INSTRUCTIONS

Because all our documentation is being reproduced on reduction, it is essential that this form is either TYPED OR PRINTED LEGIBILY IN **BLACK** INK.

Fill out only those fields marked by bullets (o).

DOCUMENT TYPE: One of the following:

AR	- Archictecture Requirments (CYBER 80)
ESL	- Baseline Documentation Change(s)
DAP	- Design Action Paper
DDD	- Design Direction Document
DR	- Design Requirements
EPP	- Evalutaion Project Plan
ERS	- External Reference Specifications
FTP	- Feature Test Plan
GDS	- General Design Spec (CYBER 80)
GID	- General Internal Specifications
IHB	- Installation Handbook
IPP	- I le en a ion Project Plan
PDA	- Product Development Assurance Plan
PJP	- Project Plan
STD	- Standards Document
STP	- System Test Plan
TIR	- Transmittal, Integration & Release Criteria
OTH	- Other, specify type

TITLE: Be concise. Be sure to include the name of the product involved.

ABSTRACT: A brief description of the contents. If the item is extremely short, it may be written in designated space with nothing else attached.

PUBLICATION #: If a document with a pub. # is being changed, write the pub. # here.

SIP & PRM Numbers: If any are associated with this document, list them here.

APPROVALS: (Minimum DCS Requirements)
Project leader for all.
Unit Manager for all.
Section Manager: Project Plans only.
DCS will assign referee.

PROJECT: The project name that the author belongs to, e.g., CLUE.

DCS Mailing List Pasted Here -
(On top of instructions)

*For ARHOPS Use Only!

CONTROL DATA CORPORATION

DOCUMENT CONTROL FORM

o DOCUMENT TYPE: OTH - DESIGN NOTE

o LOG ID: ARH6813

o TITLE: COMMAND PARAMETER CONVENT

o ABSTRACT: DEFINES CONVENTIONS FOR PARAMETER VERIFICATION AND USE IN CDCNET COMMANDS

o PRODUCT AFFECTED

o AUTHOR: J. D. Reed

o MAIL STATION: ARH207

o EXTENSION: 6062

o PROJECT: CDCNET

o SECTION: DESIGN

o RSM or PSR #

o SIP Number(s)

o PUB #

o PRM #

o Redesign* o Reimplementati

Name	Approval Initials	Date
o Internal Reviewer*		
o Project Leader		
o Unit Manager		
o Section Manager BRIAN PALMER		3/12/8
Design Team Referee BRIAN PALMER		3/12/8

DESIGN TEAM

DEFAULT CYCLE 0 WORKING DAYS

o Special Distribution:

Referee's Distribution Codes: AC-D

NUMBER	DATE	TYPE	SUBMITTED BY	DEFAULT APPROVAL DATE	CURRENT STATUS			
					REVIEW	HOLD	APPROVED	WITHDRAW
1	3-14-85	orig.	J. P. Reed	none			X	

DISTRIBUTION LIST

ARH6813

.	I	SVL107	P.L.MCNAIR
.	I	SVL107	A.E.ANDERSON
.	I	SVL107	J.M.STOWE
.	I	SVL107	K.M.MIYAHARA
.	I	SVL107	V.M.SZIGETI
.	I	SVL107	R.K.COOPER
.	I	SVL107	P.A.SHIREMAN
.	I	SVL161	E.P.LAMOUREUX
.	I	SVL170	J.C.LEE
.	I	SVL185	M.A.DUNN
.	I	ARH207	B.T.ZEMLIN
.	I	ARH207	J.P.YOUNG
.	I	ARH207	R.R.RUNDQUIST
.	I	ARH207	R.E.TATE
.	I	ARH207	R.D.SWAN
.	I	ARH207	B.S.SEKHON
.	I	ARH207	N.L.REDDY
	I	ARH207	B.G.PALMER
	I	ARH207	J.A.DAYKIN
.	I	ARH207	E.E.NELSON
.	I	ARH207	L.L.LUCAS
.	I	ARH207	G.D.JORDAHL
.	I	ARH207	J.A.WACHTER
.	I	ARH207	K.D.LAMAR
.	I	ARH207	J.A.VERPLOEGH
.	I	ARH207	A.A.WEBER
.	I	ARH207	H.J.HOMMER
.	I	ARH207	J.D.REED
.	I	ARH2C7	T.C.MCGEE
.	I	ARH207	R.V.TOBIN
.	I	ARH207	M.E.MADDEN
.	I	ARH207	K.G.VIGGERS
.	I	ARH207	ACN LIBRARY
.	I	ARH2C7	H.G.COVERSTON
.	I	ARH207	S.M.KEEFER
.	I	ARH207	D.K.TAYLOR
.	I	ARH213	C.J.DUFFY
.	I	ARH213	J.D.KNODEL
.	I	ARH219	D.W.GROUT
.	I	ARH219	P.M.O'MALLEY
.	I	ARH219	J.T.LINDMEYER
.	I	ARH219	G.J.HOLMBERG
.	I	ARH244	R.A.JAPS
.	I	ARH245	T.E.BERETT
.	I	ARH263	J.L.NADING
.	I	ARH293	D.D.THOMPSON
.	I	AHS279	J.R.HILDEBRAND
.	I	HQC02X	D.A.SEIDENSTRICKER
.	I	PLY011	R.L.CLARK
.	I	STAOPS	M.C.STEELE

