supplied for writing REQUESTs for the initial
shakedown of an NDL datacomm system. Instead of putting an input
message in the MCS queue, "TERMINATE ECHO" puts the message in the
station's output queue, so the station's output REQUEST can echo it
back. The input REQUEST is ended.

ERROR

Used to inform the NC of an unsuccessful attempt to transmit or
receive. An error message is queued for the MCS or user program. If
an MCS is present the station is marked down. Attached input
buffers are released, while attached output messages are re-queued.

NOINPUT

Signals an unsuccessful attempt to receive input, but without errors
(receiving an EOT, for example, ending a conversational line
discipline). The input buffer is de-allocated, and any output
message is re-queued for an output REQUEST. State is lost.
"TERMINATE NOINPUT" may be used in a transmit REQUEST just to
re-queue the output message, perhaps to broadcast a message to more
than one station.

RELEASE STATION

Releases a station not physically present on one line, making it
available to other lines in the line group. TERMINATE RELEASE
STATION clears the station table entry STATION(LINE) to zero.

TERMINATE OPTIONS

OPTION	INPUT MESSAGE	OUTPUT MESSAGE	STATE OF REQUEST	TYPE REQUEST
DISCONNECT	LOST	LOST	LOST	ALL
ECHO	QUEUED FOR TRANSMIT	NO ACTION	LOST	DEBUGGING
ERROR	LOST	RE-QUEUED	LOST	RECEIVE, TRANSMIT, CONVERSATIONAL
INPUT	QUEUED	LOST	LOST	RECEIVE OR CONVERSATIONAL
INPUT(RETURN)	QUEUED	NO ACTION	SAVED (IMMEDIATE RETURN)	CONVERSATIONAL OR BLOCK RECEIVE
LOGICALACK	QUEUED	LOST	SAVED	RECEIVE
OUTPUT	LOST	LOST	LOST	TRANSMIT OR CONVERSATIONAL
OUTPUT(RETURN)	NO ACTION	LOST	SAVED (CONTROL MUST CONTINUE)	CONVERSATIONAL OR BLOCK TRANSMIT
RELEASE STATION	LOST	QUEUED	LOST	RECEIVE,TRANSMIT
INPUT(RETURN, NO BUFFER)	QUEUED	NO ACTION	SAVED (IMMEDIATE RETURN)	RECEIVE
NOINPUT	LOST	RE-QUEUED IF NECESS.	LOST	RECEIVE, TRANSMIT

TRANSMIT STATEMENT

SYNTAX:

<TRANSMIT STATEMENT>::=   TRANSMIT <ITEM LIST>.

<ITEM LIST>::=   <ITEM>
                              I <ITEM>, <ITEM LIST>

<ITEM>::=   ADDRESS <ADDRESS QUALIFIER>
                              I TEXT
                              I <STRING>
                              I <CONSTANT IDENTIFIER>
                              I TRAN <TRAN QUALIFIER>
                              I CHAR
                              I CHARACTER

<ADDRESS QUALIFIER>::=   (RECEIVE)
                              I (TRANSMIT)

<TRAN QUALIFIER>::=   (MESSAGE)
                              I (RECEIVE)
                              I (TRANSMIT)
                              I <EMPTY>

SEMANTICS:

The <TRANSMIT STATEMENT> is used for output to a station. <ITEMS>
are concatenated until the <ITEM LIST> is exhausted. <ITEMS> from
subsequent <TRANSMIT STATEMENT>s are added to the concatenation
until a "FINISH TRANSMIT" statement is executed, at which time the
entire string is actually sent to the station.

TRAN (RECEIVE)/
TRAN (TRANSMIT) -   Transmission number in the station table.

TRAN <EMPTY> -   Equivalent to TRAN(TRANSMIT)

TRAN<EMPTY> -   Transmission number from 27-byte message
                              header

ADDRESS(RECEIVE)/
ADDRESS(TRANSMIT) -   Address from station table.

CHAR/CHARACTER -   Character register.

<CONSTANT IDENTIFER> -
/<STRING> -   Loads the specified characters.

UNDO STATEMENT

SYNTAX:

```
<UNDO STATEMENT>::=          UNDO <DO OPTION>.

<DO OPTION>::=               <EMPTY>
                          I  <DO IDENTIFIER>
```

SEMANTICS:

The UNDO statement is used to exit a DO-GROUP. Upon execution of an
UNDO, control transfers to the statement following the do-group's
"END".

DO OPTION:

   "UNDO." Transfers control out of the innermost DO-GROUP in
   which the statement appears.

   "UNDO <IDENTIFIER>." Transfers control out of the DO-GROUP
   with the same identifier.

EXAMPLE:

```
DO OUTER.
    DO.
        IF TOG[0] THEN UNDO. % EXITS UN-NAMED DO
        DO INNER.
            IF TOG[1] THEN UNDO.  % EXITS "INNER"
            UNDO OUTER.  % EXITS "OUTER"
        END INNER
    END.
END OUTER.
```

CONTROL SECTION

SYNTAX:

<CONTROL SECTION>::=   <CONTROL LIST>

<CONTROL LIST>::=   <CONTROL>
                             I <CONTROL> <CONTROL LIST>

<CONTROL>::=   CONTROL <CONTROL IDENTIFIER>:
                               <CONTROL STATEMENT LIST>

<CONTROL STATEMENT
LIST>::=   <CONTROL STATEMENT>
                             I <CONTROL STATEMENT>
                               <CONTROL STATEMENT LIST>

<CONTROL IDENTIFIER>::=   <IDENTIFIER>

<CONTROL STATEMENT>::=   <ASSIGNMENT STATEMENT>
                             I <CASE STATEMENT>
                             I <CONTINUE STATEMENT>
                             I <DISPLAY STATEMENT>
                             I <DO STATEMENT>
                             I <IF STATEMENT>
                             I <CONTROL INITATE STATEMENT>
                             I <NULL STATEMENT>
                             I <POLL STATEMENT>
                             I <UNDO STATEMENT>

SEMANTICS:

Each line in an NDL system must have a CONTROL associated with it by
means of the <LINE CONTROL STATEMENT> in the line section. The
user-written CONTROL is invoked by the NC when it is necessary to
decide which of a set of contending I/O functions gets the use of
the line. The duties of a CONTROL are

    1.   Creation of poll list, if any.

2.   Choosing a station on the line by testing system
     variables (LINE (CONTROL KEY), STATION (QUEUED),
     FREQUENCY, "STATION", etc.) or by polling.

3.   Initiation or restarting of the REQUEST for the
     chosen station.

REQUESTs are treated as I/O subroutines by CONTROLs. It is the
CONTROL that decides when and for which station a given REQUEST is
to be executed. The REQUEST is invoked by the <CONTROL INITIATE
STATEMENT>.

Unlike REQUESTs, CONTROLs may never save state and suspend
themselves.   CONTROLs are always entered at the first instruction
and run until they execute an <INITIATE STATEMENT> or a <CONTINUE
STATEMENT>, whereupon the CONTROL is terminated in favor of the NC
or the appropriate REQUEST.

The first thing a CONTROL should do is determine the state of the
line by checking the value of LINE(CONTROL KEY). (See SYSTEM STATUS
VARIABLES for the meaning of the different values). Certain values
of this item limit the range of action of the CONTROL or force the
CONTROL to take one course of action.

Six CONTROL statements (ASSIGNMENT, CASE, DO, IF, NULL, UNDO) are
virtually identical to the corresponding REQUEST statements. The
<CONTROL INITIATE STATEMENT> however serves an entirely different
purpose from the <INITIATE STATEMENT> in the REQUEST section.

ASSIGNMENT STATEMENT

SYNTAX:

```
<ASSIGNMENT STATEMENT>::=      <CONTROL ASSIGNABLE VARIABLE>
                               <ASSIGNER> <EXPRESSION>.

<CONTROL ASSIGNABLE
VARIABLE>::=                   (SEE VARIABLES)
```

SEMANTICS:

Used to change the value of a system variable. The variable to be
assigned, as well as any variables appearing in the expression,
must be referenceable in CONTROLs. Example:

```
        STATION:= STATION + 1.
```

CASE STATEMENT

SYNTAX:

```
<CASE STATEMENT>::=            CASE<EXPRESSION>.
                               <CONTROL STATEMENT LIST>
                               END CASE.

<CONTROL STATEMENT
LIST>::=                       <CONTROL STATEMENT>
                             I <CONTROL STATEMENT>
                               <CONTROL STATEMENT LIST>
```

SEMANTICS:

Same as case in the REQUEST section.

CONTINUE STATEMENT

SYNTAX:

<CONTINUE STATEMENT>::=   CONTINUE.

SEMANTICS:

The <CONTINUE STATEMENT> is used in conjunction with the "TERMINATE OUTPUT (RETURN)" statement in the REQUEST section. It returns control to the instruction following the previous "TERMINATE OUTPUT(RETURN)" statement in the REQUEST section.

DISPLAY STATEMENT

Same as display statement in REQUEST section.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

DO STATEMENT

SYNTAX:

```
<DO STATEMENT>::=          <DO HEAD>
                          <CONTROL STATEMENT LIST>
                          <END PART>

<DO HEAD>::=              DO<DO IDENTIFIER PART>
                          <FOREVER PART>.

<END PART>::=             END <DO IDENTIFIER PART>.

<DO IDENTIFIER PART>::=    EMPTY>
                        I <IDENTIFIER>

<FOREVER PART>::=          <EMPTY>
                        I FOREVER
```

SEMANTICS:

Same as REQUEST section <DO STATEMENT>.

IF STATEMENT

SYNTAX:

```
<IF STATEMENT>::=            IF<EXPRESSION> THEN <CONTROL STATEMENT>
                            <ELSE PART>

<ELSE PART>::=              ELSE <CONTROL STATEMENT>
                          I <EMPTY>
```

SEMANTICS:

Same as REQUEST section <IF STATEMENT>.