DISTRIBUTED COMMUNICATIONS NETWORK
(CDCNET)

COMMAND PARAMETER CONVENTIONS DESIGN NOTE

J. D. Reed
03/12/85

DISCLAIMER: This document is an internal working paper only. It is unapproved and subject to change, and does not necessarily represent any official intent on the part of Control Data Corporation.

REVISION RECORD

Revision	Description	Author	Date
A	Release for TDRB review	JDR	02/14/85
B	Release to DCS	JDR	03/12/85

Table of Contents

1.0 INTRODUCTION

For CDCNET, we have designed a fair number of CDCNET network definition, operator, and terminal commands. To make these commands easy to use and remember, we have defined conventions for use of commands and the way commands are named. General command conventions have been documented in the CDCNET Command Conventions document by B. Sekhon. The Command Conventions document also includes conventions and suggestions for command parameters; specifically, conventions for parameter names and the various types parameter values may take. The sections of the Command Conventions document pertaining to parameters have been attached to this note as Appendix A. These conventions are also briefly repeated in this note.

CDCNET development, however, has identified the need for additional parameter conventions. In particular, design direction is needed for the names and types of parameters common to several commands, for the processing of lists of values, for the processing of operator/terminal user entry errors detected by command processors and for the recovery and response to errors that occur during command execution. This document provides the design direction.

2.0 REQUIREMENTS AND CONSTRAINTS

- The conventions in this note are compatible with the conventions in the Command Conventions document.

- Conventions are simple to follow and may be applied consistently across the CDCNET commands.

- The conventions are applicable to CDCNET commands for R1 and later CDCNET releases.

- The parameter conventions are compatible with NOS/VE conventions. NOS/VE has defined parameters conventions in the NOS/VE System Interface Standard.

- The schedule impact on R1 CDCNET is minimized.

Where these requirements conflict, the requirements will be followed using the following priority (highest to lowest):

- Consistency across the CDCNET command set
- Minimum impact against R1 CDCNET
- Consistency with NOS/VE.

For example, where we have designed command parameters for R1 CDCNET in conflict with NOS/VE conventions, we will vary from the NOS/VE conventions (For R1, we reserve the right to vary from NOS/VE conventions on a case by case basis).

3.0 SOLUTION

The solution presented defines conventions for parameters, defines the controlling commands for parameters and defines connventions for user defined or CDCNET defined names for network components. This solution also defines the command actions and responses for parameter errors and redundant commands. A statement of the philsophy behind these conventions is provided in Appendix B.

3.1 CONVENTIONS

3.1.1 PARAMETER NAMES AND VALUES

The following six conventions are repeated from the Command Conventions document. The Command Conventions document also contains examples of using the first six conventions.

- The value of a command parameter should not be used to indicate the function to be performed by the command.
- Within a single command every effort should be made to minimize the illegal combinations of values of two or more parameters.
- The size of a parameter name for any CDCNET command will not exceed 31 characters.
- A parameter name will be abbreviated by using the first character of each word in the parameter name.
- For a given command, the names of all parameters will be selected so that no two names will result in the same abbreviation.
- If value of a parameter in one command is to be used as the name of a parameter in another command, then the same name will be used for the parameter value as well as the parameter name.

The following conventions are defined by this document:

- All required parameters will be placed before the optional parameters in a command.

3.0 SOLUTION
3.1.1 PARAMETER NAMES AND VALUES

- If the name of the command object is a required parameter, the object name will be the first required parameter.

- If the object name is an optional parameter, the object name should be the first optional parameter (i.e., the first parameter after all required parameters). For example, the define_hdlc_net command defines a CDCNET network solution (the object) and optionally provides a name for the network. The only required parameter for the network definition is the name of the underlying trunk definition. The name for the network should be the first optional parameter after the required trunk parameter.