NOTE: Variables in the <EXPRESSION> must be CONTROL-referenceable.

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

        CONTROL INITIATE STATEMENT

SYNTAX:

<CONTROL INITIATE
STATEMENT>::=   INITIATE <INITIATE OPTION>

<INITIATE OPTION>::=   <I-O OPTION>
                                 | <AUTOPOLL OPTION>
                                 | <IDLE OPTION>
                                 | <CANCEL OPTION>
                                 | <DISCONNECT OPTION>

<I-O OPTION>::=   INPUT
                                 | OUTPUT
                                 | INPUTOUTPUT
                                 | OUTPUTINPUT
                                 | INPUT(NO BUFFER)
                                 | OUTPUT(NO BUFFER)

<AUTOPOLL OPTION>::=   AUTOPOLL<AUTOPOLL SLAVE ADDRESS>

<AUTOPOLL SLAVE ADDRESS>::= (<INTEGER>)
                                 | <EMPTY>

<IDLE OPTION>::=   IDLE

<CANCEL OPTION>::=   CANCEL

<DISCONNECT OPTION>::=   DISCONNECT.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

SEMANTICS:

INITIATE is a statement that (1) ends the CONTROL, and (2) depending
on the option, either starts an I/O REQUEST, cancels an I/O in
progress, starts hardware autopoll, or idles the line. The initiate
option executed must be appropriate to the current line and current
station status. For instance "INITIATE OUTPUT" if there is no
message queued for the current station will cause a fatal
run-error. It is the NDL programmer's responsibilty to either test
all relevant conditions before executing an initiate, or to
structure his NDL program so that the INITIATE cannot be
inappropriate. See following for requirements of each option.

I-O OPTION:

All I-O option variants begin a REQUEST. Which REQUEST is chosen depends on the variant and the station's <TERMINAL REQUEST STATEMENT> as defined in the terminal section.

INITIATE INPUT starts the input REQUEST, attaches an input buffer and sets INPUTATTACHED to true. The station must be valid, ready, enabled, and STATION (MYUSE) must allow input (See <MYUSE STATEMENT> in the station section). Example:

        IF STATION(VALID) AND STATION(READY) AND
        STATION (ENABLED) AND (STATION(MYUSE) NE 2)
        THEN INITIATE INPUT.

INITIATE INPUT(NO BUFFER) also starts the input REQUEST, but no input buffer is attached. To get an input buffer later, the REQUEST must execute a TERMINATE INPUT(RETURN). Same requirements as "INITIATE INPUT".

INITIATE OUTPUT starts the output REQUEST, attaches an output message, and sets OUTPUTATTACHED to true. The current station must be valid, ready, queued, and STATION(MYUSE) must permit output.

INITIATE OUTPUT(NO BUFFER) is just like INITIATE OUTPUT, but no message is attached, even though one must be queued for output in order to execute this initiate. To get the message while running, the REQUEST must execute a TERMINATE OUTPUT(RETURN).

INITIATE INPUTOUTPUT starts the input REQUEST, while INITIATE OUTPUTINPUT starts the output REQUEST. Otherwise they have the same effect - both get an input buffer and an output message, and both set INPUTATTACHED and OUTPUTATTACHED true. The station must be enabled, valid, ready and queued, and STATION(MYUSE) must equal 3 (I/O). These initiates are used to start conversational line disciplines.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

AUTOPOLL OPTION:

This initiate takes the poll string built by means of the <POLL
STATEMENT> and fires an I/O starting recirculating polling
("AUTOPOLL"). Polling continues automatically until a message is
received or an error occurs. Then "STATION" is set to the proper
value and the input REQUEST is entered, as if an "INITIATE INPUT"
had been excuted. The REQUEST should check for I/O exception.
Re-initiating autopoll causes polling to pick up where it left off.
The poll string need only be re-built to add or delete a station.

<AUTOPOLL SLAVE ADDRESS> is required only when the station may reply
with address different from the polled address, as in controller -
slave station configurations. The integer specifies the position of
the first character of the slave station address in the message
received. Note--the characters in the message are numbered from 1,
so if the address began in the third character of the message, the
integer should equal "3".

IDLE OPTION:

The line is put in a state of semi-permanent rest. It is marked not
busy but left ready. While idle, the line CONTROL will not be
entered, so the line will not respond to stations attempting input,
and any messages that were queued for output on the line at the
time it was idled will not be transmitted. This command might be
used to ease the load on the system by deactivating a line on which
no traffic is expected. Initiate idle terminates the CONTROL (it is
reentered from the top). The line can be reactivated by 3 events:

1.   An MCS "CHANGE" command to the NC.

2.   A file open by a user program involving a
      station on this line, if no MCS is present

3.   An output message is queued for a station on
      this line, and this station has no other
      messages queued.

CANCEL OPTION:

Halts the read I/O that is presently in progress on this line. Since a line can only perform one task at a time, the line CONTROL is not normally entered while an I/O is in progress. An exception occurs when

1.   The line is performing a read that can go on indefinitely (autopoll, read with no timeout), and

2.   The line has no output messages queued, and

3.   A   message is now queued for   output,   in contention with the on-going read.

The CONTROL is entered with LINE (CONTROL KEY) equal to 2, giving it an opportunity to stop the read with an "INITIATE CANCEL". After cancelling the read, the NC will re-enter the CONTROL with LINE (CONTROL KEY) equal to 0.

DISCONNECT OPTION:

Shuts down a line, releases associated stations, and hangs up. Used when all stations no longer require service (not enabled, timing out, not performing output, etc.).

NULL STATEMENT

Same as null statement in the REQUEST section.

POLL STATEMENT

SYNTAX:

```
<POLL STATEMENT>::=          POLL<POLL ELEMENT LIST>.

<POLL ELEMENT LIST>::=       <POLL ELEMENT>
                           | <POLL ELEMENT>,<POLL ELEMENT LIST>

<POLL ELEMENT>::=            <CONSTANT IDENTIFIER>
                           | <STRING>
                           | <ADDRESS>

<ADDRESS>::=                ADDRESS
                           | ADDRESS(TRANSMIT)
```

SEMANTICS:

Used to create a poll string prior to initiating autopoll (See
AUTOPOL_ option of <CONTROL INITIATE STATEMENT>). The line CONTROL
should cycle through the stations on the line, executing POLL once
for each station to go in the poll string. A <POLL ELEMENT LIST>
must contain exactly one occurrence of <ADDRESS>. ADDRESS and
ADDRESS(TRANSMIT) are equivalent in the poll statement. The poll
list should be rebuilt when LINE (CONTROL KEY)= 1 (See VARIABLES,
LINE(CONTROL KEY)).

RESTRICTIONS:

1.   Every <POLL ELEMENT LIST> entered in the same
     poll string must have the same total length.

2.   Maximum   length of a <POLL ELEMENT LIST> is 15
     characters.

3.   The total length of the poll string should not
     exceed the length specified in the <AUTOPOLL
     SIZE STATEMENT> in the line section. If it is
     exceeded, ENDOFBUFFER is set true.

4.   The <POLL STATEMENT> should be executed only for
      stations that are ready, valid, and enabled, and
      myuse must permit input.

5.   Group poll: If   the   system   includes
      controller-slave   stations (TU910 - TU510,   for
      example),   STATION (TYPE) should be checked   to
      avoid polling slave stations.

   EXAMPLE:

```
      STATION := 1.
      DO FOREVER;
          IF STATION (VALID) AND STATION (READY) AND
          STATION (ENABLED) AND (STATION(MYUSE) NE 2) AND
          (STATION(TYPE)NE 1)
          THEN POLL EOT, ADDRESS,POL, 4"2E".
          IF ENDOFBUFFER THEN UNDO.
          IF STATION EQ MAXSTATIONS THEN UNDO,
          STATION:= STATION + 1.
      END.
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

UNDO STATEMENT

SYNTAX:

<UNDO STATEMENT>::=   UNDO <DO OPTION>.

<DO OPTION>::=   <DO IDENTIFIER>
                          I <EMPTY

SEMANTICS:

Same as undo in REQUEST section.

TERMINAL SECTION

SYNTAX:

```
<TERMINAL SECTION>::=          <TERMINAL LIST>

<TERMINAL LIST>::=             <TERMINAL DEFINITION>
                          I  <TERMINAL DEFINITION> <TERMINAL LIST>
                          I  <TERMINAL DEFAULT DEFINITION> <TERMINAL
                             LIST>

<TERMINAL DEFINITION>::=       TERMINAL <TERMINAL IDENTIFIER>:
                              <TERMINAL ATTRIBUTE LIST>

<TERMINAL DEFAULT>
DEFINITION>::=                 TERMINAL DEFAULT <TERMINAL IDENTIFIER>:
                              <TERMINAL ATTRIBUTE LIST>

<TERMINAL ATTRIBUTE
LIST::=                        <TERMINAL ATTRIBUTE STATEMENT>.
                          I  <TERMINAL ATTRIBUTE STATEMENT>.
                              <TERMINAL ATTRIBUTE LIST>

<TERMINAL ATTRIBUTE
STATEMENT>::=                  <HEADER SIZE STATEMENT>
                          I  <MAXINPUT STATEMENT>
                          I  <TERMINAL ADDRESS STATEMENT>
                          I  <TERMINAL DEFAULT STATEMENT>
                          I  <TERMINAL DIAGNOSTIC REQUEST STATEMENT>
                          I  <TERMINAL REQUEST STATEMENT>
                          I  <TERMINAL TYPE STATEMENT>
                          I  <TRANSMISSION NUMBER STATEMENT>