- All logical entities (e.g., networks, trunks, gateways, processes, modules, communications lines, terminal devices, etc.) will be addressed by logical name.

- All physical devices (memory banks, boards - MPB, CIM, SMM, LIM, ESCI - and ports) will be addressed by physical name. The define commands, however, will vary from this convention.

- Boolean parameters cannot be used for two or more options that are mutually exclusive (i.e., parm_a = yes means parm_b must be no). This is a case of illegal combinations of values that must be avoided. Options that are mutually exclusive must be key values for an single parameter.

- All CDCNET defined names, including titles and physical names, will be prefixed with a dollar sign ($). User defined names, including titles and logical names will not be permitted to start with a dollar sign. If a user defined name defaults to a CDCNET defined name, the default name will begin with a dollar sign.

3.1.2 CONTROLLING COMMAND

The define command for an object (or if a define command does not exist, the start command) is the controlling command for all parameters that refer to an object. Other commands for an object that share parameters with the define command will use the parameter definition from the define command.

3.1.3 SOURCE OF LOGICAL NAMES

The define command for a configurable object (or if a define command does not exist, the start command) assigns the logical name for the object.

3.1.4 PHYSICAL NAMES

The physical devices within a DI will be addressed in commands by a physical name (with the exception of the R1 define commands, which will address physical devices by their slot, LIM or port address). Physical names are ASCII strings and consist of a prefix of a dollar sign ($) followed by the concatenation of the type of the hardware device and its physical address. The physical name for a device, then, is of the form '$deviceN' where 'device' may be an SMM, MPB, CIM, ESCI or LIM card, SMM bank or LIM port and N (0 - 7) is a card slot number, SMM bank number or LIM port number. For example, the physical name for a CIM card in card slot 3 is '$CIM3'. Where a component is a subassembly of a device, the component's physical name is the concatenation of the device name and the subassembly name joined by an underscore. For example, the physical name for the second port on a LIM card in slot 5 is '$LIM5_PORT2'.

3.1.5 SINGULAR AND PLURAL FORMS OF PARAMETER NAMES

If the values for a parameter are a list, then both the singular or plural form of the parameter name should be supported. As with command names, the singular or plural form of a parameter name may be entered with one or more than one parameter value. (For R1 CDCNET, only the singular form of parameter names is required. Support of plural forms is optional.)

3.1.6 COMMON PARAMETERS

A set of parameters will be common across all CDCNET commands where the parameter is applicable. These parameters include:

message|m = an ASCII string. For example, a message to a terminal user. Message is always type <string>.

options|o = a list of options that are not mutually exclusive. Options is always type list of <key>.

display_options|do = a list of parameter names or keywords that defines the items to display. For display configuration commands, the display_options keywords are the parameter names from the associated define command (default is all), For status commands, the display_options are summary or detail (default is summary). Display_options is always of type list of <key>.

device_name|device_names|dn = the physical name of DI hardware. For example, a display_hardware_status command will address hardware devices by device_name. Device_name is always of type <name> or type list of <name>. The only exception to device_name = physical name is the device_name for a terminal device. For terminal devices, device_name is a logical name.

slot|s = a card slot number. Slot is always of type 0..7. Slot will only be used on define commands.

port|p = a LIM port number. Port is always of type 0..3. Port will only be used on define commands.

lim|l = a LIM card slot number. Lim is always of type 0..7. Lim will only be used on define commands.

line_name|line_names|ln = the logical name of a communications line (e.g., an asynchronous line) from a DI to terminating equipment (e.g., an interactive terminal, a batch workstation, etc.). Line_name is always of type <name> or type list of <name>.

trunk_name|trunk_names|tn = the logical name of the communications media between DI's, between a DI and a CYBER 180 or between a DI and a foreign communications network. Trunk_name is always of type <name> or type list of <name>.

network_name|network_names|nn = the logical name of a CDNA network solution. Network_name is always of type <name> or type list of <name>.

gateway_name|gateway_names|gn = the logical name of a CDNA gateway service. Gateway_name is always of type <name> or type list of <name>.

tip_name|tip_names|tn = the logical name of a CDNA Terminal Interface Program service. Tip_name is always of type <name> or type list of <name>.

log_message|log_messages|lm = a log message number or list of log numbers. Log_message is always of type 1..32767 or type list of 1..32767.

alarm_message|alarm_messages|am = an alarm message number or list of alarm numbers. Alarm_message is always of type 1..32767 or type list or 1..32767.

recorder_log_group|recorder_log_groups|rlg = a recorder_log_group name or list of recorder_log_group names. Recorder_log_group is always of type <name> or type list of <name>.

source_log_group|source_log_groups|slg = a source_log_group name or list of source_log_group names. Source_log_group is always of type <name> or type list of <name>.

group|g = a selection from a list of options that are mutually exclusive. Group is always type <key>.

type|types|t = a list of a class of items that are not mutually exclusive. For example, a list of file types. Type is always type list of <key>.

3.1.7 VERIFICATION OF PARAMETERS

The following are conventions for the verification and use of command parameter values. The conventions for verification apply to parameter verification by command processors (the command parser is assumed to follow the NOS/VE syntax rules for verifying command parameters). Please note the differences between the rules for display commands, the rules for fields interpreted by the Operator Support Application and the rules for all other commands.

- For NON-DISPLAY commands, all parameters will be fully parsed and verified before a command is acted on. Any parameter values that cannot be verified by the command parser will be verified by the command processor. Syntactic errors not detected by the command parser (e.g., user defined names starting with $) and semantic errors (e.g., unknown logical names) will terminate a command. For most commands, parameter errors will be errors of severity level '--ERROR--' and will result in an error response to the operator. For some commands, see below, a --WARNING-- response will be returned.

- For NON-DISPLAY commands, all values in a list of values for a command parameter will be verified before the command is acted on. Any value in the list that is not syntactically or semantically correct will terminate the command before any action takes place for legal values (i.e., commands will not partly complete). For most commands, an error in a list value will be an be a error of severity level '--ERROR--', will result in an error response to the operator. Only the first value in error will be reported. For some commands, see below, a --WARNING-- response will be returned.

- For DISPLAY commands, all parameters should also be fully parsed and verified. Any parameter values that cannot be verified by the command parser will verified by the command processor. Syntactically or semantically incorrect values detected by the command processor should NOT terminate the entire command but just replace the display for the parameter in error with an error message (i.e., display commands may partly complete). Parameter errors detected by the command processor will not result in an error response (i.e., all responses from display command processors will be of severity level --INFORMATIVE--). Instead, the response for a value in error will simply be included in the informative response. The error message should appear within the informative response at the position that a normal response would have appeared if the value had been correct. An error message should appear for each parameter in error.

- For DISPLAY commands, all values in a list of values for a command parameter will be verified. As for other incorrect values, a syntactically or semantically incorrect value in a list will NOT terminate the entire command but just replace the display for the list value in error with an error message. As above, the error message should appear within the --INFORMATIVE-- response for the display command. An error message should appear for each value in error.

- For OSA command destinations, OSA will generate a error response for each invalid destination for a command or for destinations that are unavailable (down, currently being reloaded etc.). Note that if OSA detects an invalid destination in a list of systems or communities, the command is still sent to the valid destinations entered. Commands sent to multiple destinations, including some unavailable destinations, are also sent to those available.

- For NON-DISPLAY commands, if a parameter allows a list of options, the last instance of a option entered redundantly will be the one used. For example, the define_hdlc_trunk command contains an option parameter that specifies various

HDLC protocol options. The values of the parameter are a keywords for the options. Keywords for both activating and deactivating part of the HDLC protocol are provided. If the user enters multiple instances of the same keyword, only the last entry will be used and no error response will be generated. If the user enters both the activating and an deactivating keywords for the same part of the protocol, again only the last instance is used. However, a --WARNING-- response should be given for the command.

- For DISPLAY commands, if a parameter allows a list of values, each value will be treated separately. Redundant entries will cause redundant displays. If the entries for different parameters would result in redundant displays, redundant displays will be given. The display commands should not prejudge the intent of the user.

3.1.8 'REDUNDANT' COMMANDS

For cancel commands, the result of the processing the command for a known name is to make to the name unknown. Thus, the cancelling of an unknown name results in no changes to the system. Likewise, the result of a stop command on an already stopped device makes no changes to the system. If the result of a command is unambiguous even if entry of the command is redundant or unnecessary, then a --WARNING-- response may be returned. The result of entering some commands, of course, is ambiguous if these commands are entered redundantly. For example, if a start command is addressed to an already started device, the command could be ignored or the started device could be reset and restarted (e.g., the HDLC protocol restarted). These redundant commands must be rejected with an --ERROR-- response to the operator.