<TERMINAL IDENTIFIER>::=       <IDENTIFIER>

<TERMINAL DEFAULT
IDENTIFIER>::=                 <IDENTIFIER>
```

RESTRICTIONS:

1. At least one <TERMINAL DEFINITION> must appear in the
    <TERMINAL SECTION>.

2. A <TERMINAL DEFAULT DEFINITION> must predede any
    reference to it in a <TERMINAL DEFAULT STATEMENT>.

3. The <TERMINAL DEFAULT STATEMENT> may appear in a
    <TERMINAL DEFINITION> but not in a <TERMINAL DEFAULT
    DEFINITION> (defaults may not be nested).

SEMANTICS:

The terminal section contains descriptions of every different type
of remote device connected to the user's datacomm system. For
example, if the system consists of TC-500's and TD-700's, two
terminal definitions are required in the terminal section, one for
the TC-500 and one for the TD-700. This section is linked to the
rest of the NDL program as follows: Terminals are associated with
REQUESTs by the <TERMINAL REQUEST STATEMENT> and the <TERMINAL
DIAGNOSTIC REQUEST STATEMENT>. A station is associated with a
terminal definition by the <TERMINAL STATEMENT> in the STATION
section.

ATTRIBUTE STATEMENTS:

Only the <TERMINAL REQUEST STATEMENT> and the <MAXINPUT STATEMENT> are absolutely required in every <TERMINAL DEFINITION>. Compiler warnings will be printed if the terminal address, reader size, type, and transmission number statements are omitted.

TERMINAL DEFAULT DEFINITIONS:

For terminals which have several common attributes, the common attributes can be grouped in a <TERMINAL DEFAULT DEFINITION>. The remaining attributes can be listed explicitly under each individual <TERMINAL DEFINITION>, and a <TERMINAL DEFAULT STATEMENT> will complete the definition.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

TERMINAL DEFAULT STATEMENT

SYNTAX:

<TERMINAL DEFAULT
STATEMENT>::=   DEFAULT = <TERMINAL DEFAULT IDENTIFIER>

SEMANTICS:

This statement specifies the identifier of a preceding <TERMINAL
DEFAULT DEFINITION> . The attributes in the default are added to
the explicit attributes contained in the <TERMINAL DEFINITION> in
which this statement appears.

RESTRICTIONS:

1.   If a default attribute conflicts with an explicit
      attribute in the same terminal definition, the
      explicit attribute is given precedence.

2.   This statement cannot appear in a <TERMINAL DEFAULT
      DEFINITION>.

EXAMPLE:

Assume terminals A1, A2, and A3 have the same attributes. A
<TERMINAL DEFAULT STATEMENT> could be used as illustrated below to
simplify the coding of the descriptions of these 3 terminals.

```
        TERMINAL DEFAULT A123DFLT:
        REQUEST        = TWXREAD:RECEIVE,TWXWRITE:TRANSMIT.
        MAXINPUT       = 72.
        ADDRESS        = NULL.

        TERMINAL A1:
        DEFAULT        = A123DFLT.
        MAXINPUT       = 95.

        TERMINAL A2:
        DEFAULT        = A123DFLT.
        ADDRESS        = 2

        TERMINAL A3:
        DEFAULT        = A123DFLT.
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

TERMINAL ADDRESS STATEMENT

SYNTAX:

```
<TERMINAL ADDRESS
STATEMENT>::=            ADDRESS = <RECEIVE ADDRESS>
                        <TRANSMIT ADDRESS PART>
                    I ADDRESS = NULL

<RECEIVE ADDRESS>::=     <ADDRESS SIZE>

<TRANSMIT ADDRESS PART>::=   <TRANSMIT ADDRESS> I<EMPTY>

<TRANSMIT ADDRESS>::=    <ADDRESS SIZE>

<ADDRESS SIZE>::=        1 I 2 I 3
```

SEMANTICS:

The <TERMINAL ADDRESS STATEMENT> specifies the number of characters
which are used to address a station associated with a terminal. An
empty <TRANSMIT ADDRESS PART> implies the transmit address is of
the same length as the receive address. A non-empty <TRANSMIT PART>
indicates that the transmit address may be non-identical to the
receive address. If this statement is omitted, both address lengths
are assumed to be zero.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

MAXINPUT STATEMENT

SYNTAX:

<MAXINPUT STATEMENT>::= MAXINPUT = <INTEGER>

SEMANTICS:

The <MAXINPUT STATEMENT> applies only to inputs from the terminal
and defines the maximum size (in characters) of an input message.
The <INTEGER> must be less than 4096. This statement is required in
all terminal definitions. 27 bytes for the header and 5 bytes for
the trailer are added to the maxinput size to determine the input
buffer length.

HEADER SIZE STATEMENT

SYNTAX:

<HEADER SIZE STATEMENT>::=   HEADER = <INTEGER>

SEMANTICS:

The <HEADER SIZE STATEMENT> defines the expected size of the
text   header (i.e. SOH---STX) on input from this terminal. The
<INTEGER>   must be less than or equal to 27. Size is assumed to
be zero if this statement is omitted.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

TERMINAL REQUEST STATEMENT

SYNTAX:

```
<TERMINAL REQUEST
STATEMENT>::=              REQUEST = <REQUEST SPECIFIER LIST>

<REQUEST SPECIFIER LIST>::=  <INPUT SPECIFIER>
                          I  <OUTPUT SPECIFIER>
                          I  <INPUT SPECIFIER>, <OUTPUT SPECIFIER>
                          I  <OUTPUT SPECIFIER>, <INPUT SPECIFIER>

<INPUT SPECIFIER>::=       <REQUEST IDENTIFIER>: RECEIVE

<OUTPUT SPECIFIER>::=      <REQUEST IDENTIFIER>:  TRANSMIT
```

SEMANTICS:

This required statement links the terminal with one of two line
disciplines (REQUESTs). When input is to be received from a
station, the station's terminal-table entry is checked and the
REQUEST identified in the <INPUT SPECIFIER> is called.
Similarly on output, the REQUEST from the <OUTPUT SPECIFIER> is
called.

The <REQUEST LIST> may specify either the input REQUEST or the
output REQUEST, or both. The same REQUEST can be used for both
input and output. For example:

    REQUEST = IOTTY33:RECEIVE, IOTTY33:TRANSMIT.

TRANSMISSION NUMBER STATEMENT

SYNTAX:

```
<TRANSMISSION NUMBER
STATEMENT>::=              TRANSMISSION = <TRANSMISSON NUMBER>

<TRANSMISSION NUMBER>::=   0 I 1 I 2 I 3 I NULL
```

SEMANTICS:

The <TRANSMISSION NUMBER STATEMENT> specifies the length of the
transmission number. The number zero and NULL are equivalent and
specify that no transmission number will be used. A non-NULL
transmission number must be specified if the item "TRAN" is
mentioned in a REQUEST of the terminal.

TERMINAL DIAGNOSTIC REQUEST STATEMENT

SYNTAX

```
<TERMINAL DIAGNOSTIC
REQUEST STATEMENT>::=      DIAGNOSTIC =
                          <REQUEST SPECIFIER LIST>
```

SEMANTICS:

This optional statement specifies an alternate REQUEST that can be
used for diagnostic reasons. A "DCWRITE" from the MCS is required
to initiate the diagnostic REQUESTs.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

TERMINAL TYPE STATEMENT

SYNTAX:

<TERMINAL TYPE STATEMENT>::=   TYPE = <TYPE NUMBER>

<TYPE NUMBER>::=   <INTEGER>

SEMANTICS:

On input messages, an eight-bit field in the message header is
automatically set to the terminal type specified for this current
station. The value of <TYPE NUMBER> must be less than 64. The
following hardware terminal types are recommended.

1	=	TC500
2	=	CONRAC (WIDE)
3	=	RT2000
4	=	TOUCH TONE
5	=	TU300
6	=	TC3500
7	=	DC140
8	=	TC4000
9	=	TC5100
10	=	TT102
11	=	TC700
12	=	CONRAC (NARROW)
13	=	DC110
14	=	TELETYPE
15	=	TU400
21	=	TC1700
22	=	BIDS
25	=	TU500
32	=	TD700
35	=	TU700
42	=	TD800
45	=	TU800
62	=	B1700
63	=	B6700

The user, however, has the option of assigning any hardware type and this will be passed in the message header. If this statement does not appear in a terminal definition, a compiler warning will be printed and type will be assumed zero.

STATION SECTION

SYNTAX:

```
<STATION SECTION>::=        <STATION LIST>

<STATION LIST>::=           <STATION DEFINITION>
                          I <STATION DEFINITION> <STATION LIST>
                          I <STATION DEFAULT DEFINITION> <STATION LIST>

<STATION DEFINITION>::=     STATION <STATION IDENTIFIER>:
                           <STATION ATTRIBUTE LIST>

<STATION DEFAULT
DEFINITION>::=             STATION DEFAULT <STATION
                           DEFAULT IDENTIFIER>:
                           <STATION ATTRIBUTE LIST>

<STATION ATTRIBUTE LIST>::= <STATION ATTRIBUTE STATEMENT>.
                          I <STATION ATTRIBUTE STATEMENT>.
                           <STATION ATTRIBUTE LIST>

<STATION ATTRIBUTE
STATEMENT>::=              <CONTROLLER STATEMENT>
                          I <FREQUENCY STATEMENT>
                          I <LOGICALACK STATEMENT>
                          I <MCSQUEUE STATEMENT>
                          I <MYUSE STATEMENT>
                          I <RETRY STATEMENT>
                          I <STATION ADDRESS STATEMENT>
                          I <STATION DEFAULT STATEMENT>
                          I <STATION TERMINAL STATEMENT>

<STATION IDENTIFIER>::=     <IDENTIFIER>

<STATION DEFAULT
IDENTIFIER>::=             <IDENTIFIER>
```

RESTRICTIONS:

1. At least one <STATION DEFINITION> must appear in
   the <STATION SECTION>.

2. A <STATION DEFAULT DEFINITION> must precede any
   reference   to it   in   a   <STATION   DEFAULT
   STATEMENT>.

3. The <STATION DEFAULT STATEMENT> may appear in a
   <STATION DEFINITION> but not in a <STATION
   DEFAULT DEFINITION> (no nested defaults).

SEMANTICS:

Each individual remote device in an NDL system must be defined in
the   station section. As in the terminal   section, default
definitions are provided as a coding aid.

CONTROLLER STATEMENT

SYNTAX:

<CONTROLLER STATEMENT>::=   CONTROLLER = <STATION CONTROLLER
                               TYPE>

<STATION CONTROLLER TYPE>::= TRUE
                           I FALSE
                           I <STATION IDENTIFIER>

SEMANTICS:

The statement is only required for stations that are part of a
CONTROLLER SLAVE station configuration, like TU910 - TU510's. The
<CONTROLLER STATEMENT> makes it possible to use group polling in
the line CONTROL (See <POLL STATEMENT>). A station's CONTROLLER
status can be accessed at run-time by using the variable
STATION(TYPE).

CONTROLLER TYPE   MEANING

     TRUE   This station is a controller, like TU910.

     FALSE   This station is a regular remote device.
                          This is the default value.

     <STATION IDENTIFIER> This station is a slave, like a TU510,
                          and <STATION IDENTIFIER> is its controller.
                          <STATION IDENTIFIER> must refer to
                          a preceding station definition.

FREQUENCY STATEMENT

SYNTAX:

```
<FREQUENCY STATEMENT>::=     FREQUENCY = <IO FREQUENCY>

<IO FREQUENCY>::=              <INTEGER>,<INTEGER>
                            I <INTEGER>
```

SEMANTICS:

The <FREQUENCY STATEMENT> specifies a priority for input and output
for use by the station in the line CONTROL section. The maximum
number which may be specified is 255. The first integer defines the
input priority and the second integer defines the output priority.
The default value is zero.

```
    EXAMPLE:

        FREQUENCY = 3,7.
        FREQUENCY = 3.
```

LOGICALACK STATEMENT

SYNTAX:

```
<LOGICALACK STATEMENT>::=     LOGICALACK = <LOGICAL VALUE>
```

SEMANTICS:

TRUE is required if the REQUEST is actually to be interrupted.

If this statement is omitted, LOGICALACK defaults to "FALSE".

MCSQUEUE STATEMENT

SYNTAX:

```
<MCSQUEUE STATEMENT>::=      MCSQUEUE = <QUEUE NAME>

<QUEUE NAME>::=              <QUEUE FAMILY>
                         I  <QUEUE FAMILY> <SLASH> <QUEUE OFFSPRING>

<QUEUE FAMILY>::=            <EBCDIC STRING>

<QUEUE OFFSPRING>::=         <EBCDIC STRING>
```

SEMANTICS:

This statement informs the NC of 1) the presence of an MCS in the
user's .datacomm system, and 2) the name of the MCS's input queue.
It is required if and only if an MCS is to be present. The actual
name used by the MCS queue is a 20-character field. The <QUEUE
FAMILY> is left-justified in the first 10 characters, while <QUEUE
OFFSPRING> is left-justified in the second 10 characters. Examples:

    MCSQUEUE = "USERSMCS"/ "INPUTQUEUE".

    MCSQUEUE = "MCSIN".

If the user intends to use the ILLUSTRATIVE MCS supplied by
Burroughs, set MCSQUEUE to "MCS.QUEUE".

For the V.0 release, this statement need only appear once in the
station section, in any station definition.

MYUSE STATEMENT

SYNTAX:

```
<MYUSE STATEMENT>::=        MYUSE = <IO LIST>

<IO LIST>::=               INPUT
                         I  OUTPUT
                         I  INPUT, OUTPUT
                         I  OUTPUT, INPUT
```

SEMANTICS:

This required statement specifies the allowed use of a station--
input, output, or both. It should be consistent with the <REQUEST
STATEMENT> of this station's terminal definition (See terminal
REQUEST statement) and with the I/O initiation in the line CONTROL
(See <CONTROL INITIATE STATEMENT>). MYUSE can be changed at
run-time by an MCS communicate. The MYUSE status can be checked at
run-time by accessing the variable STATION(MYUSE).

IO LIST	STATION (MYUSE) VALUE
INPUT	1
OUTPUT	2
INPUT, OUTPUT	3
OUTPUT, INPUT	3

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

RETRY STATEMENT

SYNTAX:

<RETRY STATEMENT>::=   RETRY = <INTEGER>

SEMANTICS:

The <RETRY STATEMENT> gives the initial retry count for the station.
At run time, the value may be changed by the MCS or by INITIALIZE
RETRY in a REQUEST. The <INTEGER> must be less than 256.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

STATION ADDRESS STATEMENT

SYNTAX:

```
<STATION ADDRESS
STATEMENT>::=              ADDRESS = <STATION ADDRESS LIST>

<STATION ADDRESS LIST>::=  <RECEIVE-TRANSMIT ADDRESS>
                         | <RECEIVE ADDRESS>, <TRANSMIT ADDRESS>

<RECEIVE-TRANSMIT
ADDRESS>::=               <EBCDIC STRING>

<RECEIVE ADDRESS>::=      <EBCDIC STRING>

<TRANSMIT ADDRESS>::=     <EBCDIC STRING>
```

SEMANTICS:

The <STATION ADDRESS STATEMENT> is used to specify the address of a
station for operations such as polling and selecting. If no
<STATION ADDRESS STATEMENT> is defined for a station, the mode of
the associated terminal must be contention. The length of the
address specified must be equal to the length as defined in the
<TERMINAL ADDRESS STATEMENT> of the associated terminal.

An address pair should be defined only if the station's receive and
transmit addresses are different. The first member of an address
pair is the receive address, the second the transmit address.
Examples:

        ADDRESS = "01".

        ADDRESS = "010", "011".

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

STATION DEFAULT STATEMENT

SYNTAX:

<STATION DEFAULT
STATEMENT>::=   DEFAULT = <STATION DEFAULT IDENTIFIER>

SEMANTICS:

The <STATION DEFAULT STATEMENT> is used in the same manner that a
<TERMINAL DEFAULT STATEMENT> is used. The <TERMINAL DEFAULT
STATEMENT> is discussed under the description of terminal default
semantics.

STATION TERMINAL STATEMENT

SYNTAX:

<STATION TERMINAL
STATEMENT>::=   .   TERMINAL = <TERMINAL IDENTIFIER>

SEMANTICS:

This statement is required in every <STATION DEFINITION>. The
<STATION TERMINAL STATEMENT> specifies the name of the terminal
description which defines the physical characteristics of the
station being described. More than one station description may
reference the same terminal description.

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

LINE SECTION

SYNTAX:

<LINE SECTION>::=   <LINE LIST>

<LINE LIST>::=   <LINE DEFINITION>
                                  I <LINE DEFINITION> <LINE LIST>
                                  I <LINE DEFAULT DEFINITION> <LINE LIST>

<LINE DEFINITION>::=   LINE <LINE IDENTIFIER>:
                                    <LINE ATTRIBUTE LIST>

<LINE DEFAULT DEFINITION>::= LINE DEFAULT <LINE DEFAULT
                                    IDENTIFIER>:
                                    <LINE ATTRIBUTE LIST>

<LINE ATTRIBUTE LIST>::=   <LINE ATTRIBUTE STATEMENT>.
                                  I <LINE ATTRIBUTE STATEMENT>. <LINE
                                    ATTRIBUTE LIST>

<LINE ATTRIBUTE
STATEMENT>::=   <AUTOPOLL SIZE STATEMENT>
                                  I <LINE ADDRESS STATEMENT>
                                  I <LINE CONTROL STATEMENT>
                                  I <LINE DEFAULT STATEMENT>
                                  I <LINE STATION STATEMENT>
                                  I <LINE TYPE STATEMENT>
                                  I <MAXINWORK STATEMENT>
                                  I <MAXOUTWORK STATEMENT>

<LINE IDENTIFIER>::=   <IDENTIFIER>

<LINE DEFAULT
IDENTIFIER>::=   <IDENTIFIER>

RESTRICTIONS:

1. At least one <LINE DEFINITION> must appear in the line section.

2. A <LINE DEFAULT DEFINITION> must precede a reference to it in a <LINE DEFAULT STATEMENT>.

3. The <LINE DEFAULT STATEMENT> may appear in a <LINE DEFINITION> but not in a <LINE DEFAULT DEFINITION> (defaults may not be nested).

4. The <LINE STATION STATEMENT> may not appear in <LINE DEFAULT DEFINITION>s. A station may only be on one line.

5. The <LINE ADDRESS STATEMENT> may not appear in <LINE DEFAULT DEFINITION>s.

SEMANTICS:

Each line in an NDL system must be described in the line section. As in the terminal section, default definitions are provided as a coding aid.

BURRJUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

AUTOPOLL SIZE STATEMENT

SYNTAX:

<AUTOPOLL SIZE STATEMENT>::=   AUTOPOLL = <INTEGER>

SEMANTICS:

The <AUTOPOLL SIZE STATEMENT> defines the maximum size in characters
of an autopoll buffer. This buffer will contain the poll list for
the line as supplied by the line CONTROL section through the use of
the "POLL" syntax. The maximum value of <INTEGER> is 1023.