The following chart defines the proper response for various commands by verb for redundant entry or unknown object names:

Command Verb	Redundant, object	Object Name
	in resultant state	Unknown
define	--ERROR--	--INFORMATIVE--
change	--INFORMATIVE--	--ERROR--
cancel	--WARNING--	--WARNING--
start	--ERROR--	--ERROR--
stop	--WARNING--	--ERROR--
load	--INFORMATIVE--	--ERROR--
unload	--INFORMATIVE--	--WARNING--

3.1.9 ACTION AND RESPONSE ON INTERNAL COMMAND FAILURES

Commands can fail during their execution. For example, define commands usually require allocatable memory to build tables for the network components defined. The memory allocation, however, could fail. Commands whose objects are not lists should recover from failure by returning any resources allocated by the command and, in general, by returning the state of any data structures, interlocks etc. used by the command to their state before the command was entered. A failing command will provide a --FATAL-- response.

For commands whose objects are lists, the command should process each list entry sequentially (after the list entries have been verified). Should the processing for a list element fail, the command should recover by returning resources for the failing element and by returning the state of data structures, interlocks etc. used for that element to their previous state. Successful processing for previous list elements, however, will not be undone. Instead, the --FATAL-- response will indicate the point at which the command failed. Successful completion for previous list elements will be understood, then, by convention. The list elements following a failing element will not be processed as the processing for these elements is also likely to fail. Again, failure of all list elements following the indicated element will be understood by convention.

3.1.10 OPTIONAL PARAMETERS THAT MAY BE REQUIRED

For CDCNET commands, some 'optional' parameters will be required depending on the hardware configuration of the DI where the command executes or the residency of the command. For example, the slot number of an Ethernet trunk is optional if only one ESCI card is installed in a DI, but required if more than one ESCI card is installed. For such optional parameters, we will provide a --WARNING-- response and accept the command if the parameter is not specified, the default is used and the default is allowed at the time the command is entered. We will provide an --ERROR-- response and reject the command if the parameter is not specified, but the default cannot be used at the time the command is entered.

3.1.11 SEVERITY LEVELS VERSES RESPONSE TYPES

The --INFORMATIVE-- and --WARNING-- severity levels are normal responses (i.e., the response flag is set to normal). --ERROR--,

--FATAL-- and --CATASTROPHIC-- severity levels are abnormal responses.

3.2 VARIANCES FROM NOS/VE CONVENTIONS

The following conventions derived from NOS/VE will not be followed by CDCNET commands:

- Parameter names will not be constructed by adding the type of parameter value to the item referred to. For example, a parameter to specify a communications line is 'line' not 'line_name'. For CDCNET commands, we will choose to vary from this convention.

- A parameter name will not repeat the name of the command object. For example, for all commands the parameter to give the name of the command object is 'name'. For CDCNET commands, we will choose to vary from this convention.

4.0 REQUIRED ACTIONS

The R1 CDCNET commands must be brought into compliance with these conventions with the exception that the following conventions will be optional for R1.

- Prefix of a dollar sign for all CDCNET defined names. This convention will be required by R2 CDCNET.
- Support of singular and plural forms for parameter names. This convention will be required by R3 CDCNET.

Appendix A
CDCNET Command Conventions Parameter Guidelines

A1.0 GENERAL GUIDELINES

1) The value of a command parameter should not be used to indicate the function to be performed by the command.

2) Within a single command every effort should be made to minimize the illegal combinations of values of two or more parameters. This will happen if command parmeters are not orthogonal. In some cases this type of situation may be avoided by defining a new parameter in place of the parameters which are not orthogonal and therefore can have values which conflict with each other.

3) The size of a parameter name for any CDCNET command will not exceed 31 characters.

4) A parameter name will be abbreviated by using the first character of each word in the parameter name. For example the abbreviation for the parameter name "LINE_SPEED" will be "LS".

5) For a given command, the names of all parameters will be selected so that no two names will result in the same abbreviation.

6) If value of a parameter in one command is to be used as the name of a parameter in another command, then the same name will be used for the parameter value as well as the parameter name. For example consider the command "DEFINE_COMMAND_ENVIRONMENT" used to define the command environment for an operator. This command can have "ce" or "command_echo" as a name of one of the parameters. The command "DISPLAY_COMMAND_ENVIRONMENT" used to display the command environment has a parameter called "do" or "display_options". One valid value for this parameter can be used to display the value of the "command_echo" parameter as set in the "DEFINE_COMMAND_ENVIRONMENT" command. The keyword name for this value should be "ce" or "command_echo", i.e. same as the name of the corresponding parameter name.

A2.0 PARAMETER VALUE CONVENTIONS

The values which may be specified for the command parameters will always be of a well known type. The following is a list of valid types.