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

    LINE ADDRESS STATEMENT

SYNTAX:

<LINE ADDRESS STATEMENT>::=   <LINE ADDRESS LIST>

<LINE ADDRESS LIST>::=   <LINE ADDRESS>
                            | <LINE ADDRESS> , <LINE ADDRESS LIST>

<LINE ADDRESS>::=   <PORT NUMBER> : <CHANNEL ADDRESS> :
                             <ADAPTER ADDRESS

<PORT NUMBER>::=   0|1|2|3|4|5|6|7

<CHANNEL ADDRESS>::=   0|1|2|3|4|5|6|7|8|9|10|
                             11|12|13|14|15

<ADAPTER ADDRESS>::=   0|1|2|3|4|5|6|7|8|9|10
                             11|12|13|14|15

SEMANTICS:

Required for each line, the line address statement is used to
specify the port number, the channel number and the adapter number
within the channel to which the line is attached. Two lines may not
have the same address. Example:

        ADDRESS = 2:0:15.

This statement would appear in the line description of the line
which is connected to channel number zero at adapter number 15 of
port number two.

With switched lines, a particular line may have several alternate
addresses. The addresses will be checked in the order listed.

LINE CONTROL STATEMENT

SYNTAX:

<LINE CONTROL STATEMENT>::=   CONTROL = <CONTROL IDENTIFIER>

SEMANTICS:

This required statement associates a CONTROL procedure with this
line. The <CONTROL IDENTIFIER> must have been defined in the
CONTROL section.

LINE DEFAULT STATEMENT

SYNTAX:

<LINE DEFAULT STATEMENT>::=   DEFAULT=<LINE DEFAULT IDENTIFIER>

SEMANTICS:

The <LINE DEFAULT STATEMENT> is used in the same manner that a
<TERMINAL DEFAULT STATEMENT> is used. The <TERMINAL DEFAULT
STATEMENT> is descussed under the description of terminal default
semantics.

LINE STATION STATEMENT

SYNTAX:

```
<LINE STATION STATEMENT>::=   STATION = <LIST OF STATIONS>

<LIST OF STATIONS>::=         <STATION IDENTIFIER>
                            I <LIST OF STATIONS>, <STATION IDENTIFIER>
```

SEMANTICS:

Required for each line, the <LINE STATION STATEMENT> identifies the
stations on a line. Lines designated as passive must not have a
<LINE STATION STATEMENT>. A station may be associated with only one
line and may not be defined twice for the same line. <STATION
IDENTIFIER> must have been defined in the STATION section.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

LINE TYPE STATEMENT

SYNTAX:

<LINE TYPE STATEMENT>::=   TYPE = PASSIVE

SEMANTICS:

A line may be declared in NDL but kept inactive through use of the
passive type statement.

This statement is optional and if omitted, the line will be
considered active and its type determined through the hardware.

```
EXAMPLE:
     LINE L1:
          ADDRESS = 4:2:1.
          CONTROL = AUTOPOLCTL.
          STATION = B1.
          TYPE = PASSIVE.
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

MAXINWORK STATEMENT

SYNTAX:

<MAXINWORK STATEMENT>::=   MAXINWORK = <INTEGER>

SEMANTICS:

Defines the maxium size in bytes of an input workarea for the line.
It should be set just large enough to contain the longest CONTROL
message expected from any station on the line. The workarea is
located within base-to-limits, in the line table. Workarea is
intended to be used only to accept CONTROL message (See "INITIATE
RECEIVE (CONTROL)" in the REQUEST section), since text would
overflow it.

Note: <INTEGER> may not exceed 27. The default value is zero.

MAXOUTWORK STATEMENT

SYNTAX:

<MAXOUTWORK STATEMENT>::=   MAXOUTWORK = <INTEGER>

SEMANTICS:

<MAXOUTWORK STATEMENT> defines the maximum size of an output
workarea for the line. This output workarea may be used for
transmitting CONTROL messages to a station such as "ACK" using the
syntax "TRANSMIT ACK" "FINISH TRANSMIT" in the REQUEST section. The
<INTEGER> must be at least as large as the maximum size of any
header sent with a message on the line (i.e. SOH . . . STX) and
must not be greater than 27. The default value is 0.

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

FILE SECTION

SYNTAX

```
<FILE SECTION>::=               <FILE LIST>

<FILE LIST>::=                  <FILE DEFINITION>
                             I  <FILE LIST> <FILE DEFINITION>
                             I  <FILE DEFAULT DEFINITION> <FILE LIST>

<FILE DEFINITION>::=            <FILE IDENTIFIER>:<FILE ATTRIBUTE LIST>

<FILE DEFAULT
DEFINITION>::=                  FILE DEFAULT <FILE DEFAULT IDENTIFIER>:
                               <FILE ATTRIBUTE LIST>

<FILE ATTRIBUTE LIST>::=        <FILE ATTRIBUTE STATEMENT>.
                             I  <FILE ATTRIBUTE LIST>
                               <FILE ATTRIBUTE STATEMENT>.


<FILE ATTRIBUTE
STATEMENT>::=                   <FAMILY STATEMENT>
                             I  <FILE DEFAULT STATEMENT>
                             I  <FILE RESIDENT STATEMENT>

<FILE IDENTIFIER>::=            <IDENTIFIER>

<FILE DEFAULT
IDENTIFIER>::=                  <IDENTIFIER>
```

RESTRICTION:

The <FILE DEFAULT STATEMENT> may appear in <FILE DEFINITION>s but not in <FILE DEFAULT DEFINITION>s (DEFAULTS may not be nested). <FAMILY STATEMENT> may not appear in a <FILE DEFAULT DEFINITION>.

SEMANTICS:

The file section associates stations that are part of an NDL network with remote files of user programs.

FILE DEFAULT STATEMENT

SYNTAX

<FILE DEFAULT STATEMENT>::= DEFAULT = <FILE DEFAULT IDENTIFIER>

SEMANTICS:

The <FILE DEFAULT STATEMENT> is used in the same manner that a <TERMINAL DEFAULT STATEMENT> is used. The <TERMINAL DEFAULT STATEMENT> is discussed under the description of terminal default semantics.

FAMILY STATEMENT

SYNTAX:

<FAMILY STATEMENT>::=   FAMILY = <STATION FAMILY>

<STATION FAMILY>::=   <STATION IDENTIFIER LIST>
                            I DUMMY

<STATION IDENTIFIER
LIST>::=   <STATION IDENTIFIER>
                            I <STATION IDENTIFIER>,<STATION
                               IDENTIFIER LIST>

SEMANTICS:

The <FAMILY STATEMENT> lists the names of stations which are to be
considered   members of the file being described. A station can be
associated with any number of files and it can also be referenced
any number of times in any one file. If DUMMY is used, it means
that   (1) an MCS must assign the stations to be associated with the
file at open time and, (2) the number of stations assigned must be
equal to or less than the number of stations declared in the file
statement of the opening user program.

FILE RESIDENT STATEMENT

SYNTAX:

```
<FILE RESIDENT
STATEMENT>::=              RESIDENT = CORE
                        I RESIDENT = DISK
```

SEMANTICS:

The file resident statement locks the user program in core if resident equals core. Otherwise, it is oaged out after each remote file read or write. The default is RESIDENT = CORE.

NDL LIBRARY

The Network Definition Language (NDL) Library consists of procedures
used in the operation of on-line terminals that interface with
B1700-Series computers. NDL Library is a source language library
that contains the REQUESTs and CONTROLs necessary to handle common
line disciplines for the terminal devices referenced herein. By
utilizing the library, users should find it unnecessary to write
their own REQUESTs and CONTROLs.

ACCESSING THE LIBRARY

The library of line desciplines described herein is a released
product in the NDL system and is named "NDL/LIBRARY".

This library consists of a number of groups of source statements
coded in NDL (Network Definition Language), each group constituting
a logical entity. These groups fall into two classes: those called
REQUESTs and those called CONTROLs.

Including the appropriate $ LIBRARY statement in a compile deck
places any of these entities, intact, into an NDL compile. For
instance, "$ LIBRARY POLLTD700", when placed correctly in a
compilation deck passed to the NDL compiler, will cause all source
statements in the entity named POLLTD700 from the NDL/LIBRARY file,
to be included into the source passed to the compiler.

A typical compilation deck will appear as follows:

```
    ?COMPILER HANDLER WITH NDL TO LIBRARY;
    ?DATA CARDS
    DECLARATION:
        NIF= "HANDLER"/"NIF".
    $LIBRARY POLLTD700
    $LIBRARY SELECTD700
    $LIBRARY POLSELCTL
        <TERMINAL SECTION STATEMENTS>
        <STATION SECTION STATEMENTS>
        <LINE SECTION STATEMENTS>
        <FILE SECTION STATEMENTS>
    FINI.
    ?END;
```

This deck with the <SECTION STATEMENTS> appropriately filled in will, with the aid of an applications program, communicate to a network composed of either TD700 or TD800 terminals, or some combination of both.

> Note: The $ LIBRARY cards take the place of NDL
>   executable statements in the deck. The
>   compiler will replace these $ LIBRARY cards
>   with actual source statements found in the
>   NDL/LIBRARY file under the entity named
>   POLLTD700 or SELECTD700 etc.

ASSOCIATIVE MATRICES

REQUESTS VS. CONTROL

C O N T R O L

	1	2	3	4	5	6
POLLTCTD	X	X				
POLTCTDDYN						X
READTTY			X			
TC5PTRCV			X			
TD7PTRCV			X			
SELECTCTD	X	X				X
FASTSELTCD	X	X				
WRITETTY			X			
TC5PTXMIT			X			
TD7PTXMIT			X			
TC5PT?IO				X		
DIAGTCTDIO	X	X				
RJE					X	

(left margin label: R E Q U E S T)

CONTROLS
----

1 = POLSELCTL
2 = AUTOPOLCTL
3 = PTPTCONCTL
4 = CONVERTL
5 = RJECTL
6 = AUTODYNCTL

X = RECOMMENED USAGE

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

REQUEST VS. TERMINAL

TERMINAL

	TC500 TC700 TC3500	TD700 TD800	TU500	TELETYPE	REMOTE APPLICATIONS	
POLLTCTD	X	X	X			
POLTCTDDYN	X	X	X			
READTTY				X		
TC5PTRCV	X					
TD7PTRCV		X				
SELECTCTD	X	X	X			
FASTSELTCD	X	X	X			
WRITETTY				X		
TC5PTXMIT	X					
TC7PTXMIT		X				
TC5PTRIO	X					
DIAGTCTDIO	X	X	X			
RJE					X	

(Row label down left side: R E Q U E S T)

X = RECOMMENDED USAGE

USING THE NDL COMPILER *

PREPARING PROGRAMS *

CONTROL CARDS

Control card deck to compile a Network Controller with the NDL
compiler is as follows:

```
    ?COMPILE MYHANDLER WITH NDL TO LIBRARY
    ?DATA CARDS
    ------
    ------
    (NDL SOURCE)
    ------
    ------
    ?END
```

Note: "?" indicates an invalid punch in column 1 for 80-column cards.

* Section will be completed at a later date.

COMPILER INPUT AND OUTPUT FILES

EXTERNAL NAME	INTERNAL NAME	MEDIUM	USE
CARDS	CARDS	CARDS	Primary source input
SOURCE	SOURCE	DISK	Secondary source input
NEWSOURCE	NEWSOURCE	DISK	New secondary source(output)
LINE	LINE	PRINTER	Listing
NDL/NIF	NIF	DISK	Output network information file
NDL/ADDRESS	ADDRESS	DISK	Address file output by MACRO for compiler (input)
(USER-CHOSEN)	OBJECT	DISK	Object code file of user's handler
NDL/MACRO	MACRO	DISK	Skeletal controller code file (input)
AUDIT.FPBS	AUDIT.FPBS	DISK	Workfile
NDL/LIBRARY	LIBRARY	DISK	Standard request and control source code library
CODE	CODE	DISK	Workfile

Note: NDL/MACRO and NDL/ADDRESS must be present on disk at BOJ of
       the compiler. SOURCE (or user's label-equated file) must be
       present   if $MERGE is used. NDL/LIBRARY must be present   if
       are part of the release package.

NDL SOURCE FORMAT

The format of the NDL source record is as follows: Card columns 1-72
inclusive are the free form source field. Column 72 is logically
adjacent to column 1 of the next card, so identifiers, numbers,
strings, etc. may be continued on the next card.

Card columns 73-80 are reserved for sequence numbers, if desired.

Comments: Occurrence of a percent sign (%) in columns 1-72 results
   in the compiler skipping to column 1 of the next card. Any
   desired comment text may be entered following the "%".
   Percent   signs are part of the NDL source, however, if they
   appear within an <EBCDIC STRING>. For example:

        FETCH CHARACTER. % ANY COMMENT. . .
        TRANSMIT "LEGAL %". % PERCENT WITHIN QUOTES IS LEGAL
        % THIS WHOLE CARD IS COMMENTS. . .

A dollar sign ($) in column 1 signals the compiler that the card
contains compiler CONTROL statements (See next section).

COMPILER CONTROL STATEMENTS

Compiler execution options are specified by means of primary source
records with a dollar sign ($) in column 1. The options cover the
areas   of list format error/warning handling, source maintenance,
changing stack sizes, and accessing the NDL source code library.

SYNTAX:

```
<COMPILER CONTROL
STATEMENT>::=                    <"$" IN COLUMN 1> <OPTION LIST>

<OPTION LIST>::=                 <OPTION> I NO <OPTION>
                               I <OPTION> <OPTION LIST>
                               I NO <OPTION> <OPTION LIST>

<OPTION>::=                      CHECK
                               I CODE
                               I CONTROL
                               I <DBUG OPTION>
                               I DOUBLE
                               I FORGETERRORS
                               I LIST
                               I LST
                               I MERGE
                               I NEW
                               I <SEQUENCE OPTION>
                               I SGL
                               I SINGLE
                               I <STACK SIZE OPTION>
                               I SUPPRESS
                               I <LIBRARY OPTION>
                               I <VOID OPTION>

<DBUG OPTION>::=                 DBUG <DBUG LIST>. (NON-RELEASE)

<DBUG LIST>::=                   <INTEGER>I<INTEGER>, <DBUG LIST>

<SEQUENCE OPTION>::=             SEQ <SEQ BASE> <SEQ INCREMENT>

<SEQ BASE>::=                    <INTEGER>I<EMPTY>

<SEQ INCREMENT>::=               +<INTEGER>I<EMPTY>
```

```
<STACK SIZE OPTION>::=        CSSIZE <INTEGER>
                            I DYNAMICSIZE <INTEGER>
                            I NSSIZE <INTEGER>
                            I PPSSIZE <INTEGER>
                            I ESSIZE <INTEGER>
                            I VSSIZE <INTEGER>