- Integer
- Name
- String
- Boolean
- keyword
- List type
- Record type
- Union type (Any)
- Application type

These value types are described next using the NOS/VE SCL metalanguage rules. These rules are described in the Command Conventions document. Most of the material in this section has been extracted from the NOS/VE command writer's guide.

The description of various types is preceded by a definition of various types of constants which are used in the definition of above types.

A2.1 DEFINITION OF CONSTANTS

A2.1.1 INTEGERS

An integer constant is a sequence of digits, the first of which must be a decimal digit, optionally prefixed by a sign and optionally suffixed by a radix enclosed in parentheses. The constant must be delimited at both ends. No spaces are permitted between the digits and the radix specification.

```
<integer> ::= [<sign>] <unsigned integer>

<unsigned integer> ::= <digit> [<hex digit>]...  [<(> <radix> <)>]

<sign> ::= <+|-> [<sp>]

<+|-> ::= + | -

<hex digit> ::= <digit> | A | B | C | D | E | F

                        | a | b | c | d | e | f

<radix> ::= <unsigned decimal>

<unsigned decimal> ::= <digit>...
```

An integer constant can be expressed in any radix between two and sixteen. When the radix specification is omitted, ten (decimal) is assumed. Besides ten (decimal), the most common radix values are sixteen (hexadecimal), eight (octal) and two (binary). No distinction is made between lower and upper case hex digits by the CDCNET command parser.

When a radix greater than ten is specified, a leading zero digit may be required to ensure that the constant begins with a decimal digit. This is to avoid ambiguity between, for example, the sixteenth element of the array whose name is A1 given by A1(16), and the hexadecimal representation of the decimal value 161 denoted by 0A1(16).

Spaces may not appear between the sign and unsigned integer in certain contexts, most notably in expressions for parameter values. See the syntax definition for numeric expressions for details.

Example: 63, 77(8), -63(10), 3f(16), 111111(2), 0abc(16)

A2.1.2 NAME CONSTANT

A name is a sequence of from 1 to 31 alphanumeric characters
the first of which must not be a digit and which must be
delimited at both ends.
In a name the case of letters is irrelevant and
all lower case letters appearing in a name are "folded" to
their upper case counterparts.

```
<name> ::= <alphabetic char> [<alphanumeric char>]...

<alphanumeric char> ::= <alphabetic char> | <digit>

<alphabetic char> ::= <letter>
                    | <special alphabetic char>
                    | <international alphabetic char>

<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M
           | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
           | a | b | c | d | e | f | g | h | i | j | k | l | m
           | n | o | p | q | r | s | t | u | v | w | x | y | z

<special alphabetic_char> ::= # | $ | @ | _

<international alphabetic char> ::= _[_ | \ | _]_ | ↑
                                  | ¢ T { | _|_ T } | ¬

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Example: x, $dAtE, this_is_a_semi_long_name

A2.1.3 STRING CONSTANT

A string constant is any (possibly empty) sequence of ASCII characters enclosed in apostrophes (single quote marks). The apostrophes are not part of the string but serve as delimiters. To include an apostrophe in the string, two consecutive apostrophes are used. The string must be delimited at both ends.

```
<string> ::= ' [<string char>]... '

<string char> ::= <any ascii character except '> | ''
```

```
Example: 'This is a string.'
         'Here is an enclosed apostrophe '' character.'
         ''        "a null (empty) string"
         ''''      "a string containing only an apostrophe"
```

A2.1.4 BOOLEAN CONSTANT

A boolean constant is represented by one of the names shown below. These names are interpreted as boolean values only in those contexts which require boolean values.

```
<boolean> ::= <true> | <false>

<true> ::= TRUE | YES | ON

<false> ::= FALSE | NO | OFF
```

A2.2 DEFINITION OF TYPES

A2.2.1 INTEGER TYPE

```
<integer type> ::= INTEGER [<sp> <subrange>]

<subrange> ::= <min integer> <..> <max integer>

<min integer> ::= <integer>

<max integer> ::= <integer>
```

The subrange specification can be used to restrict the range of values applicable for the type being defined. The value of <min integer> must be less than or equal to that of <max integer>.

A2.2.2 NAME TYPE

```
<name type> ::= NAME [<sp> <name size>]

<name size> ::= [<min name size> <..>] <max name size>

<min name size> ::= <integer>

<max name size> ::= <integer>
```

The name size specification can be used to restrict the length of names applicable for the type being defined. The value of <min name size> must be less than or equal to that of <max name size> and both must be in the range 1..31. If <min name size> is omitted, 1 is assumed. If <max name size> is omitted, 31 is assumed.

A2.2.3 STRING TYPE