LIBRARY OPTION>::=            LIBRARY <STANDARD IDENTIFIER>

<STANDARD IDENTIFIER>::=      <IDENTIFIER>

<VOID OPTION>::=              VOID I VOID <HIGH SEQUENCE>

<HIGH SEQUENCE>::=            <8-CHARACTER INTEGER>
```

SEMANTICS:

CHECK	Print warnings on sequence errors. A sequence error occurs when sequence of last card is greater than or equal to the current sequence number.
CODE	List generated SDL-Machine S-OPS.
CONTROL	List "$" cards
CSSIZE	Set CONTROL stack size of NC to <INTEGER> entries.
DBUG	Non-release. Compiler debugging options.
DOUBLE	Doublespace listing.
DYNAMICSIZE	Set NC's dynamic memory to <INTEGER> bits.
ESSIZE	Set NC's evaluation stack size to <INTEGER> entries.
FORGETERRORS	Generate object NC despite syntax errors.
LIBRARY	The NDL cource code specified by <STANDARD IDENTIFIER> is retrieved from the file "NDL/LIBRARY" and inserted in the user's program following the "LIBRARY" card.
LIST	List source.
LST	List source.

MERGE   Merge input (patches in file "CARDS") with secondary
                 input ("SOURCE").

NEW   Output a new secondary source file ("NEWSOURCE").

NSSIZE   Set NC's name stack size to <INTEGER> entries

PPSSIZE   Set   NC's program pointer stack size to <INTEGER>
                 entries.
SEQ   Sequence   the source beginning at <SEQ   BASE> and
                 incrementing the sequence number by <SEQ INCREMENT>.
                 Default is "SEQ 00000000 +100" (specified by just "SEQ")

SGL   Single-space listing

SINGLE   Single-space listing

SUPPRESS   Syntax warnings will not be printed.

VSSIZE   Set NC's value stack to <INTEGER> bits.

VOID   If <HIGH SEQUENCE> is not specified, only the secondary
                 source record with the current sequence number is
                 skipped by the compiler. otherwise, skip all secondary
                 source records beginning at current sequence number and
                 continuing till a secondary source record is read with
                 sequence   higher than <HIGH SEQUENCE>. Note: VOID
                 presumes $MERGE.

Prefacing an option with NO turns it off. For instance NO LIST stops
listing of source records till $LIST is encountered. NO has no
effect on VOID and LIBRARY.

Defaults--all options are off except LIST, CHECK, and DOUBLE.

Stack sizes are listed at the end of the compiler run and should
suffice for normal operation, but may be increased if stack
overflow occurs or decreased to tune to minimum memory requirements
for special reasons.

RESERVED WORDS

The following words may not be used as identifiers because of their
use in the NDL syntax.

ACCESSERR	DISCONNECT	LENGTH	RESULTDESC
ACU	DISPLAY	LINE	RETRY
ADDERR	DO	LOGICALACK	RETURN
ADDRESS	DUMMY	LOGIN	SCREEN
AND	DUMP	LOSSOFCARRIER	SEC
ASC53	EBCDIC	LOSSOFDSR	SECURITY
ASC64	ELSE	LT	SEQUENCE
ASC68	END	MAX	SIGNAL
AUDIT	ENDOFBUFFER	MAXINPUT	SPECIAL
AUDITFILE	EQ	MAXINWORK	SPO
AUTOANSWER	ERROR	MAXOUTWORK	STATION
AUTOPOLL	EVEN	MAXSTATIONS	STORE
BREAK	EXCEPTION	MCSQUEUE	SUM
BUFFER	FALSE	MEMORY	TALLY
CALL	FAMILY	MIN	TASK
CANCEL	FETCH	MODE	TERMINAL
CARRIAGE	FILE	MYUSE	TERMINALTYPE
CASE	FINI	NE	TERMINATE
CHAR	FINISH	NIF	TEXT
CHARACTER	FOR	NORMAL	THEN
CLEAR	FORMAT	NO	TIME
CODE	FORMATERR	NOT	TIMEOUT
COMMENT	FREQUENCY	NULL	TO
CONSTANT	GE	ODD	TOG
CONTINUE	GT	OR	TRACE
CONTROL	HEADER	OUTPUT	TRAN
CONTROLLER	HORIZONTAL	OUTPUTATTACHED	TRANERR
CONVERT	IDLE	PAGE	TRANSMISSION
DEBUG	IF	PARITY	TRANSMIT
DECIMAL	INCREMENT	PASSIVE	TRUE
DECLARATION	INITIALIZE	PHONE	TURNAROUND
DECREMENT	INITIATE	POLL	TYPE
DEFAULT	INPUT	RECEIVE	UNDO
DIAGNOSTIC	INPUTATTACHED	REMOTE	USER
DIALIN	IODESC	REQUEST	VERTICAL
DIALOUT	LE		

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

RUNNING AN NDL CONTROLLER *

        NORMAL OPERATIONS *

                EXECUTING *

                PRIORITY *
                ----

*Section will be completed at a later date.

MEMORY, STACK SIZES

V.0 NDL DATACOMM SYSTEM MEMORY ESTIMATION

	Memory Required in Bytes
1. BASIC SYSTEM SOFTWARE	-------
System static memory (MCP stacks, MCP zero, GISMO zero, etc.)....-------	
Conditional static (IOAT, etc.)..................................-------	
System dynamic memory	
MCP zero-page resident code...................................... 4,772	
MCP code for NDL/QUEUE/REMOTE-FILE support...................... 5,594	
MCP code for QUEUE-FILES (required for active MCS) conditional.. 1,893	
Subtotal 1 * * *-------	

2. NDL CONTROLLER

Basic - required static + required code + SDL/INTERP segments.... 8,137

NDL tables and table-related
    Line: 272 *(number lines)............ = ------
    Station: 75 *(number stns +1)......... = ------
    Terminal: 9 *(number terminals)..... = -------
    File: 13 *(number files)............ = ------

    NDL Table SUM..-------

Requests + Controls (See compilation) + Links......................-------

Queues and Messages
    Queue Directories: 27 * (open Q-files
      + open remote files).............. = ------

    NDL Input Buffers: sum of "MAXINPUT"
      for all lines..................... = ------

    NDL Output Buffers: sum of average
      output text for all lines......... = ------

    Queued Messages number * (average
      size text + 68)................... = ------

    Queue Messages SUM...-------

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

NDL CONDITIONAL ADDITIONS

    Autopoll.. 598

      + Sum autopoll buffers (all lines)............................-------
    Read no timeout (Autopoll, TTY, RJE)................................. 454

    Teletypes... 130

    Audit - at least.. 1,106

      + if 2 or 3 auditfiles.. 362

      + if "CONVERT".. 152

      + if Fibs, buffers, etc.....................................-------
      + GISMO segments sequential I/O.............................-------
                    enhanced I/O...................................-------

    IOLOG... 840

    Transmission number incrementation................................ 164

      (not ARM character or 0/1 schemes)

    Conversational line disciplines - at least........................ 273

      + if "RECEIVE ADDRESS (STATION)" is used...................... 220

    MCS - at least.. 438

      + if LOGICALACK is used... 828

NDL Conditional Addition SUM..-------

    Subtotal 2: NDL basic + tables + queues + conditional * * * -------

3. MCS, if any- - - - - - - - - - - Subtotal 3:- - - - - - - - - - - --------
    ---

4. USER PROGRAMS

    Including additonal interpreters, MCP support code
    (e.g. MICR, Random I/O)- -Subtotal 4:- - - - - - - - - - - - - - --------

5. FUDGE FACTOR

    Leftover available, memory wasted in link system + unknowns
    (required to provide non-thrashing system on real 1700
                               Subtotal 5:- - - - - - - - - - 2,000

        Add Subtotals 1-5. The sum is the minimum amount of
        S memory to hold the NDL system working set.   TOTAL....=======

        STOPPING

    DEBUGGING

        DEBUGS

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

IOLOG

The IOLOG function provides for monitoring of all datacomm IO
activity through an external disk file labelled "DC"/"AUDIT.FILE".
A program "DC"/"AUDIT" will read this file either in parallel with
NDL or subsequently, formatting the information to the user's needs
(See documentation on DC/AUDIT).

To turn on IOLOG:

    Set DEBUG #28 in DBCARDS or type in <CONTROL MIX
    NUMBER> AXIOLOG on SPO. To turn off IOLOG, just
    repeat the process: <CONTROL MIX NUMBER> AXIOLOG. A
    message will appear on the SPO indicating whether
    IOLOG was turned on or off and the date and time.

If DC/AUDIT.FILE is missing, a new file is created. A header
provides information to "DC"/"AUDIT" indicating the first record
written by this job (irrelevant of turing IOLOG off and on
subsequently), the next record to be written, and the total records
written. The file wraps around on itself, so that more records may
be written than are accounted for in the AUDIT file. The size of
the file is determined dynamically. If the file is in use by
another handler, a SPO message will so indicate.

A comment record is always the first record to be written each time
IOLOG is set on.

DUMP *

TRACE *
----

*Section will be completed at a later date.

RUNERRORS

RUN ERROR LIST

NO.	Reason	Error Originator
100	Parity Error	Memory
101	Invalid station index returned by autopoll.	Controller
103	DCWRITE error on present communicate.	Controller
104	MACRO incompatible with the MCP.	Controller
105	Initiate receive but no buffer.	NDL Code
109	No poll list.	NDL Code
110	No autopoll buffer or length zero.	NDL Code
115	Invalid DCWRITE while queing input.	Controller
116	Invalid DCWRITE while queing error message or station not enabled for input.	Controller
117	Receive text or terminate echo with no input buffer or no text.	NDL Code
118	Beginning of text misaligned.	NDL Code
119	Transmit exceeded end of buffer.	NDL Code
120	Invalid DCWRITE while returning open message.	Controller
124,125,126,127,128,136	Invalid NIF file.	Compiler
129,134	Network is too large.	Controller
130	Main control loop exited.	Controller
131	Invalid address file.	Controller
132,133	Table size is too small.	Compiler
135	Station is invalid.	NDL Code
142	Transmit text but no buffer.	NDL Code
143	Station cannot be polled or string size does not agree with previous entry.	NDL Code
144	Poll pointer is invalid.	Controller
145	Length of poll list is greater than 120 characters.	NDL Code
146	Invalid open text size length	MCP
148	Message in station queue has length zero.	MCP or User Program
149	Station attached to another line.	Controller
150	Tried to receive input after shutdown started.	NDL Code

151	Invalid DCWRITE while queing EOF for USER. PROGRAM	Controller
152	Transmit string is not 4008 bits.	NDL Code
170	Message length is zero.	MCP
177	MCS message found but not expected.	MCP
180	Request definition is missing or wrong.	NDL Code
190	Invalid initiate.	NDL Code.
191	Control changed state to a different request.	NDL Code
192	Station cannot be changed.	NDL Code
193	Line has a station not in station list.	Controller

USING AN MCS *

----

*Section will be completed at a later date.

NDL DUMP

NDL/DUMP is a SDL program used for analyzing dump files created by
MCP   V.0 of the MACRO V.0. This analysis gives the NDL programmer
information regarding his system status variables, his present
network configuration and his lines, stations and terminals status.
The dump analysis also contains information regarding the status of
the MACRO, which is transparent to the NDL user. This information
can be used to debug the MACRO.

This document is for helping an NDL programmer understand the dump
analysis. It is not however, for explaining how the MARCO works to
an NDL user. Consequently the explanation of some variables is
omitted.

OPERATION

Given a dump file named DUMPFILE/<NUMBER>, to use NDL/DUMP enter

                          PM <NUMBER> NC

on the SPO. This command will start up NDL/DUMP. On EOJ, the dump
file will not be removed.

If the dump file is not made of the MACRO or is of a   MACRO of a
wrong level   or some essential constant(s) (e.g. PPBPTR) are
incorrect, NDL/DUMP will terminate prematurely indicating that the
dump file was not valid.

DATA FORMATS

There are a number of formats which are used to print the various
data items of the MACRO. In general, the obvious interpretation
should be correct.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

DECIMAL

All data items which are bit(24) or less and are not absolute addresses are printed as decimal numbers. Fixed fields are also printed as decimal numbers, but they are preceeded by a "+" or a "-".

HEX

All data items which are surrounded by "ə" signs are printed in hex.

CHARACTER

All character fields which contain valid characters are printed as they are; character fields which contains invalid character(s) are printed in hex.

ASCII

Datacomm buffers are printed in this format. In this format a printable character is printed as itself and an invalid character is printed vertically in hex. For example, the string:

ə01C1F102C1C2C303ə

is printed as:

```
0A10ABC0
1  2   3
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

SPECIAL FORMATS

All 20 bit data items which contain the time of day in tenths of seconds are printed in standard HH:MM:SS format.

The fields of LINE.TYPE and LINE.STATUS are interpreted.

Debug booleans are printed by name when set.

DUMP ANALYSIS OUTLINE

The following is an outline of the output of NDL/DUMP. Note that the last three items are printed only when applicable.

     1. Header page
     2. Global data items
     3. Terminal tables
     4. Station tables
     5. Line tables
     6. Queue headers
     7. Active CONTROL or REQUEST
     8. Local variables

HEADER PAGE

The MACRO version data is obtained from the NIF. If the NIF is not present, this date field will be printed as blanks. If the NIF indicates that the MACRO level is not compatible with NDL/DUMP, then an error message is printed out and NDL/DUMP will terminate. The other parameters in this page are obtained from the PPB.

GLOBAL VARIABLES

The global variables are printed out in order of their declaration in the MACRO.

TABLE INDEXING

The network description as specifed by an NDL program is stored in
the MACRO tables. These tables are declared as one data item in the
MAIN.CONTROL procedure of the MACRO. The information in these
tables is accessed by using indexing. The global variables L.I, S.I
and T.I are respectively the line table index, station table index
and terminal table index. The terminal tables are stored
sequentially starting at the index TERMINAL.TABLE.FIRST (a global
variable). There are MAX.TERMINALS terminal tables. Similary
STATION.TABLE.FIRST (LINE.TABLE.FIRST) and MAX.STATIONS (MAX.LINES)
indicate the number and starting index of the station (line)
tables.

NDL/DUMP prints out the terminal tables first followed by the
station tables followed by the line tables. All references to these
tables are done through the index. In other words, the index of the
table of the third line identifes that line. A table index is
always printed in decimal.