```
<string type> ::= STRING [<sp> <string size>]

<string size> ::= <min string size> <..> <max string size>
                | <fixed string size>

<min string size> ::= <integer>

<max string size> ::= <integer>
```

```
<fixed string size> ::= <integer>
```

The string size specification can be used to restrict the length of strings applicable for the type being defined. The value of <min string size> must be less than or equal to that of <max string size> and both must be in the range 0..256. If the string must be of one particular length, the <fixed string size> specification can be used to give values for both <min string size> and <max string size>. If the <string size> specification is omitted, 0 is assumed for <min string size> and 256 for <max string size>.

A2.2.4 BOOLEAN TYPE

```
   <boolean type> ::= BOOLEAN
```

A2.2.5 KEYWORD TYPE

The keyword type is generally used for designating a set of options.

```
<keyword type> ::= KEY <sp|nl>
                     <keyword groups>
                   KEYEND

<keyword groups> ::= <keyword group> <,|sp|nl>
                     [<keyword group> <,|sp|nl>]...

<keyword group> ::= <(> <keyword> [<,|sp> <keyword>]...  <)>
                  | <keyword>

<keyword> ::= <name>
```

A2.2.6 LIST TYPE

Lists should be used for structuring data whose elements are, in general, to be accessed sequentially, or in cases where the actual number of elements in the list will vary from one usage to the another.

```
<list type> ::= LIST [<list type qualifier>]

<list type qualifier> ::=
      [<sp> <list size qualifier>] <sp> OF <sp> <type>

<list size qualifier> ::= <min list size> <..> <max list size>

<min list size> ::= <integer>

<max list size> ::= <integer>
```

The list must have at least <min list size> elements. If <min list size> is not specified, zero is assumed. The list may not have more than <max list size> elements. If <max list size> is not specified, there is no limit to how many elements the list may have.

The list type qualifier may be omitted when defining a type for a parameter of a command. The omission means that the parameter may be passed a list with any element type.

A2.2.7 RECORD TYPE

Records provide a structuring mechanism for grouping data items of different types together. Each item is called a field of the record and is referenced via its field name.

```
<record type> ::= RECORD <sp|nl>
                    <field definition> <,|sp|nl>
                    [<field definition> <,|sp|nl>]...
                  RECEND

<field definition> ::=
      <field name> [<sp|nl>] <:> [<sp|nl>] <type>
```

A2.2.8 UNION TYPE

The union type provides for the case where any one of a number of types is applicable.

```
<union type> ::= ANY [<union type qualifier>]

<union type qualifier> ::= <sp> OF <sp|nl>
                             <member definition> <,|sp|nl>
                             [<member definition> <,|sp|nl>]...
                           ANYEND

<member definition> ::= <type>
```

If the union type qualifier is omitted the union consists of all possible types.

The order in which the members of the union type are defined is significant. If an expression for a union type can be successfully interpreted for more than one of its member types, it is given the first such interpretation.

```
Example:  var
            x: any of
              name
              file
            anyend
          varend

          x = fred
          display_value $type(x)
          NAME
          x = $work.fred
          display_value $type(x)
          FILE
```

A2.2.9 APPLICATION TYPE

Application types may be defined to deal with those situations where no other type known by the CDCNET command parser can be used.

```
<application type> ::=
      APPLICATION [<sp> <application value evaluator name>]
```

The application value evaluator designates a procedure supplied by the application (product, utility, user program, etc.) to parse and evaluate an expression of the particular application type. This procedure assumes total responsibility for the evaluation of the "expression" for the application value.

A2.2.10 MAXIMUM SIZE OF PARAMETERS OF TYPE NAME

Even though the definition of type name allows one to have names whose maximum size may vary between 1 and 31 characters, all name type parameters (in CDCNET commands) which are externally visible to an end user or a CDCNET operator must support a maximum size of 31 characters. In other words the size of all externally visible name type parameters may range from 1 to 31 characters.

A2.2.11 PARAMETER NAME ORDERING

The following rules are used to position values of parameters when a command is used. These rules are being provided here as things to be kept in mind when specifying a command and its parameters. Other than that this information has no purpose in this document.

Parameters for a command are specified as a sequence of individual parameters separated by commas or spaces. Parameters may be specified positionally or by name. When a parameter is not specified by name its position is taken to be one beyond that of the preceeding parameter. The significance of explicitly omitting a parameter (e.g. by placing two commas together) is only to establish the position of the next parameter. Giving a parameter by name has the effect of "tabbing" to that parameter position.

Appendix B
Philosophy of Parameter Conventions

B1.0 PARAMETER CONSISTENCY

The requirements for consistency of parameter names and uses come from the concept from which most CDCNET commands were formed: a matrix of command verbs applied across a set of command objects. That is, the majority of CDCNET commands can be derived from a matrix as follows (note that the command objects also include modifiers not shown):

```
          OBJECT |trunk  |network| gw     | metric| status| .....
    VERB ---------|-------------------------------------------
        define   |       |       |       |       |       |
                 |       |       |       |       |       |
        display  |       |       |       |       |       |
                 |       |       |       |       |       |
        change   |       |       |       |       |       |
                 |       |       |       |       |       |
        start    |       |       |       |       |       |
                 |       |       |       |       |       |
        stop     |       |       |       |       |       |
         .       |       |       |       |       |       |
         .       |       |       |       |       |       |
         .
```

Each matrix intersection defines a command. Of course, some intersections arrive at meaningless or redundant commands; these commands are discarded. In any case, the command matrix technique provides a consistent set of commands using a small number of verbs and objects. The conventions for command parameters also reflects the command matrix by requiring that consistent parameter conventions are applied across the commands. That is:

```
          OBJECT |trunk  |network| gw     | metric| status| .....
    VERB ---------|-------------------------------------------
        define   | ---> consistency across verb
                 |     . |       |       |       |       |
        display  |       | .     |       |       |       |
                 |  |    |    .  |       |       |       |
        change   |  |    |       .       |       |       |
                 |  V    |       |  .    |       |       |
        start    |consis-|       |     . |       |       |
                 |tency  |       |       .       |       |
        stop     |across |       |       | .     |       |
         .       |object |       |          consistency
         .       |       |       |       |  across all
         .                                  commands
```

B2.0 RETAINING THE MAXIMUM CONSISTENCY

The maximum possible consistency between commands should be maintained even if exceptions occur. For example, the parameter or parameters for some item could be defined and used in the following manner for the commands to which the item applies:

- A single parameter for the item used consistently across all commands.
- One parameter used for the item consistently across the commands with the same command verb with a different parameter or parameters used consistently for commands with other verbs.
- One parameter used for the item consistently across the commands with the same command object with a different parameter or parameters used consistently for commands with other objects.
- One parameter used for the item as an exception in one command, all other commands used a different parameter.
- One parameter used for the item is some commands where the parameter seems to fit, other parameters used for the item in other commands.

Clearly, it is most desirable to use the same parameter name and type to represent the same item in any command the item appears. If we feel, however, that an exception to using the same parameter in all relevant commands is justified, our first exception choice should be to replace consistency across all commands with consistency across all commands of the same verb. That is, we should walk down the levels of consistency given above and choose the first exception that works. The least desirable exception is one applied, as the consistency level suggests, haphazardly across the relevant commands.

For example, we have the convention that hardware devices within a DI will be addressed by physical name, a concatenation of device type and device address (slot number, port, etc.). However, the define command for an Ethernet trunk and the define command for a Channel trunk only require a slot number; the device type is implied by the command name. On the other hand, commands that refer to all devices in a DI (e.g., display_hardware_status) require both the device type and address. Also, the define_hdlc_trunk command requires both a port address and the LIM providing the port.

The desired exception was the allow slot number as a parameter on the define_ether_trunk and define_channel_trunk so that the device type need not be redundantly entered. Our solutions for this exception might be as follows:

B2.0 RETAINING THE MAXIMUM CONSISTENCY

- Combine the Channel and Ethernet trunk commands so that device type is meaningful. This solution was rejected because the commands have different parameters; therefore, the device_name would determine the validity of use of other parameters.

- Use slot as a parameter only on the Channel and Ethernet trunk commands. Use device_name = physical name on all others.

- Use slot as a parameter on all commands. Drop physical names. This solution was rejected because we desire to be consistent with NOS/VE device addressing on diagnostic or other hardware oriented commands.

- Use slot or lim number/port number on all define commands where device addresses are relevant.

- Allow slot number on all commands where Ethernet or Channel trunks could be the the object of the command (e.g., cancel_trunk).

Using slot on the define_ether_trunk and define_channel_trunk commands and lim and port numbers on the define_hdlc_trunk command was chosen. This solution applied the exception across all commands with the same command verb. The solution did not disrupt the consistency of other command verbs. For example, the cancel commands remained as requiring the logical name (which defaults to the physical name) to cancel an object.