DEBUGS

If any debug booleans are set or debug table has a non-zero entry,
the appropriate name of the debug boolean will be printed out in
the global variable section. These booleans are printed out between
FILE.RESEDENT and HDR.DCW.

MACRO TABLES

The MACRO tables are printed in the order described in TABLE
INDEXING. The entries in these tables are either status indicator
or system constants set by NDL. In section NDL-MACRO VARIABLES
XREF, the cross-reference between NDL and the MACRO is presented.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

TABLE HEADER

Each table has a header of the following format:

    TERMINAL
    STATION   TABLE <NUMBER> [<IDENTIFER>] <INDEX>
    LINE

The line table has in addition below the header, the hardware address of this particular line. This address is printed in the following format:

        port   channel   adapter
             :   :
        number   number   number

The identifier in the header is the actual NDL source name for the line station or terminal to which the table is referring to. This identifier is obtained from the NIF. If the NIF is not present during execution of NDL/DUMP, blanks are put in place of the identifier. The index in the header is the indexed described in TABLE INDEXING.

STATION QUEUE HEADER

If a station has an active queue, its header will be printed out after the station TALLYS and TOGGLES. The format of the header is described in QUEUE HEADERS.

LINE WORK AREAS AND BUFFERS

For each line, the input and output work areas, poll list, and input and output buffers (if present) are printed out in ASCII format at the end of each line table.

THE STATION ACTIVE ON A LINE

To find the station attached to a given line at dump time, use the following formula:

index=STATION.LIST(FIRST.STATION.NO+LINE.CURRENT.STATION.NO-1)

This index identifies the desired station.

GENERAL INFORMATION

This section of the dump analysis is optional--it is only printed when the information needed is present.

QUEUE HEADERS

When active, the following queue headers are going to be printed out:

1. General input queue
2. General output queue
3. MCS queue
4. Control queue

The format of these headers is as follows:

 ∂Q.HEAD∂ Q.FAMILY.NAME/Q.OFFSPRING.NAME ∂Q.TAIL∂ Q.USER.COUNT

CONTROL AND REQUEST NUMBERS

If the NDL program has an active CONTROL or an active REQUEST, the
dump analysis will indicate the number of such a control and/or
request. This number is a index to the order in which the controls
or requests are declared. For example, the first request in the NDL
program is numbered 1.

LOCAL VARIABLES

The local variables of all procedures that are envoked after the
MAIN.CONTROL procedures are printed out at the end of the dump
analysis.

PROCEDURE HEADER FORMAT

Each procedure's data items is preceeded by the header of the
following format:

EXITED _EXIC LEVEL<number>.ENTERED LEXIC LEVEL<number>[<procedure name>]

Note that the procedure name is an option which depends on whether
the NMPRC conditional compile is set for the MACRO compile.

DATA ITEMS FORMAT

The format of the local data items is as follows:

    <Data Type> <bit length> [(decimal value)] <value>

The decimal value is printed if <bit length> is less than or equals
24. The value is printed in hex unless <data type> equals CHARACTER
and value has valid characters. In this case <value> is printed out
directly.

NDL-MACRO VARIABLES XREF

This cross reference is by no means a complete map between the MACRO's variables and NDL. It is ment to be helpful to the NDL programmer who is not familiar with the MACRO inner works.

SYSTEM STATUS VARIABLES

NDL System Status Variables Names	NDL/DUMP Section (See Outline 4.0)	MACRO Variables Names	Special Notes*
CHARACTER	LINE TABLE	LINE.CHAR.BUFFER	
FREQUENCY(INPUT)	STATION TABLE	INPUT.PRIORITY	
FREQUENCY(OUTPUT)	STATION TABLE	OUTPUT.PRIORITY	
IODESC	LINE.TABLE	IO.DESCRIPTOR	#1
INPUTATTACHED	LINE.TABLE	INPUT.BUFFER.VALID	
OUTPUTATTACHED	LINE.TABLE	OUTPUT.BUFFER.VALID	
LENGTH(INPUT)	LINE.TABLE	INPUT.BUFFER.LENGTH	
LENGTH(OUTPUT)	LINE.TABLE	DCW.SIZE	
LINE	LINE.TABLE	NUMBER	
LINE(CONTROL KEY)	LINE.TABLE	CONTROL.KEY	
LINE(QUEUED)	LINE.TABLE	QUEUED	#2
MAXSTATIONS	LINE.TABLE	MAX.STATIONS	
RESULTDESC	LINE.TABLE	IO.REQUEST	#3
RETRY	STATION TABLE	RETRY.COUNT	
STATION	LINE.TABLE	LINE.CUR.STATION.NO	
STATION(ENABLED)	STATION TABLE	ENABLED	
STATION(MYUSE)	STATION TABLE	MYUSE	
STATION(QUEUED)	GLOBAL ARRAY	STATION.Q	#4
STATION(READY)	STATION TABLE	READY	
STATION(TYPE)	STATION TABLE	CONTROL.LINK	#5
STATION(VAILD)	STATION TABLE	VALID	
TERMINAL_TYPE	TERMINAL TABLE	TYPE	
TIME	from MCP	-------	
TRAN(RECEIVE)	STATION TABLE	INPUT.TRAN.NO	
TRAN(TRANSMIT)	STATION TABLE	OUTPUT.TRAN.NO	
ERROR FLAGS	GLOBAL VARIABLE	DESCRIPTION.ERRORS	#6
TOG[<INDEX>]	STATION TABLE	STATION TOGGLES	
LINE(TOG[<INDEX>])	LINE.TABLE	LINE TOGGLES	
TALLY[<INDEX>]	STATION TABLE	STATION TALLYS	
LINE(TA_LY[<INDEX>])	LINE.TABLE	LINE TALLYS	
TIME(TA_LY)	STATION TABLE	TIME.TALLY	

BURROUGHS CORPORATION   COMPANY CONFIDENTIAL
COMPUTER SYSTEMS GROUP   B1700 NETWORK DEFINITION LANGUAGE
SANTA BARBARA PLANT   P.S. 2212 5223

*SPECIAL NOTES:
----

1.   Part of the first two descriptions is stored in
     this field. Each of these parts consists of the
     following consecutive 24 bit fields:

        IO.ACTUAL.END
        IO.RESULT
        IO.LINK
        IO.OP
        IO.BEGIN
        IO.END

2.   If this field is greater than 0, the QUEUED has
     a value of true, otherwise QUEUED is false.

3.   This   field is a concatination of the IO.RESULT
     fields of the first two IO.DESCRIPTOR fields.

4.   If this field is greater than 0, STATION(QUEUED)
     is true, otherwise STATION(QUEUED) is false.

5.   The   following are the possible values   and
     corresponding meanings of this field.

        a0a means NORMAL
        aFa means CONTROLLER
        anything else means SLAVE

6.   The   specific   error   flags   are   bits   of
     RESULT.DESCRIPTOR:

        Bit 1 : PARITY   Bit 7 : LOSSOFDSR
        Bit 2 : ACCESSERR.   Bit 8 : LOSSOFCARRIER
        Bit 3 : Not used   Bit 9 : ADDERR
        Bit 4 : TIMEOUT   Bit 10: TRANERR
        Bit 5 : READ   Bit 11: FORMATERR
        Bit 6 : ENDOFBUFFER   Bit 12: Not used
                                Bit 13: EXCEPTION

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 NETWORK DEFINITION LANGUAGE
P.S. 2212 5223

TERMINAL SECTION

NDL Terminal Statement	NDL/DUMP Terminal Table Variable
IDENTIFIER	Table Identifier
HEADER SIZE	HEADER.SIZE
MAXINPUT ADDRESS	INPUT.MAX.MSG
DIAGNOSTIC REQUEST	DIAG.OUTPUT.REQUEST(DIAG.INPUT.REQUEST)
REQUEST	OUTPUT.REQUEST(INPUT.REQUEST)
TYPE	TYPE
TRANSMISSION NUMBER	NO.SIZE
ADDRESS	OUTPUT.ADRS.SIZE(INPUT.ADRS.SIZE)

STATION SECTION

NDL Station Statement	NDL/DUMP Station Table Variable
IDENTIFIER	Table identifier
CONTROLLER	CONTROLLER
FREQUENCY	INPUT.PRITORITY(OUTPUT.PRIORITY)
LOGICALACK	LOGICAL.ACK
MCS QUEUED	MCS.EXPECTED (this is a global variable)
MYUSE	MYUSE
ADDRESS	INPUT.ADDRESS(OUTPUT.ADDRESS)
TERMINAL	TERMINAL.LINK
RETRY	RETRY.COUNT

LINE SECTION

NDL Line Statement	NDL/DUMP Line Table Variable
IDENTIFIER	Table Identifier
AUTOPOLL SIZE	POLL.LIST.LENGTH
ADDRESS	Port:Channel:Adapter
CONTROL	CONTROL.LINK
STATION	STATION.LINK
TYPE	TYPE
MAXINWORK	INPUT.WA.LENGTH
MAXOUTWORK	OUTPUT.WA.LENGTH
MAXSTATIONS	MAX.STATIONS

FILE SECTION

The arrays in the global section contain information regarding which
station belong to which family. The following is a method to find
this information.

1.   Look in FILE.NAME array for the family name.

2.   Get the index into this array which points to the
      given family name. This is the family number.

3.   The station number is the index into the station
      identifier list in the NDL family statement. Note
      that this index starts with 0 value.

4.   FILE.STATION.LIST.PTR (family number + station
      number) is your station pointer.

5.   FILE.STATION.LIST   (station pointer)   is   the
      station list pointer.

6.   STATION.LIST (station list pointer) is the index
      into the station table.

ALPHABETIC INDEX: