CA Mainframe Network Management

Network Control Language Reference Guide

Release 12.1

CA Technologies Product References

This document references the following CA Technologies products:

- CA SOLVE:NetMail™
- CA SYSVIEW® Performance Management (CA SYSVIEW)
- CA TCPaccess™ Communications Server for z/OS (CA TCPaccess CS)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 7: Dataset Services Interface 1047

Chapter 1: Introduction

This section contains the following topics:

About NCL

NCL is a high-level interpretive language integrated into many components to provide a fast, comprehensive, and advanced development tool to implement the specific requirements of an installation. NCL is the vehicle through which your product region is rapidly tailored to the needs of the installation.

NCL is based on free-form statement syntax that can process both system and user-supplied data. Data is maintained in variables, which is manipulated and changed as required.

Collections of NCL statements, which may include system commands, are termed procedures and are stored in partitioned data sets (z/OS) or CMS files (z/VM) called procedure libraries, which is edited and updated while your product region is operational. Each NCL procedure is a separate member within a procedure library.

There is one principal procedure library (or concatenation of libraries) used by your product region. In addition to this system library, individual users under z/OS is allocated an individual procedure library for their own use, as part of the definition of their user ID.

NCL procedures can take many forms. They is a simple collection of comment statements that provide an effective means of online documentation. They is a collection of your product region commands in exactly the same format as entered from a terminal. They is extended to include logical decision making capabilities, the display of full-screen panels and the use of file processing capabilities.

An NCL procedure can call or nest to another procedure to improve modularity.

In addition, certain NCL procedures are reserved for performing special functions such as interfacing to unsolicited messages from VTAM (PPOPROC), intercepting and reacting to other messages sent to the user terminal (MSGPROC), and processing messages destined for the activity logs (LOGPROC).

Format

The following information is presented for each item described in this book:

- On the left is the name of the verb or built-in. To the right of the name are the permissible operands for that verb or built-in.
- Each item contains some or all of the following section headings:

  Operands

  Description of operands.

  Return Codes

  Return code options set on completion of the item, with an explanation.

  Examples

  Examples of interface syntax.

  Notes

  Additional information about the item.

Example: Format

```
&INTCLEAR [ TYPE={ ALL | REQ | RESP | ANY } ]
```

UPPERCASE characters

Must be entered as shown for verb or built-in names or operands consisting of uppercase characters, but can be entered in uppercase or lowercase.

***Italic* characters**

Are variables that show the kind of information, rather than the exact information that must be supplied. The actual entry replaces the italic description. Valid types of data are described for each verb or built-in within the operands section.

Underscored values

Indicate the defaulted value that is assumed for an operand if it is not specified in the verb or built-in.

{Braces}

Indicate the available options for an operand. One of the alternatives must be selected. Do not include the braces when entering a specification.

[Square brackets]

Indicate optional specifications. Do not include square brackets when entering a specification.

Or signs (|)

Separate options for an optional or mandatory specification. If a group of options is enclosed by square brackets, and each is separated by an or sign, none of the options have to be chosen. If none are coded, the default value (underscored) is used.

Commas (,) and Equal (=) signs

Must be entered as shown. If commas or equal signs appear in brackets, they are optional and used only if the accompanying optional operand is used.

Ellipsis (...)

Denotes items that are repeated.

Verbs

Keywords denote and initiate a specific action. Depending on the verb, the processing that occurs to carry out an action may require additional operands in the form of values or variables. The result of the action can also modify or create variables.

Examples: Verbs

The following example deletes a message received by MSGPROC:

```
&MSGDEL
```

The following example suspends the NCL process at this statement awaiting operator input, which is placed in variables &1, &2, ...

```
&PAUSE ARGS
```

Built-in Functions

Built-in functions are special-purpose verbs that operate on one or more parameters to give a result in a target variable; therefore, built-in functions must be specified on the right of an assignment statement.

The following shows the general syntax for statements with built-in functions:

&target = &built-in &parm ... &parm

&target

Specifies the name of a variable that receives the result.

&built-in

Specifies the name of the built-in function invoked.

&parm

Represents one or more variables or values used as input parameters to the function.

Note: The precise syntax for the function is discussed for each function, if required.

Example: Built-in Functions

The following example shows the variable &C is the target of the &CONCAT built-in function, which concatenates the values of the two variables &A and &B, then puts result, ABCDEF, into &C.

```
&A = ABC
&B = DEF
&C = &CONCAT &A &B
```

System Variable Format

System variables represent read-only information that is available to all procedures in the system. These variables cannot be modified by procedures and the information in the variable is either constant (for example, &TIME always contains the current time, regardless of which procedure references the variable) or procedure-dependent (for example, &LUNAME always returns the name of the terminal associated with the NCL region in which the procedure is executing).

The system variables documented are general purpose variables available to all NCL procedures unless specifically noted otherwise. There are also special system variables that occur only on completion of certain verbs and that provide message profile information.

Related Documentation

For more information about writing and maintaining NCL procedures, see the *Network Control Language Programming Guide*.

Chapter 2: Verbs and Built-in Functions

Summary Table

The following table is a list of the verbs and built-in functions available in Network Control Language, with a brief description of their function.

The Feature/Component column indicates whether a specific product or component must be included in the initialization parameters at region startup before you can use the verb or built-in function.

Note: For more information about the initialization parameters, see the *Reference Guide*.

Automation Services (AS) is an internal component that is enabled if any of the following products are configured in the region: FT, NETSPY, OPSCICS, OPSOS, SNA, SNAAUTO, TCPIP.

The column headed V/B indicates whether the item is a verb or built-in function.

Name	Description	Feature/ Component	V/B
&AOMALERT	Generates or simulates an AOM event, WTO, VM MSG, or MVS DOM, and routes it as required.	AS	V
&AOMCONT	Releases a message from an AOMPROC for delivery, passes the message to another AOMPROC for processing, or sends a copy of the current message to an ISR connected system.	AS	V
&AOMDEL	Deletes the message that an AOMPROC is processing.	AS	V
&AOMFLAG	Alters the value of an AOM global flag.	AS	V
&AOMFLAG	Inspects the value of an AOM global flag.	AS	B
&AOMGVAR	Alters the value of an AOM global variable.	AS	V
&AOMGVAR	Inspects the value of an AOM global variable.	AS	B
&AOMINIT	Indicates to regard the current procedure as an AOMPROC, and registers the procedure for message delivery.	AS	V
&AOMMIGID	Determines whether a migration ID is required.	AS	B
&AOMMINLN	Accesses the text of a specific minor line of a multiline WTO message in an AOMPROC.	AS	B

Name	Description	Feature/ Component	V/B
&AOMMINLT	Accesses the line type of a specific minor line of a multiline WTO message in an AOMPROC.	AS	B
&AOMREAD	Requests to make the next message available to an AOMPROC.	AS	V
&AOMREPL	Alters the text of a message and release the message for local delivery.	AS	V
&APPC	Provides access to LU6.2 conversations.		V
&APPSTAT	Returns the status for a VTAM application.		B
&ASISTR	Assigns a multiword string into a variable, retaining leading blanks.		B
&ASSIGN	Updates lists of variables in one operation.		V
&BOOLEXPR	Evaluates a Boolean expression.		B
&CALL procedure	Invokes an NCL procedure.		V
&CALL program	Invokes a user program.		V
&CMDLINE	Writes text into your OCS command input line.		V
&CNMALERT	Sends a CNM record directly to CNMPROC in a local or remote NEWS system for processing.	SNA	V
&CNMCLEAR	Requests to clear all outstanding CNM reply data solicited by this NCL user.	SNA	V
&CNMCONT	Directs the current CNM record across a specific ISR link.	SNA	V
&CNMDEL	Deletes a CNM record or stops ISR delivery of the record to a remote region.	SNA	V
&CNMPARSE	Requests to parse the MDO data supplied into user variables.	SNA	V
&CNMREAD	Requests to make the next CNM record available to an NCL procedure.	SNA	V
&CNMSEND	Requests to send the data supplied across the CNM interface.	SNA	V
&CNMVECTR	Requests to vector the data supplied into user variables.	SNA	V
&CONCAT	Concatenates multiple variables/constants.		B
&CONTROL	Sets NCL procedure control characteristics.		V
&DATECONV	Changes a date format.		B
&DEC	Converts a hexadecimal number to its decimal equivalent.		B
&DECODE	Decodes part or all of an MDO.		V
&DELAY	Interrupts processing of a procedure for a specified period of time.		V

Name	Description	Feature/ Component	V/B
&DO	Groups a sequence of NCL statements to form a logical program function.		V
&DOEND	Signifies the logical end of a group of statements.		V
&DOM	Issues an MVS DOM to erase a non-roll delete (NRD) WTO.		V
&DOUNTIL	Builds a conditional loop with a test at the bottom.		V
&DOWHILE	Builds a conditional loop with a test at the top.		V
&ELSE	Specifies that the code following is the alternative path after &IF, where the &IF condition is false.		V
&ENCODE	Encodes all or part of an MDO.		V
&END	Terminates the current nesting level.		V
&ENDAFTER	Terminates the current nesting level after executing the command following the &ENDAFTER.		V
&EVENT	Signals an event occurrence.		V
&EXIT	Terminates the current nesting level.		V
&FILE	Connects, disconnects, switches, accesses, modifies, and deletes file records.		V
&FLUSH	Terminates all nesting levels within an NCL process.		V
&FNDSTR	Determines whether a string occurs within one or more variables.		V
&GOSUB	Branches to a subroutine within the procedure.		V
&GOTO	Branches to another statement within the procedure.		V
&HEX	Converts a decimal number to its hexadecimal equivalent.		B
&HEXEXP	Converts a character string to its hexadecimal equivalent.		B
&HEXPACK	Converts a hexadecimal string into equivalent characters.		V
&IF	Tests the truth of a logical expression.		V
&INTCLEAR	Clears messages queued to a dependent processing environment.		V
&INTCMD	Schedules a command to execute in the dependent environment of the issuing process.		V
&INTCONT	Propagates a message to the next higher processing environment.		V
&INTREAD	Retrieves the next message queued from the dependent processing environment of the issuing process.		V
&INTREPL	Propagates a message to the next higher processing environment, and changes the message text.		V

Name	Description	Feature/ Component	V/B
&INVSTR	Inverts a string.		B
&LBLSTR	Removes leading blanks from a string.		B
&LENGTH	Tells you the length of a variable or constant.		B
&LOCK	Obtains or releases access to a resource.		V
&LOGCONT	Resumes normal processing of a message delivered to LOGPROC.		V
&LOGDEL	Deletes a log record that LOGPROC is processing.		V
&LOGON	Passes control of a terminal to another application.		V
&LOGREAD	Makes the next log message available to LOGPROC.		V
&LOGREPL	Replaces the text of the last log message delivered to LOGPROC.		V
&LOOPCTL	Sets a new runaway loop control limit.		V
&MAICMD	Specifies an MAI primary command.	SNAACCESS	V
&MAICONT	Sends the current data stream on to the terminal and/or the application.	SNAACCESS	V
&MAICURSA	Sets up the cursor address to send to the application.	SNAACCESS	V
&MAIDEL	Signifies not to deliver a data stream.	SNAACCESS	V
&MAIDSFMT	Places the entire current data stream into variables.	SNAACCESS	V
&MAIFIND	Determines whether a data stream contains a given string.	SNAACCESS	V
&MAIINKEY	Sets the attention key that is to be simulated in a data stream.	SNAACCESS	V
&MAIPUT	Builds a data stream to send to the PLU (application).	SNAACCESS	V
&MAIREAD	Waits for the next data stream.	SNAACCESS	V
&MAIREPL	Replaces a data stream destined for the terminal.	SNAACCESS	V
&MAISADD	Adds a new session definition, based on user variables.	SNAACCESS	V
&MAISCMD	Specifies an MAI session command against the current session.	SNAACCESS	V
&MAISGET	Retrieves details of the specified session into user variables.	SNAACCESS	V
&MAISPUT	Updates MAI session list entries.	SNAACCESS	V
&MASKCHK	Tests a data string against a wildcard mask.		B
&MSGCONT	Resumes normal processing of a message delivered to MSGPROC.		V
&MSGDEL	Deletes a message that MSGPROC is processing.		V
&MSGREAD	Makes the next message available to MSGPROC.		V
&MSGREPL	Replaces the text of a message delivered to MSGPROC.		V

Name	Description	Feature/ Component	V/B
&NBLSTR	Removes leading and trailing blanks from a string.		B
&NDBADD	Adds a record to an NDB database.		V
&NDBCLOSE	Signs off (disconnects) from an NDB database.		V
&NDBCTL	Alters NDB processing characteristics.		V
&NDBDEF	Adds, updates, or deletes field definitions.		V
&NDBDEL	Deletes a record from an NDB database.		V
&NDBFMT	Defines a list of fields for an &NDBGET to retrieve.		V
&NDBGET	Retrieves a record from an NDB database.		V
&NDBINFO	Retrieves information about an NDB database.		V
&NDBOPEN	Signs on (connects) to an NDB database.		V
&NDBPHON	Allows you to return a phonetic value for a character string, typically a name.		V
&NDBQUOTE	Places quotes around data to protect special characters.		B
&NDBSCAN	Scans an NDB database for all records matching a search argument.		V
&NDBSEQ	Defines, deletes, or resets a sequential retrieval path for an NDB database.		V
&NDBUPD	Updates a record in an NDB database.		V
&NPFxCHK	Tests a user's network partitioning authority for a resource.		B
&NRDDEL	Deletes NRD messages.		V
&NUMEDIT	Edits the format of a real number or integer.		B
&OVERLAY	Replaces a section of a data string with data from another string.		B
&PANEL	Displays a full-screen panel.		V
&PANELEND	Gives up exclusive control of a display window.		V
&PARSE	Parses tokenized strings into variables.		V
&PAUSE	Suspends an NCL process.		V
&PPI	Allows exchange of data between programs.		V
&PPOALERT	Generates a simulated VTAM PPO message.		V
&PPOCONT	Resumes normal processing of a VTAM PPO message.		V
&PPODEL	Deletes a VTAM PPO message, or blocks its delivery.		V
&PPOREAD	Makes the next VTAM PPO message available to PPOPROC.		V

Name	Description	Feature/ Component	V/B
&PPOREPL	Resumes normal VTAM PPO message processing, after replacing message text.		V
&PROMPT	Writes text to a user's terminal and awaits input.		V
&QEXITR	Terminates this procedure, plus all higher levels.		V
&REMSTR	Splits a data string and returns the end portion.		B
&RETCODE	Returns or resets the system return code.		V
&RETSUB	Returns from a subroutine within a procedure.		V
&RETURN	Passes variables to a higher nesting level.		V
&RSCCHECK	Tests the access of a user to a resource.		B
&SECCALL	Communicates with the security subsystem or the installation security exit.		V
&SELSTR	Splits a data string and returns the front portion.		B
&SETBLNK	Explicitly sets a variable to blank.		B
&SETLENG	Sets the length of a variable.		B
&SETVARS	Extracts named keywords and associated data from a data string.		V
&SMFWRITE	(z/OS only) Writes a record to the SMF data set.		V
&SNAMS	Provides the client/server interface to invoke object-oriented services.	SNA	V
&SOCKET	Provides NCL control over allocation and management of communications using TCP/IP.		V
&STR	Assigns a multiword string.		B
&SUBSTR	Extracts part of a variable or constant.		B
&TBLSTR	Removes trailing blanks from a string.		B
&TRANS	Translates characters within a string.		B
&TYPECHK	Tests variables and returns their type.		B
&VARTABLE	Creates and maintains vartables and vartable entries.		V
&WRITE	Writes a message.		V
&WTO	Issues a WTO.		V
&WTOR	Issues a WTOR and waits for a reply.		V
&ZAMCHECK	Indicates whether support is enabled for a specified access method.		B
&ZFEATURE	Returns availability status of a feature.		B

Name	Description	Feature/ Component	V/B
&ZNCLKWD	Indicates whether the string is an NCL keyword.		B
&ZOSCHK	Indicates whether support is enabled for a specified operating system.		B
&ZPSKIP	Sets new active panel skip data.		V
&ZQUOTE and &ZQUOTE2	Places quotes around a string.		B
&ZSHRINK	Removes leading and trailing spaces and reduces multiple spaces within a string.		B
&ZSOCINFO	Obtains information about the specific socket owned by the process.		B
&ZSUBST	Returns a string with substituted data.		B
&ZSYSPARM	Returns the value of a systems parameter (SYSPARMS).		B
&ZTCPERDS	Returns a short message for a TCP/IP error code.		B
&ZTCPERNM	Returns the logical name of a TCP/IP error code.		B
&ZTCPINFO	Obtains information about the local host or TCP/IP vendor stack.		B
&ZTCPSUPP	Determines whether the current TCP/IP vendor stack supports a function.		B
&ZUNQUOTE	Removes one level of quotes from a string and undoes &ZQUOTE.		B

&AOMALERT

The &AOMALERT verb generates an event, message, WTO, or DOM occurrence, as if it had been produced by either the local operating system interface, or received across an AOM ISR link. The verb can also be used to send an occurrence directly across an ISR link or to a specific AOMPROC.

Any NCL procedure can use &AOMALERT. It is not restricted to an AOMPROC, or any particular environment. This allows processes outside of AOM to provide input to AOM.

By default, &AOMALERT generates an EVENT, with a ROUTE option of PROCONLY. Thus, the primary AOMPROC sees the event, and can process it.

A comprehensive set of operands allows specification of the exact occurrence to generate, and the attributes of that occurrence.

This verb has the following format:

```
&AOMALERT [ TYPE={ EVENT | WTO | MSG | DOM } ]
          [ SOS={ OS | VM } ]
          [ STATUS={ YES | NO } ]
          [ DOMAIN=domain | LINK=link | NCLID=nclid ]
          [ ROUTE={ PROCONLY | route } [ LCLROUTE=route ] [ RMTROUTE=route ] ]
          [ ALARM={ YES | NO } ]
          [ ASID=asid ]
          [ CLASS=class ]
          [ COLOR={ NO | BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE } ]
          [ DESC={ NONE | ALL | list } ]
          [ DOM-TRACK={ YES | NO } ]
          [ DOMID={ * | mvsdomid ]
          [ HLITE={ NO | USCORE | REVERSE | BLINK } ]
          [ ID={ AOMALERT | identifier } ]
          [ INTENS={ NORMAL | HIGH } ]
          [ JOBNAME=jobname ]
          [ JOBID=jobid ]
          [ JSTCB=jstcb ]
          [ LDOMAIN=domain ]
          [ MONITOR={ YES | NO } ]
          [ MSGCLASS={ 30 | msgclass } ]
          [ MSGCODE={ 00 | nn } ]
          [ MSGID=msgid ]
          [ MSGLEVEL={ IN | msglevel } ]
          [ NRD={ NO | OPER | YES } ]
          [ ODOMAIN=domain ]
          [ ROUTCDE = { 2 | NONE | ALL | list }]
          [ RMTCLASS={ NONE | ALL | list } ]
          [ SCAN={ YES | NO } ]
          [ SOURCE={ PROP | GCS } ]
          [ TIME=hhmmss ]
          [ TRACE=* ]
          [ UFLAGS=nn | UFLAGn={ YES | NO } ]
          [ USERID=userid ]
          [ USERNODE=node ]
          [ DATA=msg, wto or event data ]
```

Operands:

TYPE={ EVENT | WTO | MSG | DOM }

Specifies the type of occurrence to generate.

TYPE=EVENT (the default)

Generates an AOM event message. An event message allows notification to an AOMPROC of an event, but the message is never delivered to authorized AOM receivers.

TYPE=WTO

Generates a simulated WTO message. In this case, several operands allow specification of job name, JES job ID, message attributes, as well as address space and DOMID information.

TYPE=MSG

Generates a simulated VM message. In this case, several operands allow specification of VM user ID and node, message attributes and VM message class.

TYPE=DOM

Generates a simulated MVS DOM. A DOMID must be specified, and it is used to cause deletion of NRD messages.

Note: TYPE=DOM does not send the generated DOM to MVS. Use the &DOM NCL verb for this.

SOS={ OS | VM }

Indicates the sourcing operating system. By default, this is the operating system that this product region is running on.

SOS=OS

Indicates that the WTO, EVENT, or DOM is to be marked as originating from an OS type system (including z/OS, MSP, and VOS3). In this case, operands that imply or are relevant to TYPE=MSG (which implies SOS=VM) are not permitted.

SOS=VM

Indicates that the MSG or EVENT is to be marked as originating from a VM type system. In this case, operands that imply or are relevant to TYPE=WTO or TYPE=DOM are not permitted.

For TYPE=EVENT, messages are marked as coming from either OS or VM. Source information, such as job name, user ID, and so on, must be consistent with the indicated or defaulted operating system. For example, if SOS=VM is specified, JOBNAME and JOBID cannot be specified.

STATUS={ YES | NO }

Indicates whether this is a status only message. This parameter is ignored for TYPE=DOM.

Status only messages are never delivered to Local AOM receivers (regardless of the ROUTE options), and have an implied RMTCLASS=8 if the RMTCLASS parameter is not specified.

STATUS=YES

Makes the message equivalent to the internally generated status messages for such things as AOM START, AOM STOP, SYSTEM IN SHUTDOWN that are queued to AOMPROC. This setting is useful if some other procedure in the system, such as LOGPROC needs to send status information to AOMPROC.

STATUS=NO

Specifies that the message is to be delivered normally.

DOMAIN=*domain* | LINK=*link* | NCLID=*nclid*

Allows the alert to be sent to a specific destination, either a specific AOMPROC, or a specific ISR link identified either by LINK name (as defined on this system), or DOMAIN ID (as defined on the remote system). None of these operands is used with TYPE=DOM. Only one of these three operands is specified.

DOMAIN=*domain*

Queues the message to the ISR link where the remote system has the indicated domain ID. This operand makes the sending end independent of the link name used locally to identify the remote system. If the message cannot be queued, a non-zero &ZFDBK value indicates the reason.

LINK=*link*

Queues the message to the ISR link with the specified local LINK name. This operand makes the sending end dependent on the link name. This is advantageous when different remote systems could be (at different times) connected with the same link name. If the message cannot be queued, a non-zero &ZFDBK value indicates the reason.

NCLID=*nclid*

Queues the message to the nominated AOMPROC in this system. If the specified NCL ID is not found, is equal to the NCL ID of the issuing NCL process, or is not a current primary or secondary AOMPROC, the &ZFDBK system variable is set to 8 and the message not queued.

This option allows individual AOMPROCs to hold conversations with each other, independent of the incoming message flow.

These options allow AOMPROCs in separate, ISR-connected systems to communicate with each other independently of the unsolicited message flow.

ROUTE={ PROCONLY | *route* } | [LCLROUTE=*route*] [RMTROUTE=*route*]

Specifies the AOM routing option for the current WTO, message, or event.

Specifying a single value for ROUTE sets both the local and remote route options to the same value. Specifying values for LCLROUTE and RMTROUTE allows individual setting of the local and remote AOM route options. Valid values are:

NO

Does not deliver the message. If set from the screening table, the message is not passed to Automation Services.

LOG

Delivers the message to the activity log only.

Note: If your product region has SYSPARMS AOMLOG=NO set, the message is never logged.

MSG

Delivers the message to authorized AOM receivers and to the activity log.

PROC

Queues the message to the primary AOMPROC, if it is active. If not, the message is delivered as if ROUTE=MSG was specified. Following AOM processing, ROUTE=PROC is treated as ROUTE=MSG.

PROCONLY

Queues the message to the primary AOMPROC, if it is active. If not, the message is delivered as if ROUTE=NO was specified. Following AOM processing, ROUTE=PROCONLY is treated as ROUTE=NO.

BOTH

Treats the message as if ROUTE=MSG was specified—it is immediately queued to all eligible AOM receivers and logged. Following this, it is also queued to the primary AOMPROC, if it is active. Regardless of the action taken by any AOMPROC that handles the message, it is never redelivered locally—it is treated as if ROUTE=NO was specified, and this cannot be overridden by the ROUTE operand on any other verb.

If NCLID= is specified, the message is always queued to the specified NCL procedure (if there), regardless of the ROUTE value specified. For example, if ROUTE=NO is coded, the NCL procedure still gets the message. The same applies for LINK= and DOMAIN=, although the message is discarded at the remote end depending on the ISR command LCLROUTE and RMTROUTE options in effect.

ALARM={ YES | NO }

Allows specification of the ALARM attribute for a WTO or MSG. If not specified, the default, ALARM=NO, is assumed.

ASID=*asid*

Allows specification of an address space ID. This operand is only permitted for TYPE=WTO or TYPE=EVENT with SOS=MVS. *asid* must contain exactly four hexadecimal digits. This corresponds to the values provided by the &AOMASID system variable. By specifying an address space ID that ties in to MVS, any address-space DOM correctly correlates to this message, if it is also marked as NRD=YES, DOM-TRACK=YES, or an AOMPROC uses DOM-NOTIFY=YES.

If this operand is specified, the JSTCB operand must also be specified.

CLASS=*class*

Allows specification of an event class. This operand is only permitted for TYPE=EVENT (which is the default if TYPE= is not specified).

The class value must be from 0 to 12 characters in length.

COLOR={ NO | BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE }

Allows specification of a color attribute for TYPE=WTO or TYPE=MSG. The operand can also be spelt COLOUR.

DESC={ NONE | ALL | *list* }

Allows specification of a descriptor code list, for TYPE=WTO or TYPE=MSG. The list is a single descriptor number, or a list of numbers or ranges:

- DESC=5
- DESC=(3,6,12–14)

Note: For a list of valid descriptor codes, see the *Administration Guide*.

DOM-TRACK={ YES | NO }

Lets you indicate that, for TYPE=WTO, or TYPE=EVENT with SOS=MVS, any subsequent DOM that matches this message follows it across any ISR links automatically, even if the message or event is not marked NRD=YES.

DT is an acceptable abbreviation for this operand.

DOMID={ * | *mvsdomid* }

Allows specification of an MVS DOMID for TYPE=WTO, or TYPE=EVENT with SOS=MVS, or TYPE=DOM (in which case it is required).

DOMID=*

Generates a DOMID internally. This DOMID will not match any valid DOMID that MVS generates. This format is not valid for TYPE=DOM.

DOMID=*mvsdomid*

Allows specification of your own DOMID. *mvsdomid* must be eight hexadecimal digits. The first two correspond to a system ID, the last six are the DOMID.

Regardless of the specification of DOMID, after &AOMALERT the system variable &ZDOMID contains the DOMID that was used, in the MVS (eight hexadecimal digits) format. Thus, if &AOMALERT generated a DOMID, you can save it for a future &AOMALERT TYPE=DOM.

Default: *

HLIGHT={ NO | USCORE | REVERSE | BLINK }

Allows specification of a highlight attribute for TYPE=WTO or TYPE=MSG. The operand can also be spelt HLITE.

ID={ AOMALERT | *identifier* }

Allows specification of an optional 1- to 12-character identifier. This value is made available to AOMPROC in &AOMID, and to an MSGPROC in &ZMAOMID.

INTENS={ NORMAL | HIGH }

Specifies whether the intensity attribute is TYPE=WTO or TYPE=MSG.

JOBNAME=*jobname*

Allows specification of the job name that is to be regarded as the source of a TYPE=WTO or TYPE=EVENT with SOS=MVS alert. This operand cannot be specified if TYPE=MSG or SOS=VM is specified.

jobname is either null or one through eight characters in length.

This value corresponds to, and is derived from, the &AOMJOBNM and &ZMAOMJN system variables.

JOBID=*jobid*

Allows specification of the JES job ID that is to be regarded as the source of a TYPE=WTO or TYPE=EVENT with SOS=MVS alert. This operand cannot be specified if TYPE=MSG or SOS=VM is specified.

jobid is either null or in the format *typnnnnn* where:

typ is JOB, STC, or TSU (abbreviated to J, S, or T), and *nnnnn* is a one- to five-digit number.

This value corresponds to, and is derived from, the &AOMJOBID and &ZMAOMJI system variables.

JSTCB=*jstcb*

Allows specification of a job-step TCB address. This operand is only permitted for TYPE=WTO or TYPE=EVENT with SOS=MVS. *jstcb* must be exactly eight hexadecimal digits. This corresponds to the values provided by the &AOMJSTCB system variable. By specifying a job-step TCB address that ties in to MVS, any JSTCB-specific DOMs correctly correlate to this message, if it is also marked as NRD=YES, DOM-TRACK=YES, or an AOMPROC uses DOM-NOTIFY=YES.

If this operand is specified, the ASID operand must also be specified.

LDOMAIN=*domain*

Allows specification of a specific last-handler domain value. If omitted, the current domain ID is used (that is, the value in &ZNMDID).

The value must be 1 through 4 characters, the first alphabetic or national, the remainder alphanumeric or national.

This operand corresponds to, and is initialized from, the &AOMLDID system variable.

MONITOR={ YES | NO }

This operand allows specification of the MONITOR attribute for TYPE=WTO or TYPE=MSG. If YES, the message is eligible for delivery to MONITOR class message receivers, and AOM class receivers.

MSGCLASS={ 30 | *msgclass* }

Specifies of a specific VM *msgclass*, for TYPE=MSG. This corresponds to the IUCV *MSG message class. Classes 1 through 8 are IUCV message types; class 30 is a programmable operator facility message type. Valid values are:

1

Message sent using CP MESSAGE and CP MSGNOH

2

Message sent using CP WARNING

3

Asynchronous CP messages, CP responses to a CP command executed by the programmable operator facility virtual machine, and any other console I/O initiated by CP

4

Message sent using CP SMSG command

5

Any data directed to the virtual console by the virtual machine (WRTERM, LINEDIT, and others)

6

Error messages from CP (EMSG)

7

Information messages from CP (IMSG)

8

Single Console Image Facility (SCIF) message from CP 30 Message coming from Automation Services

MSGCODE={ 00 | *nn* }

Allows specification of an MSGCODE value for TYPE=WTO or TYPE=MSG. This value is used to restrict the delivery of the message to receivers with at least one common MSGCODE in their user ID definition.

nn must be two hexadecimal digits. If omitted, 00 is assumed.

MSGID=*msgid*

Allows specification of a specific message id for type=WTO, MSG, or EVENT. If omitted, no message ID value is set.

msgid is null, or 1 through 12 characters.

The value set by this parameter is available to an AOMPROC in the &AOMMSGID system variable, and to an MSGPROC in the &ZMAOMMID system variable.

MSGLEVEL={ IN | *msglevel* }

Allows specification of a message severity level, corresponding to the &AOMMSGLV system variable, and the user profile AOMMSGLV values.

Only one message level is assigned. The default, IN, means informational.

NRD={ NO | OPER | YES }

Allows specification of the non-roll delete attribute for a message or event.

NRD=NO

Means the message is not to be regarded as NRD.

NRD=OPER

Causes the message to remain on OCS screens, but the system retains no copy. When an operator deletes the message from the screen, it is not recallable.

NRD=YES

Causes the message to remain on OCS screens, until the appropriate DOM is sent (it could be produced by &AOMALERT TYPE=DOM, or an MVS DOM). The system remembers the message, and it is recalled to OCS screens with the NRDRET command.

ODOMAIN=*domain*

Allows specification of an originator domain value. If this operand is omitted, the current domain ID is used (that is, the value in &ZNMDID).

The value must be 1 through 4 characters, the first alphabetic or national, the remainder alphanumeric or national.

This operand corresponds to, and is initialized from, the &AOMODID system variable.

ROUTCDE={ 2 | NONE | ALL | *list* }

Allows specification of a list of routing codes for the generated WTO or MSG. Either a single value or list of values is specified as follows:

```
ROUTCDE=5
```

```
ROUTCDE=(1,5,16-29,112)
```

RMTCLASS={ NONE | ALL | *list* }

Allows specification of a list of remote ISR delivery classes, for automatic ISR delivery of WTO, MSG, or EVENT alerts. Remote classes are numbered from 1 to 8.

RMTCLASS=NONE means no remote classes. RMTCLASS=ALL is equivalent to RMTCLASS=1–8.

Either a single value, or a list of values and ranges is specified:

```
RMTCLASS=5
```

```
RMTCLASS=(3,5-7)
```

If this parameter is omitted, RMTCLASS=ALL is assumed if STATUS=NO is specified, else RMTCLASS=8 is assumed if STATUS=YES is specified or defaulted.

SCAN={ YES | NO }

Allows specification of the scan-for-highlight (@) characters for message display. If SCAN=YES is specified, when the message text is displayed on an OCS screen, text between @ characters is displayed in high intensity.

SOURCE={ PROP | GCS }

Indicates the source of the message for TYPE=MSG or TYPE=EVENT with SOS=VM.

PROP

Indicates the message is to be regarded as originating from the PROP virtual machine IUCV interface.

GCS

Indicates the message is to be regarded as originating from the GCS IUCV interface.

TIME=*hhmmss*

Specifies the message issue time. If omitted, the current time is used. This parameter must be in the format *hhmmss*.

TRACE=*

Specifies that the current message is to be traced on the local system. Trace messages are written to the activity log.

UFLAGS=*nn* | UFLAG*n*={ YES | NO }

Assigns a value to the 8 user flags, either by using a hexadecimal value to assign all 8, or individually using UFLAG1, UFLAG2, and so on.

In an AOMPROC, the system variables &AOMUFLG*n* contains the set values.

In MSGPROC, the system variables &ZMAOMUFM and &ZMAOMUF*n* contain the user flags in various formats. Variables are also set after an &INTREAD if receiving unsolicited messages.

USERID=*userid*

Specifies the VM user ID that is to be regarded as the source of a TYPE=MSG or TYPE=EVENT with SOS=VM alert. This operand cannot be specified if TYPE=WTO or SOS=MVS is specified. *userid* is null, else it must be from 1 to 8 characters in length.

This value corresponds to, and is derived from, the &AOMUSERI and &ZMAOMUI system variables.

USERNODE=*node*

Specifies the VM RSCS node that is to be regarded as the source of a TYPE=MSG or TYPE=EVENT with SOS=VM alert. This operand cannot be specified if TYPE=WTO or SOS=MVS is specified. *node* is null; otherwise, it must be from 1 to 8 characters in length.

This value corresponds to, and is derived from, the &AOMUSERN and &ZMAOMUN system variables.

DATA=*msg, wto or event data*

The optional message text to be associated with the WTO, MSG, or EVENT.

For TYPE=WTO, ensure that the first character of the text is the MVS flag character (which is blank).

Examples: &AOMALERT

```
&AOMALERT DATA=MSG001 SYSTEM UP

&AOMALERT TYPE=EVENT STATUS=NO LINK=NEWYORK DATA=LA IS ONLINE

&AOMALERT TYPE=WTO STATUS=NO ROUTE=MSG DATA=*IEF233D COUNTERFEIT MSG
```

Notes:

If NCLID, DOMAIN, or LINK were specified, &ZFDBK is set as follows:

0

Message queued successfully.

4

Destination domain same as ODOMAIN or LDOMAIN value.

8

Link/domain not found or NCL ID not found, not an AOMPROC, or same as issuing process.

12

Link/domain not enabled outbound.

16

ISR storage shortage. Message not sent.

20

Queue overflow on ISR link.28 Internal ISR error.

In all other cases, &ZNCLID is set to 0.

The &ZDOMID system variable is always cleared to null by &AOMALERT, and, if TYPE=WTO or TYPE=ALERT with OS=MVS message is generated, the supplied or internally assigned MVS DOMID is placed in &ZDOMID in the MVS format (eight hexadecimal digits).

When an AOMPROC reads a message produced by &AOMALERT, the &AOMSALRT system variable is always set to YES. Thus, AOMPROCs can always distinguish &AOMALERT-sourced messages.

More information:

&AOMREAD (see page 65)

&AOMCONT

The &AOMCONT verb is used in an AOMPROC procedure to request the processing of a message in *one* of the following ways. The message could be a WTO or WTOR, VM MSG, MVS DOM notify, or AOM EVENT that was previously delivered for processing by an &AOMREAD.

- Passed to your product region for normal delivery, including possible automatic ISR delivery. The message is no longer available to the issuing AOMPROC.
- Passed to your product region for local delivery only. No automatic ISR delivery is performed. The message is no longer available to the issuing AOMPROC.
- Passed to your product region for possible automatic ISR delivery only. The message is not delivered to local AOM receivers. The message is no longer available to the issuing AOMPROC.
- Copied to a specified ISR link, or to all active ISR links. The message remains available to the issuing AOMPROC.
- Passed to another AOMPROC for further processing. The message is no longer available to the issuing AOMPROC.

An AOM message passed to your product region for normal delivery is sent to all OCS and ROF users with the following authorities:

- AOM message receipt authority
- Authorized to receive messages with the relevant routing codes and message levels

The processing attributes of the message, set by the screening table when the message was selected for routing to AOMPROC, is changed by specifying the appropriate &AOMCONT operand. When this is done with the LINK= or DOMAIN= operands, only the copy of the message that is sent to the ISR link has its attributes altered.

When processing a multiline WTO message, only the current line of the message is altered. However, if this is the first line, and the next &AOMREAD specifies MINOR=NO, all lines will have the same attribute alterations applied.

Any operand specified overrides the current value of that message attribute. Attribute modification operands are ignored if the current message is an MVS DOM-notify message.

The LINK, DOMAIN, and NCLID options are only valid when processing the first line of a multiline WTO. The entire message (or a copy of it) is sent.

This verb has the following format:

```
&AOMCONT [ ALL | LOCAL | REMOTE | NCLID=nclid | LINK={ * | link } | DOMAIN={ * | domain } ]
         [ ALARM={ YES | NO } ]
         [ COLOR={ NONE | BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE } ]
         [ DOM-NOTIFY={ NO | YES } ]
         [ DOM-TRACK={ NO | YES } ]
         [ HLIGHT={ NONE | USCORE | BLINK | REVERSE } ]
         [ ID=identifier ]
         [ INTENS={ LOW | HIGH } ]
         [ MONITOR={ YES | NO } ]
         [ MSGCODE=nn ]
         [ MSGID=msgid]
         [ MSGLEVEL=msglevel ]
         [ NRD={ NO | OPER | YES } ]
         [ RMTCLASS={ ALL | NONE | LIST } ]
         [ ROUTCDE={ NONE | ALL | list } ]
         [ ROUTE=route | [  LCLROUTE=route ] [ RMTROUTE=route ] ]
         [ UFLAGn={ YES | NO } ]
```

Operands:

ALL

Indicates that the current message is to be (possibly) altered as requested, and released for both local and automatic remote ISR delivery. Following this statement, this AOMPROC no longer owns the message, and until another &AOMREAD is executed, the message-related system variables are null.

For a multiline message, only the current line is processed, and the next line is made available by issuing &AOMREAD MINOR=YES. If this was the last line, or if the next &AOMREAD has MINOR=NO specified, then the entire message is released.

LOCAL

Indicates that the current message is to be (possibly) altered as requested, and released for local delivery only. No automatic ISR delivery is performed. Following this statement, this AOMPROC no longer owns the message, and until another &AOMREAD is executed, the message-related system variables are null.

This option is invalid when processing minor lines of a multiline message. If specified for the major line, the entire message is released.

REMOTE

Indicates that the current message is to be (possibly) altered as requested, and released for automatic ISR remote delivery only. No local delivery is performed. Following this statement, this AOMPROC no longer owns the message, and until another &AOMREAD is executed, the message-related system variables are null.

This option is invalid when processing minor lines of a multiline message. If specified for the major line, the entire message is released.

NCLID=*nclid*

Indicates that the current message is to be passed to the specified secondary AOMPROC. The *nclid* must correspond to a secondary AOMPROC (that is, a procedure, executing in the AOMP environment, that has issued &AOMINIT).

If the message is successfully queued to the secondary AOMPROC, this AOMPROC no longer owns the message, and until another &AOMREAD is executed, the message-related system variables are null. The &ZFDBK system variable is set to 0. If the specified NCL ID cannot be found, or is not a secondary AOMPROC, or is the same as the NCL ID of the issuing AOMPROC, the &ZFDBK system variable is set to 8. The message remains owned by this AOMPROC.

Note: If NCLID= is specified, you cannot specify other operands. This means that the message attributes cannot be altered in this case.

This option is not valid for a minor line of a multiline message. If specified for the major (first) line, the entire message is enqueued.

LINK={ * | *link* } | DOMAIN={ * | *domain* }

Enqueues a copy of the message to the indicated ISR link, specified by LINK=link, or DOMAIN=domain; or enqueues a copy of the message to all presently active and enabled (for AOM unsolicited outbound from this system) ISR links, specified by LINK=* or DOMAIN=* (no route code, message level, or RMT class filtering is done in this case).

In either case, the message remains under the control of the current AOMPROC.

When LINK=link or DOMAIN=domain is specified, the &ZFDBK system variable is set as follows:

0

Message enqueued to specified ISR link.

1

Multiline message enqueued successfully to ISR link, but some minor lines were lost due to overflows, and so on.

4

Not enqueued, either the destination domain matched the message originator or last-handler domain, or the message has already been sent to that link.

8

Not enqueued, link or domain name not found.

12

Not enqueued, cannot send AOM unsolicited outbound to this ISR link.

16

Not enqueued, storage shortage.

20

Not enqueued, outbound queue overflow.

24

Specific delete done for specified link or all links.

28

Not enqueued, ISR internal error.

When LINK=* or DOMAIN=* is specified, The &ZFDBK system variable is set as follows:

0

Message enqueued to all eligible (and at least one) ISR links.

4

Message enqueued to some eligible links, but not delivered to some because of some error such as already sent or came from that link, and so on.

8

Not enqueued to any link. There were none, or all eligible links already received a copy, or an &AOMDEL LINK/DOMAIN=* was previously issued for this message (possibly by a previous AOMPROC that passed this message to this AOMPROC).

If attribute alteration options are specified, only the copy enqueued to the remote systems has its attributes altered. The current message is not changed.

These options are not valid for minor lines of a multiline message. If used for the major (first) line, a copy of the entire message is sent.

Note: After a message has been enqueued to any ISR link, it is no longer eligible for any automatic ISR delivery. Also, it cannot be enqueued twice to any link.

ALARM={ YES | NO }

Modifies the alarm attribute of the current message or event.

COLOR={ NONE | BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE }

Modifies the color attribute for the current message or event. This operand can also be spelt COLOUR.

DOM-NOTIFY={ NO | YES } DOM-NOTIFY=YES

Indicates that this AOMPROC wants a DOM-notify message queued to it when an appropriate MVS DOM is received.

This operand is only valid when ALL, LOCAL, or REMOTE is specified (or ALL defaulted).

If the current message is not a WTO, WTOR, or MVS-sourced EVENT, no DOM-notify occurs, because only MVS issues DOMs.

This operand is abbreviated to DN.

Note: Many messages never have a matching DOM generated, because they were never intended to be non-roll delete. Careless use of DOM-NOTIFY=YES can cause excessive storage wastage in the system as it notes all messages that are to be DOM-NOTIFIED.

DOM-TRACK={ NO | YES }

Alters the current value of the DOM-TRACK option for an MVS-sourced WTO, WTOR, or EVENT.

DOM-TRACK=YES causes the first following MVS DOM that matches this message to follow it across the relevant ISR links.

This operand is abbreviated to DT.

Note: Many messages never have a matching DOM generated, because they were never intended to be non-roll delete. Careless use of DOM-TRACK=YES can cause excessive storage wastage in the sending system, as it notes the sending of the message across the links.

HLIGHT={ NONE | USCORE | BLINK | REVERSE }

Modifies the highlight attribute for the current message or event. This operand can also be spelt HLITE.

ID=*identifier*

Modifies the value of the AOM ID attribute, as seen in the &AOMID and &ZMAOMID system variables.

The value is null, or from 1 to 12 characters.

INTENS={ LOW | HIGH }

Modifies the intensity attribute for the current message or event.

MONITOR={ YES | NO }

Alter the MONITOR attribute of the current message or event. The monitor attribute determines whether MONITOR class OCS receivers also receive the message even if they are not AOM receivers.

MSGCODE=*nn*

Modifies the MSGCODE mask for the current message or event.

MSGID=*msgid*

Allows specification of a specific message id for type=WTO, MSG, or EVENT. If omitted, no message ID value is set. *msgid* is null, or 1 through 12 characters. The value set by this parameter is available to an AOMPROC in the &AOMMSGID system variable, and to an MSGPROC in the &ZMAOMMID system variable.

MSGLEVEL=*msglevel*

Modifies the message level of the current message.

NRD={ NO | OPER | YES }

Alters the NRD (non-roll delete) attribute of the current message.

Note: If the message has the NRD=YES attribute (already, or set by the NRD=YES operand), then, following execution of &AOMCONT ALL, LOCAL, or REMOTE, &ZDOMID contains the region-assigned DOMID, which the &NRDDEL verb uses to delete the message from OCS screens.

RMTCLASS={ ALL | NONE | *list* }

Alters the ISR automatic delivery routing classes of the current message or event. This operand can affect the links that receive this message, if ISR automatic delivery occurs later.

ROUTCDE= {NONE | ALL | *list* }

Modifies the routing codes currently assigned to the current message.

ROUTE=*route* | [LCLROUTE=*route*] | [RMTROUTE=*route*]

Modifies the AOM routing option for the current message or event.

Specifying a single value for ROUTE sets both the local and remote route options to the same value. Specifying values for LCLROUTE and RMTROUTE allows individual setting of the local and remote AOM route options. Valid values are:

NO

Does not deliver the message. If set from the screening table, the message is not passed to Automation Services.

LOG

Delivers the message to the activity log only.

Note: If your product region has SYSPARMS AOMLOG=NO set, the message is never logged.

MSG

Delivers the message to authorized AOM receivers and to the activity log.

PROC

Queues the message to the primary AOMPROC, if it is active. If not, the message is delivered as if ROUTE=MSG was specified. Following AOM processing, ROUTE=PROC is treated as ROUTE=MSG.

PROCONLY

Queues the message to the primary AOMPROC, if it is active. If not, the message is delivered as if ROUTE=NO was specified. Following AOM processing, ROUTE=PROCONLY is treated as ROUTE=NO.

BOTH

Treats the message as if ROUTE=MSG was specified—it is immediately queued to all eligible AOM receivers and logged. Following this, it is also queued to the primary AOMPROC, if it is active. Regardless of the action taken by any AOMPROC that handles the message, it is never redelivered locally—it is treated as if ROUTE=NO was specified, and this cannot be overridden by the ROUTE operand on any other verb.

UFLAG*n*={ YES | NO }

Modifies one of the eight user flags that is initially set in the AOM screening table.

Examples: &AOMCONT

```
&AOMCONT COLOR=PINK MONITOR=YES

&AOMCONT LINK=NY

&AOMCONT LINK=LA

&AOMCONT LOCAL COLOR=RED

&AOMCONT NCLID=&SUBID1
```

Notes:

- If &AOMCONT is issued when no AOM message is current, the procedure is terminated abnormally.
- If the message had the NRD=YES attribute assigned or overridden, the system variable &ZDOMID has the region-assigned DOMID that is used to delete the message later from OCS consoles.
- &AOMCONT REMOTE and &AOMCONT LOCAL complement &AOMDEL LOCAL and &AOMDEL REMOTE. See &AOMDEL for more information.
- The &AOMCONT verb always sets the system variable &AFDBK. If LINK=, DOMAIN=, or NCLID= was not specified, it is always set to 0.

More information:

&AOMREAD (see page 65)
&AOMREPL (see page 69)
&AOMDEL (see page 55)

&AOMDEL

&AOMDEL is used within an AOMPROC procedure to request that the message that was previously delivered for processing by &AOMREAD be deleted.

Options allow deletion as follows:

- Completely, thus indicating that no further delivery of the message occurs
- From the local system only. Automatic ISR delivery can still occur.
- From ISR automatic delivery only. The message can still be locally delivered.
- From a specific ISR link, or from all ISR links.

If the current message is an MVS DOM-notify message, &AOMDEL is treated as an &AOMCONT.

This verb has the following format:

```
&AOMDEL [ ALL | LOCAL | REMOTE | LINK={ * | link } | DOMAIN={ * | domain } ]
        [ DOM-NOTIFY={ NO | YES } ]
```

Operands:

ALL

Deletes the current message completely. No further processing takes place. The message is not delivered to authorized AOM receivers, nor is it logged. No automatic ISR delivery takes place. Following this statement, this AOMPROC no longer owns the message, and until another &AOMREAD is executed, the message-related system variables are null.

For a multiline message, only the current line is processed, and the next line is made available by issuing &AOMREAD MINOR=YES. If this line was the last line, or if the next &AOMREAD has MINOR=NO specified, then the entire message is released.

LOCAL

Deletes the current message from the local system. Possible automatic ISR delivery is still to be processed. Following this statement, this AOMPROC no longer owns the message, and until another &AOMREAD is executed, the message-related system variables are null.

This operand is invalid when processing minor lines of a multiline message. If specified for the major line, the entire message is processed.

REMOTE

Delivers the current message to the local system. Possible automatic ISR delivery is not performed. Following this statement, this AOMPROC no longer owns the message, and until another &AOMREAD is executed, the message-related system variables are null.

This operand is invalid when processing minor lines of a multiline message. If specified for the major line, the entire message is processed.

Because the message is locally delivered, if the message had the NRD=YES attribute, the system variable &ZDOMID has the region-assigned DOMID. This DOMID is used to delete the message later from OCS consoles.

LINK={ * | *link* | DOMAIN={ * | *domain* }

Allows the prevention of automatic or specific delivery to a specific ISR link, specified by LINK=link or DOMAIN=domain, or prevents any further automatic or specific delivery to all ISR links, specified by LINK=* or DOMAIN=*.

In either case, the message remains under the control of the current AOMPROC.

When LINK=link or DOMAIN=domain is specified, the &ZFDBK system variable is set as follows:

0

Indicates that the message is blocked from delivery to the specified ISR link.

4

Indicates that the specified ISR link is not found or not enabled for AOM flow.

24

Indicates that an &AOMCONT verb or previous automatic ISR delivery has already sent a copy of the current message.

When LINK=* or DOMAIN=* is specified, the &ZFDBK system variable is always set to 0.

DOM-NOTIFY={ NO | YES }

Indicates whether this AOMPROC wants to be notified of any future MVS DOM for this message. This option is not valid with LINK= or DOMAIN=.

DOM-NOTIFY=YES causes the procedure to receive a DOM=notify message when AOM receives an MVS DOM that matches the message.

This operand is abbreviated to DN.

Note: Many messages never have a matching DOM generated, because they were never intended to be NRD messages. Careless use of DOM-NOTIFY=YES can waste excessive storage in the system as it notes all messages that are to be DOM-NOTIFIED.

Examples: &AOMDEL

```
&AOMDEL

&AOMDEL LOCAL DN=YES

&AOMDEL LINK=NY
```

Notes:

- &AOMDEL LOCAL and &AOMDEL REMOTE complement &AOMCONT REMOTE and &AOMCONT LOCAL, except that no message attributes is altered with &AOMDEL.
- &AOMDEL LINK=* or DOMAIN=* also prevents the following &AOMCONT LINK= or DOMAIN= from sending a message. The &ZFDBK system variable is set to 4.
- Issuing an &AOMDEL when no message is current causes the procedure to terminate abnormally with an error message.

More information:

&AOMCONT (see page 48)
&AOMREAD (see page 65)
&AOMREPL (see page 69)

&AOMGFLAG

&AOMGFLAG alters or inspects the value of an AOM global flag.

&AOMFLAG is used as a verb using the following format:

```
&AOMGFLAGn { ON | OFF }
```

When used as a built-in function, &AOMFLAG must be coded to the right of an assignment statement:

```
&variable =&AOMGFLAGn
```

Used as a verb, &AOMGFLAG is coded within an AOMPROC procedure to alter the value of an AOM global flag. These global flags are visible to the screening table and is tested and altered by screening table statements.

Used as a built-in function, any NCL procedure can inspect the value of an AOM global flag.

Operands:

n

Specifies the number of the global flag to alter or inspect. *n* must be from 1 to 32.

ON | OFF

Specifies the new value of the indicated global flag.

ON sets the flag on, and OFF sets it off. The interpretation of these settings is entirely dependent on the user.

Examples: &AOMGFLAG

```
&AOMGFLAG 5 ON       -* set global flag 5 on

&AOMGFLAG 32 OFF     -* set global flag 32 off

&FLAG7 = &AOMGFLAG 7 -* inspect the current value of
                     -* global flag 7
```

Notes:

- Screening table statements can test the value of an AOM global flag using the GFLAG(*n*) screening criteria, or alter the value of the first 16 global flags using SET GFLAG(*n*) = ON | OFF.
- If used as a verb in a procedure that is not an AOMPROC, the procedure is terminated with an error message.

&AOMGVAR

&AOMGVAR alters or inspects the value of an AOM global variable.

&AOMGVAR is used as a verb using the following format:

```
&AOMGVARn [ value ]
```

When used as a built-in function, &AOMGVAR must be coded to the right of an assignment statement:

```
&variable =&AOMGVARn
```

Used as a verb, &AOMGVAR is coded within an AOMPROC procedure to alter the value of an AOM global variable. These global variables are visible to the screening table and is tested and altered by screening table statements.

Used as a built-in function, any NCL procedure can inspect the value of an AOM global variable.

Operands:

n

Specifies the number of the global variable to alter or inspect. *n* must be from 1 to 16.

value

Specifies the new value of the indicated global variable. value is null, which will set the global variable to all blanks. Otherwise, the value must be from 1 to 16 characters. The value will be stored in the indicated global variable, padded with blanks if necessary. The interpretation of AOM global variables is entirely dependent on the user.

Examples: &AOMGVAR

```
&AOMGVAR 3 CHICAGO  -* set global variable 3 to 'CHICAGO'

&AOMGVAR 15 START   -* set global variable 15 to 'START'

&VAR4 = &AOMGVAR 4  -* get current value of global
                    -* variable 4
```

Notes:

- Screening table statements can test the value of an AOM global variable using the GVAR(*n*) screening criteria, or alter the value of a global variable using SET GVAR(*n*) = value. The values can also be used in screening table substitution, by using the &GVAR*n* variable names.
- If used as a verb, by a procedure that is not an AOMPROC, the procedure is terminated with an error message.

&AOMINIT

&AOMINIT indicates that the current procedure is to be regarded as an AOMPROC, and registers the procedure for message delivery.

This verb has the following format:

```
&AOMINIT
```

The &AOMINIT verb performs the following functions:

- Registers the executing NCL procedure as a secondary AOMPROC if:
  - The procedure is executing in the AOMP environment. The procedure is started by the primary AOMPROC, or &INTCMD started by the primary AOMPROC, or was started by SUBMIT AOMP START *procname*.
  - The procedure is not the primary AOMPROC. The primary AOMPROC is started by the SYSPARMS AOMPROC=*procname* command.
- Marks the procedure (including the primary AOMPROC) as ready for AOM message delivery. Typically, AOM messages marked for delivery to an AOMPROC are not queued until an &AOMREAD has been executed. Messages with ROUTE=PROC are treated as ROUTE=MSG, and ROUTE=PROCONLY messages are discarded.

Sometimes, however, the primary AOMPROC have housekeeping work to perform before it is ready to process messages. By issuing &AOMINIT, following messages are queued, ready for the first &AOMREAD.

Note: In the procedure, do not delay too long before issuing &AOMREAD. The maximum queued message limit (as set by SYSPARMS AOMPQLIM=) could be exceeded.

Registering a procedure as a secondary AOMPROC always makes it eligible for messages. Only another AOMPROC can deliver these messages by issuing &AOMCONT NCLID or by issuing &AOMALERT NCLID.

Notes:

The &ZFDBK system variable is set to the following values after executing &AOMINIT:

0

Indicates that the primary AOMPROC issues the &AOMINIT verb. Messages are now queued.

4

Indicates that a valid procedure in AOMP environment issues the &AOMINIT verb. Procedure is now a secondary AOMPROC. Messages are queued (using &AOMCONT NCLID). This value is also returned if the procedure was already a secondary AOMPROC.

An &AOMINIT issued by an NCL procedure that is not executing in the AOMP environment causes the procedure to terminate with an error message.

If the primary AOMPROC terminates for any reason, all secondary AOMPROCs are also terminated, as the AOMP environment is terminated.

Registered secondary AOMPROCs that terminate have any queued messages processed by normal AOM delivery. If the procedure terminates abnormally, the message indicating the procedure error termination is queued to the primary AOMPROC as an AOM status message.

&AOMMIGID

&AOMMIGID parses a system command string to determine whether a migration ID is required.

This built-in function has the following format:

&*result* = &AOMMIGID *command-string*

The return value is NO (meaning no migration ID required) or YES (meaning a migration ID is required). This value is used directly as the value for the MIGID operand on a SYSCMD command.

If one of the following is true, then NO is always returned:

- The current console type (as set by the AOMCTYPE SYSPARM) does not support migration IDs (that is, the type is NOT EXTMCS).
- No consoles are currently acquired.

If no command text is supplied, or if it is too long (more than 126 characters), then an error is raised. The NCL procedure terminates.

Example: &AOMMIGID

```
&CMD=STR D J,L

&MIG=&AOMMIGID &CMD
```

&AOMMINLN

The &AOMMINLN built-in function returns the current value of the specified minor line, when processing a multiline WTO message in an AOMPROC.

This built-in function has the following format:

&*variable* = &AOMMINLN *n*

Operand:

n

Specifies the minor line number.

0 returns the text of the major line. 1 to the value in the &AOMNMIN system variable returns that minor line. A greater value returns a null value.

Note: &AOMMINLN is used when any line of the multiline WTO is current in an AOMPROC. This means, if you use &AOMREAD MINOR=YES to read successive minor lines (to access all attributes), any preceding or following line text is accessed.

The returned value is the text of the specified minor line. An index of 0 returns the text of the major line.

Only an AOMPROC can use &AOMMINLN. If the current message is not a multiline WTO, a null value is returned.

Examples: &AOMMINLN

```
.IEC995I            -* indicative dump mlwto
&MAJ = &AOMMINLN 0  -* get major line
&MIN1 = &AOMMINLN 1 -* get minor line 1
&MIN2 = &AOMMINLN 2 -* get minor line 2
&MIN3 = &AOMMINLN 3 -* get minor line 3
```

More information:

&AOMMINLT (see page 64)
&AOMREAD (see page 65)

&AOMMINLT

The &AOMMINLT built-in function returns the line type of the specified minor line, when processing a multiline WTO message in an AOMPROC.

This built-in function has the following format:

&*variable* = &AOMMINLT *n*

Operand:

n

Specifies the minor line number. 0 returns the type of the major line. A value between 1 and the value in the &AOMNMIN system variable returns the type of that minor line. A greater value returns a null value.

The returned values are:

C

Indicates a CONTROL line.

CE

Indicates a CONTROL plus END line.

L

Indicates a LABEL line.

LE

Indicates a LABEL plus END line.

D

Indicates a DATA line.

E

Indicates an END line.

DE

Indicates a DATA/END line.

Only an AOMPROC can use &AOMMINLT. If the current message is not a multiline WTO, a null value is returned.

Examples: &AOMMINLT

```
.IEC995I             -* indicative dump mlwto
&MAJT = &AOMMINLT 0  -* get major line type
&MINT1 = &AOMMINLT 1 -* get minor line 1 type
&MINT2 = &AOMMINLT 2 -* get minor line 2 type
&MINT3 = &AOMMINLT 3 -* get minor line 3 type
```

Note:

&AOMMINLT is used when any line of the multiline WTO is current in an AOMPROC. This means, if you use &AOMREAD MINOR=YES to read successive minor lines (to access all attributes), any preceding or following line type is accessed.

&AOMREAD

&AOMREAD is used within an AOMPROC to request delivery of the next AOM message. If no message is immediately available, processing of the procedure is suspended and then resumes when the next message arrives.

Any messages and associated attributes routed to PROC, PROCONLY, or BOTH from the screening table are delivered to &AOMREAD.

Multiple &AOMREAD statements are present within an AOMPROC but we recommend that only one is used within a closed loop environment.

On completion of &AOMREAD, the system variable &ZVARCNT are set to the number of variables created, unless the SET operand is specified.

The profile of the message received by &AOMREAD is set in a suite of reserved system variables. The message profile provides a complete description of all the message attributes in addition to the message text. (The attributes reflect the processing attributes such as color, highlighting, source, and delivery information that the screening table sets.)

This verb has the following format:

```
&AOMREAD [ WAIT={ YES | NO | 0 | nn } ]
         [ MINOR={ YES | NO } ]
       { [ VARS=prefix* [ RANGE=(start,end) ] ] |
         [ VARS={ name | var-name-list } ] |
         [ STRING { &name | string-name-list } ] |
         [ ARGS [ RANGE=(start,end) ] ] |
         [ SET ] }
```

Operands:

WAIT={ YES | NO | 0 | *nn.nn* }

Specifies what to do if no message is immediately available to satisfy the &AOMREAD.

- WAIT=YES causes the procedure to be suspended indefinitely, pending receipt of a message.
- WAIT=NO or WAIT=0 causes the procedure to continue if no message is pending. If no message was pending, the system variable &RETCODE is set to 4. The procedure is never suspended in this case.
- WAIT=*nn.nn* causes the procedure to wait for up to *nn.nn* seconds for a message. If none have arrived in this time, &RETCODE is set to 4 and the procedure resumes execution. The range of *nn.nn* is 0.01 to 9999.99.

If a message satisfies &AOMREAD, &RETCODE is set to 0.

Default: YES

MINOR={ YES | NO }

Specifies whether &AOMREAD is to return minor lines of a multiline WTO message.

- MINOR=YES presents the individual lines of a multiline WTO message in order, following the &AOMREAD of the major line. This setting is useful if you want to examine attributes of each line.
- MINOR=NO causes the next &AOMREAD after reading a major line to skip all the minor lines, and read the next message.

Default: YES

Note: Regardless of this setting, the &AOMMINLN and &AOMMINLT built-in functions allow access to the text and line type of any minor line of the current message. The &AOMNMIN system variable indicates how many minor lines there are. The operand has no effect except in the case where the previous line read was a multiline WTO major line. Thus, you can use &AOMREAD MINOR=NO in a main processing loop, and enter an inner processing loop, to read &AOMNMIN minor lines.

VARS=

Tokenizes the message into the nominated variables before control is returned to the procedure. Each word of the message is tokenized into the nominated variables from left to right. If insufficient variables are provided, some data is lost. Excess variables are set to a null value. The format of the operands that can be coded with VARS= are as follows:

prefix*

Denotes that variables are generated automatically during the tokenization process, and that the variable names are *prefix*1, *prefix*2, and so on. The RANGE= operand is specified to indicate a starting and ending suffix number. *prefix** cannot be used with other variable names.

name

Specifies the name of a variable, excluding the ampersand (&).

***name*(*n*)**

Is the same as *name,* but *n* denotes the length of the data that is placed in the variable.

***(*n*)**

Denotes a skip operation, where *n* represents the number of units to skip during the tokenization process. On VARS= statements, *n* denotes 'skip this number of words'. An asterisk (*) by itself is the same as *(1).

STRING

Specifies no tokenization. The entire text of the message is returned as a single string to the procedure in the nominated variables. The format of the operands associated with STRING are:

&name

Specifies the variables, including the leading &, into which to place the string text. Text is placed into each variable for the maximum length of a variable.

ARGS

Denotes that the message received is tokenized and placed word by word into automatically generated variables of the form &1 through &*n*, depending on how many are required to hold the message. The RANGE= operand is coded to designate a start number and optionally, an end number, which delimits the number of variables that are generated.

SET

Specifies that no tokenization of the incoming AOM message is performed, but that the &AOMREAD statement is to return only the AOMPROC system variables that relate to the message.

If SET is not coded, instructions must be coded on the &AOMREAD statement specifying the tokenization requirements for the message by using other &AOMREAD operands.

Examples: &AOMREAD

```
&AOMREAD VARS=(A,B,C,D,E,F,G,H,I,J) MINOR=NO

&AOMREAD VARS=A*

&AOMREAD ARGS RANGE=(2,60)

&AOMREAD STRING &DATA

&AOMREAD SET
```

Notes:

Following an &AOMREAD, you can use an &GOTO statement, using the ID of the message, to go to the routine that processes the message. (The ID is set in &AOMID by a screening table GLOBAL, MSGGROUP, or SET statement ID=*id*.)

When a message is delivered to &AOMREAD, a suite of system variables is set, including one for the message prefix (&AOMMSGID). No request for further tokenization is necessary unless specific processing is required on the message text. On this basis, using &AOMREAD SET is sufficient.

The following example illustrates the use of both &AOMMSGID and &AOMID:

```
&CONTROL NOLABEL
.READ
&AOMREAD MINOR=NO SET
&GOTO .&AOMMSGID
&GOTO .&AOMID
&AOMCONT
&GOTO .READ
 :
-* Special msg processing
.IOS000I
 : &GOTO .READ
-* Special msg processing for screening table set ID.
.HASPMSGS

 :
&GOTO .READ
```

While testing and developing the primary AOMPROC, you can terminate the current version and invoke a new updated copy. The SYSPARMS AOMPROC= command with FLUSH specified terminates the primary AOMPROC. To start the primary AOMPROC, the same command is used, specifying the member name from the relevant NCL library.

More information:

&AOMCONT (see page 48)
&AOMDEL (see page 55)
&AOMREPL (see page 69)

&AOMREPL

&AOMREPL requests that a message, which was previously delivered for processing by an &AOMREAD, have the message text replaced and then delivered locally. The message could be a WTO, a WTOR, or a VM message. The verb is used within an AOMPROC procedure. No automatic ISR delivery is performed.

An AOM message for delivery is sent to all OCS and ROF users with the following authorities:

- AOM message receipt authority
- Authorized to receive messages with the relevant routing codes and message levels

The screening table sets the processing attributes of the message when the message was selected for routing to AOMPROC. These attributes are changed by specifying the appropriate &AOMREPL operand.

When processing a multiline WTO message, only the current line of the message is altered. If this line is the first line, and the next &AOMREAD specifies MINOR=NO, all lines have the same attribute alterations applied. The text of these lines, however, is not altered. The text of individual lines is altered by issuing &AOMREAD MINOR=YES to obtain each line in turn, and then issuing &AOMREPL for each line.

Any operand specified overrides the current value of that message attribute. This statement is treated as &AOMDEL for an MVS DOM-notify message.

This verb has the following format:

```
&AOMREPL [ LOCAL ]
         [ ALARM={ YES | NO } ]
         [ COLOR={ NONE | BLUE | RED | PINK | GREEN |TURQUOISE | YELLOW | WHITE } ]
         [ DOM-NOTIFY={ NO | YES } ]
         [ HLIGHT={ NONE | USCORE | BLINK | REVERSE } ]
         [ ID=identifier ]
         [ INTENS={ LOW | HIGH } ]
         [ MONITOR={ YES | NO } ]
         [ MSGCODE=nn ]
         [ MSGID=msgid ]
         [ MSGLEVEL=msglevel ]
         [ NRD={ NO | OPER | YES } ]
         [ ROUTCDE={ NONE | ALL | list } ]
         [ ROUTE=route | LCLROUTE=route ]
         [ SCAN={ YES | NO } ]
         [ UFLAGn={ YES | NO } ]
           DATA=replacement message text
```

Operands:

LOCAL

Specifies that the current message is to be (possibly) altered as requested and released for local delivery only. This process occurs regardless of the specification of LOCAL or not. This operand is provided for compatibility with &AOMCONT LOCAL. No automatic ISR delivery is performed.

Following this statement, this AOMPROC no longer owns the message, and until another &AOMREAD is executed, the message-related system variables are null.

ALARM={ YES | NO }

Modifies the alarm attribute of the current message or event.

COLOR={ NONE | BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE }

Modifies the color attribute for the current message or event. This operand can also be spelt COLOUR.

DOM-NOTIFY={ NO | YES }

DOM-NOTIFY=YES indicates that this AOMPROC wants a DOM-notify message queued to it when an appropriate MVS DOM is received.

If the current message is not a WTO, WTOR, or MVS-sourced EVENT, no DOM-notify ever occurs, as only MVS issues DOMs.

This operand is abbreviated to DN.

Note: Many messages never have a matching DOM generated, as they were never intended to be NRD. Careless use of DOM-NOTIFY=YES can waste excessive storage in the system as it notes all messages that are to be DOM-NOTIFIED.

HLIGHT={ NONE | USCORE | BLINK | REVERSE }

Modifies the highlight attribute for the current message or event. This operand can also be spelt HLITE.

ID=*identifier*

Modifies the value of the AOM ID attribute, as seen in the &AOMID and &ZMAOMID system variables.

The value is null, or from 1 through 12 characters.

INTENS={ LOW | HIGH }

Modifies the intensity attribute for the current message or event.

MONITOR={ YES | NO }

Alters the MONITOR attribute of the current message or event. The monitor attribute determines whether MONITOR class OCS receivers also receive the message, even if they are not AOM receivers.

MSGCODE=*nn*

Modifies the MSGCODE mask for the current message or event.

MSGID=*msgid*

Allows specification of a specific message id for type=WTO, MSG, or EVENT. If omitted, no message ID value is set.

msgid is null, or 1 through 12 characters.

The value set by this parameter is available to an AOMPROC in the &AOMMSGID system variable, and to an MSGPROC in the &ZMAOMMID system variable.

MSGLEVEL=*msglevel*

Modifies the message level of the current message.

For a list of valid message levels, see the *Administration Guide*.

NRD={ NO | OPER | YES }

Alters the NRD attribute of the current message.

Note: If the message has the NRD=YES attribute, then, following execution of &AOMCONT ALL, LOCAL, or REMOTE, &ZDOMID contains the region-assigned DOMID. The &NRDDEL verb uses DOMID to delete the message from OCS screens.

ROUTCDE={ NONE | ALL | list }

Modifies the routing codes currently assigned to the current message.

ROUTE=*route* | LCLROUTE=*route*

Modifies the AOM routing option for the current message or event.

Because the message is only delivered locally, ROUTE= or LCLROUTE= have the same effect. Valid values are:

NO

Does not deliver the message. If set from the screening table, the message is not passed to Automation Services.

LOG

Delivers the message to the activity log only.

Note: If your product region has SYSPARMS AOMLOG=NO set, the message is never logged.

MSG

Delivers the message to authorized AOM receivers and to the activity log.

PROC

Queues the message to an active primary AOMPROC. If the primary AOMPROC is not active, the message is delivered as if ROUTE=MSG was specified. Following AOM processing, ROUTE=PROC is treated as ROUTE=MSG.

PROCONLY

Queues the message to an active primary AOMPROC. If the primary AOMPROC is not active, the message is delivered as if ROUTE=NO was specified. Following AOM processing, ROUTE=PROCONLY is treated as ROUTE=NO.

BOTH

Treats the message as if ROUTE=MSG was specified—it is immediately queued to all eligible AOM receivers and logged. The message is also queued to an active primary AOMPROC. Regardless of the action taken by any AOMPROC that handles the message, it is never redelivered locally. The message is treated as if ROUTE=NO was specified, which cannot be overridden by the ROUTE operand on any other verb.

SCAN={ YES | NO }

Indicates whether the replacement text has highlight characters (@) to highlight parts of the text selectively.

UFLAG*n*={ YES | NO }

Modifies one of the eight user flags that is initially set in the AOM screening table.

DATA=*replacement message text*

Specifies the replacement text for the message. Any text is specified, up to 256 characters. If the current message is part of a multiline WTO, only the current message text is replaced.

If this operand is omitted, &AOMREPL acts like &AOMDEL. The message is deleted.

Examples: &AOMREPL

```
&AOMREPL COLOR=PINK MONITOR=YES DATA=&NEWMSG

&AOMREPL DATA=*&AOMMSGID MESSAGE SUPPRESSED BY AOMPROC
```

Notes:

- IF &AOMREPL is issued when no AOM message is current, the procedure is terminated abnormally.
- If the message had the NRD=YES attribute assigned or overridden, the system variable &ZDOMID have the region-assigned DOMID. The DOMID is used to delete the message from OCS consoles later.

More information:

&AOMCONT (see page 48)
&AOMDEL (see page 55)
&AOMREAD (see page 65)

&APPC

&APPC provides access to LU6.2 conversations.

This verb has the following formats:

```
&APPC ALLOCATE_DELAYED
      TRANSID=transid
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]

&APPC ALLOCATE_IMMEDIATE
      TRANSID=transid
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM }  ]
    [ USERID=userid  [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]

&APPC ALLOCATE_NOTIFY
      TRANSID=transid
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid  [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME= luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

```
&APPC { ALLOCATE_SESSION | ALLOCATE }
        TRANSID=transid
      [ MODENAME=modename ]
      [ SYNC={ NONE | CONFIRM } ]
      [ WAIT=nn ]
      [ USERID=userid [ PASSWORD=password ] ]
      [ PROFILE=profile ]
      [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
        LUNAME=luname | LINK=linkname | DOMAIN=domain ]
      [ PARMS=(parm1,parm2,...,parmn) |
        VARS=var | VARS=(var1, var2, ..., varn ) |
        VARS=prefix* [ RANGE=(start,end) ] ]

&APPC ATTACH_DELAYED
      PROC=procname
    [ SERVER=servername
     [ SCOPE={ REGION | USER | SYSTEM } ] ]
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]

&APPC ATTACH_IMMEDIATE
      PROC=procname
    [ SERVER=servername
     [ SCOPE={ REGION | USER | SYSTEM } ] ]
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

```
&APPC ATTACH_NOTIFY
      PROC=procname
    [ SERVER=servername
     [ SCOPE={ REGION | USER | SYSTEM } ] ]
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]

&APPC { ATTACH_SESSION | ATTACH }
        PROC=procname
      [ SERVER=servername [ SCOPE={ REGION | USER | SYSTEM } ] ]
      [ MODENAME=modename ]
      [ SYNC={ NONE | CONFIRM } ]
      [ WAIT=nn ]
      [ USERID=userid [ PASSWORD=password ] ]
      [ PROFILE=profile ]
      [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
        LUNAME=luname | LINK=linkname | DOMAIN=domain ]
      [ PARMS=(parm1,parm2,...,parmn) |
        VARS=var | VARS=(var1, var2, ..., varn ) |
        VARS=prefix* [ RANGE=(start,end) ] ]

&APPC CONFIRM
    [ ID=id ]

&APPC CONFIRMED
    [ ID=id ]
```

```
&APPC CONNECT_DELAYED
    { NCLID=nclid | SERVER=servername }
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]

&APPC CONNECT_IMMEDIATE
    { NCLID=nclid | SERVER=servername }
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]

&APPC CONNECT_NOTIFY
    { NCLID=nclid | SERVER=servername }
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]

&APPC { CONNECT_SESSION | CONNECT }
    { NCLID=nclid | SERVER=servername }
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ WAIT=nn ]
    [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]

&APPC DEALLOCATE
    [ TYPE={ SYNC | FLUSH | CONFIRM | ABEND | LOCAL } ]
    [ ID=id ]
    [ LOG=msg ]

&APPC DEREGISTER

&APPC FLUSH
    [ ID-id ]

&APPC { PREPARE_TO_RECEIVE | PREPARE }
      [ TYPE={ SYNC | FLUSH | CONFIRM } ]
      [ ID=id ]

&APPC { RECEIVE_AND_WAIT | RECEIVE }
      [ ID={ ANY | CLIENTS | SERVERS | id } ]
      [ WAIT=nn ]
      [ MDO=mdoname [ MAP=mapname ] | VARS=var | VARS=(var1, var2, ..., varn ) |
        VARS=prefix* [ RANGE=(start,end) ] ]

&APPC RECEIVE_IMMEDIATE
    [ ID=id ]
    [ MDO=mdoname [ MAP=mapname ] | VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]

&APPC RECEIVE_NOTIFY
    [ ID={ ANY | CLIENTS | SERVERS | id } ]
```

```
&APPC REGISTER
      SERVER=servername [ SCOPE={ REGION | USER | SYSTEM } ]
    [ CONNECT={ ACCEPT | NOTIFY | REJECT } ]
    [ RETRY={ YES  | NO } ]
    [ CONVLIM={ 100 | nnn } ]

&APPC REQUEST_TO_SEND
    [ ID=id ]

&APPC RPC
      PROC=procname
    [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ SHARE | SHARE=(shrvars1,shrvars2,...,shrvarsn) |
      NOSHARE=(shrvars1,shrvars2,...,shrvarsn) ]
    [ RETCODE=varname ]
    [ PARMS=(parm1,parm2,...,parmn) ]

&APPC SEND_AND_CONFIRM
    [ ID=id ]
    [ MDO=mdoname [ MAP=mapname ] |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]

&APPC SEND_AND_DEALLOCATE
    [ ID=id ]
    [ MDO=mdoname [ MAP=mapname ] |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
    [ TYPE={ SYNC | FLUSH | CONFIRM } ]
    [ LOG=msg ]

&APPC SEND_AND_FLUSH
    [ ID=id ]
    [ MDO=mdoname [ MAP=mapname ] |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
    [ CONT={ YES | NO } ]

&APPC { SEND_AND_PREPARE_TO_RECEIVE |  SEND_AND_PREPARE }
      [ ID=id ]
      [ TYPE={ SYNC | FLUSH | CONFIRM } ]
      [ MDO=mdoname [ MAP=mapname ] |
        VARS=var | VARS=(var1, var2, ..., varn ) |
        VARS=prefix* [ RANGE=(start,end) ] ]
```

```
&APPC { SEND_DATA | SEND }
      [ ID=id ]
      [ MDO=mdoname [ MAP=mapname ] |
        VARS=var | VARS=(var1, var2, ..., varn ) |
        VARS=prefix* [ RANGE=(start,end) ] ]
      [ CONT={ YES | NO } ]

&APPC SEND_ERROR
    [ ID=id ]
    [ LOG=msg ]

&APPC SET_SERVER_MODE
    [ CONNECT={ ACCEPT | NOTIFY | REJECT } ]
    [ RETRY={ YES | NO } ]
    [ CONVLIM=nnn ]

&APPC START
      PROC=proc
    [ SERVER=servername SCOPE={ REGION | USER | SYSTEM } ] ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ NOTIFY={ NO | YES } ]
    [ VARS=(genvars1,genvars2,...,genvarsn) ]
    [ PARMS=(parm1,parm2,...,parmn) ]

&APPC TEST
    [ ID=id ]

&APPC TRANSFER_ACCEPT
    [ ID=id ]
    [ NCLID=nclid | SERVER=servername ]
    [ ARGS | VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]

&APPC TRANSFER_CONNECT
    [ ID=id ]
    [ NCLID=nclid | SERVER=servername ]
    [ WAIT=nn ]

&APPC TRANSFER_REJECT
      ID=id
    [ NCLID=nclid | SERVER=servername ]
    [ RETRY={ YES | NO } ]

&APPC { TRANSFER_REQUEST | TRANSFER }
      [ NCLID=nclid | SERVER=servername ]
      [ ID=id ]
      [ WAIT=nn ]
```

&APPC Return Code Information

All &APPC verb options complete by setting a number of NCL variables with return code information.

General completion code information is contained within the &RETCODE system variable, and is qualified by the &ZFDBK system variable. &RETCODE values of 0 and 4 occur in normal operation, while &RETCODE values of 8 or higher indicate an error condition. If an error is detected, the &SYSMSG user variable is set providing an explanation of the error condition.

A number of APPC system variables are available providing information about the current conversation being operated. For example, following a receive operation, the &ZAPPCWR and &ZAPPCWRI (the what-received indicators) provide information about what satisfied the receive.

&RETCODE and &ZFDBK

The &RETCODE system variable provides general completion information as follows:

0

Operation successful

4

Conversation ended (or operation unsuccessful)

8

Remote program error

12

State error

16

LU6.2 architected error

These values can provide sufficient feedback information to control simple procedures; however, more detailed information is available to assist in debugging or real-time recovery of certain errors. This information is provided by the &ZFDBK system variable, which is set in conjunction with &RETCODE.

&RETCODE	&ZFDBK	Meaning
0	0	Operation successful
4	0	Normal conversation deallocation
	4	Immediate request failure

&RETCODE	&ZFDBK	Meaning
8	0	Program_error_purging
	4	Program_error_no_truncation
	8	Program_error_truncation
12	0	State error
16	0	Parameter error
	4	Allocation_failure_retry
	8	Allocation_failure_no_retry
	12	Sync_level_not_supported_by_LU
	16	Deallocate_abend_prog
	20	Deallocate_abend_svc
	24	Deallocate_abend_timer
	28	Svc_error_purging
	32	Svc_error_no_truncation
	36	Svc_error_truncation
	40	Resource_failure_retry
	44	Resource_failure_no_retry
	48	FMH_data_not_supported
	52	Mapping_not_supported
	56	Map_not_found
	60	Map_execution_failure
	64	Security_not_valid
	68	TPN_not_recognized
	72	PIP_not_allowed
	76	PIP_not_specified_correctly
	80	Conversation_type_mismatched
	84	Sync_level_not_supported_by_program
	88	Trans_pgm_not_avail_retry
	92	Trans_pgm_not_avail_no_retry

&RETCODE 0 always signifies a successful operation and is equivalent to the LU6.2 architected RETURN_CODE value of OK.

&RETCODE 4 with &ZFDBK 0 is normally set when, following a receive operation, a deallocation flush is received indicating normal conversation termination.

&RETCODE 4 with &ZFDBK 4 is set when an immediate request is unsuccessful, for example, an ALLOCATE_IMMEDIATE or RECEIVE_IMMEDIATE cannot be satisfied.

&RETCODE 8 always signifies that the remote program issued an error through the SEND_ERROR request.

&RETCODE 12 always signifies that the procedure has issued a verb from an invalid state. This value usually indicates a programming error; however, in some cases the state is changed internally when severe errors occur. Recovery is possible depending upon the sophistication of the procedure.

&RETCODE 16 always signifies a serious error. In most cases, these errors are unrecoverable; however, some recovery is possible depending upon the sophistication of the procedure.

Note: For more information about &ZFDBK values, see IBM's *SNA Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084-4).

&APPC ALLOCATE_DELAYED

A conversation is started when an allocate request is issued to attach a transaction program in the conversation partner system. ALLOCATE_DELAYED indicates a conversation allocation request where delayed session allocation is permitted. Provided the request is valid, control is returned to the procedure immediately and conversation operation is permitted pending a session becoming available for the conversation.

&APPC ALLOCATE_DELAYED has the following format:

```
&APPC ALLOCATE_DELAYED
      TRANSID=transid
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid  [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

TRANSID=*transid*

Provides the local transaction identifier for this conversation, as defined in the TCT. If no transaction is known by that identifier, then the allocation fails; otherwise the TCT entry is used to complete details for the allocation. This operand is required.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the TCT or OSCT is used to select a default mode name. If present, *modename* must correspond to a valid mode defined for the APPC link, or the allocation fails.

SYNC= { NONE | CONFIRM }

Specifies the synchronization level for the conversation. If omitted, the TCT is used to nominate the default SYNC level. If SYNC level of NONE is specified, then the following requests cannot be used:

- &APPC CONFIRM and CONFIRMED
- &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM
- DEALLOCATE TYPE=CONFIRM

Default: Set by TCT

USERID=*userid* [PASSWORD=*password*]

Specifies the target user ID and, optionally, the password under which the transaction starts on the remote system.

If the transaction being allocated is defined in the TCT with SECURITY=NONE, these operands are ignored.

If the TCT entry is defined with SECURITY=SAME, SECURITY=USERPSWD, or SECURITY=USER, APPC includes the supplied user ID and password in the attach header for validation on the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

ENV={ CURRENT | DEPENDENT | BACKGROUND } | LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The ENV operand is the default destination, indicating that the allocation is to take place entirely within the local region of the allocating procedure. If the allocation request came from a remote system, this operand is ignored. Otherwise, the ENV operand specifies the environment where the procedure is attached.

If ENV=CURRENT, or defaulted, the attached target procedure has the same relationship to the allocating procedure as it would have if a START command invokes it.

If ENV=DEPENDENT, the new process is attached in a dependent environment. The process has the same relationship to the allocating procedure as it would have if an &INTCMD START command invokes it.

If ENV=BACKGROUND, the new process is attached in the background APPC server region for the requesting user ID, as if the request was sourced from a remote system.

The LUNAME operand specifies the name of the network LU on which to allocate the remote transaction. When LUNAME is specified, the LINK parameter cannot be used. LUNAME is abbreviated to LU.

If LUNAME=*luname* is specified, then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization. Otherwise, the allocation fails. If *luname* resolves to the local LU, the conversation proceeds exactly as it would for a remote system, not as described for the ENV parameter. That is, it executes the remote end of the conversation in the appropriate APPC environment within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LU (where you want to allocate the remote transaction). If LINK is specified, then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK). The definition enables the link to be initialized by the request.

The DOMAIN operand enables identification of the target APPC system by domain name.

The ENV, LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Default: ENV=CURRENT

PARMS=(*parm1*,*parm2*,...,*parmn*) | VARS=*var* |
VARS=(*var1*, *var2*, ..., *varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Alternatively, the VARS operand is used to specify the allocation parameters:

VARS=*var* | VARS=(*var1*,*var2*,...,*varn*) |
VARS=*prefix [RANGE=(*start*,*end*)]**

Provides the list of any Program Initialization Parameters (PIP data) to be passed to the transaction processor in the remote end upon initialization of the conversation. Each token implicated by the VARS option is passed as a separate parameter in the PIP subfield list. If the remote conversation partner is an NCL procedure, PIP data is available when the remote procedure is invoked as &1,&2....&*n* in the usual manner.

Examples: &APPC ALLOCATE_DELAYED

```
&APPC ALLOCATE_DELAYED TRANSID=DBQUERY

&APPC ALLOCATE_DELAYED TRANSID=DBQUERY LINK=NMA
```

Return Codes:

The request has the following return codes:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is set, plus all APPC system variables.

This request sets the system variable &ZAPPCID, which returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation where the NCL process is operating more than one conversation concurrently.

State Transition:

The request is deemed to be performed from reset state, and following successful completion of this request, the conversation enters send state locally. When the remote program or procedure is attached to service the conversation, it is in receive state.

Notes:

- If no target system information is supplied, TCT must provide a default destination; otherwise the request fails.
- When using the request for a valid transaction, a return code of 0 (request successful) is normal. The allocation is assumed complete, and the conversation enters send state. This allows the procedure to continue processing pending session assignment. The procedure can consume system resource (such as storage) by sending data before any output path is established. However, in controlled situations, this can usefully lead to overlapped processing and can provide the most efficient LU6.2 usage in certain cases.
- No information is returned on the request about whether a session was assigned. Subsequent conversation requests, such as a SEND_DATA, can fail with a return code of 16 indicating an allocation failure. Subsequent conversation requests can also cause procedure execution to be suspended pending actual session assignment. This situation occurs, for example, when a CONFIRM request is issued following a number of SEND_DATA requests, or when internal buffering limits are reached.

Relationship to LU6.2 Verb Set:

&APPC ALLOCATE_DELAYED is equivalent to the LU6.2 verb MC_ALLOCATE with the RETURN_CONTROL(DELAYED_ALLOCATION_PERMITTED) option.

More information:

&APPC ALLOCATE_SESSION (see page 99)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC ALLOCATE_IMMEDIATE

ALLOCATE_IMMEDIATE indicates an immediate conversation allocation request and that the allocation is conditional upon a session being immediately available. If no session is immediately assigned to the conversation, the allocation fails.

&APPC ALLOCATE_IMMEDIATE has the following format:

```
&APPC ALLOCATE_IMMEDIATE
      TRANSID=transid
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

TRANSID=*transid*

Provides the local transaction identifier for this conversation, as defined in the TCT. If no transaction is known by that identifier, then the allocation fails; otherwise the TCT entry is used to complete details for the allocation. This operand is required.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the TCT or OSCT is used to select a default mode name. If present, *modename* must correspond to a valid mode defined for the APPC link, or the allocation fails.

SYNC= { NONE | CONFIRM }

Specifies the synchronization level for the conversation. If omitted, the TCT is used to nominate the default SYNC level. If SYNC level of NONE is specified, then the following requests cannot be used:

- &APPC CONFIRM and CONFIRMED
- &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM
- DEALLOCATE TYPE=CONFIRM

Default: Set by TCT

USERID=*userid* [PASSWORD=*password*]

Specifies the target user ID and, optionally, the password under which the transaction starts on the remote system.

If the transaction being allocated is defined in the TCT with SECURITY=NONE, these operands are ignored.

If the TCT entry is defined with SECURITY=SAME, SECURITY=USERPSWD, or SECURITY=USER, APPC includes the supplied user ID and password in the attach header for validation on the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

ENV={ CURRENT | DEPENDENT | BACKGROUND } | LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The ENV operand is the default destination, indicating that the allocation is to take place entirely within the local region of the allocating procedure. If the allocation request came from a remote system, this operand is ignored. Otherwise, the ENV operand specifies the environment where the procedure is attached.

If ENV=CURRENT, or defaulted, the attached target procedure has the same relationship to the allocating procedure as it would have if a START command invokes it.

If ENV=DEPENDENT, the new process is attached in a dependent environment. The process has the same relationship to the allocating procedure as it would have if an &INTCMD START command invokes it.

If ENV=BACKGROUND, the new process is attached in the background APPC server region for the requesting user ID, as if the request was sourced from a remote system.

The LUNAME operand specifies the name of the network LU on which to allocate the remote transaction. When LUNAME is specified, the LINK parameter cannot be used. LUNAME is abbreviated to LU.

If LUNAME=*luname* is specified, then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization. Otherwise, the allocation fails. If *luname* resolves to the local LU, the conversation proceeds exactly as it would for a remote system, not as described for the ENV parameter. That is, it executes the remote end of the conversation in the appropriate APPC environment within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LU (where you want to allocate the remote transaction). If LINK is specified, then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK). The definition enables the link to be initialized by the request.

The DOMAIN operand enables identification of the target APPC system by domain name.

The ENV, LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Default: ENV=CURRENT

PARMS=(*parm1*,*parm2*,...,*parmn*) | VARS=*var* |
VARS=(*var1*, *var2*, ..., *varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Alternatively, the VARS operand is used to specify the allocation parameters:

VARS=*var* | VARS=(*var1*,*var2*,...,*varn*) |
VARS=*prefix [RANGE=(*start*,*end*)]**

Provides the list of any Program Initialization Parameters (PIP data) to be passed to the transaction processor in the remote end upon initialization of the conversation. Each token implicated by the VARS option is passed as a separate parameter in the PIP subfield list. If the remote conversation partner is an NCL procedure, PIP data is available when the remote procedure is invoked as &1,&2....&*n* in the usual manner.

Examples: &APPC ALLOCATE_IMMEDIATE

```
&APPC ALLOCATE_IMMEDIATE TRANSID=DBQUERY

&APPC ALLOCATE_IMMEDIATE TRANSID=DBPUT VARS=DATA
```

Return Codes:

The request has the following return codes:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is set, plus all APPC system variables.

This request sets the system variable &ZAPPCID, which returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation where the NCL process is operating more than one conversation concurrently.

State Transition:

The request is deemed to be performed from reset state, and following successful completion of this request, the conversation enters send state locally. When the remote program or procedure is attached to service the conversation, it is in receive state.

Notes:

- If no target system information is supplied, TCT must provide a default destination; otherwise the request fails.
- An ALLOCATE_IMMEDIATE is used to perform conditional allocation where the procedure wants to keep executing even when there is no session immediately available for communication. The allocation is retried at some later time. Consider also the use of the ALLOCATE_NOTIFY request.
- When no session is available, this request completes with &RETCODE set to 4, and &ZFDBK set to 0.

Relationship to LU6.2 Verb Set:

&APPC ALLOCATE_IMMEDIATE is equivalent to the LU6.2 verb MC_ALLOCATE with the RETURN_CONTROL(IMMEDIATE) option.

More information:

&APPC ALLOCATE_SESSION (see page 99)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC ALLOCATE_NOTIFY

ALLOCATE_NOTIFY allows the procedure to request an asynchronous allocation which does not complete until a session is assigned for the conversation.

A conversation is started when an allocate request is issued to attach a transaction program in the conversation partner system. ALLOCATE_NOTIFY indicates that this conversation allocation request is asynchronous and that the procedure regains control immediately. However conversation operation is unavailable until a session is assigned to the conversation and the procedure is notified of this event.

&APPC ALLOCATE_NOTIFY has the following format:

```
&APPC ALLOCATE_NOTIFY
      TRANSID=transid
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid  [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME= luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

TRANSID=*transid*

Provides the local transaction identifier for this conversation, as defined in the TCT. If no transaction is known by that identifier, then the allocation fails; otherwise the TCT entry is used to complete details for the allocation. This operand is required.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the TCT or OSCT is used to select a default mode name. If present, *modename* must correspond to a valid mode defined for the APPC link, or the allocation fails.

SYNC= { NONE | CONFIRM }

Specifies the synchronization level for the conversation. If omitted, the TCT is used to nominate the default SYNC level. If SYNC level of NONE is specified, then the following requests cannot be used:

- &APPC CONFIRM and CONFIRMED
- &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM
- DEALLOCATE TYPE=CONFIRM

Default: Set by TCT

USERID=*userid* [PASSWORD=*password*]

Specifies the target user ID and, optionally, the password under which the transaction starts on the remote system.

If the transaction being allocated is defined in the TCT with SECURITY=NONE, these operands are ignored.

If the TCT entry is defined with SECURITY=SAME, SECURITY=USERPSWD, or SECURITY=USER, APPC includes the supplied user ID and password in the attach header for validation on the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

ENV={ CURRENT | DEPENDENT | BACKGROUND } | LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The ENV operand is the default destination, indicating that the allocation is to take place entirely within the local region of the allocating procedure. If the allocation request came from a remote system, this operand is ignored. Otherwise, the ENV operand specifies the environment where the procedure is attached.

If ENV=CURRENT, or defaulted, the attached target procedure has the same relationship to the allocating procedure as it would have if a START command invokes it.

If ENV=DEPENDENT, the new process is attached in a dependent environment. The process has the same relationship to the allocating procedure as it would have if an &INTCMD START command invokes it.

If ENV=BACKGROUND, the new process is attached in the background APPC server region for the requesting user ID, as if the request was sourced from a remote system.

The LUNAME operand specifies the name of the network LU on which to allocate the remote transaction. When LUNAME is specified, the LINK parameter cannot be used. LUNAME is abbreviated to LU.

If LUNAME=*luname* is specified, then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization. Otherwise, the allocation fails. If *luname* resolves to the local LU, the conversation proceeds exactly as it would for a remote system, not as described for the ENV parameter. That is, it executes the remote end of the conversation in the appropriate APPC environment within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LU (where you want to allocate the remote transaction). If LINK is specified, then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK). The definition enables the link to be initialized by the request.

The DOMAIN operand enables identification of the target APPC system by domain name.

The ENV, LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Default: ENV=CURRENT

PARMS=(*parm1*,*parm2*,...,*parmn*) | VARS=*var* |
VARS=(*var1*, *var2*, ..., *varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Alternatively, the VARS operand is used to specify the allocation parameters:

VARS=*var* | VARS=(*var1*,*var2*,...,*varn*) |
VARS=*prefix [RANGE=(*start*,*end*)]**

Provides the list of any Program Initialization Parameters (PIP data) to be passed to the transaction processor in the remote end upon initialization of the conversation. Each token implicated by the VARS option is passed as a separate parameter in the PIP subfield list. If the remote conversation partner is an NCL procedure, PIP data is available when the remote procedure is invoked as &1,&2....&*n* in the usual manner.

Examples: &APPC ALLOCATE_NOTIFY

```
&APPC ALLOCATE_NOTIFY TRANSID=DBQUERY
&APPC ALLOCATE_NOTIFY TRANSID=DBPUT VARS=DATA LINK=NMA
…
&INTREAD TYPE=REQ VARS=NFYMSG
```

Return Codes:

The request has the following return codes:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is set, plus all APPC system variables.

This request sets the system variable &ZAPPCID, which returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation where the NCL process is operating more than one conversation concurrently.

State Transition:

The request is deemed to be performed from reset state, and following successful completion of this request, the conversation enters send state locally. When the remote program or procedure is attached to service the conversation, it is in receive state.

Notification Message Format:

When an ALLOCATE_NOTIFY request completes (with return code=0), and subsequent session allocation completes either successfully or unsuccessfully, the following message is placed on the request queue of the internal environment for the NCL procedure:

```
N00101 NOTIFY: APPC EVENT: ALLOCATE RESOURCE: &zappcid
```

&zappcid contains the conversation identifier returned by the ALLOCATE_NOTIFY request.

Notes:

- If no target system information is supplied, TCT must provide a default destination; otherwise the request fails.
- Using ALLOCATE_NOTIFY has no significant advantages over ALLOCATE_SESSION unless concurrent processing using other NCL resources is required. The internal environment of the NCL procedure receives the notification, which is accessed by issuing an &INTREAD TYPE=REQUEST. No information is provided on the notification as to the success or otherwise of the allocation. An &APPC TEST is used to set the return code on the next operation attempted. If the allocation was unsuccessful, an allocation error is returned.
- The allocation request is canceled by issuing an &APPC DEALLOCATE TYPE=ABEND request for the conversation concerned. If the NCL process terminates, the request is automatically canceled.

Relationship to LU6.2 Verb Set:

&APPC ALLOCATE_NOTIFY has no exact equivalent in the LU6.2 verb options. The request is merely an asynchronous form of the LU6.2 verb MC_ALLOCATE with the RETURN_CONTROL(WHEN_SESSION_ ALLOCATED) option.

More information:

&APPC ALLOCATE_SESSION (see page 99)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC ALLOCATE_SESSION

The ALLOCATE_SESSION request does not complete until a session is assigned for the conversation.

ALLOCATE_SESSION indicates a synchronous conversation allocation request. Control is not returned to the procedure until a session has been successfully assigned to the conversation. This request is the most usual method of conversation allocation. The ALLOCATE_SESSION verb option is abbreviated to ALLOCATE.

&APPC ALLOCATE_SESSION has the following format:

```
&APPC { ALLOCATE_SESSION | ALLOCATE }
        TRANSID=transid
      [ MODENAME=modename ]
      [ SYNC={ NONE | CONFIRM } ]
      [ WAIT=nn ]
      [ USERID=userid [ PASSWORD=password ] ]
      [ PROFILE=profile ]
      [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
        LUNAME=luname | LINK=linkname | DOMAIN=domain ]
      [ PARMS=(parm1,parm2,...,parmn) |
        VARS=var | VARS=(var1, var2, ..., varn ) |
        VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

TRANSID=*transid*

Provides the local transaction identifier for this conversation, as defined in the TCT. If no transaction is known by that identifier, then the allocation fails; otherwise the TCT entry is used to complete details for the allocation. This operand is required.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the TCT or OSCT is used to select a default mode name. If present, *modename* must correspond to a valid mode defined for the APPC link, or the allocation fails.

SYNC= { NONE | CONFIRM }

Specifies the synchronization level for the conversation. If omitted, the TCT is used to nominate the default SYNC level. If SYNC level of NONE is specified, then the following requests cannot be used:

- &APPC CONFIRM and CONFIRMED
- &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM
- DEALLOCATE TYPE=CONFIRM

Default: Set by TCT

WAIT=*nn*

Specifies the time, in seconds (for example, 10), or seconds and hundredths (for example, 1.25), for which the procedure is prepared to wait for a session. If not successful before this interval expires, the request is canceled. An unsuccessful return code results (&RETCODE is set to 4, and &ZFDBK is set to 0).

USERID=*userid* [PASSWORD=*password*]

Specifies the target user ID and, optionally, the password under which the transaction starts on the remote system.

If the transaction being allocated is defined in the TCT with SECURITY=NONE, these operands are ignored.

If the TCT entry is defined with SECURITY=SAME, SECURITY=USERPSWD, or SECURITY=USER, APPC includes the supplied user ID and password in the attach header for validation on the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

ENV={ CURRENT | DEPENDENT | BACKGROUND } | LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The ENV operand is the default destination, indicating that the allocation is to take place entirely within the local region of the allocating procedure. If the allocation request came from a remote system, this operand is ignored. Otherwise, the ENV operand specifies the environment where the procedure is attached.

If ENV=CURRENT, or defaulted, the attached target procedure has the same relationship to the allocating procedure as it would have if a START command invokes it.

If ENV=DEPENDENT, the new process is attached in a dependent environment. The process has the same relationship to the allocating procedure as it would have if an &INTCMD START command invokes it.

If ENV=BACKGROUND, the new process is attached in the background APPC server region for the requesting user ID, as if the request was sourced from a remote system.

The LUNAME operand specifies the name of the network LU on which to allocate the remote transaction. When LUNAME is specified, the LINK parameter cannot be used. LUNAME is abbreviated to LU.

If LUNAME=*luname* is specified, then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization. Otherwise, the allocation fails. If *luname* resolves to the local LU, the conversation proceeds exactly as it would for a remote system, not as described for the ENV parameter. That is, it executes the remote end of the conversation in the appropriate APPC environment within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LU (where you want to allocate the remote transaction). If LINK is specified, then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK). The definition enables the link to be initialized by the request.

The DOMAIN operand enables identification of the target APPC system by domain name.

The ENV, LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Default: ENV=CURRENT

PARMS=(*parm1*,*parm2*,...,*parmn*) | VARS=*var* |
VARS=(*var1*, *var2*, ..., *varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Alternatively, the VARS operand is used to specify the allocation parameters:

VARS=*var* | VARS=(*var1*,*var2*,...,*varn*) |
VARS=*prefix [RANGE=(*start*,*end*)]**

Provides the list of any Program Initialization Parameters (PIP data) to be passed to the transaction processor in the remote end upon initialization of the conversation. Each token implicated by the VARS option is passed as a separate parameter in the PIP subfield list. If the remote conversation partner is an NCL procedure, PIP data is available when the remote procedure is invoked as &1,&2....&*n* in the usual manner.

Examples: &APPC ALLOCATE_SESSION

```
&APPC ALLOCATE_SESSION TRANSID=DBQUERY

&APPC ALLOCATE TRANSID=DBPUT VARS=DATA LINK=NMA
```

Return Codes:

The request has the following return codes:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is set, plus all APPC system variables.

This request sets the system variable &ZAPPCID, which returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation where the NCL process is operating more than one conversation concurrently.

State Transition:

The request is deemed to be performed from reset state, and following successful completion of this request, the conversation enters send state locally. When the remote program or procedure is attached to service the conversation, it is in receive state.

Notes:

- If no target system information is supplied, TCT must provide a default destination; otherwise the request fails.
- The ALLOCATE_SESSION request is the simplest allocation option to use. The request is generally the most resource efficient because resources are not allocated until a session is available for the conversation. While the request is queued pending session availability, the procedure is suspended. Upon completion, the procedure can continue processing based purely upon the success or otherwise of the allocation request.

Relationship to LU6.2 Verb Set:

&APPC ALLOCATE or &APPC ALLOCATE_SESSION is equivalent to the LU6.2 verb MC_ALLOCATE with the RETURN_CONTROL (WHEN_SESSION_ALLOCATED) option.

The &APPC TRANSID parameter is equivalent to the LU6.2 verb MC_ALLOCATE TPN(*tpn*) option. However, rather than specifying the remote transaction program name directly, the TRANSID parameter allows a level of indirection and validation. The request first examines the Transaction Control Table for the TRANSID, then inserts the GLOBALID from the TCT as the TPN parameter for the request.

The &APPC ENV parameter is equivalent to the LU6.2 verb MC_ALLOCATE LU_NAME(OWN) option.

The &APPC LUNAME=*luname* parameter is equivalent to the LU6.2 verb MC_ALLOCATE LU_NAME(OTHER(*luname*)) option.

The &APPC LINK=*linkname* parameter is equivalent to the LU6.2 verb MC_ALLOCATE LU_NAME(OTHER(*luname*)) option. *linkname* is considered to be the locally known name for the actual network *luname* concerned.

The &APPC PARMS or VARS=(*var1*,*var2*,...,*varn*) parameter is equivalent to the LU6.2 verb MC_ALLOCATE PIP(YES(*var1*,*var2*,...,*varn*)) option.

The &APPC MODENAME and SYNC parameters are equivalent to the LU6.2 verb MC_ALLOCATE MODE_NAME and SYNC_LEVEL parameters respectively.

The &APPC USERID, PASSWORD and PROFILE operands are equivalent to the LU6.2 verb MC_ALLOCATE SECURITY parameter. NCL conversation security options can also be set through the TCT entry (see the DEFTRANS command).

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC ATTACH_DELAYED

ATTACH_DELAYED allows the requester to specify the target procedure name to be attached to the conversation. The request is only useful when both ends of the conversation reside in APPC systems.

ATTACH_DELAYED indicates a request to attach an NCL procedure to service the APPC conversation. This request is a more direct way of creating a conversation where both ends are to execute within an APPC system.

&APPC ATTACH_DELAYED has the following format:

```
&APPC ATTACH_DELAYED
      PROC=procname
    [ SERVER=servername
     [ SCOPE={ REGION | USER | SYSTEM } ] ]
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid  [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

PROC=*procname*

(Mandatory) Nominates the target procedure by name. The operand is used where the conversation is known to take place between two NCL procedures, and where the target procedure is either within the local system, or a remote APPC system, as specified by the LOCAL, LU, or LINK operands. This operand is used instead of the TRANSID operand to target an explicit procedure. However, bypassing the Transaction Control Table is often not desirable. The default security parameter for this type of allocation is SAME. The parameter indicates that a signed on user allocates the target procedure in a signed on region, while an unsecured region allocates the procedure in the background APPC region.

SERVER=*servername*

Specifies the logical name for this NCL process. The name must be up to 32 characters long and unique within the scope as determined by the SCOPE operand, or the request fails.

SCOPE={ REGION | USER | SYSTEM }

Provides the scope for the registration of *servername* (which must be unique within the scope indicated) and is valid only if the SERVER operand is present. Valid scopes are:

REGION

Indicates that *servername* is to be unique within the current session, as defined by a particular connection to your product region.

USER

Indicates that *servername* is to be unique across all sessions associated with the particular user ID

SYSTEM

Indicates that *servername* is to be unique within this product region.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the system transaction ATTACH TCT entry is used to select a default mode name in the usual manner. If present, *modename* must correspond to a valid one defined for the APPC link or the allocation fails.

SYNC= { NONE | CONFIRM }

Nominates the synchronization level for the conversation. If omitted, the system ATTACH TCT entry is used to nominate the default SYNC level. If SYNC level of NONE is specified, or defaulted, then the &APPC CONFIRM and CONFIRMED requests, the &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM request, and the DEALLOCATE TYPE=CONFIRM request cannot be used.

USERID=*userid* [PASSWORD=*password*]

Specifies the target user ID and, optionally, the password, under which the transaction starts on the remote system.

The ATTACH system transaction uses SECURITY=SAME processing. Therefore, APPC includes the supplied user ID and password in the attach header for validation in the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

ENV={ CURRENT | DEPENDENT | BACKGROUND } | LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The ENV operand is the default destination, indicating that the allocation is to take place entirely within the local region of the allocating procedure. If the allocation request came from a remote system, this operand is ignored. Otherwise, the ENV operand specifies the environment where the procedure is attached.

If ENV=CURRENT, or defaulted, the attached target procedure has the same relationship to the allocating procedure as it would have if a START command invokes it.

If ENV=DEPENDENT, the new process is attached in a dependent environment. The process has the same relationship to the allocating procedure as it would have if an &INTCMD START command invokes it.

If ENV=BACKGROUND, the new process is attached in the background APPC server region for the requesting user ID, as if the request was sourced from a remote system.

The LUNAME operand specifies the name of the network LU on which to allocate the remote transaction. When LUNAME is specified, the LINK parameter cannot be used. LUNAME is abbreviated to LU.

If LUNAME=*luname* is specified, then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization. Otherwise, the allocation fails. If *luname* resolves to the local LU, the conversation proceeds exactly as it would for a remote system, not as described for the ENV parameter. That is, it executes the remote end of the conversation in the appropriate APPC environment within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LU (where you want to allocate the remote transaction). If LINK is specified, then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK). The definition enables the link to be initialized by the request.

The DOMAIN operand enables identification of the target APPC system by domain name.

The ENV, LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Default: ENV=CURRENT

PARMS=(*parm1*,*parm2*,...,*parmn*) | VARS=*var* |
VARS=(*var1*, *var2*, ..., *varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Alternatively, the VARS operand is used to specify the allocation parameters:

VARS=*var* | VARS=(*var1*,*var2*,...,*varn*) |
VARS=*prefix [RANGE=(*start*,*end*)]**

Provides the list of any Program Initialization Parameters (PIP data) to be passed to the transaction processor in the remote end upon initialization of the conversation. Each token implicated by the VARS option is passed as a separate parameter in the PIP subfield list. If the remote conversation partner is an NCL procedure, PIP data is available when the remote procedure is invoked as &1,&2....&*n* in the usual manner.

Examples: &APPC ATTACH_DELAYED

```
&APPC ATTACH_DELAYED PROC=MYPROC LINK=SOLSYD
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

This request also sets the system variable &ZAPPCID, which returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation, if the NCL process is operating more than one conversation concurrently.

State Transition:

The request is deemed to be performed from the reset state. Following successful completion of this request, the conversation enters send state locally. When the remote program or procedure is attached to service the conversation, it is in receive state.

Notes:

- If no target system information is supplied, TCT must provide a default destination; otherwise, the request fails.
- When using the request for a valid transaction, a return code of 0 (request successful) is normal. The allocation is assumed complete, and the conversation enters send state. This allows the procedure to continue processing pending session assignment. The procedure can consume system resource (such as storage) by sending data before any output path is established. However, in controlled situations, this can usefully lead to overlapped processing and can provide the most efficient LU6.2 usage in certain cases.
- No information is returned on the request about whether a session was assigned. Subsequent conversation requests, such as a SEND_DATA, can fail with a return code of 16 indicating an allocation failure. Subsequent conversation requests can also cause procedure execution to be suspended pending actual session assignment. This situation occurs, for example, when a CONFIRM request is issued following a number of SEND_DATA requests, or when internal buffering limits are reached.

Relationship to LU6.2 Verb Set:

&APPC ATTACH_DELAYED is equivalent to the LU6.2 verb MC_ALLOCATE with the RETURN_CONTROL(DELAYED_ ALLOCATION_PERMITTED) option.

More information:

&APPC ATTACH_SESSION (see page 123)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC ATTACH_IMMEDIATE

ATTACH_IMMEDIATE allows the requester to specify the target procedure name to be attached to the conversation. The request is useful only when both ends of the conversation reside in APPC systems.

ATTACH_IMMEDIATE indicates a request to attach an NCL procedure to service the APPC conversation. This request is a more direct way of creating a conversation where both ends are to execute within an APPC system.

&APPC ATTACH_IMMEDIATE has the following format:

```
&APPC ATTACH_IMMEDIATE
      PROC=procname
    [ SERVER=servername
     [ SCOPE={ REGION | USER | SYSTEM } ] ]
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid  [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

PROC=*procname*

(Mandatory) Nominates the target procedure by name. The operand is used where the conversation is known to take place between two NCL procedures, and where the target procedure is either within the local system, or a remote APPC system, as specified by the LOCAL, LU, or LINK operands. This operand is used instead of the TRANSID operand to target an explicit procedure. However, bypassing the Transaction Control Table is often not desirable. The default security parameter for this type of allocation is SAME. The parameter indicates that a signed on user allocates the target procedure in a signed on region, while an unsecured region allocates the procedure in the background APPC region.

SERVER=*servername*

Specifies the logical name for this NCL process. The name must be up to 32 characters long and unique within the scope as determined by the SCOPE operand, or the request fails.

SCOPE={ REGION | USER | SYSTEM }

Provides the scope for the registration of *servername* (which must be unique within the scope indicated) and is valid only if the SERVER operand is present. Valid scopes are:

REGION

Indicates that *servername* is to be unique within the current session, as defined by a particular connection to your product region.

USER

Indicates that *servername* is to be unique across all sessions associated with the particular user ID

SYSTEM

Indicates that *servername* is to be unique within this product region.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the system transaction ATTACH TCT entry is used to select a default mode name in the usual manner. If present, *modename* must correspond to a valid one defined for the APPC link or the allocation fails.

SYNC= { NONE | CONFIRM }

Nominates the synchronization level for the conversation. If omitted, the system ATTACH TCT entry is used to nominate the default SYNC level. If SYNC level of NONE is specified, or defaulted, then the &APPC CONFIRM and CONFIRMED requests, the &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM request, and the DEALLOCATE TYPE=CONFIRM request cannot be used.

USERID=*userid* [PASSWORD=*password*]

Specifies the target user ID and, optionally, the password, under which the transaction starts on the remote system.

The ATTACH system transaction uses SECURITY=SAME processing. Therefore, APPC includes the supplied user ID and password in the attach header for validation in the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

ENV={ CURRENT | DEPENDENT | BACKGROUND } | LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The ENV operand is the default destination, indicating that the allocation is to take place entirely within the local region of the allocating procedure. If the allocation request came from a remote system, this operand is ignored. Otherwise, the ENV operand specifies the environment where the procedure is attached.

If ENV=CURRENT, or defaulted, the attached target procedure has the same relationship to the allocating procedure as it would have if a START command invokes it.

If ENV=DEPENDENT, the new process is attached in a dependent environment. The process has the same relationship to the allocating procedure as it would have if an &INTCMD START command invokes it.

If ENV=BACKGROUND, the new process is attached in the background APPC server region for the requesting user ID, as if the request was sourced from a remote system.

The LUNAME operand specifies the name of the network LU on which to allocate the remote transaction. When LUNAME is specified, the LINK parameter cannot be used. LUNAME is abbreviated to LU.

If LUNAME=*luname* is specified, then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization. Otherwise, the allocation fails. If *luname* resolves to the local LU, the conversation proceeds exactly as it would for a remote system, not as described for the ENV parameter. That is, it executes the remote end of the conversation in the appropriate APPC environment within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LU (where you want to allocate the remote transaction). If LINK is specified, then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK). The definition enables the link to be initialized by the request.

The DOMAIN operand enables identification of the target APPC system by domain name.

The ENV, LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Default: ENV=CURRENT

PARMS=(*parm1*,*parm2*,...,*parmn*) | VARS=*var* |
VARS=(*var1*, *var2*, ..., *varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Alternatively, the VARS operand is used to specify the allocation parameters:

VARS=*var* | VARS=(*var1*,*var2*,...,*varn*) |
VARS=*prefix [RANGE=(*start*,*end*)]**

Provides the list of any Program Initialization Parameters (PIP data) to be passed to the transaction processor in the remote end upon initialization of the conversation. Each token implicated by the VARS option is passed as a separate parameter in the PIP subfield list. If the remote conversation partner is an NCL procedure, PIP data is available when the remote procedure is invoked as &1,&2....&*n* in the usual manner.

Example: &APPC ATTACH_IMMEDIATE

```
&APPC ATTACH_IMMEDIATE PROC=MYPROC
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

This request also sets the system variable &ZAPPCID, which returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation, if the NCL process is operating more than one conversation concurrently.

State Transition:

The request is deemed to be performed from the reset state. Following successful completion of this request, the conversation enters send state locally. When the remote program or procedure is attached to service the conversation, it is in receive state.

Notes:

- If no target system information is supplied, TCT must provide a default destination; otherwise, the request fails.
- An ATTACH_IMMEDIATE is used to perform conditional allocation where the procedure wants to keep executing, even when there is no session immediately available for communication. The allocation is retried at some later time. Consider also the use of the ATTACH_NOTIFY request.
- When no session is available, this request completes with &RETCODE set to 4, and &ZFDBK set to 0.

Relationship to LU6.2 Verb Set:

&APPC ATTACH_IMMEDIATE is equivalent to the LU6.2 verb MC_ALLOCATE with the RETURN_CONTROL(IMMEDIATE) option.

More information:

&APPC ATTACH_SESSION (see page 123)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC ATTACH_NOTIFY

ATTACH_NOTIFY allows the requester to specify the target procedure name to be attached to the conversation. The request is useful only when both ends of the conversation reside in APPC systems.

ATTACH_NOTIFY indicates a request to attach an NCL procedure to service the APPC conversation. This request is a more direct way of creating a conversation where both ends are to execute within an APPC system.

&APPC ATTACH_NOTIFY has the following format:

```
&APPC ATTACH_NOTIFY
      PROC=procname
    [ SERVER=servername
     [ SCOPE={ REGION | USER | SYSTEM } ] ]
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ PARMS=(parm1,parm2,...,parmn) |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

PROC=*procname*

(Mandatory) Nominates the target procedure by name. The operand is used where the conversation is known to take place between two NCL procedures, and where the target procedure is either within the local system, or a remote APPC system, as specified by the LOCAL, LU, or LINK operands. This operand is used instead of the TRANSID operand to target an explicit procedure. However, bypassing the Transaction Control Table is often not desirable. The default security parameter for this type of allocation is SAME. The parameter indicates that a signed on user allocates the target procedure in a signed on region, while an unsecured region allocates the procedure in the background APPC region.

SERVER=*servername*

Specifies the logical name for this NCL process. The name must be up to 32 characters long and unique within the scope as determined by the SCOPE operand, or the request fails.

SCOPE={ REGION | USER | SYSTEM }

Provides the scope for the registration of *servername* (which must be unique within the scope indicated) and is valid only if the SERVER operand is present. Valid scopes are:

REGION

Indicates that *servername* is to be unique within the current session, as defined by a particular connection to your product region.

USER

Indicates that *servername* is to be unique across all sessions associated with the particular user ID

SYSTEM

Indicates that *servername* is to be unique within this product region.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the system transaction ATTACH TCT entry is used to select a default mode name in the usual manner. If present, *modename* must correspond to a valid one defined for the APPC link or the allocation fails.

SYNC= { NONE | CONFIRM }

Nominates the synchronization level for the conversation. If omitted, the system ATTACH TCT entry is used to nominate the default SYNC level. If SYNC level of NONE is specified, or defaulted, then the &APPC CONFIRM and CONFIRMED requests, the &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM request, and the DEALLOCATE TYPE=CONFIRM request cannot be used.

USERID=*userid* [PASSWORD=*password*]

Specifies the target user ID and, optionally, the password, under which the transaction starts on the remote system.

The ATTACH system transaction uses SECURITY=SAME processing. Therefore, APPC includes the supplied user ID and password in the attach header for validation in the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

ENV={ CURRENT | DEPENDENT | BACKGROUND } | LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The ENV operand is the default destination, indicating that the allocation is to take place entirely within the local region of the allocating procedure. If the allocation request came from a remote system, this operand is ignored. Otherwise, the ENV operand specifies the environment where the procedure is attached.

If ENV=CURRENT, or defaulted, the attached target procedure has the same relationship to the allocating procedure as it would have if a START command invokes it.

If ENV=DEPENDENT, the new process is attached in a dependent environment. The process has the same relationship to the allocating procedure as it would have if an &INTCMD START command invokes it.

If ENV=BACKGROUND, the new process is attached in the background APPC server region for the requesting user ID, as if the request was sourced from a remote system.

The LUNAME operand specifies the name of the network LU on which to allocate the remote transaction. When LUNAME is specified, the LINK parameter cannot be used. LUNAME is abbreviated to LU.

If LUNAME=*luname* is specified, then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization. Otherwise, the allocation fails. If *luname* resolves to the local LU, the conversation proceeds exactly as it would for a remote system, not as described for the ENV parameter. That is, it executes the remote end of the conversation in the appropriate APPC environment within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LU (where you want to allocate the remote transaction). If LINK is specified, then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK). The definition enables the link to be initialized by the request.

The DOMAIN operand enables identification of the target APPC system by domain name.

The ENV, LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Default: ENV=CURRENT

PARMS=(*parm1*,*parm2*,...,*parmn*) | VARS=*var* |
VARS=(*var1*, *var2*, ..., *varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Alternatively, the VARS operand is used to specify the allocation parameters:

VARS=*var* | VARS=(*var1*,*var2*,...,*varn*) |
VARS=*prefix [RANGE=(*start*,*end*)]**

Provides the list of any Program Initialization Parameters (PIP data) to be passed to the transaction processor in the remote end upon initialization of the conversation. Each token implicated by the VARS option is passed as a separate parameter in the PIP subfield list. If the remote conversation partner is an NCL procedure, PIP data is available when the remote procedure is invoked as &1,&2....&*n* in the usual manner.

Example: &APPC ATTACH_NOTIFY

```
&APPC ATTACH_NOTIFY PROC=MYPROC LINK=SOLSYD
…
&INTREAD TYPE=REQ VARS=NFYMSG
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

This request also sets the system variable &ZAPPCID, which returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation, if the NCL process is operating more than one conversation concurrently.

State Transition:

The attach is deemed to be performed from reset state and, following notification and successful completion, the conversation enters send state locally. When the remote program or procedure is attached to service the conversation, it is in receive state.

Notification Message Format:

When an ATTACH_NOTIFY request completes (with return code=0), and subsequent session attachment completes, either successfully or unsuccessfully, the following message is placed on the request queue of the internal environment for the NCL procedure:

```
N00101 NOTIFY: APPC EVENT: ALLOCATE RESOURCE: &zappcid
```

The value *&zappcid* is the conversation identifier returned by the ATTACH_NOTIFY request.

Notes:

- If no target system information is supplied, TCT must provide a default destination; otherwise, the request fails.
- Using ATTACH_NOTIFY has no significant advantages over ATTACH_SESSION, unless concurrent processing using other NCL resources is required. Notification is provided by a request sent to the internal environment of the NCL procedure, and is accessed by issuing an &INTREAD TYPE=REQUEST. No information is provided on the notification as to the success or otherwise of the allocation. An &APPC TEST is used to set the return code on the next operation attempted. If the allocation was unsuccessful, an allocation error is returned.
- The allocation request is canceled by issuing an &APPC DEALLOCATE TYPE=ABEND request for the conversation concerned. If the NCL process terminates, the request is automatically canceled.

Relationship to LU6.2 Verb Set:

&APPC ATTACH_NOTIFY has no exact equivalent in the LU6.2 verb options. The request is merely an asynchronous form of the LU6.2 verb MC_ALLOCATE with the RETURN_CONTROL(WHEN_SESSION_ ALLOCATED) option.

More information:

&APPC ATTACH_SESSION (see page 123)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC ATTACH_SESSION

ATTACH_SESSION allows the requester to specify the target procedure name to be attached to the conversation. The request is useful only when both ends of the conversation reside in APPC systems.

ATTACH_SESSION indicates a request to attach an NCL procedure to service the APPC conversation. This request is a more direct way of creating a conversation where both ends are to execute within an APPC system.

&APPC ATTACH_SESSION has the following format:

```
&APPC { ATTACH_SESSION | ATTACH }
        PROC=procname
      [ SERVER=servername
       [ SCOPE={ REGION | USER | SYSTEM } ] ]
      [ MODENAME=modename ]
      [ SYNC={ NONE | CONFIRM } ]
      [ WAIT=nn ]
      [ USERID=userid [ PASSWORD=password ] ]
      [ PROFILE=profile ]
      [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
        LUNAME=luname | LINK=linkname | DOMAIN=domain ]
      [ PARMS=(parm1,parm2,...,parmn) |
        VARS=var | VARS=(var1, var2, ..., varn ) |
        VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

PROC=*procname*

(Mandatory) Nominates the target procedure by name. The operand is used where the conversation is known to take place between two NCL procedures, and where the target procedure is either within the local system, or a remote APPC system, as specified by the LOCAL, LU, or LINK operands. This operand is used instead of the TRANSID operand to target an explicit procedure. However, bypassing the Transaction Control Table is often not desirable. The default security parameter for this type of allocation is SAME. The parameter indicates that a signed on user allocates the target procedure in a signed on region, while an unsecured region allocates the procedure in the background APPC region.

SERVER=*servername*

Specifies the logical name for this NCL process. The name must be up to 32 characters long and unique within the scope as determined by the SCOPE operand, or the request fails.

SCOPE={ REGION | USER | SYSTEM }

Provides the scope for the registration of *servername* (which must be unique within the scope indicated) and is valid only if the SERVER operand is present. Valid scopes are:

REGION

Indicates that *servername* is to be unique within the current session, as defined by a particular connection to your product region.

USER

Indicates that *servername* is to be unique across all sessions associated with the particular user ID

SYSTEM

Indicates that *servername* is to be unique within this product region.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the system transaction ATTACH TCT entry is used to select a default mode name in the usual manner. If present, *modename* must correspond to a valid one defined for the APPC link or the allocation fails.

SYNC= { NONE | CONFIRM }

Nominates the synchronization level for the conversation. If omitted, the system ATTACH TCT entry is used to nominate the default SYNC level. If SYNC level of NONE is specified, or defaulted, then the &APPC CONFIRM and CONFIRMED requests, the &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM request, and the DEALLOCATE TYPE=CONFIRM request cannot be used.

WAIT=*nn*

Specifies the time, in seconds (for example, 10), or seconds and hundredths (for example, 1.25), for which the procedure is prepared to wait for a session. If not successful before this interval expires, the request is canceled. An unsuccessful return code results (&RETCODE is set to 4, and &ZFDBK is set to 0).

USERID=*userid* [PASSWORD=*password*]

Specifies the target user ID and, optionally, the password, under which the transaction starts on the remote system.

The ATTACH system transaction uses SECURITY=SAME processing. Therefore, APPC includes the supplied user ID and password in the attach header for validation in the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

ENV={ CURRENT | DEPENDENT | BACKGROUND } | LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The ENV operand is the default destination, indicating that the allocation is to take place entirely within the local region of the allocating procedure. If the allocation request came from a remote system, this operand is ignored. Otherwise, the ENV operand specifies the environment where the procedure is attached.

If ENV=CURRENT, or defaulted, the attached target procedure has the same relationship to the allocating procedure as it would have if a START command invokes it.

If ENV=DEPENDENT, the new process is attached in a dependent environment. The process has the same relationship to the allocating procedure as it would have if an &INTCMD START command invokes it.

If ENV=BACKGROUND, the new process is attached in the background APPC server region for the requesting user ID, as if the request was sourced from a remote system.

The LUNAME operand specifies the name of the network LU on which to allocate the remote transaction. When LUNAME is specified, the LINK parameter cannot be used. LUNAME is abbreviated to LU.

If LUNAME=*luname* is specified, then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization. Otherwise, the allocation fails. If *luname* resolves to the local LU, the conversation proceeds exactly as it would for a remote system, not as described for the ENV parameter. That is, it executes the remote end of the conversation in the appropriate APPC environment within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LU (where you want to allocate the remote transaction). If LINK is specified, then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK). The definition enables the link to be initialized by the request.

The DOMAIN operand enables identification of the target APPC system by domain name.

The ENV, LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Default: ENV=CURRENT

PARMS=(*parm1*,*parm2*,...,*parmn*) | VARS=*var* |
VARS=(*var1*, *var2*, ..., *varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Alternatively, the VARS operand is used to specify the allocation parameters:

VARS=*var* | VARS=(*var1*,*var2*,...,*varn*) |
VARS=*prefix [RANGE=(*start*,*end*)]**

Provides the list of any Program Initialization Parameters (PIP data) to be passed to the transaction processor in the remote end upon initialization of the conversation. Each token implicated by the VARS option is passed as a separate parameter in the PIP subfield list. If the remote conversation partner is an NCL procedure, PIP data is available when the remote procedure is invoked as &1,&2....&*n* in the usual manner.

Examples: &APPC ATTACH_SESSION

```
&APPC ATTACH_SESSION PROC=MYPROC LINK=SOLMELB
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

This request also sets the system variable &ZAPPCID, which returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation, if the NCL process is operating more than one conversation concurrently.

State Transition:

The request is deemed to be performed from the reset state. Following successful completion of this request, the conversation enters send state locally. When the remote program or procedure is attached to service the conversation, it is in receive state.

Notes:

- If no target system information is supplied, TCT must provide a default destination; otherwise, the request fails.
- The ATTACH_SESSION request is the simplest allocation option to use. The request is generally the most resource efficient, as resources are not allocated until a session is available for the conversation. While the request is queued pending session availability, the procedure is suspended. Upon completion, the procedure can continue processing based purely upon the success or otherwise of the allocation request.

Relationship to LU6.2 Verb Set:

&APPC ATTACH_SESSION is equivalent to the LU6.2 verb MC_ALLOCATE with the RETURN_CONTROL (WHEN_SESSION_ALLOCATED) option.

The &APPC PROC parameter is equivalent to the LU6.2 verb MC_ALLOCATE TPN(*tpn*) option. The parameter specifies the name of the procedure to attach on the remote APPC system to service the transaction.

The &APPC ENV parameter is equivalent to the LU6.2 verb MC_ALLOCATE LU_NAME(OWN) option.

The &APPC LUNAME=*luname* parameter is equivalent to the LU6.2 verb MC_ALLOCATE LU_NAME(OTHER(*luname*)) option.

The &APPC LINK=*linkname* parameter is equivalent to the LU6.2 verb MC_ALLOCATE LU_NAME(OTHER(*luname*)) option. *linkname* is considered to be the locally known name for the actual network *luname* concerned.

The &APPC PARMS or VARS=(*var1*,*var2*,...,*varn*) parameter is equivalent to the LU6.2 verb MC_ATTACH PIP(YES(*var1*,*var2*,...,*varn*)) option.

The &APPC MODENAME and SYNC parameters are equivalent to the LU6.2 verb MC_ALLOCATE MODE_NAME and SYNC_LEVEL parameters respectively.

The &APPC USERID, PASSWORD and PROFILE operands are equivalent to the LU6.2 verb MC_ALLOCATE SECURITY parameter. NCL conversation security options can also be set through the system TCT entry (see the DEFTRANS command).

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC CONFIRM

&APPC CONFIRM sends a request for confirmation to the conversation partner and waits for the reply.

CONFIRM indicates this is a conversation confirmation request.

&APPC CONFIRM has the following format:

```
&APPC CONFIRM [ ID=id ]
```

Operand:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted the current (last referenced, or only) conversation is assumed.

Examples: &APPC CONFIRM

```
&APPC CONFIRM
&APPC CONFIRM ID=999
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The CONFIRM request can only be issued from send state. When the remote program responds with the CONFIRMED request, the return code is set to 0 and no local state change occurs. If the remote program issues SEND_ERROR the return code is set to 8 and the local state is changed to receive state.

Notes:

- This request is useful in synchronizing processing in both ends of the conversation, and provides a means to request verification of the receipt of conversation data.
- If following a successful confirmation, the next conversation action is to receive data, consider using the PREPARE_TO_RECEIVE TYPE= CONFIRM or TYPE=FLUSH request, for improved communication efficiency.
- If following a successful confirmation, the next conversation action is to deallocate the conversation, consider using the DEALLOCATE TYPE=CONFIRM request for improved communication efficiency.

Relationship to LU6.2 Verb Set:

&APPC CONFIRM is equivalent to the LU6.2 verb MC_CONFIRM.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC CONFIRMED

&APPC CONFIRMED sends a reply to the conversation partner's request for a reply (&APPC CONFIRM).

CONFIRMED indicates this is a conversation confirmation reply.

&APPC CONFIRMED has the following format:

```
&APPC CONFIRMED [ ID=id ]
```

Operand:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted the current (last referenced, or only) conversation is assumed.

Examples: &APPC CONFIRMED

```
&APPC CONFIRMED
&APPC CONFIRMED ID=999
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The CONFIRMED request can only be issued from one of the confirm states, namely, confirm, confirm_send, or confirm_deallocate.

Confirm state is entered as a result of the remote program sending the CONFIRM request, and following the CONFIRMED reply the local program re-enters receive state.

Confirm_send state is entered as a result of the remote program sending the PREPARE_TO_RECEIVE TYPE=CONFIRM request, and following the CONFIRMED reply the local program enters send state.

Confirm_deallocate state is entered as a result of the remote program sending the DEALLOCATE TYPE=CONFIRM request, and following the CONFIRMED reply the local program enters deallocate state.

Notes:

- This request is useful when synchronizing processing in both ends of the conversation, and provides a means to verify the receipt of conversation data.
- To send a negative acknowledgement, use the SEND_ERROR request.

Relationship to LU6.2 Verb Set:

&APPC CONFIRMED is equivalent to the LU6.2 verb MC_CONFIRMED.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC CONNECT_DELAYED

CONNECT_DELAYED allows the requester to start a conversation with an existing NCL process in either the local or a connected APPC system.

CONNECT_DELAYED indicates that this is a request to connect to an active NCL process. Unless the target system cannot be reached, this request will normally be accepted. However, once accepted, the connection request remains pending a transfer of information to the target process. After such a transfer, the conversation might subsequently fail due to communication failures, or the inability to locate the desired active process.

&APPC CONNECT_DELAYED has the following format:

```
&APPC CONNECT_DELAYED
    { NCLID=nclid | SERVER=servername }
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]
```

Operands:

NCLID=*nclid*

Identifies the target process for the APPC connection request, by specifying its 6-digit NCL ID. The NCL ID referenced is in the local system or in a connected APPC system.

SERVER=*servername*

Specifies the logical name for this NCL process. It must be 1 to 32 characters long and unique within the current scope.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the system CONNECT TCT entry is used to select a default mode name in the usual manner. If present, *modename* must correspond to a valid one defined for the APPC link or the allocation will fail.

SYNC= { NONE | CONFIRM }

Nominates the synchronization level for the conversation. If omitted, the system CONNECT TCT entry is used to nominate the default SYNC level. If SYNC level of NONE is specified, or defaulted, then the &APPC CONFIRM and CONFIRMED requests, the &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM request, and the DEALLOCATE TYPE=CONFIRM request cannot be used.

LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The LUNAME operand nominates the network LU name where the process to be connected is executing. When LUNAME is specified, the LINK parameter cannot be used. If *LUNAME=luname* is specified then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization, otherwise the allocation fails.

If *luname* resolves to the local LUNAME, the process being connected must be within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LUNAME. When LINK is specified then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LUNAME name (that is, not LUMASK) to allow it to be initialized by the allocate request.

The DOMAIN operand allows the target APPC system to be identified by domain name.

The ENV LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Example: &APPC CONNECT_DELAYED

```
&APPC CONNECT_DELAYED NCLID=&TARGETID
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

The system variable &ZAPPCID is also set by this request and returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation where the NCL process is operating more than one conversation concurrently.

State Transition:

The allocation is deemed to be performed from reset state and, following successful completion of this request, the conversation enters send state locally. When the remote program or procedure is attached to service the conversation, it is in receive state.

Notes:

- Following a successful connect request, the &ZAPPCID system variable contains the new conversation identifier. The conversation is allocated in send state as usual, and is subsequently operated in the same manner as any other allocated conversation. Similarly, once the connection request is accepted in the target process, the APPC conversation is operated like any other attached conversation in that process.
- If confirmation of the connection to the target process is required, use the CONFIRM protocol following the connect request.

More information:

&APPC ALLOCATE_DELAYED (see page 82)
&APPC ATTACH_DELAYED (see page 105)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC CONNECT_IMMEDIATE

&APPC CONNECT_IMMEDIATE allows the requester to start a conversation with an existing NCL process in either the local or a connected APPC system.

CONNECT_IMMEDIATE indicates that this is a request to connect to an active NCL process. Unless the target system cannot be reached, this request will normally be accepted. However, once accepted, the connection request remains pending a transfer of information to the target process. After such a transfer, the conversation might subsequently fail due to communication failures, or the inability to locate the desired active process.

This verb has the following format:

```
&APPC CONNECT_IMMEDIATE
    { NCLID=nclid | SERVER=servername }
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]
```

Operands:

NCLID=*nclid*

(Mandatory) Identifies the target process for the APPC connection request by specifying its 6-digit NCL ID. The NCL ID referenced is in the local system or in a connected APPC system.

SERVER=*servername*

Provides an alternative way to identify the target process for the APPC connection request by specifying its registered server name.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the system CONNECT TCT entry is used to select a default mode name. If present, *modename* must correspond to a valid one defined for the APPC link or the allocation will fail.

SYNC= { NONE | CONFIRM }

Nominates the synchronization level for the conversation. If omitted, the system CONNECT TCT entry is used to nominate the default SYNC level. If SYNC level of NONE is specified, or defaulted, then the &APPC CONFIRM and CONFIRMED requests, the &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM request, and the DEALLOCATE TYPE=CONFIRM request cannot be used.

LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The LUNAME operand nominates the network LU name where the process to be connected is executing. When LUNAME is specified, the LINK parameter cannot be used.

If LUNAME=*luname* is specified, the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization, otherwise the allocation fails.

If *luname* resolves to the local LUNAME, the process being connected must be within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LUNAME. When LINK is specified then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK) to allow it to be initialized by the allocate request.

The DOMAIN operand allows the target APPC system to be identified by domain name.

The ENV LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Example: &APPC CONNECT_IMMEDIATE

```
&APPC CONNECT_IMMEDIATE NCLID=&TARGETID
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

The system variable &ZAPPCID is also set by this request and returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation where the NCL process is operating more than one conversation concurrently.

State Transition:

The allocation is deemed to be performed from reset state and following successful completion the conversation enters send state locally. When the remote program or procedure is attached to service the conversation it is in receive state.

Notes:

- Following a successful connect request, the &ZAPPCID system variable contains the new conversation identifier. The conversation is allocated in send state as usual, and is subsequently operated in the same manner as any other allocated conversation. Similarly, once the connection request is accepted in the target process, the APPC conversation is operated like any other attached conversation in that process.
- If confirmation of the connection to the target process is required, use the CONFIRM protocol following the connect request.

More information:

&APPC ALLOCATE_IMMEDIATE (see page 88)
&APPC ATTACH_IMMEDIATE (see page 111)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC CONNECT_NOTIFY

&APPC CONNECT_NOTIFY allows the requester to start a conversation with an existing NCL process in either the local or a connected APPC system.

CONNECT_NOTIFY indicates that this is a request to connect to an active NCL process. Unless the target system cannot be reached, this request will normally be accepted. However, once accepted, the connection request remains pending a transfer of information to the target process. After such a transfer, the conversation might subsequently fail due to communication failures, or the inability to locate the desired active process.

This verb has the following format:

```
&APPC CONNECT_NOTIFY
    { NCLID=nclid | SERVER=servername }
    [ MODENAME=modename ]
    [ SYNC={ NONE | CONFIRM } ]
    [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]
```

Operands:

NCLID=*nclid*

(Mandatory) Identifies the target process for the APPC connection request by specifying its 6-digit NCL ID. The NCL ID referenced is in the local system or in a connected APPC system.

SERVER=*servername*

Provides an alternative way to identify the target process for the APPC connection request by specifying its registered server name.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the system CONNECT TCT entry is used to select a default mode name. If present, *modename* must correspond to a valid one defined for the APPC link or the allocation will fail.

SYNC= { NONE | CONFIRM }

Nominates the synchronization level for the conversation. If omitted, the system CONNECT TCT entry is used to nominate the default SYNC level. If SYNC level of NONE is specified, or defaulted, then the &APPC CONFIRM and CONFIRMED requests, the &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM request, and the DEALLOCATE TYPE=CONFIRM request cannot be used.

LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The LUNAME operand nominates the network LU name where the process to be connected is executing. When LUNAME is specified, the LINK parameter cannot be used.

If LUNAME=*luname* is specified then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization, otherwise the allocation fails. If *luname* resolves to the local LUNAME, the process being connected must be within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LUNAME. When LINK is specified then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK) to allow it to be initialized by the allocate request.

The DOMAIN operand allows the target APPC system to be identified by domain name.

The ENV LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Examples: &APPC CONNECT_NOTIFY

```
&APPC CONNECT_NOTIFY NCLID=&TARGETID
 .
 .
 .
&INTREAD TYPE=REQ VARS=NFYMSG
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

The system variable &ZAPPCID is also set by this request and returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation where the NCL process is operating more than one conversation concurrently.

State Transition:

The allocation is deemed to be performed from reset state and following notification and successful completion the conversation enters send state locally. When the remote program or procedure is attached to service the conversation it is in receive state.

Notes:

- Following a successful connect request, the &ZAPPCID system variable contains the new conversation identifier. The conversation is allocated in send state as usual, and is subsequently operated in the same manner as any other allocated conversation. Similarly, once the connection request is accepted in the target process, the APPC conversation is operated like any other attached conversation in that process.
- If confirmation of the connection to the target process is required, use the CONFIRM protocol following the connect request.

More information:

&APPC ALLOCATE_NOTIFY (see page 93)
&APPC ATTACH_NOTIFY (see page 117)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC CONNECT_SESSION

&APPC CONNECT_SESSION allows the requester to start a conversation with an existing NCL process in either the local or a connected APPC system.

CONNECT_SESSION indicates that this is a request to connect to an active NCL process. Unless the target system cannot be reached, this request will normally be accepted. However, once accepted, the connection request remains pending a transfer of information to the target process. After such a transfer, the conversation might subsequently fail due to communication failures, or the inability to locate the desired active process.

This verb has the following format:

```
&APPC { CONNECT_SESSION | CONNECT  }
      { NCLID=nclid | SERVER=servername }
      [ MODENAME=modename ]
      [ SYNC={ NONE | CONFIRM } ]
      [ WAIT=nn ]
      [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]
```

Operands:

NCLID=*nclid*

Identifies the target process for the APPC connection request by specifying its 6-digit NCL ID. The NCL ID referenced is in the local system or in a connected APPC system.

SERVER=*servername*

Provides an alternative way to identify the target process for the APPC connection request by specifying its registered server name.

MODENAME=*modename*

Nominates the session mode name for the conversation. If omitted, the system CONNECT TCT entry or OSCT is used to select a default mode name. If present, *modename* must correspond to a valid one defined for the APPC link or the allocation will fail.

SYNC= { NONE | CONFIRM }

Nominates the synchronization level for the conversation. If omitted, the system CONNECT TCT entry is used to nominate the default SYNC level. If SYNC level of NONE is specified, or defaulted, then the &APPC CONFIRM and CONFIRMED requests, the &APPC PREPARE_TO_RECEIVE TYPE=CONFIRM request, and the DEALLOCATE TYPE=CONFIRM request cannot be used.

WAIT=*nn*

Specifies the time, in seconds (for example, 10), or seconds and hundredths (for example, 1.25), for which the procedure is prepared to wait for a session. If not successful before this interval expires, the connect request is canceled, and an unsuccessful return code results (&RETCODE is set to 4, &ZFDBK is set to 0).

LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

The LUNAME operand nominates the network LU name where the process to be connected is executing. When LUNAME is specified, the LINK parameter cannot be used.

If LUNAME=*luname* is specified then the target system must be connected to the local system with an APPC link, or must have an appropriate entry in the Dynamic Link Table to permit link initialization, otherwise the allocation will fail. If *luname* resolves to the local LUNAME, the process being connected must be within the local system.

The LINK operand nominates the name of the APPC link connecting the local system with the target LUNAME. When LINK is specified then the LUNAME parameter cannot be used. The link must either be active, or be defined in the Dynamic Link Table with an explicit link and LU name (that is, not LUMASK) to allow it to be initialized by the allocate request. The DOMAIN operand allows the target APPC system to be identified by domain name.

The ENV LUNAME, LINK, and DOMAIN operands are mutually exclusive.

Example: &APPC CONNECT_SESSION

```
&APPC CONNECT_SESSION NCLID=&TARGETID LINK=SOLSYD
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

The system variable &ZAPPCID is also set by this request and returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation where the NCL process is operating more than one conversation concurrently.

State Transition:

The allocation is deemed to be performed from reset state and following successful completion the conversation enters send state locally. When the remote program or procedure is attached to service the conversation it is in receive state.

Notes:

- Following a successful connect request, the &ZAPPCID system variable contains the new conversation identifier. The conversation is allocated in send state as usual, and is subsequently operated in the same manner as any other allocated conversation. Similarly, once the connection request is accepted in the target process, the APPC conversation is operated like any other attached conversation in that process.
- If confirmation of the connection to the target process is required, use the CONFIRM protocol following the connect request.

More information:

&APPC ALLOCATE_SESSION (see page 99)
&APPC ATTACH_SESSION (see page 123)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC DEALLOCATE

&APPC DEALLOCATE requests conversation termination and deallocation of its resources.

&APPC DEALLOCATE is used when an NCL procedure detects an error situation. The process can abnormally terminate the conversation with the remote application. In this case, it can provide a text string to be included in the Error Log GDS variable.

This verb has the following format:

```
&APPC  DEALLOCATE
     [ TYPE={ SYNC | FLUSH | CONFIRM | ABEND | LOCAL } ]
     [ ID=id ]
     [ LOG=msg ]
```

Operands:

TYPE={ SYNC | FLUSH | CONFIRM | ABEND | LOCAL }

Specifies the deallocate option as follows:

- TYPE=SYNC is issued from send state only, and is the default. If the conversation sync level is CONFIRM it is equivalent to the DEALLOCATE TYPE=CONFIRM option, otherwise a DEALLOCATE TYPE=FLUSH is assumed.
- TYPE=FLUSH is issued from send state only, and results in all data being flushed, forcing its transmission to the conversation partner before unconditional deallocation occurs (see Notes for this option).
- TYPE=CONFIRM is issued from send state only, and results in all data being flushed and a confirmation being requested before deallocation occurs (see Notes for this option).
- TYPE=ABEND is issued from the send, defer, receive, or any of the confirm states to abnormally terminate the conversation.
- TYPE=LOCAL is issued from the deallocate state only to terminate the conversation and free any locally held resources after a remote deallocation has occurred.

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted, the current (last referenced, or only) conversation is assumed.

LOG=*msg*

Allowed for TYPE=ABEND only. It must be the last operand specified for the verb. All data following the LOG= operand is placed unchanged into the message area of the Error Log GDS variable sent to the remote application with the abnormal termination indication.

If the remote application is an NCL procedure, it can access this text, after receiving an error return code in the &ZAPPCELM system variable.

Examples: &APPC DEALLOCATE

```
&APPC DEALLOCATE TYPE=FLUSH

&APPC DEALLOCATE TYPE=LOCAL

&APPC DEALLOCATE TYPE=ABEND LOG=&ERRMSG
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The DEALLOCATE request is issued in certain states according to the TYPE option used. Following successful completion, the conversation enters reset state, that is, it becomes inoperable.

Notes:

- A DEALLOCATE TYPE=LOCAL request is the only conversation action permissible once the deallocate state is entered. This state follows receipt of a DEALLOCATE TYPE=FLUSH or TYPE=ABEND from the remote conversation partner.
- Normal deallocation (that is, of TYPE=SYNC, FLUSH or CONFIRM) can fail if an error is received from the remote program before the deallocation is processed.
- A DEALLOCATE TYPE=FLUSH (or TYPE=SYNC where the conversation was allocated with SYNC_LEVEL=NONE), is not subject to any confirmation from the remote partner. Hence, as it is issued from send state, if no intervening error conditions have arisen it is always successful.
- When a DEALLOCATE TYPE=CONFIRM (or TYPE=SYNC where the conversation was allocated with SYNC_LEVEL=CONFIRM) is issued, the deallocation is subject to a confirmation from the remote partner. Once confirmation is received locally (that is, the remote program issues CONFIRMED), the deallocation completes successfully. However, if the remote program instead issues a SEND_ERROR the conversation is not terminated and locally enters receive state.
- If a DEALLOCATE ABEND is issued, or a procedure terminates before the conversation has been allocated to a session, no data will flow.

Relationship to LU6.2 Verb Set:

&APPC DEALLOCATE is equivalent to the LU6.2 verb MC_DEALLOCATE.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC DEREGISTER

&APPC DEREGISTER allows any executing NCL server process to DEREGISTER its server name to prevent further client access.

DEREGISTER indicates that this is a request to DEREGISTER the NCL server process, so that access to clients is no longer available. Any pending conversations that have not been notified to the server are immediately rejected, but active conversations are unaffected.

This verb has the following format:

```
&APPC  DEREGISTER
```

Return Codes:

The return codes are as follows:

0

DEREGISTER accepted

16

DEREGISTER rejected (not registered)

Note:

When &APPC DEREGISTER is issued, any pending conversations are abended with the appropriate RETRY option.

More information:

&APPC REGISTER (see page 162)
&APPC SET_SERVER_MODE (see page 184)

&APPC FLUSH

&APPC FLUSH flushes any locally buffered information and forces its transmission to the remote conversation partner.

FLUSH indicates a conversation flush request.

This verb has the following format:

```
&APPC FLUSH [ ID-id ]
```

Operand:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted the current (last referenced, or only) conversation is assumed.

Example: &APPC FLUSH

```
&APPC SEND_DATA VARS=DATA
&APPC FLUSH
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The FLUSH request can only be issued from send state and no state changes occur.

Note:

Plan carefully before you use this request. Usually, the system manages the requirement to flush send buffers based on session parameters, conversation parameters, and the current state of processing. As data accumulates for a conversation or if some form of reply is required from the remote end, all queued output is scheduled automatically for transmission in optimum message units. Issuing the FLUSH request unnecessarily can reduce such efficiencies.

Relationship to LU6.2 Verb Set:

&APPC FLUSH is equivalent to the LU6.2 verb MC_FLUSH.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC PREPARE_TO_RECEIVE

&APPC PREPARE_TO_RECEIVE changes processing from sending to receiving data.

PREPARE_TO_RECEIVE indicates that the conversation has terminated send operations and is next expecting to receive data.

This verb has the following format:

```
&APPC { PREPARE_TO_RECEIVE | PREPARE }
      [ TYPE={ SYNC | FLUSH | CONFIRM }  ]
      [ ID=id ]
```

Operands:

TYPE={ SYNC | FLUSH | CONFIRM }

Specifies the level of confirmation required for any data previously sent to the remote conversation partner as follows:

- TYPE=SYNC is the default and, if the conversation sync level is CONFIRM, then it is equivalent to the PREPARE_TO_RECEIVE TYPE=CONFIRM option; otherwise a PREPARE_TO_RECEIVE TYPE=FLUSH is assumed.
- TYPE=FLUSH results in all data being flushed, forcing its transmission to the conversation partner, but no confirmation reply is required (see Notes for this option).
- TYPE=CONFIRM results in all data being flushed and a confirmation being requested (see Notes for this option).

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted, the current (last referenced, or only) conversation is assumed.

Examples: &APPC PREPARE_TO_RECEIVE

```
&APPC PREPARE_TO_RECEIVE TYPE=FLUSH
```

```
&APPC PREPARE_TO_RECEIVE TYPE=CONFIRM
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The PREPARE_TO_RECEIVE request is issued from send state only, and results in receive state being entered.

Notes:

- In many cases the use of PREPARE_TO_RECEIVE is eliminated without loss of function. Since it implies that the next action on the conversation is to receive data, the eventual receipt of such data from the remote conversation partner is interpreted as implicit confirmation of any previous data sent, thus eliminating the need for the remote program to issue an explicit CONFIRMED request before sending its data.
- Further, if the next action locally is to receive, the PREPARE_TO_RECEIVE (TYPE=FLUSH) request itself is implied by simply issuing a RECEIVE request following the completion of all SEND requests.

Relationship to LU6.2 Verb Set:

&APPC PREPARE_TO_RECEIVE is equivalent to the LU6.2 verb MC_PREPARE_TO_RECEIVE. The LU6.2 verb has a LOCKS parameter which is not supported by &APPC.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC RECEIVE_AND_WAIT

This option of the &APPC verb is used to receive any data sent by the conversation partner, or await its arrival if none is currently available. The short form is &APPC RECEIVE.

RECEIVE_AND_WAIT indicates a synchronous conversation receive request. If data is available, it is returned in the variables indicated. If no data is immediately available, the procedure is suspended until data arrives, or some error condition arises, at which time the request completes. A process can identify a class of conversation that satisfies a receive request. Only one event will satisfy the receive; any other events remain pending.

This verb has the following format:

```
&APPC{ RECEIVE_AND_WAIT | RECEIVE }
     [ ID={ ANY | CLIENTS | SERVERS | id } ]
     [ WAIT=nn ]
     [ MDO=mdoname [ MAP=mapname ] |
       VARS=var | VARS=(var1, var2, ..., varn ) |
       VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

ID={ ANY | CLIENTS | SERVERS | *id* }

If ID=ANY, no state changes occur. Any conversation, either client or server, that is in receive state satisfies the request.

If ID=CLIENTS, no state changes occur. Any client conversation that is in receive state satisfies the request. If operating in automatic connection accept mode, a new conversation can provide the data that satisfies such a receive.

If ID=SERVERS, no state changes occur. Any server conversation that is in the receive state satisfies the request.

If ID=*id* (or allowed to default to the current conversation), then this conversation must be in receive state, or it is automatically placed in receive state, if possible, before the request is accepted.

WAIT=*nnn*

Specifies the time, in seconds (for example, WAIT=10), or seconds and hundredths (for example, WAIT=1.25), for which the procedure is prepared to wait for a receive to be satisfied.

If not successful before this interval expires, the receive request is canceled with an unsuccessful return code (that is, &RETCODE is set to 4, and &ZFDBK is set to 4).

MDO=*mdoname* [MAP=*mapname*] | VARS=*var* |
VARS=(*var*1,*var*2,...,*varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

One of the options must be specified. The operand specifies how the incoming data is formatted.

If MDO= is used, the data is formatted into an MDO with the name *mdoname*. If the incoming data is mapped (that is, a map name is sent with the data), and MAP= is not specified, then the received map name is used to connect to Mapping Services Mapping Support. If the incoming data is not mapped, then it is the responsibility of the requester to connect to Mapping Services Mapping Support, if necessary, using the MAP operand.

The VARS= operand is used to provide the list of NCL variables that contains the data received on the conversation (the usual NCL VARS definitions apply). If the incoming data is not mapped, or the map name is other than $NCL, then the data stream received is segmented into the variables nominated. If the data is mapped and the map name is $NCL, then each variable is reconstructed as it was on the send request in the conversation partner. Unused variables are set to null.

Examples: &APPC RECEIVE_AND_WAIT

```
&APPC RECEIVE_AND_WAIT VARS=DATA

&APPC RECEIVE_AND_WAIT ID=CLIENTS VARS=DATA
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The RECEIVE_AND_WAIT request is issued from either send or receive state, and the conversation enters or remains in receive state while the WHAT_RECEIVED indicator (&ZAPPCWR) contains DATA (that is, DATA_COMPLETE, DATA_TRUNCATED, or DATA_INCOMPLETE).

If the WHAT_RECEIVED indicator contains SEND (where the remote conversation partner has issued PREPARE_TO_RECEIVE) the local state changes to send.

If the WHAT_RECEIVED indicator contains CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE, then the local procedure should issue the CONFIRMED request following which the state changes to receive, send, or deallocate respectively.

Notes:

- When RECEIVE_AND_WAIT is issued from send state, an implied PREPARE_TO_RECEIVE TYPE=FLUSH is issued. This is the most efficient way of changing the direction of conversation flow.
- Following the interval expiry for a RECEIVE_AND_WAIT request that specified a WAIT interval, the &RETCODE is set to 4, and &ZFDBK to 0 when no data is available to satisfy the request.
- If the remote conversation partner deallocates the conversation normally, &RETCODE is set to 4 and &ZFDBK to 4.
- Server conversations are those that have been initiated locally (by issuing &APPC ALLOCATE, &APPC ATTACH, or &APPC CONNECT) while client conversations are those that have been initiated remotely. The system variable &ZAPPCCSI indicates whether a conversation is a CLIENT or SERVER conversation.
- When using the CLIENT, SERVER, or ANY operands, the process must be in automatic ACCEPT connect mode, otherwise the request is rejected with a state check. The verbs &APPC REGISTER, &APPC DEREGISTER, and &APPC SET_SERVER_MODE is used to alter the connect mode of the process.

Relationship to LU6.2 Verb Set:

&APPC RECEIVE_AND_WAIT is equivalent to the LU6.2 verb MC_RECEIVE_AND_WAIT.

More information:

&APPC REGISTER (see page 162)
&APPC DEREGISTER (see page 149)
&APPC SET_SERVER_MODE (see page 184)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC RECEIVE_IMMEDIATE

&APPC RECEIVE_IMMEDIATE receives data sent from the remote conversation partner, if any is queued; otherwise returns to the procedure for processing to continue.

RECEIVE_IMMEDIATE indicates that this is an immediate conversation receive request. If data is available, it is returned on this request; otherwise the operation completes unsuccessfully.

This verb has the following format:

```
&APPC RECEIVE_IMMEDIATE
    [ ID=id ]
    [ MDO=mdoname [ MAP=mapname ]| VARS=var |
      VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted, the current (last referenced, or only) conversation is assumed.

MDO=*mdoname* [MAP=*mapname*] | VARS=*var* |
VARS=(*var*1,*var*2,...,*varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Mandatory-one of the options must be specified. It indicates how the incoming data should be formatted.

If MDO= is used, the data is formatted into an MDO with the name *mdoname*. If the incoming data is mapped (that is, a map name is sent with the data), and MAP= is not specified, then the received map name is used to connect to Mapping Services Mapping Support. If the incoming data is not mapped, then it is the requester's responsibility to connect to Mapping Services Mapping Support, if required, using the MAP operand.

The VARS= operand is used to provide the list of NCL variables that will contain the data received on the conversation (the usual NCL VARS definitions apply). If the incoming data is not mapped, or the map name is other than $NCL, then the data stream received is segmented into the variables nominated. If the data is mapped and the map name is $NCL, then each variable is reconstructed as it was on the send request in the conversation partner. Unused variables are set to null.

Example: &APPC RECEIVE_IMMEDIATE

```
&APPC RECEIVE_IMMEDIATE VARS=DATA
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The RECEIVE_IMMEDIATE request is issued from receive state only and the conversation remains in receive state while the WHAT_RECEIVED indicator (&ZAPPCWR) contains DATA (that is, DATA_COMPLETE, DATA_TRUNCATED or DATA_INCOMPLETE).

If the WHAT_RECEIVED indicator contains SEND (where the remote conversation partner has issued PREPARE_TO_RECEIVE) the local state changes to send.

If the WHAT_RECEIVED indicator contains CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE then the local procedure should issue the CONFIRMED request following which the state changes to receive, send, or deallocate respectively.

Notes:

- Following a RECEIVE_IMMEDIATE request, the &RETCODE is set to 4, and &ZFDBK to 0 when no data is available to satisfy the request.
- If the remote conversation partner deallocates the conversation normally, the &RETCODE is set to 4, and &ZFDBK to 4.

Relationship to LU6.2 Verb Set:

&APPC RECEIVE_IMMEDIATE is equivalent to the LU6.2 verb MC_RECEIVE_IMMEDIATE.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC RECEIVE_NOTIFY

&APPC RECEIVE_NOTIFY requests that a notification be sent to the procedure when data is available to be received (for example, by a RECEIVE_IMMEDIATE request).

RECEIVE_NOTIFY indicates this is an asynchronous conversation receive request. If the request is accepted, the procedure continues execution and is notified of data arrival, or other completion condition, by a notification event being queued to the procedure's internal environment (see below for more details on the form of notification). No data is ever returned by this request. When the notify event is received, the RECEIVE_IMMEDIATE or RECEIVE_AND_WAIT requests is used to return any data that arrived.

This verb has the following format:

```
&APPC RECEIVE_NOTIFY
    [ ID={ ANY | CLIENTS | SERVERS | id } ]
```

Operand:

ID={ ANY | CLIENTS | SERVERS | *id* }

If ID=ANY is specified, no state changes occur, but the receive request is satisfied by any conversation, either client or server, that is in receive state.

If ID=CLIENTS is specified, no state changes occur, but the receive request is satisfied by any client conversation that is in receive state. If operating in automatic connection accept mode, a new conversation can provide the data that satisfies such a receive.

If ID=SERVERS is specified, no state changes occur, but the receive request is satisfied by any server conversation that is in the receive state.

ID=id Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted, the current (last referenced, or only) conversation is assumed.

Example: &APPC RECEIVE_NOTIFY

```
&APPC RECEIVE_NOTIFY
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The RECEIVE_NOTIFY request is issued from either send or receive state and the conversation enters receive state while the WHAT_RECEIVED indicator (&ZAPPCWR) contains DATA (that is, DATA_COMPLETE, DATA_TRUNCATED or DATA_INCOMPLETE).

If the WHAT_RECEIVED indicator contains SEND (where the remote conversation partner has issued PREPARE_TO_RECEIVE) the local state changes to send.

If the WHAT_RECEIVED indicator contains CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE then the local procedure should issue the CONFIRMED request following which the state changes to receive, send, or deallocate respectively.

Notification Message Format:

When a RECEIVE_NOTIFY request is satisfied, either successfully or unsuccessfully, the following message is placed on the request queue of the internal environment for the NCL procedure:

`N00101 NOTIFY: APPC EVENT: RECEIVE RESOURCE: &zappcid`

where the value *&zappcid* is the conversation identifier returned by the original ALLOCATE request, or set when this procedure was attached as the remote end of the conversation.

Notes:

- When RECEIVE_NOTIFY is issued from send state, an implied PREPARE_TO_RECEIVE TYPE=FLUSH is issued. This is the most efficient way of changing the direction of conversation flow.
- If the remote conversation partner deallocates the conversation normally, the &RETCODE is set to 4, and &ZFDBK to 4.

Relationship to LU6.2 Verb Set:

&APPC RECEIVE_NOTIFY is equivalent to the LU6.2 verb MC_POST_ON_RECEIPT.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC REGISTER

&APPC REGISTER allows any executing NCL process to attempt to register itself with a server name, and optionally provides its conversation options for client/server processing.

This verb has the following format:

```
&APPC REGISTER
      SERVER=servername
    [ SCOPE={ REGION | USER | SYSTEM } ]
    [ CONNECT={ ACCEPT | NOTIFY | REJECT } ]
    [ RETRY={ YES | NO } ]
    [ CONVLIM={ 100 | nnn } ]
```

Operands:

SERVER=*servername*

(Mandatory) Specifies the logical name to register for the NCL process to be recognized as a server. It may be up to 32 characters long and unique within the scope as determined by the SCOPE operand, or the registration is rejected. If the registration is accepted, the other operands determine the server condition for accepting client connections.

SCOPE={ REGION | USER |SYSTEM }

Provides the scope for the registration of *servername* (which must be unique within the scope indicated). Valid scopes are:

REGION

Indicates that *servername* is to be unique within the current session, as defined by a particular connection to your product region.

USER

Indicates that *servername* is to be unique across all sessions associated with the particular user ID

SYSTEM

Indicates that *servername* is to be unique within this product region.

CONNECT={ ACCEPT | NOTIFY | REJECT }

Sets the client connection mode for the server process issuing the registration request.

If CONNECT=ACCEPT is specified, or defaulted, any client conversations directed to this server are automatically accepted. They are connected to the process as a standard conversation and immediately satisfy an appropriate receive request.

If CONNECT=NOTIFY is specified, any queued or new client connection requests are notified to the server's &INTREAD queue in the same manner as a transfer request. In this case the server can choose to accept or reject the connection by use of the &APPC TRANSFER options. Until the conversation is accepted by the &APPC TRANSFER_ACCEPT, it cannot be operated by the process.

Note: This mode of connection is the only one that allows the server to obtain any PIP data carried with the new conversation.

If CONNECT=REJECT is specified, any queued or new conversations are rejected with a condition of retry, or no retry, as determined by the RETRY operand. This is normally done only when the server is closing down to indicate end of processing.

RETRY={ YES | NO }

Sets the retry status for rejected connection requests. Rejection is due to an explicit CONNECTION=REJECT state, or due to the process conversation limit being reached.

If RETRY=YES is specified, or defaulted, connections are failed with a return code conveyed to the initiator of:

```
TRANS_PGM_NOT_AVAIL_RETRY
```

For RETRY=NO, the conversation fails with:

```
TRANS_PGM_NOT_AVAIL_NO_RETRY
```

CONVLIM={ 100 | *nnn* }

Sets the conversation limit for the process. If this limit is reached at any stage during processing, subsequent connection requests are automatically placed in the pending queue.

Example: & APPC REGISTER

```
&APPC REGISTER SERVER=PRINTSERVER
```

Return Codes:

The return codes are as follows:

0

Register accepted

16

Duplicate server name error

Notes:

An NCL process is registered as a server in the following ways:

- By starting the process using the START command and specifying a server name
- By starting the process using the &APPC START verb, issued from another process and specifying a server name
- By specifying a server name in the TCT entry that will service a transaction
- Directly, by issuing the &APPC REGISTER verb

More information:

&APPC DEREGISTER (see page 149)
&APPC SET_SERVER_MODE (see page 184)

&APPC REQUEST_TO_SEND

&APPC REQUEST_TO_SENT notifies the remote program that the local conversation partner would like to send data.

REQUEST_TO_SEND indicates that the local procedure wants to enter send state.

This verb has the following format:

```
&APPC REQUEST_TO_SEND [ ID=id ]
```

Operand:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted the current (last referenced, or only) conversation is assumed.

Example: &APPC REQUEST_TO_SEND

```
&APPC REQUEST_TO_SEND
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The REQUEST_TO_SEND request can only be issued from receive state or one of the confirm states. No state changes occur as a result of this request.

Notes:

- Use this request only in controlled situations where the local program wants to send data. However, no state changes occur. The remote conversation partner has the discretion to detect that the REQUEST_TO_SEND was issued (by examining the REQUEST_TO_SEND_RECEIVED indicator, &ZAPPCRTS) and issuing the appropriate requests to turn the conversation around.
- The SEND_ERROR request is used (with great discretion) to force the send state if necessary.

Relationship to LU6.2 Verb Set:

&APPC REQUEST_TO_SEND is equivalent to the LU6.2 verb MC_REQUEST_TO_SEND.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC RPC

&APPC RPC allows the calling procedure to call an NCL procedure in either the same or any connected APPC system.

As part of the remote procedure call, the initiating procedure can pass any number of NCL variables, MDOs, or both, as shared variables, to be presented in the context of the new process.

This verb has the following format:

```
&APPC RPC
      PROC=procname
    [ LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ SHARE | SHARE=(shrvars1,shrvars2,...,shrvarsn) |
      NOSHARE=(shrvars1,shrvars2,...,shrvarsn) ]
    [ RETCODE=varname ]
    [ PARMS=(parm1,parm2,...,parmn) ]
```

Operands:

PROC=*procname*

(Mandatory) Specifies the NCL procedure to call.

LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

Specifies where the procedure executes.

USERID=*userid* [PASSWORD=*password*]

Specifies an alternate user ID and, optionally, the password, under which this process starts. The security exit on the target system verifies the user ID and password before the new process is initiated.

The RPC system transaction uses SECURITY=SAME processing. Therefore, APPC includes the supplied user ID and password in the attach header for validation on the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

SHARE | SHARE=(*shrvars*1,*shrvars*2,...,*shrvarsn*) |
NOSHARE=(*shrvars*1,*shrvars*2,...,*shrvarsn*)

Specifies the set of NCL variables, and/or MDOs, to be shared or not shared with the called procedure.

If the SHARE or NOSHARE operands are omitted, no variable sharing occurs for this call. If the SHARE keyword is coded without parameters, the current &CONTROL SHRVARS (or NOSHRVARS) setting is used for the call.

If the SHARE or NOSHARE keyword is used with parameters, any current &CONTROL SHRVARS settings are ignored for this execution. When SHARE is used the called procedure obtains a copy of each variable, and/or MDO, referenced by the SHARE operand. When NOSHARE is used, the called procedure obtains a copy of all NCL variables and MDO data not specified by the operand.

The parameter values for SHARE and NOSHARE cannot be substituted on the statement.

The operand can specify a single value or a list of values. Each item in the list can reference a single variable or MDO, or multiple variables or MDOs. A single variable is referenced by including its entire name. For example, an entry in the list of 'ABC' refers to the single NCL variable &ABC, as though VARS=ABC were coded. A single MDO is referenced by including its entire name followed by a full stop. For example, an entry in the list of 'ABC.' refers to the single MDO named ABC, as though MDO=ABC were coded.

A range of NCL variables is referenced by including a name prefix followed by an asterisk. For example, an entry in the list of '$CNM*' refers to $CNM1...$CNM*nnn*. The range is set explicitly, for example, '$CNM*(1,10)' (or '$CNM(1,10)') refers to the variables $CNM1 to $CNM10. If not set explicitly, all variables of the form $CNM*nnn* are assumed.

A generic list of NCL variables and MDOs is referenced by including a name prefix followed by a > symbol. For example, an entry in the list of '$NW>' refers to all variables, and MDOs, with names beginning with &$NW.

RETCODE=*varname*

Specifies the name of a local NCL variable that contains the &RETCODE value from the called procedure.

PARMS=(*parm*1,*parm*2,...,*parmn*)

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Examples: &APPC RPC

```
&APPC RPC PROC=SHOWUSER PARMS=(ABC)
```

Return Codes:

The return codes are as follows:

0

Request successful

8

Remote program error

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

This request sets the system variable &ZAPPCID, which returns the system conversation identifier allocated to the conversation. This value is saved and supplied on the ID operand of subsequent &APPC requests to nominate a specific conversation where the NCL process is operating more than one conversation concurrently.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC SEND_AND_CONFIRM

&APPC SEND_AND_CONFIRM sends a single data record and a request for confirmation to the conversation partner and waits for the reply.

This verb has the following format:

```
&APPC SEND_AND_CONFIRM
    [ ID=id ]
    [ MDO=mdoname [ MAP=mapname ] |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted the current (last referenced, or only) conversation is assumed.

MDO=*mdoname* [MAP=*mapname*] | VARS=*var* |
VARS=(*var*1,*var*2,...,*varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Indicates how the outgoing data is formatted. If the MDO operand is used, the data is formatted into an MDO with the name *mdoname*. If the outgoing data is mapped (that is, a map name is sent with the data) and MAP= is not specified, then the received map name is used to connect to Mapping Services Mapping Support. If the outgoing data is not mapped, then it is the responsibility of the requester to connect to Mapping Services Mapping Support, if necessary, using the MAP operand.

The VARS= operand provides the list of NCL variables that contain the data received on the conversation (the usual NCL VARS definitions apply). If the outgoing data is not mapped, or the map name is other than $NCL, then the data stream received is segmented into the variables nominated. If the data is mapped and the map name is $NCL, then each variable is reconstructed as it was on the send request in the conversation partner. Unused variables are set to null.

Examples: &APPC_SEND_AND_CONFIRM

```
&APPC_SEND_AND_CONFIRM MDO=CLIENTRQST

&APPC_SEND_AND_CONFIRM VARS=ALERT
```

Return Codes:

The return codes are as follows:

0

Data sent and CONFIRMED has been received

4

Request unsuccessful

8

Other program issued SEND_ERROR

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

See &APPC SEND (see page 179) and &APPC CONFIRM (see page 130).

Note:

The SEND_AND_CONFIRM is a convenient way of combining two basic APPC operations, that of sending data and of confirming receipt of that data. The verb is equivalent to issuing an &APPC SEND_DATA followed by an &APPC CONFIRM request.

Relationship to LU6.2 Verb Set:

- &APPC SEND_DATA is equivalent to the LU6.2 verb MC_SEND_DATA.
- The &APPC SEND_DATA operands of VARS and MDO are used in place of the LU6.2 verb SEND_DATA options DATA and LENGTH.
- The &APPC SEND_DATA operand MAP=*mapname* is equivalent to the LU6.2 verb SEND_DATA option MAP(YES(*mapname*)).
- &APPC does not support the FMH_DATA parameter of the LU6.2 verb SEND_DATA.
- &APPC CONFIRMED is equivalent to the LU6.2 verb MC_CONFIRMED.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC SEND_AND_DEALLOCATE

&APPC SEND_AND_DEALLOCATE sends a single data record to the remote conversation partner and requests conversation termination and deallocation of its resources.

This verb has the following format:

```
&APPC SEND_AND_DEALLOCATE
    [ ID=id ]
    [ MDO=mdoname [ MAP=mapname ] |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
    [ TYPE={ SYNC | FLUSH | CONFIRM } ]
```

Operands:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted the current (last referenced, or only) conversation is assumed.

MDO=*mdoname* [MAP=*mapname*] | VARS=*var* |
VARS=(*var*1,*var*2,...,*varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Indicates how the outgoing data should be formatted. If the MDO operand is used, the data is formatted into an MDO with the name *mdoname*. If the outgoing data is mapped (that is, a map name is sent with the data), and MAP= is not specified, then the received map name is used to connect to Mapping Services Mapping Support. If the outgoing data is not mapped, then it is the requester's responsibility to connect to Mapping Services Mapping Support, if required, using the MAP operand.

The VARS= operand is used to provide the list of NCL variables that will contain the data received on the conversation. (The usual NCL VARS definitions apply). If the outgoing data is not mapped, or the map name is other than $NCL, then the data stream received will be segmented into the variables nominated. If the data is mapped and the map name is $NCL, then each variable is reconstructed as it was on the send request in the conversation partner. Unused variables are set to null.

TYPE={ SYNC | FLUSH | CONFIRM }

Specifies the deallocate option. TYPE=SYNC is the default. If the conversation sync level is CONFIRM, it is equivalent to the DEALLOCATE TYPE=CONFIRM option; otherwise a DEALLOCATE TYPE=FLUSH is assumed.

TYPE=FLUSH results in all data being flushed, forcing its transmission to the conversation partner before unconditional deallocation occurs.

TYPE=CONFIRM results in all data being flushed and a confirmation being requested before deallocation occurs.

Examples: &APPC SEND_AND_DEALLOCATE

```
&APPC SEND_AND_DEALLOCATE MDO=ALERT

&APPC SEND_AND_DEALLOCATE TYPE=FLUSH
```

Return Codes:

The return codes are as follows:

0

Data sent and conversation deallocated

8

Other program issued SEND_ERROR

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

See &APPC SEND (see page 179) and &APPC DEALLOCATE (see page 146).

Note:

The SEND_AND_DEALLOCATE is a convenient way of combining two basic APPC operations, that of sending data and of deallocating the conversation resources. It is equivalent to issuing an &APPC SEND_DATA followed by an &APPC DEALLOCATE request.

Relationship to LU6.2 Verb Set:

- &APPC SEND_DATA is equivalent to the LU6.2 verb MC_SEND_DATA.
- The &APPC SEND_DATA operands of VARS and MDO are used in place of the LU6.2 verb SEND_DATA options DATA and LENGTH.
- The &APPC SEND_DATA operand MAP=*mapname* is equivalent to the LU6.2 verb SEND_DATA option MAP(YES(*mapname*)).
- The FMH_DATA parameter of the LU6.2 verb SEND_DATA is not supported by &APPC.
- &APPC DEALLOCATE is equivalent to the LU6.2 verb MC_DEALLOCATE.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC SEND_AND_FLUSH

&APPC SEND_AND_FLUSH sends a single data record to the remote conversation partner and flushes any locally buffered information and forces its transmission to the remote conversation partner.

This verb has the following format:

```
&APPC SEND_AND_FLUSH
    [ ID=id ]
    [ MDO=mdoname [ MAP=mapname ] |
      VARS=var | VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
    [ CONT={ YES | NO } ]
```

Operands:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted, the current (last referenced, or only) conversation is assumed.

MDO=*mdoname* [MAP=*mapname*] | VARS=*var* |
VARS=(*var*1,*var*2,...,*varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Indicates how the outgoing data should be formatted. If the MDO operand is used, the data is formatted into an MDO with the name *mdoname*. If the outgoing data is mapped (that is, a map name is sent with the data), and MAP= is not specified, then the received map name is used to connect to Mapping Services Mapping Support. If the outgoing data is not mapped, then it is the requester's responsibility to connect to Mapping Services Mapping Support, if required, using the MAP operand.

The VARS= operand is used to provide the list of NCL variables that will contain the data received on the conversation. (The usual NCL VARS definitions apply). If the outgoing data is not mapped, or the map name is other than $NCL, then the data stream received will be segmented into the variables nominated. If the data is mapped and the map name is $NCL, then each variable is reconstructed as it was on the send request in the conversation partner. Unused variables are set to null.

CONT={ YES | NO }

Applies to OPERTYPE=GDS conversations only, and indicates whether the current GDS variable is to be continued with the next send. If CONT=NO is specified (or defaulted), the current send is the last or only logical record sent for the GDS variable. CONT=YES indicates that the continuation bit is to be set in the current logical record.

Example: &APPC SEND_AND_FLUSH

```
&APPC SEND_AND_FLUSH VARS=DATA
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

See &APPC SEND (see page 179) and &APPC FLUSH (see page 150).

Note:

The SEND_AND_FLUSH is a convenient way of combining two basic APPC operations, that of sending data and of flushing the conversation. It is equivalent to issuing an &APPC SEND_DATA followed by an &APPC FLUSH request.

Relationship to LU6.2 Verb Set:

- &APPC SEND_DATA is equivalent to the LU6.2 verb MC_SEND_DATA.
- The &APPC SEND_DATA operands of VARS and MDO are used in place of the LU6.2 verb SEND_DATA options DATA and LENGTH.
- The &APPC SEND_DATA operand MAP=*mapname* is equivalent to the LU6.2 verb SEND_DATA option MAP(YES(*mapname*)).
- The FMH_DATA parameter of the LU6.2 verb SEND_DATA is not supported by &APPC.
- &APPC FLUSH is equivalent to the LU6.2 verb MC_FLUSH.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC SEND_AND_PREPARE_TO_RECEIVE

&APPC SEND_AND_PREPARE_TO_RECEIVE sends a single data record to the remote conversation partner and changes processing from sending to receiving data.

This verb has the following format:

```
&APPC { SEND_AND_PREPARE_TO_RECEIVE | SEND_AND_PREPARE }
      [ ID=id ]
      [ TYPE={ SYNC | FLUSH | CONFIRM } ]
      [ MDO=mdoname [ MAP=mapname ] |
        VARS=var | VARS=(var1, var2, ..., varn ) |
        VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted the current (last referenced, or only) conversation is assumed.

TYPE={ SYNC | FLUSH | CONFIRM }

Specifies the level of confirmation required for any data previously sent to the remote conversation partner as follows:

TYPE=SYNC is the default and ,if the conversation sync level is CONFIRM, it is equivalent to the PREPARE_TO_RECEIVE TYPE=CONFIRM option; otherwise a PREPARE_TO_RECEIVE TYPE=FLUSH is assumed.

TYPE=FLUSH results in all data being flushed, forcing its transmission to the conversation partner but no confirmation reply is required (see Notes for this option).

TYPE=CONFIRM results in all data being flushed and a confirmation being requested (see Notes for this option).

MDO=*mdoname* [MAP=*mapname*] | VARS=*var* |
VARS=(*var*1,*var*2,...,*varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Indicates how the outgoing data should be formatted. If the MDO operand is used, the data is formatted into an MDO with the name *mdoname*. If the outgoing data is mapped (that is, a map name is sent with the data), and MAP= is not specified, then the received map name is used to connect to Mapping Services Mapping Support. If the outgoing data is not mapped, then it is the requester's responsibility to connect to Mapping Services Mapping Support, if required, using the MAP operand.

The VARS= operand is used to provide the list of NCL variables that will contain the data received on the conversation. (The usual NCL VARS definitions apply). If the outgoing data is not mapped, or the map name is other than $NCL, then the data stream received will be segmented into the variables nominated. If the data is mapped and the map name is $NCL, then each variable is reconstructed as it was on the send request in the conversation partner. Unused variables are set to null.

Examples: &APPC SEND_AND_PREPARE_TO_RECEIVE

```
&APPC SEND_AND_PREPARE_TO_RECEIVE TYPE=FLUSH

&APPC SEND_AND_PREPARE_TO_RECEIVE TYPE=CONFIRM
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

See &APPC SEND (see page 179) and &APPC PREPARE_TO_RECEIVE (see page 152).

Note:

The SEND_AND_PREPARE_TO_RECEIVE is a convenient way of combining two basic APPC operations, that of sending data and of preparing to receive data. It is equivalent to issuing an &APPC SEND_DATA followed by an &APPC PREPARE_TO_RECEIVE request.

Relationship to LU6.2 Verb Set:

- &APPC SEND_DATA is equivalent to the LU6.2 verb MC_SEND_DATA.
- The &APPC SEND_DATA operands of VARS and MDO are used in place of the LU6.2 verb SEND_DATA options DATA and LENGTH.
- The &APPC SEND_DATA operand MAP=*mapname* is equivalent to the LU6.2 verb SEND_DATA option MAP(YES(*mapname*)).
- The FMH_DATA parameter of the LU6.2 verb SEND_DATA is not supported by &APPC.
- &APPC PREPARE_TO_RECEIVE is equivalent to the LU6.2 verb MC_PREPARE_TO_RECEIVE. The LU6.2 verb has a LOCKS parameter which is not supported by &APPC.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC SEND_DATA

&APPC SEND_DATA sends a single data record to the remote conversation partner.

This verb has the following format:

```
&APPC{ SEND_DATA | SEND }
     [ ID=id ]
     [ MDO=mdoname [ MAP=mapname ] |
       VARS=var | VARS=(var1, var2, ..., varn ) |
       VARS=prefix* [ RANGE=(start,end) ] ]
     [ CONT={ YES | NO } ]
```

Operands:

SEND_DATA

Indicates this is a conversation request to send data to the remote conversation partner.

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted, the current (last referenced, or only) conversation is assumed.

MDO=*mdoname* [MAP=*mapname*] | VARS=*var* |
VARS=(*var*1,*var*2,...,*varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Specifies the name of the MDO (which is an element or field) or variables to be sent as application data on this send request. The named MDO item is extracted and sent as a contiguous byte stream as the data for this send operation.

MAP=*mapname* nominates the map name to be sent to the remote end. If omitted, but the MDO operand is used, then the fully qualified element (or field) name derived from Mapping Services Mapping Support is assumed as the map name. If omitted, and the VARS operand is used, then MAP=$NCL is assumed (as described for VARS=). No map name is sent if the remote application does not support data mapping. Map names is up to 64 characters long. Valid Mapping Services map names can consist of up to 8 name segments, each of up to 12 characters, and each separated by a period.

The VARS= operand is used to provide the list of NCL variables to be passed to the transaction processor in the remote end for this send operation (the usual NCL VARS definitions apply). If the remote conversation partner supports data mapping, and no map name is supplied, MAP=$NCL is assumed, and the data sent is formatted as a series of vectors, one per token referenced. If some other map name is used, or the conversation partner does not support data mapping, then all tokens are concatenated together to form the data contents to be sent to the other end.

CONT={ YES | NO }

Applies to OPERTYPE=GDS conversations only, and indicates whether the current GDS variable is to be continued with the next send. If CONT=NO is specified (or defaulted), the current send is the last or only logical record sent for the GDS variable. CONT=YES indicates that the continuation bit is to be set in the current logical record.

Example: &APPC SEND_DATA

```
SEND_DATA VARS=DATA
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The SEND_DATA request can only be issued from send state. No state changes occur as a result of this request.

Note:

This request does not necessarily result in any actual transmission taking place as data is normally buffered within the system until some trigger condition results in session transmission becoming necessary. The FLUSH request is used to ensure data transmission if required.

Relationship to LU6.2 Verb Set:

- &APPC SEND_DATA is equivalent to the LU6.2 verb MC_SEND_DATA.
- The &APPC SEND_DATA operands of VARS and MDO are used in place of the LU6.2 verb SEND_DATA options DATA and LENGTH.
- The &APPC SEND_DATA operand MAP=*mapname* is equivalent to the LU6.2 verb SEND_DATA option MAP(YES(*mapname*)).
- The FMH_DATA parameter of the LU6.2 verb SEND_DATA is not supported by &APPC.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC SEND_ERROR

&APPC SEND_ERROR signals an error condition to the remote conversation partner.

SEND_ERROR indicates this is a request to signal an error condition to the conversation partner. A text string can also be provided which will be included in the Error Log GDS variable.

This verb has the following format:

```
&APPC SEND_ERROR
    [ ID=id ]
    [ LOG=msg ]
```

Operands:

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted the current (last referenced, or only) conversation is assumed.

LOG=*msg*

When this operand is used it must be the last specified for the verb. All data following the LOG= operand is placed unchanged into the message area of the Error Log GDS variable sent to the remote application with the error indication.

If the remote application is an NCL procedure, it can access this text after receiving an error return code, in the &ZAPPCELM system variable.

Example: &APPC SEND_ERROR

```
&APPC SEND_ERROR LOG=&ERRMSG
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

The SEND_ERROR request is issued from send, receive, confirm, confirm_send, or confirm_deallocate states, and as a result the conversation is placed in send state.

Notes:

- After receiving a SEND_ERROR message an &APPC procedure returns a program error condition (&RETCODE set to 8) on the next request.
- The SEND_ERROR request is used to indicate a negative acknowledgement to a received CONFIRM request. No data is supplied on the SEND_ERROR request itself. However, on completion, the procedure will be in send state and can send any data required to the conversation partner.
- When used from confirm_deallocate state, (that is, having received a DEALLOCATE TYPE=CONFIRM request), a SEND_ERROR causes the deallocate to be rejected and processing continues with the local conversation in send state.
- SEND_ERROR is issued from send state. In this case no state changes occur but the remote conversation partner still receives an error condition (&RETCODE set to 8 for &APPC procedures) on the next request.
- SEND_ERROR is used to force send state, but this should only be done in a controlled manner.

Relationship to LU6.2 Verb Set:

&APPC SEND_ERROR is equivalent to the LU6.2 verb MC_SEND_ERROR.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC SET_SERVER_MODE

&APPC SET_SERVER_MODE allows a server process to declare its APPC operational mode concerning client connection.

SET_SERVER_MODE indicates that this is a request to set the mode of operation for APPC client conversations concerning this server process. This request can only be issued by a process that is a registered server. If the server was registered using the START command, or through the TCT option on attaching the transaction, any client conversations directed to the server remain in a pending mode until the SET_SERVER_MODE request is issued. For these servers, this should be the first &APPC option used to set the server's client connection mode.

This verb has the following format:

```
&APPC SET_SERVER_MODE
    [ CONNECT={ ACCEPT | NOTIFY | REJECT }  ]
    [ RETRY={ YES | NO } ]
    [ CONVLIM=nnn ]
```

Operands:

CONNECT={ ACCEPT | NOTIFY | REJECT }

Changes the client connection mode for the server process. Unless this operand is specified, the connection mode remains unchanged.

If the server was registered by other than an explicit &APPC REGISTER request, queued client conversations remain in a pending state until the connection option is explicitly set by this request.

If CONNECT=ACCEPT is specified, any queued or new client connection requests are automatically accepted by the process and satisfy an appropriate receive request.

If CONNECT=NOTIFY is specified, any queued or new client connection requests are notified to the server's &INTREAD queue in the same manner as a transfer request. In this case the server can choose to accept or reject the connection by use of the &APPC TRANSFER options. Until the conversation is accepted by the &APPC TRANSFER_ACCEPT, it cannot be operated by the process.

This mode of connection is the only one that allows the server to obtain any PIP data carried with the new conversation.

If CONNECT=REJECT is specified, any queued or new conversations are rejected with a condition of retry, or no retry, as determined by the RETRY operand.

RETRY={ YES | NO }

Sets the retry status for rejected connection requests. Rejection could be due to an explicit CONNECTION=REJECT state, or due to the process conversation limit being reached.

If RETRY=YES is specified, or defaulted, connections are failed with a return code conveyed to the initiator of:

TRANS_PGM_NOT_AVAIL_RETRY

For RETRY=NO, the conversation fails with:

TRANS_PGM_NOT_AVAIL_NO_RETRY

CONVLIM=*nnn*

Sets the conversation limit for the process. If this limit is reached at any stage during processing, subsequent connection requests are automatically placed in the pending queue.

Example: & APPC SET_SERVER_MODE

```
&APPC SET_SERVER_MODE CONNECT=ACCEPT
```

Return Codes:

The return codes are as follows:

0

Set server mode accepted

16

Set server mode error

Notes:

- If the process is not already registered as a server, the SET_SERVER_MODE verb fails.
- This verb does not provide defaults, so to change the value of an option, its operand must be specified.

More information:

&APPC REGISTER (see page 162)
&APPC DEREGISTER (see page 149)

&APPC START

&APPC START allows the calling procedure to start a new NCL process in either the same or any connected APPC system.

As part of the start request, the initiating procedure can pass any number of NCL variables, MDOs, or both, which are copied and created in the context of the new process. The initiating procedure can issue the request and continue processing without any indication that the new process has started. The procedure can also specify that the request not complete until the new process has successfully started, or failed to start.

This verb has the following format:

```
&APPC START
      PROC=proc
    [ SERVER=servername
     [ SCOPE={ REGION | USER | SYSTEM } ] ]
    [ ENV={ CURRENT | DEPENDENT | BACKGROUND } |
      LUNAME=luname | LINK=linkname | DOMAIN=domain ]
    [ USERID=userid [ PASSWORD=password ] ]
    [ PROFILE=profile ]
    [ NOTIFY={ NO | YES } ]
    [ VARS=(genvars1,genvars2,...,genvarsn) ]
    [ PARMS=(parm1,parm2,...,parmn) ]
```

Operands:

PROC=*proc*

Specifies the name of the NCL procedure to start.

SERVER=*servername*

Specifies the logical name for this NCL process. The name must be up to 32 characters long and unique within the scope as determined by the SCOPE operand, or the request fails.

SCOPE={ REGION | USER | SYSTEM }

Provides the scope for the registration of *servername* (which must be unique within the scope indicated) and is valid only if the SERVER operand is present. Valid scopes are:

REGION

Indicates that *servername* is to be unique within the current session, as defined by a particular connection to your product region.

USER

Indicates that *servername* is to be unique across all sessions associated with the particular user ID

SYSTEM

Indicates that *servername* is to be unique within this product region.

ENV={ CURRENT | DEPENDENT | BACKGROUND } | LUNAME=*luname* | LINK=*linkname* | DOMAIN=*domain*

Specifies where to initiate the started procedure.

If ENV=CURRENT (or defaulted), the new process starts as a peer of the requesting process.

If ENV=DEPENDENT, the new process starts as a dependent of the requesting process. If either the USERID or PROFILE operand is present, and specifies a user other than the requesting user ID, the ENV operand is ignored. The new process is started in the APPC region for the target user ID.

If ENV=BACKGROUND, the new process is attached in the background APPC server region for the requesting user ID, as if the request was sourced from a remote system.

With one of the following parameters, the new process is always started in the APPC region for the target user on the indicated APPC system:

- LUNAME=*luname*
- LINK=*linkname*
- DOMAIN=*domain*

If an existing APPC region for the target user does not exist, one is created through this function.

This behavior is true even where LUNAME=*luname* is used to specify the local system, and hence behaves differently from the use of the ENV operand.

If the DOMAIN operand is used but the APPC system cannot be located by domain name, the request fails. For example, an INMC link has not been activated to the system and therefore the domain is unknown.

USERID=*userid* [PASSWORD=*password*]

Specifies an alternate user ID and, optionally, the password, under which this process is to start. The security exit in the target system verifies the user ID and password before the new process is initiated.

The START system transaction uses SECURITY=SAME processing. Therefore, APPC includes the supplied user ID and password in the attach header for validation on the remote system.

If the user ID is specified with no password, the user ID is included in the attach header. The already verified indicator is not set unless the user ID is that of the requesting user environment. However, for a same LU transaction, APPC performs lock and key processing to determine whether the target user ID is signed on by the requester.

Note: For more information about APPC security, see the *Network Control Language Programming Guide*.

PROFILE=*profile*

Specifies the profile name to place in the attach header access security fields. If omitted, no profile is used. APPC makes no use of the PROFILE operand.

NOTIFY={ NO | YES }

Specifies whether synchronization with the start of the new process is required. If NOTIFY=NO is specified (or defaulted), no information is returned to the user regarding whether the new process started successfully or not. If NOTIFY=YES is specified, the &APPC START request does not complete until the new process has been loaded and is about to commence execution. In addition, the NCL ID and domain of the started process is returned to &SYSMSG.

VARS={ *genvars* | (*genvars*1,*genvars*2,...,*genvarsn*) }

Specifies the set of NCL variables, MDOs, or both to pass to the started process. In the new process, a copy of each variable or MDO is created with the same name and value.

The data following the VARS operands is a single value, or a list of one or more values enclosed in parentheses. Each item in the list can reference a single variable or MDO, or multiple variables or MDOs. A single variable is referenced by including its entire name. For example, an entry in the list of ABC, refers to the single NCL variable &ABC, as though VARS=ABC was coded.

A single MDO is referenced by including its entire name followed by a full stop. For example, an entry in the list of XYZ. refers to the single MDO named XYZ, as though MDO=XYZ was coded.

A range of variables is referenced by including a name prefix followed by an asterisk. For example, an entry in the list of $CNM* refers to all variables with names beginning with $CNM, as though VARS=$CNM was coded.

A specific range of variables is referenced by including a name prefix followed by an asterisk, and the range in parentheses. For example, an entry in the list of UVW*(1,10) refers to the range of variables &UVW1 through &UVW10 as if VARS=UVW RANGE=(1,10) was coded.

A numeric range of variables is referenced by including a name prefix followed by an asterisk. For example, an entry in the list of $CNM* refers to all variables with names of the form $CNM*n*, where *n* is a numeric suffix. All such variables are located, unless a specific range is coded (for example, $CNM*(1,20)). A generic range of variables is referenced by including a name prefix, followed by >. For example, A> refers to all tokens beginning with A, plus all MDOs beginning with A.

PARMS=(*parm*1,*parm*2,...,*parmn*)

Specifies a list of parameters to pass to the procedure. Parentheses enclose the parameter list, and a comma separates each pair of parameters.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on. The list is analyzed before substitution occurs and each parameter is isolated, by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results. If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted. Otherwise, it is treated as unquoted. If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself.

Once isolated, substitution is performed, if necessary (allowing transparent data to be passed as parameters). The result is placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply. That is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string.

For example:

```
PARMS=(&USER,,PROC=&0,“variable ““&FRED”” in error”)
```

The example sets the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable “&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

If used, the PARMS operand must be the last operand on the &APPC statement.

Example: & APPC START

```
&APPC START PROC=POSTMAN PARMS=(&USERID,&MAILMSG)
```

Return Codes:

The return codes are as follows:

- If NOTIFY=NO was specified, or defaulted, no indication of the success or failure of the started process is provided. In this case, the return codes and &SYSMSG are set as follows:

  0

  Accepted

  16

  Transaction error (&SYSMSG is set)

- If NOTIFY=YES was specified, the return codes and &SYSMSG are set as follows:

  0

  OK (&SYSMSG contains N23Q01)

  8

  Start failed (&SYSMSG contains N23Q03)

  16

  Transaction error (&SYSMSG is set)

N23Q01 indicates process start. N23Q03 indicates process failure. Both messages carry the NCL ID and domain where the process was created.

More information:

&APPC RPC (see page 166)

&APPC TEST

&APPC TEST sets the default conversation in NCL, usually as a prelude to interrogating the LU6.2 system variables.

This verb has the following format:

```
&APPC TEST
    [ ID=id ]
```

Operands:

TEST

Indicates this is a conversation test request.

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted the current (last referenced, or only) conversation is assumed.

Example: &APPC TEST

```
&APPC TEST ID=999
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

Note:

The &APPC TEST request is normally only necessary where multiple conversations are being operated concurrently by the NCL procedure. By specifying the conversation identifier being tested on the ID operand this conversation becomes the default operation conversation for all &APPC requests. In addition, this ensures that all system variables associated with &APPC processing relate to that particular conversation.

Relationship to LU6.2 Verb Set:

&APPC TEST is equivalent to the LU6.2 verbs MC_TEST and MC_GET_ATTRIBUTES.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC TRANSFER_ACCEPT

&APPC TRANSFER_ACCEPT accepts the transfer of ownership of an LU6.2 conversation offered by another NCL process.

This verb has the following format:

```
&APPC TRANSFER_ACCEPT
    [ ID=id ]
    [ NCLID=nclid | SERVER=servername ]
    [ ARGS | VARS=var |
      VARS=(var1, var2, ..., varn ) |
      VARS=prefix* [ RANGE=(start,end) ] ]
```

Operands:

TRANSFER_ACCEPT

Indicates this is a request to accept the transfer of ownership of the conversation specified by the ID parameter. If the request completes successfully, the conversation identified is then available to this procedure for operation. See Notes on the &APPC TRANSFER_REQUEST verb for more details about the transfer process.

ID=*id*

Specifies the conversation identifier (as provided by the N00101 notification message) that references the particular conversation being transferred. This parameter is required to indicate precisely which conversation is being accepted.

ARGS | VARS=*var* | VARS=(*var*1,*var*2,...,*varn*) | VARS=*prefix [RANGE=(*start*,*end*)]**

Nominates the NCL variables into which any PIP variable data present in the attach request, will be placed. If none is present, or a previous receive operation has been performed against this conversation, no variable data is set.

Example: &APPC TRANSFER_ACCEPT

```
&APPC TRANSFER_ACCEPT ID=999
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

None. The transferred conversation will be in send or receive state which is determined by examining the &ZAPPCSTA system variable.

Note:

The TRANSFER_ACCEPT gains ownership of the conversation being transferred, and logically completes the &APPC TRANSFER_REQUEST for the original owner.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC TRANSFER_CONNECT

&APPC TRANSFER_CONNECT allows an attached conversation, on which no APPC verb has been issued, to be transferred to another NCL process.

If a RECEIVE has already been issued on the conversation, this request fails. Otherwise, the conversation is passed to the target process as a connection request, and is connected according to the server connection mode, if set. Otherwise, it is placed in the pending queue.

This verb has the following format:

```
&APPC TRANSFER_CONNECT
    [ ID=id ]
    [ NCLID=nclid | SERVER=servername ]
    [ WAIT=nn ]
```

Operands:

TRANSFER_CONNECT

Indicates this is a request to transfer the conversation identified to the target NCL process as a client connection request. If the conversation has already been operated, the request fails.

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted, the current (last referenced, or only) conversation is assumed.

NCLID=*nclid* | SERVER=*servername*

Nominates the target NCL process to which ownership of the LU6.2 conversation is to be transferred. The notification message (see below) is queued to the internal environment of this process to indicate it that has been targeted for a transfer request.

SERVER=*servername* is an alternative way to nominate the target NCL process. If this operand is used, a search is performed for the server name registered for the region, user or system, in that order.

WAIT=*nn*

Specifies the time, in seconds (for example, 10), or seconds and hundredths (for example, 1.25), for which the procedure is prepared to wait for the transfer to be accepted or rejected. If not successful before this interval expires, the transfer request is canceled, and an unsuccessful return code (&RETCODE is set to 4, &ZFDBK is set to 0) results.

Example: &APPC TRANSFER_CONNECT

```
&APPC TRANSFER_CONNECT NCLID=123
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

This request is valid from send or receive state only, and no state changes occur as a result.

Notification Message Format:

When a TRANSFER_CONNECT is issued, the following message is placed on the request queue of the internal environment for the NCL process targeted:

```
N00101 NOTIFY: APPC EVENT: TRANSFER RESOURCE: id
```

id is the system unique conversation identifier being transferred. This value must be supplied on any subsequent TRANSFER_CONNECT or TRANSFER_REJECT request by the target procedure.

Note:

The main use of this verb is to transfer a conversation to an active server. For example, if an NCL process was started as a result of a remotely initiated conversation (and the relevant TCT entry does not specify a server name), it is possible to have two or more instances of the process started. The first one to register becomes the server. When the process attempts to register, the &APPC REGISTER verb fails with a duplicate server name error, and the conversation is transferred to the active server by using the &APPC TRANSFER_CONNECT verb. For example:

```
&APPC REGISTER SERVER=PRINTSERVER
&IF &RETCODE = 16 &THEN +
&DO
&APPC TRANSFER_CONNECT SERVER=PRINTSERVER . . .
```

More information:

&APPC REGISTER (see page 162)
&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC TRANSFER_REJECT

&APPC TRANSFER_REJECT rejects the transfer of ownership of an LU6.2 conversation offered by another NCL process.

This verb has the following format:

```
&APPC TRANSFER_REJECT
    [ ID=id ]
    [ NCLID=nclid | SERVER=servername ]
    [ RETRY={  YES | NO } ]
```

Operands:

TRANSFER_REJECT

Indicates that this is a rejection of the conversation transfer of ownership offered by another NCL process through the &APPC TRANSFER_REQUEST option. The original owner of the conversation maintains ownership. Whether you can retry the condition, or not, is indicated when the conversation is rejected.

ID=*id*

Specifies the conversation identifier (as provided by the N00101 notification message) that references the particular conversation being rejected for transfer. This parameter indicates precisely which conversation is being rejected.

NCLID=*nclid* | SERVER=*servername*

Nominates the target NCL process to which ownership of the LU6.2 conversation is to be transferred. The notification message (see below) is queued to the internal environment of this process to indicate it has been targeted for a transfer request.

SERVER=*servername* is an alternative way to nominate the target NCL process. If this operand is used, a search is performed for the server name registered for the region, user or system, in that order.

RETRY={ YES | NO }

Indicates whether or not you can retry the reject condition. The default is RETRY=YES, and this is reflected to the remote end in the APPC return code:

```
TRANS_PGM_NOT_AVAIL_RETRY
```

Otherwise the return code is set to:

```
TRANS_PGM_NOT_AVAIL_NO_RETRY
```

Example: &APPC TRANSFER_REJECT

```
&APPC TRANSFER_REJECT ID=999 RETRY=NO
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

Note:

The TRANSFER_REJECT formally rejects ownership of the conversation being transferred, and logically completes the &APPC TRANSFER_REQUEST for the original owner.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPC TRANSFER_REQUEST

&APPC TRANSFER_REQUEST requests the transfer of ownership of an LU6.2 conversation from one NCL process to another. The short form is &APPC TRANSFER.

This verb has the following format:

```
&APPC{ TRANSFER_REQUEST | TRANSFER }
    [ NCLID=nclid | SERVER=servername ]
    [ ID=id ]
    [ WAIT=nn ]
```

Operands:

TRANSFER_REQUEST

Indicates this is a request to transfer ownership. The procedure is suspended pending completion of the request. The NCLID parameter must be supplied to indicate the target process to which the conversation is to be passed. If the ID parameter is omitted, the current conversation for this procedure is assumed. A notification is sent to the target process indicating that a transfer request has been issued. If the request completes successfully, the conversation is no longer available to this procedure for operation.

NCLID=*nclid* | SERVER=*servername*

Nominates the target NCL process to which ownership of the LU6.2 conversation is to be transferred. The notification message (see below) is queued to the internal environment of this process to indicate it has been targeted for a transfer request.

SERVER=*servername* is an alternative way to nominate the target NCL process. If this operand is used, a search is performed for the server name registered for the region, user or system, in that order.

ID=*id*

Specifies the conversation identifier (as first returned by the system variable &ZAPPCID after successful allocation) that references a particular conversation. If this parameter is omitted, the current (last referenced, or only) conversation is assumed.

WAIT=*nn*

Specifies the time, in seconds (for example, 10), or seconds and hundredths (for example, 1.25), for which the procedure is prepared to wait for the transfer to be accepted or rejected. If not successful before this interval expires the transfer request is canceled, and an unsuccessful return code (&RETCODE is set to 4, &ZFDBK is set to 0) results.

Examples: &APPC TRANSFER_REQUEST

```
&APPC TRANSFER_REQUEST NCLID=123

&APPC TRANSFER NCLID=123
```

Return Codes:

The return codes are as follows:

0

Request successful

4

Request unsuccessful

8

Remote program error

12

State check

16

Request or conversation error

&ZFDBK is also set, plus all APPC system variables.

State Transition:

This request is valid from send or receive state only, and no state changes occur as a result.

Notification Message Format:

When a TRANSFER_REQUEST is issued the following message is placed on the request queue of the internal environment for the NCL process targeted:

```
N00101 NOTIFY: APPC EVENT: TRANSFER RESOURCE: id
```

id is the system unique conversation identifier being transferred. This value must be supplied on any subsequent TRANSFER_ACCEPT or TRANSFER_REJECT request by the target procedure.

Notes:

- The &APPC TRANSFER_REQUEST is not an LU6.2 function, but an extension offered by your product. It allows the procedure that started the conversation to transfer control to another procedure under controlled conditions. The targeted procedure must issue an &APPC TRANSFER_ACCEPT or TRANSFER_REJECT to complete the transfer process.
- This request is useful when a remote allocation attaches a procedure within the local system, and that procedure, having made some determinations about the conversation, decides that another process should be started and given control of the conversation to service the transaction.
- This request can also be used so that a single procedure can accumulate control of more than one attached conversation if desirable.

More information:

&APPC Return Code Information (see page 80)
&RETCODE and &ZFDBK (see page 80)

&APPSTAT

&APPSTAT returns the current status for a specific VTAM application.

&APPSTAT is a built-in function and must be used to the right of an assignment statement.

This built-in function has the following format:

```
&APPSTAT applname
```

&APPSTAT lets you interrogate the current status of VTAM applications within the network. The status is determined when the &APPSTAT function is processed.

The status of the specified application (*applname*) is returned in the variable specified as the assignment target.

The status returned is one of the following defaulted values:

ACTIVE

The application is available and accepting logons.

INACTIVE

The application is not available.

NOLOGONS

The application is available but not accepting logons.

NOTAPPL

The resource specified is not an application.

UNAVAIL

Unable to determine status.

UNKNOWN

Resource is not known to VTAM.

If the SYSPARMS APPTXT*n* command has been used to change these default values, the substituted values are used.

Operand:

applname

Specifies the name for the VTAM application whose status is to be determined.

Examples: &APPSTAT

```
&1 = &APPSTAT IMS

&STATUS = &APPSTAT TSO2
```

Notes:

Application status is determined when the &APPSTAT statement is processed. If the application specified resides in another processor elsewhere in the network, it could take sometime to determine the application status. (If the other processor in which that application resides has stopped, this period is indeterminable.)

Important! Do not use &APPSTAT in EASINET procedures, as unacceptable delays can occur. To avoid this, the Application Status Monitoring facilities identify application names for which the system is to maintain current status. This status is updated at an installation-defined frequency. These facilities generate a global variable showing the status of the application for each application nominated. Global variables can then be referenced from a full-screen panel without incurring any of the delays that can occur with &APPSTAT.

&ASISTR

&ASISTR returns the following string, yet retains leading blanks.

&ASISTR is a built-in function and must be used to the right of an assignment statement.

&ASISTR returns the complete text, as is.

&ASISTR assumes any leading blanks are intended-assignment commences from the second character position after the &ASISTR keyword.

This built-in function has the following format:

```
&ASISTR text
```

Operand:

text

Specifies one or more words or variables for assignment to the target variable.

Examples: &ASISTR

```
&SYSMSG = &ASISTR  INVALID DATA ENTERED

&PRTLINE = &ASISTR NETWORK SUMMARY REPORT
```

Notes:

- &ASISTR is ideal when constructing error messages where the text is to be offset for display formats.
- &STR and &ASISTR are useful ways of assigning multiple words into a single string-do not use &CONCAT because it eliminates blanks between the fields being assigned.
- The total size of the constructed variable or constant cannot exceed the maximum size for a variable, that is, 256 characters (if it exceeds the maximum, it is truncated to 256 characters).
- &ASISTR differs from the &STR function in that it assigns data as is, and retains leading blanks (&STR scans for data starting after the &STR keyword and starts its assignment from the first data character).
- If &CONTROL DBCS, DBCSN, or DBCSP is in effect, &ASISTR is sensitive to the presence of DBCS data (see page 1197).

More information:

&STR (see page 649)
&LBLSTR (see page 386)
&NBLSTR (see page 442)
&TBLSTR (see page 651)

&ASSIGN

&ASSIGN sets the data values of a list or range of variables from another list or range, or a data constant.

&ASSIGN updates lists or ranges of variables in one operation. &ASSIGN is used to copy data between variables, update variable attributes, and determine the names of variables flagged as MODFLDs.

The &ASSIGN options allow the actual structure of an MDO, or its structure according to the map, to be derived during NCL execution. All MDO query functions must use tokens as the target for the information to be returned.

&ASSIGN manipulates multiple variables in one operation. The default function, OPT=DATA, is used to assign values to a target list or range of variable names from the values of a string of data, or a source list, or a range of variables. If no source data is provided or exists, then the target variable is set to nulls.

OPT=MODFLD uses the names of variables, flagged as modified fields, as the source data and extends the use of the &ZMODFLD built-in function.

The SETERR, SETMOD, RESETERR, and RESETMOD functions provide the ability to update the attributes of the target variables. The SETOUT and RESETOUT functions provide the ability to set or reset the output attribute on the target variables.

The maximum number of variables (range limit) that &ASSIGN can create in one operation is governed by the &CONTROL RNGLIM | NORNGLIM operand. If this is set to RNGLIM, then the range limit is 64. If very large numbers of variables are to be processed in one operation, then use NORNGLIM (the default value of this operand) to remove the limit of 64.

This verb has the following formats.

To set the data values of a list or range of variables from another list or range, or a data constant:

```
&ASSIGN [ OPT= DATA ]
        { VARS={ name | ( name, name, ..., name ) } |
          VARS=prefix* [ RANGE=( start, end ) ] |
          GENERIC | REPLACE | ADD | UPDATE |
          ARGS [ [ RANGE=( start, end ) ] | MDO=aaa.bbb.ccc |
                   MDO=aaa.bbb.ccc.* | MDO=aaa.bbb[*].ccc |
                   MDO=aaa [ MAP=map ] ] }
        [ DATA=value |
          FROM { VARS={ name | ( name, name, ..., name ) } |
                 VARS=prefix* [ RANGE=( start, end ) | GENERIC ] |
                 ARGS [ RANGE=( start, end ) ] |
                 MDO=aaa.bbb.ccc | MDO=aaa }
          MDO=aaa.bbb.ccc.* | MDO=aaa.bbb[*].ccc ]
```

To set a list or range of variables to the names of MODFLD variables:

```
&ASSIGN OPT= MODFLD [ NONULLS | NORESET ]
      { VARS={ name | ( name, name, ..., name ) } |
        VARS=prefix* [ RANGE=( start, end ) ] |
        ARGS  [ RANGE=( start, end ) ] }
      [ FROM { VARS=[ name | ( name, name, ..., name ) ] |
        VARS=prefix* [ RANGE=( start, end ) | GENERIC ] |
        ARGS [ RANGE=( start, end) ] } ]
```

To update the attributes of a list or range of variables:

```
&ASSIGN OPT={ SETERR | RESETERR | SETMOD | RESETMOD }
      { VARS={ name | ( name, name, ..., name ) } |
        VARS=prefix* [ RANGE=( start, end ) ] |
        ARGS [ RANGE=( start, end ) ] }
```

To set or reset the output attribute on the specified target variable:

```
&ASSIGN OPT={ SETOUT | RESETOUT }
      { VARS={ name | ( name, name, ..., name ) } |
        VARS=prefix* [ RANGE=( start, end ) ] |
        ARGS [ RANGE=( start, end ) ] }
```

The &ASSIGN syntax for querying MDO and map names:

```
&ASSIGN OPT={ NAMES,TAGS,TYPES,LENGTHS,#ITEMS,NAMEDVALUES }
      { VARS=( aaa, bbb, ..., ccc ) |
        VARS=aaa* [ RANGE=( start, end ) ] |
        ARGS [ RANGE=( start, end ) ] }
      [ FROM | DEFINED_IN | PRESENT_IN | MANDATORY
       { MDO=aaa | MDO=aaa.bbb.ccc }]
```

Operands:

If specified, the OPT keyword changes the nominated attribute of the target variables. The following options are available:

OPT=DATA

(Default option) Specifies that the &ASSIGN operation is to transfer the contents of source variables or data constant (as supplied by the FROM specification) to the target variables. If either a DATA= or a FROM option is not specified, then target variables are set to null.

OPT=MODFLD

Specifies that the values that the target variables are to receive are the names of variables updated as MODFLDs by certain NCL verbs, such as &PANEL and &NDBGET. If only one target variable is supplied, then this function is equivalent to referencing the &ZMODFLD system variable. If source variables are specified by using the FROM option, then only these variables are scanned as MODFLDs. Enough names are used to exhaust the target list or range. Remaining target variables are set to null.

Variables flagged as MODFLDs by panel operations are returned in the order in which they appear on-screen, that is, line-by-line from top-left to bottom-right. The order for MODFLD variables returned by &NDBGET is unspecified.

For OPT=MODFLD the following options are supported:

NONULLS

Specifies that null variables are not used to satisfy the MODFLD option. The target variables assigned by the MODFLD operation therefore do not contain the names of null variables.

NORESET

Specifies that source variables are not to have the MODFLD attribute reset once the MODFLD operation is completed. The default resets the MODFLD attribute to prevent it being selected further by either &ZMODFLD or &ASSIGN.

OPT={ SETERR | RESETERR | SETMOD | RESETMOD }

OPT=SETERR

Specifies that target variables are to be given the ERRFLD attribute if displayed on a panel. No source variables are associated with this operation.

OPT=RESETERR

Specifies that target variables are to have the ERRFLD attribute reset. No source variables are associated with this operation.

OPT=SETMOD

Specifies that target variables are to be given the MODIFIED attribute. If displayed on a subsequent panel, these variables appear to the procedure to have been modified even if not modified by the user. No source variables are associated with this operation.

OPT=RESETMOD

Specifies that the MODIFIED attribute is to be reset for the target variables. If variables are displayed on a panel, then these do not appear as modified fields on subsequent input, unless they have been modified on the panel. No source variables are associated with this operation.

OPT={ SETOUT | RESETOUT }

OPT=SETOUT

Specifies that the target variables are to be assigned the output attribute. This attribute indicates that if the name appears in an input field, it is protected and displayed as an output field.

OPT=RESETOUT

Specifies that the target variables are to have the OUTPUT attribute reset.

The OUTPUT attribute affects panels displayed by procedures running with the &CONTROL FLDCTL option. This attribute is reset on all variables after a panel is displayed even if the variable did not appear on the panel.

When you switch a field from INPUT to OUTVAR using this method, the field attributes (for example, color and extended highlighting) may not be suitable for the output field. The ***UNDOCUMENTED*** standard input attributes are tested against the attributes of the field being altered. If the field has the same attributes as the standard input low intensity field, it is given the attributes of the standard low intensity output field. Likewise for the high intensity field attributes. If the field attributes do not match either standard, then the field is only protected, and no other attribute is changed.

OPT={ NAMES,TAGS,TYPES,LENGTHS,#ITEMS,NAMEDVALUES }

OPT=NAMES (*or* OPT=NAME)

Applies to PRESENT_IN, DEFINED_IN, and MANDATORY options, and returns the component names associated with the target MDO name, as follows:

If only an MDO stem name is specified (for example, MDO=abc), then the name of the currently connected map is returned as the defined component name on this query, but only if the MDO exists.

If the MDO name is a compound name (for example, MDO=a.b.c), then the name of the last component in the name list (that is, c) is returned, depending upon the PRESENT_IN, DEFINED_IN, or MANDATORY option.

If the MDO name is a compound generic name (for example, MDO=a.b.c.*), then multiple names may be returned, where each name returned is a sub-component of the nominated component (for example, for MDO=a.b.c.*, all *x* where *x* is a component defined within c). This is useful for determining the names of all components that are either present in, or defined within, a given structure. It is also useful for determining which component is within a structure that is a CHOICE type.

Note: For SEQUENCE OF and SET OF items, it is possible to have null-named components, as the SEQUENCE or SET OF items are processed by index value only.

A compound variable indexed name (for example, MDO=a.b{*}.c, or MDO=a.b.{*}) is not supported on this query.

OPT=TAGS

Applies to PRESENT_IN, DEFINED_IN, and MANDATORY options and returns the component tags used by Mapping Services associated with the target MDO name. Component selection is as for OPT=NAMES.

OPT=TYPES

Applies to PRESENT_IN, DEFINED_IN, and MANDATORY options and returns the component type defined within the map and associated with the target MDO name. Component selection is as for OPT=NAMES.

OPT=LENGTHS

Applies only when PRESENT_IN is specified and returns the local form data length within the MDO of the target components. Component selection is as for OPT=NAMES.

OPT=#ITEMS

Applies only when PRESENT_IN is specified, and returns the number of items within a nominated component as follows:

If the MDO name is a stem name (for example, MDO=stem), or a compound name (for example, MDO=a.b.c), then a count of 0 is returned if the component does not exist; otherwise it is 1.

If the MDO name is a compound generic name (for example, MDO=a.b.c.*), then a count of 0 is returned if the nominated component a.b.c is in one of the following conditions:

- Does not exist
- Exists but is empty
- Exists but is not constructed Otherwise it provides the number of components present within the structure a.b.c.
- If the MDO name is a compound variable indexed name (for example, MDO=a.b{*}, or MDO=a.b.{*}), then the number of components present in the SET OF or SEQUENCE OF structure is returned, or 0 if the structure does not exist or is empty. The variable index must be in the last name segment.

OPT=NAMEDVALUES

Applies to components that have named values associated with their type. These types are limited to BIT STRING, INTEGER, and ENUMERATED. Other types return null results. No generic indexes or generic names are allowed on this option.

If DEFINED_IN is specified, then a list of the named values defined in the map for the specified component is returned.

PRESENT_IN is invalid for OPT=NAMEDVALUES.

OPT=VALIDVALUES

Applies to string types that can have their character set constrained to particular characters or strings. It only works in conjunction with the DEFINED keyword. If a string type (for example, GraphicString) has been constrained to a particular set of characters or strings, then this option returns the valid characters or strings in the target variable. If there are no constraints, then no values are returned on assignment. The &ZVARCNT system variable is set to indicate the number of target variables set by the assignment.

Example:

If there is a component defined as follows:

```
datax GraphicString (“ABCD” | “xyz” | “QQQ” )
```

then &ASSIGN VARS=X* OPT=VALIDVALUES DEFINED MDO=... datax returns three variables set as follows:

```
&X1 = ABCD
&X2 = xyz
&X3 = QQQ
```

Example:

If there is a component defined as follows:

```
datax GraphicString ( FROM ( “A”c | “C” | “Y”C | “X” ) )
```

then &ASSIGN VARS=X* OPT=VALIDVALUES DEFINED MDO=... datax returns four variables set as follows:

```
&X1 = A
&X2 = C
&X3 = Y
&X4 = X
```

When updating the attributes of a list or range of variables:

VARS=

Specifies the names of the variables to be the target of the assignment operation. If insufficient variables are provided, some data is not available to the procedure. Excess variables are set to a null value. The formats for operands that may be coded with VARS= are:

prefix*

Supplies leading characters terminated by an asterisk to denote either a numeric or generic range of variables. If the RANGE= operand is specified or allowed to default, then a numeric range is generated. *prefix** cannot be used with other variable names.

RANGE=(*start*,*end*)

Specifies the generation of an ascending numeric range by concatenating the supplied prefix with a numeric suffix that is sequentially incremented within the supplied start and end values. The start and end values must be in the range 0 to 32767 and the end value must be equal to or higher than the start value.

Note: If VARS=* is specified, then the range is restricted to 1 to 32767. Specifying a start value of 0 causes an attempt to create an &0 variable. As &0 is a system variable name, this causes the process to fail.

Example:

```
&ASSIGN VARS=AB* RANGE=(1,3)
```

creates variable names &AB1, &AB2, and &AB3.

GENERIC

Specifies that the supplied prefix applies to all variables beginning with that prefix. All existing variables in the target range are set to null and target variables starting with the nominated prefix are created by using the FROM variables suffix and data. This option is valid only with OPT=DATA and when both source and target variables are specified as VARS=*prefix**.

The MDO operand is also valid.

Note: For more information about using the MDO operand, see the *Network Control Language Programming Guide*.

REPLACE

Specify that all target variables starting with this prefix, and whose suffix matches the FROM variable suffix, are updated with the contents of the matching FROM variable. This option is valid only with OPT=DATA and when both source and target variables are specified as VARS=*prefix**. No variables are created using this option-they can only be updated.

ADD

Specifies that target variables starting with this prefix are created with the FROM variable suffix and data, if such a named variable does not already exist. This option is valid only with OPT=DATA and when both source and target variables are specified as *VARS=prefix**.

UPDATE

Specifies that target variables starting with this prefix are updated and/or created using the FROM variables suffix and data. This option is valid only with OPT=DATA and when both source and target variables are specified as VARS=*prefix**.

name

Defines a variable, without the ampersand (&). A variable list is specified by enclosing in brackets multiple names separated by commas, for example:

```
VARS=(VAR1,P$USER,P$TERM)
```

ARGS

Denotes the assignment operation modifying or creating variables of the form &1 to &n, depending on how many are needed to satisfy the operation. The RANGE= operand is coded to designate a start number and an end number to delimit the number of variables generated.

RANGE=(*start*,*end*)

Specifies the generation of an ascending numeric range by concatenating the supplied prefix with a numeric suffix that is sequentially incremented within the supplied start and end values. Start and end values must be in the range 1 to 32767, and the end value must be equal to or higher than the start value. For example, the variable names for RANGE=(1,3) are prefix1, prefix2, prefix3.

DATA=*value*

(For OPT=DATA) Specifies a data constant to be assigned to all target variables. The default is a null value if no FROM or DATA operands are specified.

FROM

Indicate that source variables are used as the source of assignment data (OPT=DATA), or to restrict the search for panel-modified variables (OPT=MODFLD). ARGS, VARS=, or MDO= must be coded after FROM.

DEFINED_IN | PRESENT_IN | MANDATORY

If the DEFINED_IN keyword is specified, then the information is returned for all those components defined within the connected map, regardless of their presence or absence in the MDO itself.

If the PRESENT_IN keyword is specified, then the information is returned only for those components that are present within the MDO.

If the MANDATORY keyword is specified, then only those defined components that are mandatory are selected.

VARS=

Specifies a variable list or range, as described previously.

GENERIC

Is a range modifier used to select a non-numeric range of currently existing variables as the source. This operand is mutually exclusive if GENERIC, REPLACE, ADD, and UPDATE are used as target variable specifications. For OPT=MODFLD, GENERIC specifies a non-numeric range of variables to be scanned as MODFLDs. If used with OPT=DATA, source variables are sorted in name order on a character basis.

ARGS

Denotes that the source variables have the form &1 to &n. The RANGE= operand is coded to designate a start number and end number to delimit the number of variables to be used.

MDO=*aaa*

Specifies the entire data section of an MDO as the source or target of the assignment. The value *aaa* is from 1 to 12 characters long, and is ended with a period (a period is assumed if it is omitted).

MAP=*map*

Identifies the map that is to be attached to this MDO during the assignment operation.

MDO=*aaa.bbb.ccc*

Specifies an MDO with 1 to 30 component name segments, each separated by a period.

Note: For more information about using the MDO operand, see the *Network Control Language Programming Guide*.

An index is specified in brackets following any but the first name segment (for example, aaa.bbb{3}.ccc{5}). The index is used to indicate a component within an MDO structure as the source or target of an assignment. See later in this section, for more information on assignment to and from MDOs.

Examples: & ASSIGN

```
&ASSIGN VARS=(NAME1,USER,PHONE) DATA= -* clear user details
&ASSIGN ARGS                         -* clear &1 - &64
&ASSIGN VARS=S* RANGE=(1,&MAXROWS)   -* clear panel selection
&ASSIGN VARS=(&ERRLST) OPT=SETERR    -* flag fields in error
   -* &ERRLST contains a name list 'n1,n2,n3' of fields in error.

ASSIGN VARS=D* RANGE=(1,&MAXROWS) +  -* set display lines
       FROM VARS=DATALINES* +
       RANGE=(&TOP,&BOT)

&ASSIGN VARS=SELECT OPT=MODFLD NONULLS +
        FROM VARS=S*                 -* pick up next selection
                                     -* and ignore blanked out
                                     -* selections
&ASSIGN ARGS OPT=MODFLD              -* pick up all modified fields
&ASSIGN VARS=SAVE* GENERIC +         -* create backup copy variables
        FROM VARS=FILEA*             -* from file record
   -* where &A1 = XXX, &AB = YYY, &DD2 = ZZZ, &DDB = $$$

&ASSIGN VARS=A* BYNAME FROM VARS=DD*  -* &AB = &DDB
&ASSIGN VARS=A* OVERLAY FROM VARS=DD* -* &AB = &DDB
                                      -* &A2 = &DD2
&ASSIGN VARS=A* MERGE FROM VARS=DD*   -* &A2 = &DD2
&ASSIGN VARS=A* FROM VARS=DD* GENERIC -* &A1 = &DDB
                                      -* &A2 = &DD2
```

Valid forms of MDO component reference:

```
MDO=CNM.ALERT.

MDO=CNM.ALERT.PSID{2}.PRODUCT

MDO=PANEL.LINE{15}

MDO=CNM.ALERT.TYPE

MDO=CNM.ALERT.PSID{2}.PRODUCT.SWNAME
MDO=PANEL.LINE{15}.RESOURCE{1}

MDO=PPO.TEXT
```

Notes:

In all functions, the system variable &ZVARCNT is set to indicate the number of target variables updated.

Using OPT=DATA assignment with overlapping ranges gives results as if all assignments are performed in parallel. This lets you shift data values up and down variable ranges. For example:

```
&ASSIGN VARS=A* RANGE=(2,5) FROM VARS=A* RANGE=(1,4)
```

gives the same results as:

```
&A5=&A4, &A4=&A3, &A3=&A2, &A2=&A1
```

Setting and resetting ERRFLD and MODIFIED attributes is independent of any MODFLD attribute for a variable. It is therefore possible to flag variables without affecting the MODFLD processing that an NCL procedure may have.

A variable flagged as ERRFLD or MODIFIED is not returned by a MODFLD function until it has been displayed on a panel. A variable retains the MODFLD attribute when a panel is redisplayed until it is reset by &ZMODFLD, by &ASSIGN OPT=MODFLD, or by a panel display that does not contain the variable. Null variables is flagged as ERRFLD or MODIFIED.

If a panel field is overtyped or erased so that the variable contents are unaltered, then it is not returned by the MODFLD function. Also, &CONTROL FLDCTL must be in effect during a panel display for the MODFLD attribute to be updated by panel services.

More information:

&ZMODFLD (see page 940)
&CONTROL (see page 266)

&ASSIGN Statement for MDO Assignments

The MDO options are NAMES, TAGS, TYPES, LENGTHS, #ITEMS, and NAMEDVALUES.

&ASSIGN Syntax for MDO Data Assignments

When &ASSIGN OPT=DATA is specified, the MDO operand is used to indicate that a source or target data item is all or part of an MDO. The full syntax allowable is as follows:

```
&ASSIGN [ OPT=DATA ]
        { VARS=aaa |
          VARS=(aaa,bbb,...,ccc) |
          VARS=aaa* [ RANGE=(start, end) | GENERIC ] |
          REPLACE | ADD | UPDATE |
          ARGS [ RANGE=(start, end) ] |
          MDO=aaa.bbb.ccc |
          MDO=aaa.bbb.ccc.* |
          MDO=aaa.bbb{*}.ccc [ RANGE=(start, end) ] |
          MDO=aaa [ MAP=map ] }
        { FROM
          { VARS=aaa |
            VARS=(aaa,bbb,...,ccc) |
            VARS=aaa* [ RANGE=(start, end) | GENERIC ] |
            ARGS [ RANGE=(start, end) ] |
            MDO=aaa.bbb.ccc |
            MDO=aaa.bbb.ccc.* |
            MDO=aaa.bbb{*}.ccc [ RANGE=(start, end) ] |
            MDO=aaa }|
          DATA=data }
```

Example: A stem name MDO with no other &ASSIGN operands

```
&ASSIGN MDO=aaa
```

When no other operands are present, this assignment statement is interpreted as an MDO deletion operation. This is the only way to delete an MDO.

Example: A stem name MDO as the target of &ASSIGN

Using this form of assignment, the entire MDO with name aaa is created, or updated:

```
&ASSIGN MDO=aaa MAP=map
```

This has two functions:

If the MDO does not exist, then it is created and assigned to the map name provided, but contains no data.

If the MDO already exists, then it is assigned to the map name provided, but its data remains unchanged.

If the map name is unknown, then the MDO is unmapped, and its data can only be referenced in entirety by the MDO stem name.

```
&ASSIGN MDO=aaa [ MAP=map ] FROM MDO=xxx
```

Allows the MDO named *aaa* to be created or updated as a copy of the entire contents of the MDO named *xxx*. If the MAP operand is omitted, then the map name of the source MDO is assumed.

```
&ASSIGN MDO=aaa [ MAP=map ] FROM MDO=xxx.yyy.zzz
```

Allows the MDO named *aaa* to be created or updated as a copy of only a section of the source MDO named *xxx.yyy.zzz*. If the MAP operand is omitted, then the map name of the source MDO is assumed.

```
&ASSIGN MDO=aaa [ MAP=map ] DATA=xxx
```

Allows the MDO named aaa to be created or updated to contain only the data specified. If the MAP operand is omitted, then the MDO is unmapped, and its data can only be referenced in entirety by the MDO stem name.

```
&ASSIGN MDO=aaa [ MAP=map ] FROM VARS=xxx
```

Allows the MDO named *aaa* to be created or updated to contain only the data from the token *xxx*. If the MAP operand is omitted, then the MDO is unmapped, and its data can only be referenced in entirety by the MDO stem name.

```
&ASSIGN MDO=aaa [ MAP=map ]
        FROM { ARGS [ RANGE=(start,end) ] |
               VARS=xxx* [ RANGE=(start,end) ] |
               VARS=(aaa,bbb,...,ccc) }
```

Allows the MDO named aaa to be created or updated to contain the data from all the tokens in the range or list specified. If MAP=$NCL is specified, then special processing occurs to encapsulate all the source tokens in an MDO such that their individual names, attributes, and data contents are preserved. If the MAP operand is omitted, or if a map name other than $NCL is specified, then the source tokens are treated as a list of data items concatenated together to form the MDO contents. If the MAP operand is omitted, then the MDO is unmapped, and its data can only be referenced in entirety by the MDO stem name.

Example: A compound name MDO as the target of &ASSIGN

Using this form of assignment, the MDO with name *aaa* must exist or a null operation takes place. If the MDO exists, then the structure named *aaa.bbb.ccc* that is referenced is created, updated, or deleted. If no structure named *aaa.bbb.ccc* is known to Mapping Services, then no assignment takes place. If the structure is known, but the data source specified resolves to nulls, then a null assignment takes place but components are not deleted.

```
&ASSIGN MDO=aaa.bbb.ccc
```

This is used to delete the MDO structure named *aaa.bbb.ccc*. If it is a fixed length positional component, then it is set to nulls. If it is a variable length, keyed component, then it is deleted, and thus removed from the MDO completely.

```
&ASSIGN MDO=aaa.bbb.ccc FROM MDO=xxx
```

```
&ASSIGN MDO=aaa.bbb.ccc FROM MDO=xxx.yyy.zzz
```

These examples allow the MDO structure named *aaa.bbb.ccc* to be created or updated from the MDO structure nominated.

```
&ASSIGN MDO=aaa.bbb.ccc DATA=xxx
```

Allows the MDO structure named *aaa.bbb.ccc* to be created or updated to contain only the data specified.

```
&ASSIGN MDO=aaa.bbb.ccc FROM VARS=xxx
```

Allows the MDO structure named *aaa.bbb.ccc* to be created or updated to contain only the data from the token *xxx*.

```
&ASSIGN MDO=aaa.bbb.ccc
        FROM {ARGS [ RANGE=(start,end) ] |
              VARS=xxx* [ RANGE=(start,end) ] |
              VARS=(aaa,bbb,...,ccc) }
```

Allows the MDO structure named aaa.bbb.ccc to be created or updated to contain the data from all the tokens in the range or list specified. The source tokens are treated as a list of data items concatenated together to form the MDO structure contents.

Example: A stem name MDO as the source of &ASSIGN

In these assignments the MDO stem name specified following the FROM keyword is used to select the data for the assignment.

```
&ASSIGN MDO=aaa [ MAP=map ] FROM MDO=aaa
```

Allows the MDO named *aaa* to be created or updated as a copy of the entire contents of the MDO named *xxx*. If the MAP operand is omitted, then the map name of the source MDO is assumed.

```
&ASSIGN MDO=aaa.bbb.ccc FROM MDO=aaa
```

Allows the MDO structure named *aaa.bbb.ccc* to be created or updated from the MDO structure nominated.

```
&ASSIGN { VARS=aaa |
          VARS=(aaa,bbb,...,ccc) |
          VARS=aaa* [ RANGE=(start,end) ] |
          ARGS [ RANGE=(start,end) ] }
        FROM MDO=aaa
```

The variables that are the target of the assignment are assigned from consecutive 256-byte sections of the MDO name *aaa*.

Example: A compound name MDO as the source of &ASSIGN

In these assignments, the MDO compound name specified following the FROM keyword is used to select the data for the assignment.

```
&ASSIGN MDO=aaa [ MAP=map ] FROM MDO=aaa.bbb.ccc
```

Allows the MDO structure named *aaa.bbb.ccc* to be created or updated from the MDO structure nominated.

```
&ASSIGN MDO=aaa.bbb.ccc FROM MDO=aaa.bbb.ccc
```

Allows the MDO structure named *aaa.bbb.ccc* to be created or updated from the MDO structure nominated.

```
&ASSIGN { VARS=aaa |
          VARS=(aaa,bbb,...,ccc) |
          VARS=aaa* [ RANGE=(start,end) ] |
          ARGS [ RANGE=(start,end) ] }
        FROM MDO=aaa.bbb.ccc
```

The variables that are the target of the assignment are assigned from consecutive 256-byte sections of the MDO component *aaa.bbb.ccc*.

After using any verb that references an MDO, the MDO return code (&ZMDORC) and feedback (&ZMDOFDBK) system variables are set. Therefore you should check most verbs when using operations involving MDOs. The possible values of the return code and feedback system variables and their meanings are shown in the following table.

&ZMDORC	&ZMDOFDBK	Meaning
0	0	OK
4	0	Null: optional component present but empty, or null data assigned to optional component
	1	Null: optional component not present
	2	Null: mandatory component present but empty, or null data assigned to mandatory component
	3	Null: mandatory component not present
	4	String was truncated (applies to FIX offset or length components only)
8	0	Type check: data is invalid for type
	1	Data check: data is invalid structurally-a common cause is data too long or too short
	2	Length check: maximum MDO length exceeded
12	0	Name check: component not defined
	1	Name check: index position invalid or value is out of range
16	0	Map check: map not found
	1	Map check: map contains errors, load failed
	2	Map check: map/data mismatch

If &CONTROL MDOCHK is in effect, then an &ZMDORC value of 8 or more causes the NCL procedure to abend. If &CONTROL NOMDOCHK (the default) is in effect, you should check the values of &ZMDORC and &ZMDOFDBK after any verb involving MDOs.

&BOOLEXPR

The &BOOLEXPR built-in function validates and/or tests a Boolean expression.

This built-in function has the following format:

```
&BOOLEXPR [ SUBCHAR={ & | c } ]
          [ EVAL={ YES | NO } ]
          [ FOLD={ * | YES | NO } ]
          { DATA=expression | VARS=prefix* RANGE=(start,end) }
```

The &BOOLEXPR built-in function allows you to write a complex Boolean expression, and supply it, either directly or indirectly, through a set of variables. The expression is analyzed for syntactical correctness, or evaluated, and the logical result made available.

The expression syntax is flexible and supports the full use of AND, OR, and NOT, and unlimited complexity and use of parentheses.

The function returns one of the following values:

VALID

Indicates that the Boolean expression is valid, but the EVAL=NO operand suppressed evaluation.

INVALID

Indicates that the Boolean expression has an error. &SYSMSG contains a description of the error.

BAD

Indicates that the Boolean expression is valid, but a data error has been encountered during evaluation. &SYSMSG contains a description of the error.

0

Indicates that the Boolean expression has been evaluated and is false.

1

Indicates that the Boolean expression has been evaluated and is true.

The &BOOLEXPR function does not change the content of &SYSMSG when the return values are VALID, 0, or 1.

Operands:

SUBCHAR={ <u>&</u> | *c* }

(Optional) Allows you to alter the variable substitution character from its default value of an ampersand (&). This ability prevents the substitution of variable names with their values before the parsing of the Boolean expression, which can lead to syntax and other errors.

The only characters permitted for the substitution character are & (the default), %, !, ?, ~, and '.

None of these characters are valid outside quoted data in the expression in any other context.

When supplying an expression using the DATA= operand, without the expression itself being contained in variables, this operand is probably required. For example, the following statement would result in the &A, &B, &C, and &D variables being replaced with their current contents before the evaluation of the expression. This processing sequence could possibly result in syntax errors:

```
&RESULT = &BOOLEXPR DATA=&A = &B AND &C = &D
```

Alternatively, if you specify the following statement, &BOOLEXPR itself processes the variables:

```
&RESULT = &BOOLEXPR SUBCHAR=% DATA=%A = %B AND %C = %D
```

When supplying the expression itself in variables, any embedded ampersand-prefixed variable names are correctly processed, unless &CONTROL RESCAN is in effect, for example:

```
&BOOLEXPR DATA=&EXPRESSION
```

EVAL={ <u>YES</u> | NO }

Indicates whether the Boolean expression is to be evaluated if syntactically correct or validated for syntactical correctness.

EVAL=YES

Evaluates the expression if no syntax errors are encountered. If the expression is valid, evaluation is attempted, which leads to either the 0, 1, or BAD return values.

EVAL=NO

Suppresses evaluation. If the expression is valid, the function return value is VALID.

If the expression is not valid (that is, a syntax error), the return value is INVALID in any case.

Default: YES

FOLD={ * | YES | NO }

Controls the default action to take, regarding the uppercasing of variable and quoted operands, when evaluating the expression.

FOLD=*

Specifies that the setting of &CONTROL IFCASE is honored for the standard relational operators, and the CONTAINS and LIKE operators. The strict operators do not uppercase data in this case.

FOLD=YES

Specifies that everything is uppercased.

FOLD=NO

Specifies that nothing is uppercased.

The setting of the FOLD operand is overridden on individual tests in the expression, using the FOLD and NOFOLD modifiers.

If variable uppercase translation is performed automatically on assignment, the effect of the &CONTROL UCASE option can override the intended effect of IFCASE.

Default: *

DATA=*expression* | VARS=*prefix RANGE=(*start*,*end*)**

These operands supply the source data of the Boolean expression. Either the DATA=*expression*, or the VARS=*prefix** RANGE=(*start*,*end*) operands must be supplied.

If DATA=*expression* is used, it must be the last operand, as all data following it to the end of the NCL statement is regarded as the Boolean expression.

If VARS=*prefix** RANGE=(*start*,*end*) is used, the Boolean expression is contained in a set of nominated NCL variables. The data in these variables is extracted, and concatenated together with a single blank inserted between them, and is used as the Boolean expression. The VARS= and RANGE= operands is in any order, and need not be the last operands. The start and end values must be numbers in the range 0 through 32767, and end must be greater than or equal to start.

BOOLEAN Expression Syntax

The Boolean expression must conform to the following syntax:

```
expression := term [ OR term … ]

term := factor [ AND factor … ]

factor := [ NOT … ] exp2

exp2 := ( expression ) | test
```

```
test := [ IGNORE { TRUE | FALSE } ]
        [ ANY | ALL ]
          loperand [ ,loperand,… ]
          operator
        [ ANY | ALL ]
          roperand [ ,roperand,… ]

loperand := operand

operator := = ==
            ¬= ¬==
            < <<
            > >>
            <= <<=
            >= >>=
            CONTAINS
            LIKE
            IS [ NOT ]

roperand := (For all operators except IS [ NOT ]):
              operand
              [ GENERIC | : operand ]
              [ { CHARACTER | NUMERIC | FOLD | NOFOLD } …]
              [ , … ]

            (For the IS [ NOT ] operators):
              type-name | &variable
              [ , … ]

type-name := 'constant' | “constant” | &variable | number
```

```
operand := ALPHA ALPHANUM ALPHANUMNAT
           BITLIST16 DATE1 DATE2
           DATE3 DATE4 DATE5
           DATE6 DATE7 DATE8
           DATE9 DATE10 DOMAIN
           DSN HEX MIXED
           MSGLVL N NAME
           NAME12 NAME256 NULL
           NUM REAL ROUTCDE
           SIGNNUM TIME1 TIME2
           TIME3 Y
```

Notes:

- Tests can be connected using AND (&), OR (|), NOT (¬), and parentheses (). If the ampersand (&) is a variable substitution character (as set by the SUBCHAR operand), use of an ampersand (&) for AND requires a trailing blank if a word (such as NOT) follows it.
- The expression or the nesting of parentheses can be of any complexity.
- IGNORE allows a test to be ignored and treated as if it were true (return code 1) or false (return code 0).
- The ANY and ALL prefixes allow you to override the default processing of lists of operands on the left and right of an operator.

  The left default is ANY, meaning that if any left-operand passes the test, then the test is true (evaluation stops once this is satisfied). ALL means that all left-operands must pass the test for the test to be true (any operand that fails causes evaluation to stop).

  The right default depends on the operator. For =, ==, CONTAINS, LIKE, and IS, the default is ANY. For all others, the default is ALL. If ANY is specified or defaulted, then any operator and right-operand that result in a true result for a left operand results in a true result for that operand. If ALL is specified or defaulted, then any given left-operand must pass all combinations of operator and right-operand to pass a test.
- You can have constants on both sides of any operator. In this case, the expression can be always true or false. The constants can be there if you use substitution before evaluation. However, if mixed constant types are found, for example, '123' = 123, then a syntax error is raised.

  The IS and IS NOT operators do not support constants to the left of them.

- Constant values must be quoted if they are non-numeric, or if they are numeric but a character comparison is wanted. Numbers must always be unquoted.

  Quoted constants can use either single quotes (') or double quotes (”). Use two adjacent quote characters to represent a quote character in the value. Alternatively, use the alternate quote character to surround the value.

  Numbers are recognized and validated based on the current &CONTROL INTEGER/REAL setting.

- The GENERIC modifier is only permitted with a quoted constant or variable to the right of an operator, and only for the equal (=) and not equal (¬=) operators. The modifier implies a CHARACTER test.

- The RANGE test (:) is permitted with the equal and not equal operators only. Both sides of the range must be the same type, that is, you cannot say *constant* : *number*, or *number* : *constant*. If one side of the range is numeric and the other is a variable, then the variable must have a numeric value. Otherwise, BAD is returned.

  If the from and to range values are reversed, they are reversed before making the test, rather than raising an error.

- If a variable is compared to a number and the variable value is null or not numeric, then BAD is returned.

- The CHARACTER modifier forces a character compare and is only required when comparing a variable to another variable. This modifier avoids the default action of testing both sides for numerics and performing a compare based on the result.

  Similarly, the NUMERIC modifier forces a numeric compare. If either variable is not numeric, BAD is returned. The NUMERIC modifier is invalid with the GENERIC modifier.

- The FOLD and NOFOLD modifiers force uppercasing (FOLD) or prevent uppercasing (NOFOLD) of character values. The modifiers override the following settings:

  - Default for the operator
  - As set by the FOLD= operand on the &BOOLEXPR invocation
  - Default based on the &CONTROL IFCASE setting

- The operators are:

  = (equal)—Standard equality test

  ¬= (not equal)—Standard inequality test

  < (less than)—Standard less than test

  > (greater than)—Standard greater than test

  <= (less than or equal)—Standard less than or equal test

  >= (greater than or equal)—Standard greater than or equal test

  The standard operators strip both leading and trailing blanks, except for equal and not equal, when used with the GENERIC modifier. The comparison then pads with trailing blanks if necessary.

  When variables are on both sides of the operator, the standard operators, by default, type check both variables. If both are numeric, the operators perform a numeric compare. Otherwise, a character compare is performed. Specification of the GENERIC or CHARACTER modifiers overrides this test to force a character compare. Specification of the NUMERIC operator forces a numeric compare. In this case, the operands must be valid numbers as determined by the current &CONTROL INTEGER or REAL setting.

  For character compares, these operators honor the setting of &CONTROL IFCASE by default. The FOLD and NOFOLD modifiers override this setting, as does the FOLD=YES or NO operand. If variable uppercase translation is performed automatically on assignment, the effect of the &CONTROL UCASE option can override the intended effect of IFCASE.

== (strictly equal to)

¬== (strictly not equal to)

<< (strictly less than)

>> (strictly greater than)

<<= (strictly less than or equal to)

>>= (strictly greater than or equal to)

The strict operators always perform a character compare. No blank stripping or padding is performed. Numeric constants are invalid with these operands. If one value is physically shorter than another value, but equal for the shortest length, then the shorter value is regarded as logically less than the longer value. By default, values are not uppercased, although the FOLD and NOFOLD modifiers, and the FOLD= operand can override this default. The setting of &CONTROL IFCASE is ignored for these operators.

CONTAINS

The CONTAINS operator searches the left operands for the right-hand values. Variable values on the left have one leading and one trailing blank added to their value before to the search. CONTAINS honors &CONTROL IFCASE for folding, by default, which can be overridden as for the standard operators.

LIKE

The LIKE operator performs a pattern-match, as for the ANSI SQL LIKE operator. The standard wildcard characters are used. (They are: the per cent sign (%) which matches 0 or more characters, and the underscore character (_) which matches one character.) LIKE honors &CONTROL IFCASE for folding, by default, which can be overridden as for the standard operators.

IS and IS NOT

The IS and IS NOT operators perform a type check on the nominated variables. Only variables are specified to the left of these operators. The listed types are verified, as for &TYPECHK (see page 654).

If a variable is specified in the type list, then during expression evaluation it must contain a list of type names, separated by blanks or commas. If it is null, it is treated as if it contained types that the left variables all passed. If it contains invalid data, then the expression returns a BAD result.

Use of these operators can prevent BAD checks on non-numeric variables.

Examples: &BOOLEXPR

This example shows how you can use &BOOLEXPR to evaluate a complex condition that is beyond the capabilities of the &IF or &DOWHILE/&DOUNTIL NCL statements:

```
&R = &BOOLEXPR SUBCHAR=% DATA=+
%NAME = 'FRED SMITH' AND +
  %DOB < 700101 AND +
 (ANY %SKILL1, %SKILL2, %SKILL3, %SKILL4 = 'PROGRAMMER' +
OR +
   %RELATIVES CONTAINS 'Managing Director')
&IF &R = 1 &THEN &DO
:
&DOEND
```

This example shows how a dynamic expression is validated for syntactical correctness. The expression is contained in the NCL variables &EXPR1 to &EXPR10.

```
&R = &BOOLEXPR SUBCHAR=% EVAL=NO VARS=EXPR* +
 RANGE=(1,10)
&IF &R = INVALID &THEN &DO
:
&DOEND
```

This example shows how the dynamic expression of the last example is processed at a later time. The result is checked for the BAD value in case a non-numeric variable value or nonvalid type list variable was encountered. Otherwise, whether the result is true is checked.

```
&R = &BOOLEXPR SUBCHAR=% EVAL=YES VARS=EXPR* +
 RANGE=(1,10)
&IF &R = BAD &THEN &DO
   :
&DOEND
&ELSE
&IF &R = 1 &THEN &DO
   :
    &DOEND
```

Return Codes:

The &BOOLEXPR function typically returns a value that is assigned to the target variable on the assignment statement.

Syntax errors in the various operands, such as SUBCHAR or FOLD, result in the NCL process being abnormally terminated. Syntax errors in the Boolean expression or data validity errors never result in process termination.

The following return values are possible when using EVAL=YES (the default):

INVALID

The Boolean expression is not valid. &SYSMSG contains a descriptive message.

BAD

The Boolean expression is valid, but a variable referred to in the expression contained data that is not valid for the operator. Only the following conditions can return this value:

- Non-numeric data or null value when a numeric value is required (such as when a test has &*variable* = *number*)
- A variable is expected to contain a list of valid type names

&SYSMSG contains a message describing the error.

0

The Boolean expression is valid and evaluated to false.

1

The Boolean expression is valid and evaluated to true.

The following return values are possible when using EVAL=NO:

INVALID

The Boolean expression is not valid. &SYSMSG contains a message with additional information.

VALID

The Boolean expression is valid. If EVAL=YES had been specified, evaluation would have been attempted.

No other values are returned.

Notes:

&BOOLEXPR processes in the following order:

1. First, the operands of the &BOOLEXPR function itself are verified.

   That is, the EVAL, FOLD, SUBCHAR, VARS, RANGE, and DATA operands. Errors in these operands result in the NCL process being terminated.

2. If the VARS/RANGE operands are used, the nominated variables are retrieved. Their values are concatenated together with blanks between them to form the source of the Boolean expression.

3. The Boolean expression is then compiled into a parse tree. Errors during compilation result in the INVALID return value.

4. If EVAL=NO was specified, execution is complete and the return value is VALID.

5. If EVAL=YES was specified, the parse tree is interpreted. Variables are retrieved and tests are performed.

   Invalid variable values (numeric or type lists) result in the interpretation being terminated with a BAD return value.

   Evaluation of AND and OR lists is from left to right. The first false (for AND) or true (for OR) test will short-circuit the rest of the tests at that level.

   This left-to-right order means that you can specify, for example:

   %VAR IS SIGNNUM AND %VAR = 1:10

   This expression does not return a BAD value. Only if the value in &VAR is both numeric and 1 through 10, is the value 1 (true) returned. In all other cases, the value 0 (false) is returned.

   ANY/ALL evaluations are processed left-to-right, with the list entries on the right being performed for each list entry on the left in order.

   Note: Use of &*variables* with DATA= can cause too-early evaluation of variable contents, leading to syntax errors in the expression.

More information:

&TYPECHK (see page 654)
&IF (see page 364)
&DOWHILE (see page 306)
&DOUNTIL (see page 304)

&CALL

The &CALL verb invokes an NCL procedure or a user program.

This verb has the following formats:

```
&CALL PROC=procname
    [ SHARE | SHARE=(sharvars1,sharvars2,...,shrvarsn) |
      NOSHARE=(sharvars1,sharvars2,...,shrvarsn) ]
    [ PARMS=(parm1, parm2,...,parmn) ] }

&CALL progname
    [ data1  data2 ... datan  ]

&CALL PGM=pgmname
    [ PARMLIST={ OLD | NEW }
    [ data1  data2 ... datan  ] ]

&CALL SUBSYS=ssname
    [ data1  data2 ... datan  ]
```

An NCL procedure uses &CALL to nest to another procedure or to invoke a program developed within your installation, for performing specialized processing.

More information:

&CALL procedure (see page 235)
&CALL program (see page 238)

&CALL procedure

This format invokes an NCL procedure to nest to another procedure.

```
&CALL PROC=procname
    [ SHARE | SHARE=(sharvars1,sharvars2,...,shrvarsn) |
      NOSHARE=(sharvars1,sharvars2,...,shrvarsn) ]
    [ PARMS=(parm1, parm2,...,parmn) ] }
```

When using &CALL to nest to another NCL procedure, the resulting call path is equivalent to the result from an EXEC command. However, the ability to pass parameters and variables across the call boundary is more structured using the &CALL verb than the EXEC command.

Operands:

PROC=*procname*

Specifies a nested NCL procedure call to the procedure *procname*. The &CALL verb does not complete until the called procedure ends.

SHARE | SHARE=(*shrvars*1,*shrvars*2,...,*shrvarsn*) |
NOSHARE=(*shrvars*1,*shrvars*2,...,*shrvarsn*)

Specifies the set of NCL variables and/or MDOs, to be shared or not shared, with the called procedure.

If the SHARE or NOSHARE keywords are omitted, no variable sharing occurs for this call. If the SHARE keyword is coded without parameters, the current &CONTROL SHRVARS (or NOSHRVARS) setting is used for the call.

If the SHARE or NOSHARE keyword is used with parameters, any current &CONTROL SHRVARS settings are ignored for this execution. When SHARE is used, the called procedure obtains a copy of each variable and/or MDO referenced by the SHARE operand. When NOSHARE is used, the called procedure obtains a copy of all NCL variables and MDO data not specified by the operand.

Note: The parameter values for SHARE and NOSHARE cannot be substituted on the statement.

The operand can specify a single value or a list of values. Each item in the list can reference a single variable or MDO, or multiple variables or MDOs. A single variable is referenced by including its entire name. For example, an entry in the list of 'ABC' refers to the single NCL variable &ABC, as though VARS=ABC were coded. A single MDO is referenced by including its entire name followed by a full stop. For example, an entry in the list of 'ABC.' refers to the single MDO named ABC, as though MDO=ABC were coded.

A range of NCL variables is referenced by including a name prefix followed by an asterisk. For example, an entry in the list of '$CNM*' refers to $CNM1...$CNM*nnn*. The range is set explicitly. For example, '$CNM*(1,10)' (or '$CNM(1,10)') refers to the variables $CNM1 to $CNM10. If not set explicitly, all variables are assumed to be of the form $CNM*nnn*.

A generic list of NCL variables and MDOs is referenced by including a name prefix followed by a > symbol. For example, an entry in the list of '$NW>' refers to all variables, and MDOs, with names beginning with &$NW.

PARMS=(*parm*1,*parm*2,...,*parmn*)

Specifies a list of parameters to pass to the procedure. You enclose the parameter list by parentheses and separate each parameter by a comma.

The parameter list can contain any combination of characters, including variables. The first parameter isolated is placed in &1 in the target procedure, the next in &2, and so on.

The list is analyzed before substitution occurs, and each parameter is isolated by scanning for a comma or the closing parenthesis. If another opening parenthesis is encountered, a syntax error results.

If a single or double quote is encountered as the first character of a parameter, the entire parameter is assumed to be quoted, otherwise it is treated as unquoted.

If an unquoted parameter is encountered, the next comma or closing parenthesis delimits it. Any other characters are considered part of the parameter itself (including embedded blanks).

When a parameter does not start with a quote, but contains quotes, normal quoting rules apply and the data is substituted. In the following example, &1 is PARM1='ABC' and &2 is PARM2=XY Z for MYPROC:

```
&VAL = C
&CALL PROC=MYPROC PARMS=(PARM1='AB&VAL',PARM2=XY Z)
```

Once isolated, substitution is performed (if necessary, allowing transparent data to be passed as parameters) and the result placed in the next initialization parameter in the called procedure.

A closing quote of the same type as the opening quote terminates a quoted parameter. Only a comma delimiting the next parameter, or a closing parenthesis terminating the entire parameter list, can immediately follow the closing quote. The entire quoted string is passed to the target procedure unchanged, except that the delimiting quotes are removed.

Normal quote rules apply, that is, two consecutive quotes of the same type as the opening quote are treated as a single occurrence in the resulting string. No substitution is performed on the contents of the quoted string. For example:

```
PARMS=(&USER,,PROC=&0,”variable ””&FRED”” in error”)
```

would set the following variables in the called procedure (assume &USER has the value 'ADMIN', &0 'MYPROC', and &FRED 'xyz'):

```
&1 ADMIN
&2
&3 PROC=MYPROC
&4 variable ”&FRED” in error
```

&RETCODE or &END in the called procedure sets the return codes.

Note: If used, the PARMS operand must be the last operand on the &CALL statement.

Example: &CALL

```
&CALL PROC=PROC01 SHARE=(???,msg(1,10),VARS>) +
      PARMS=(&USER,,”inactive”)
```

&CALL program

These formats invoke a user program developed at your site for performing specialized processing:

```
&CALL progname
    [ data1 data2 ... datan ]

&CALL PGM=pgmname
    [ PARMLIST={ OLD | NEW } ]
    [ data1 data2 ... datan ]

&CALL SUBSYS=ssname
    [ data1 data2 ... datan ]
```

While NCL can perform many functions, individual installations can have particular processing requirements that need a unique facility for that function. The called program receives data and the contents of variables by including them on the &CALL statement. The called program updates this data and returns the updated data for further NCL processing.

Data returned from a called program is always returned in variables &1, &2, through to &*n*, regardless of the variable names that have been specified in the &CALL statement itself.

The number of variables available after the &CALL statement matches the number of data fields specified on the &CALL statement.

Variables with null values when the &CALL statement was issued always return a variable (for example, &5) equivalent to their position on the &CALL statement. This variable has a null value unless data is supplied from the called program.

The called program or subsystem can set a return code (in Register 15 if written in Assembler) in the &RETCODE system variable when processing resumes at the statement after the &CALL statement. This return code is in the range 0 through 99. Anything outside this range is an error, and either aborts the process or sets &SYSMSG. In this case, &RETCODE is set to 100.

The program can also return 4 bytes of information (termed a correlator) that are saved and passed to any program called later within the NCL process.

Operands:

progname

Indicates the &CALL target. If the program abends or returns a code greater than 100, the NCL process is terminated with an error.

If *pgmname* is equal to the name of a subsystem, which was defined with the CALLREPL=YES option, the call is rerouted to that subsystem.

PGM=*pgmname*

Causes the call to be performed as for *progname* except that, if the program abends or returns an invalid return code, the NCL process is not terminated. Instead, &RETCODE is set to 100 and &SYSMSG contains the error message that would have been issued.

No SUBSYS replacement takes place for PGM=*pgmname*. A one-time attach always occurs.

This operand must be specified if you want to use the new format parameter list. Specifying *progname* causes a default to PARMLIST=OLD.

SUBSYS=*ssname*

Queues the call to the nominated subsystem. If it is not found, the process is aborted. Other errors result in &RETCODE of 100 and &SYSMSG being set. PARMLIST= is not permitted in this case. If specified, it is treated as the first parameter to the subsystem program. The subsystem definition defines the parameter list format.

PARMLIST={ OLD | NEW }

Indicates the format of the parameter list to use. The operand is not recognized if specified with SUBSYS=*ssname*, but is treated as the first user-program parameter.

Default: OLD

PARMLIST=OLD

Specifies to use the old-format parameter list.

Note: This format parameter list is not compatible with high-level-language programs.

The format of this type of parameter list is:

```
SECURITY EXIT CORRELATOR
&CALL CORRELATOR AREA
Register 1 ----> COUNT
ADDR1 ----> xxxxdata (xxxx is length of data)
ADDR2 ----> xxxxdata (xxxx is length of data)
 .
 .
ADDRn ----> xxxxdata (xxxx is length of data)
```

All passed parameters are padded with blanks to 256 characters. The program can return up to the number of passed parameters, by resetting the values and lengths.

PARMLIST=NEW

Specifies to use the new-format parameter list. This parameter list contains much new information.

The parameter list is as follows:

```
R1 ---> A(parmlist) (that is, a word that points to itself)
 A(NCPF)
 A(NCPE)
 A(NCPS)
 A(NCPU)
 A(NCPN)
 A(NCPZ)
 A(NCPC)
 A(data1)--->f'len',cl256'data'
 A(data2)--->f'len',cl256'data'
```

Each of the parameter areas is as follows:

- A(*parmlist*) points to the parameter list itself.

  This area is the key to allowing a program to determine whether the parameter list is in the old or new formats. Recall that the old format parameter list had as the first parameter the number of passed parameters.

  This value is a binary fullword in the range from 0 to approximately 1000 (maximum with a 1024-byte statement buffer).

  In the new format, the first parameter points to itself. Because a parameter list cannot start in the first 4096 bytes of storage (page 0), the following simple test allows you to determine easily whether the PARMLIST is old or new format:

  ```
  C R1,0(,R1)
  BE NEWFORMAT
  B OLDFORMAT
  ```

- A(NCPF) points to an area that contains function code information, mapped as follows:

  ```
  NCPF     DESECT
  NCPFFUNC DS     F  function code
  NCPFFCAL EQU    00 00 - &CALL
  NCPFFTRM EQU    04 04 - NCL proc terminate
                          (SUBSYS only)
  NCPFFSIN EQU    08 08 - SUBSYS initialize
  NCPFFSTM EQU    12 12 - SUBSYS terminate
  NCPFFSST EQU    16 16 - system shutdown
                          (SUBSYS only)
  ```

  Only defined subsystems ever receive any code other than 0.

- A(NCPE) points to an area that contains environment information. This information is used to find out about your product region. Some VTAM information is also provided.

  The NCPE is mapped as follows:

  ```
  NCPE     DSECT
  NCPEVERS DS    CL4  version, for example, 'V5.1'
  NCPEID   DS    CL12 NMID
  NCPEDMN  DS    CL4  domain ID
  NCPEACB  DS    CL8  ACBNAME
  NCPESSCP DS    C8   SSCP name
  NCPENETI DS    CL8  NETID name
  ```

- A(NCPS) points to an area containing subsystem environment information. If this call is not to a defined subsystem, this area is still provided. The code can detect that a subsystem is not defined by testing NCPSNAME. If not a subsystem, the value CL8'*' is inserted. The subsystem can use the correlator field to anchor subsystem-related control blocks across calls (for example, to preserve complete reentrancy).

  The NCPS maps as follows:

```
NCPS     DSECT
NCPSNAME DS    CL8 subsystem name or asterisk (*)
                   if not a subsys call
NCPSPGMN DS    CL8 program name
NCPSSCOR DS    F   subsystem correlator
```

- A(NCPU) points to an area that maps user ID information. This information is about the user ID that is executing the procedure that issues the &CALL.

  The NCPU is mapped as follows:

```
NCPU     DSECT
TNCPUID  DS    CL8 User ID
NCPULU   DS    CL8 Logical unit (terminal name)
NCPUWIND DS    F   Window number (0 to 1)
NCPUSECC DS    F   Security correlator
NCPUTOKN DS    F   Address of Security Exit User
                   Token supplied by security exit.
```

- A(NCPN) points to information about the NCL process that issued this &CALL. This information includes the name of the current procedure, and the NCL ID. Also, a shared correlator and a private correlator (SUBSYS only) are provided.

  The NCPN maps as follows:

```
NCPN     DSECT
NCPNPROC DS    CL8 Procedure name (that issued
                   this &CALL)
         DS    CL4 Reserved
NCPNNCLI DS    F   NCLID (in binary) 1-999999
NCPNSCOR DS    F   Correlator shared with all
                   other &CALLS for proc
NCPNPCOR DS    F   SUBSYS only private correlator
                   for this proc
```

- A(NCPZ) points to a null area. This area is reserved for future expansion. The pointer is set (it is not 0). In a high-level language define it as a single binary number (S9(9) COMP in COBOL, FIXED BIN(31) in PL/1).

- A(NCPC) points to the count area. This section contains the count of the number of user parameters passed.

  The NCPC maps as follows:

  ```
  NCPC    DSECT
  NCPCCNT DS    F Number of passed user parameters
                  (0 to n)
  ```

  Following the pointer to the NCPC, is a list of n pointers to user parameters. Each points to a 4-byte (binary) length (0 through 256), followed by the data, padded with blanks to 256 characters. As for old-format parameter lists, these values can be updated for return in &1, &2, &3, and so on. Only as many parameters as are passed are returned.

A macro, $NMNCPL, is distributed with your product. This macro maps out the new format parameter list.

data*1 *data*2 ... *datan

(Optional) Specifies other parameters to pass on the call. Each is a constant, a simple variable name, or constant data followed by a variable name:

- *constant*
- &*variable*
- *constant*&*variable*

No complex substitution is allowed. The variable can contain any hexadecimal values. These values are passed unchanged. Similarly, the returned data is placed, unchanged, in the returned variables &1, &2, through to &*n*.

Examples: &CALL

```
&CALL &PROCNAME USERID &2
&CALL &DATECONV &MYDATE
```

Notes:

Variables passed to and from the called program can contain any hexadecimal values, printable or non-printable. No translation is made of the data returned by the program - it is the responsibility of the NCL procedure to determine any meaning for data passed to and from the program.

&CALL always returns variables named &1, &2 through to &n. If the procedure has already defined variables with these names, the contents of those variables are replaced with the information returned from &CALL. Therefore, be careful when using such variable names with &CALL.

In z/OS and z/VM environments, the specified program is attached to your product region as a subtask, and can therefore issue waits or I/O requests without affecting the rest of the system. If storage is GETMAINed for subsequent calls to this or another program, the &CALL correlator is used to remember its address.

However, obtain the storage in subpool 50 to avoid freeing it automatically when the program returns to your product region. The program is responsible for ensuring that any storage obtained in subpool 50 is freed when no longer required.

By default, subpool zero is not shared between your product region and the &CALL subtask. If you are using operating system functions in the subtask (for example, native VSAM) that require sharing subpool zero, you can remove this restriction with the SYSPARMS CALLSHR0=YES command.

If the program terminates abnormally, the invoking NCL procedure terminates with an appropriate error message which includes the failure reason.

The specified program must be available in the standard load module libraries available to your product region.

If many concurrent users of the program are anticipated, consider making the program reentrant and loading it into your product region in advance. You can use the LOAD MOD=*module_name* command to specify the name of a module to load into your product region.

When the loaded module has the RENT or REUS attributes, subsequent references to this module from &CALL use the loaded copy, avoiding further retrieval. The LOAD commands are typically placed in the NMINIT initialization procedure. If these modules are link edited with the reentrant attribute, they must be reentrant. Otherwise, unpredictable results occur when they are invoked. Use the UNLOAD MOD=*module_name* command to delete previously loaded programs from storage.

Complex variables cannot be used in &CALL statements.

See also UTIL0005 in the distribution library.

&CMDLINE

The &CMDLINE verb writes the supplied text into your OCS command input line.

This verb has the following format:

```
&CMDLINE command_text
```

The &CMDLINE verb gives you the ability to put command text from within an NCL procedure into the command input line of your OCS window. Any command text is written and the cursor positioned on any location within that text.

The verb is used to prompt for operator input. A default command text is put directly into the command line for immediate entry or modification before entry. These features are used to simplify operator responses.

Operands:

command_text

Specifies any uppercase and lowercase text to place in the operator command entry line. The text starts from the second position after the &CMDLINE keyword and leading blanks are preserved. If the specified text exceeds the length of the command entry line it is truncated. Any text currently in the command entry line is erased when the &CMDLINE statement is executed.

The cursor is positioned at any location within the text by the including an underscore (_) at the point where the cursor is required. Only the first occurrence of an underscore is interpreted. If an underscore is not specified or lies outside the current command entry line boundary, the cursor is positioned at the end of the text.

Examples: &CMDLINE

```
&CMDLINE INACT &NODE
```

```
&CMDLINE INTQ ID=&ZNCLID DATA=_CONTINUE
```

Notes:

&CMDLINE generates a standard message.

The underscore character does not take up a character position within the displayed text (it is deleted before display).

&CMDLINE can be used in a procedure executing under a Remote Operator Facility region. The text returned to the OCS command line is automatically prefixed with a ROUTE command to the remote system.

If used within a procedure that is not executing under OCS (for example, system level NCL processes or NCL processes executed in a dependent processing environment), the owning environment receives the message.

A procedure can use the &ZOCS system variable to determine whether it is running under an OCS environment.

&CNMALERT

The &CNMALERT verb sends a CNM record directly to CNMPROC in a local or remote NEWS system for processing.

This verb has the following format:

```
&CNMALERT [ MODE={ CREATE | INSERT } ]
          [ LINKNAME=linkname | SSCPNAME=sscpname | DOMAIN=domain ]
          [ &var &var .... &var |
            MDO=mdoname | VARS=... |
            ARGS [ RANGE=(start,end) ] ]
```

The &CNMALERT verb gives you the ability to queue a CNM record to a CNMPROC, running in a local or remote NEWS system, for processing. Any NCL procedure can use this verb. Using this verb, a procedure can generate alerts to make operators aware of events in the system or network, and alerts that simulate a network event to test NEWS processing procedures.

Operands:

MODE={ CREATE | INSERT }

Determines the context in which the receiving CNMPROC processes the alert.

MODE=CREATE

Places the alert on the queue of records waiting to be processed.

MODE=INSERT

Places the alert as the first on the queue so the next &CNMREAD issued by CNMPROC results in CNMPROC receiving this alert. This option is used to create a record that maintains some context with the record CNMPROC is processing.

The MODE=INSERT option is only valid when &CNMALERT is issued from CNMPROC and procedures which are nested (executed) under CNMPROC. Any other NCL procedure that attempts to use the MODE=INSERT option terminates with an error message.

LINKNAME=*linkname* | SSCPNAME=*sscpname* | DOMAIN=*domain*

Are mutually exclusive operands used to direct the alert to the CNMPROC running in the specified region. NEWS uses the Inter-System Routing (ISR) feature to communicate with other product regions running on remote hosts, and therefore can route any CNM request to a remote host for processing. The link, SSCP, or domain ID uniquely identifies the destination host. For remote processing to be possible, the INMC link to the remote host must be active and the NEWS facilities in ISR must be enabled for unsolicited flows. If the destination name is that of the local system, no ISR delivery is necessary and the record is delivered to the local CNMPROC in the usual manner.

&var &var .... &var

Specifies user variables that contain the CNM record to forward to VTAM across the CNM interface.

Each variable contains valid hexadecimal characters that, when concatenated, form the entire CNM RU. Where a Network Services RU (NS RU) is sent to VTAM embedded in a Forward RU, the variables must include the Forward RU with the target resource and PU vectors appended. In addition, set all relevant RU byte values and length fields appropriately, with the exception of the 12 bits in the CNM header comprising the procedure correlator identifier (PRID). Where NEWS recognizes from the supplied RU data that this RU is a solicitation request, it automatically generates a PRID and inserts the value in the CNM RU. The PRID, echoed by the resource in any reply RUs, is used to correlate reply data with the &CNMSEND request.

MDO=*mdoname* | VARS=... | ARGS [RANGE=(*start*,*end*)]

Specifies the data to place in the alert. If the MDO= operand is used, the data is located as formatted in the MDO.

Specifying VARS= or ARGS results in a $NCL MDO being built and delivered in the $MSG MDO, containing the named variables or arguments.

The MDO, VARS, and ARGS operands are mutually exclusive.

RANGE=(*start*,*end*) is specified with ARGS, to denote an argument range.

Examples: &CNMALERT

```
&CNMALERT &var &var ....

&CNMALERT MODE=INSERT &var &var ....

&CNMALERT LINKNAME=NMP &var &var ....
```

Notes:

&CNMALERT does not solicit information. No response is received from any &CNMALERT. The data supplied on the &CNMALERT statement is queued as is to the appropriate NEWS system for processing.

When control is returned to the NCL procedure following a &CNMALERT statement the &RETCODE system variable is set to one of the following:

0

The alert has been successfully delivered to the target CNMPROC.

8

The alert has not been successful. The &SYSMSG variable contains an explanatory message.

See also the ISR command description in the online help.

More information:

&CNMSEND (see page 261)

&CNMCLEAR

The &CNMCLEAR verb clears all outstanding CNM reply data solicited by this NCL user.

This verb has the following format:

```
&CNMCLEAR
```

The &CNMCLEAR verb is used within a procedure to clear all outstanding CNM reply data previously solicited using an &CNMSEND request. This verb would typically be invoked when a procedure in which &CNMSEND has been used is about to terminate and some possibility exists that not all solicited data has been processed. It is also useful in purging any such unprocessed data in order that the results of a subsequent &CNMSEND is guaranteed to be the next available for processing with an &CNMREAD statement.

&CNMCLEAR can be issued at any time to clear the CNM reply environment, regardless of the status of &CNMSEND/&CNMREAD processing.

No useful purpose is served by issuing an &CNMCLEAR from CNMPROC.

More information:

&CNMSEND (see page 261)
&CNMREAD (see page 255)

&CNMCONT

The &CNMCONT verb directs the current CNM record across a specific ISR link.

This verb has the following format:

```
&CNMCONT [ LOCAL |
           REMOTE |
           LINKNAME=linkname |
           SSCPNAME=sscpname |
           DOMAIN=domain ]
         [ NVC={ YES |NO } ]
```

The &CNMCONT verb is used only within the CNMPROC procedure to send the current CNM record across all or specific ISR links.

Operands:

LOCAL | REMOTE | LINKNAME=*linkname* | SSCPNAME=*sscpname* | DOMAIN=*domain*

(Optional) Specifies the system that the current CNM record is targeted at. If omitted and the CNM record contains a SNAMS MDS-MU, it is repackaged and sent to the remote SNAMS focal point, if one is active. (That is, it is equivalent to the &SNAMS SEND CATEGORY='23F0F3F1'x operation.) Otherwise, it is sent to all remote systems.

LOCAL

Sends the current CNM record to only the local CNMPROC. This option is equivalent to the &CNMDELREMOTE verb, that is, the CNM record is discarded.

REMOTE

Sends the current CNM record to all remote CNMPROCs across ISR links known to the local system.

LINKNAME=*linkname* | SSCPNAME=*sscpname* | DOMAIN=*domain*

These operands are used to direct the CNM record only to the CNMPROC running in the specified system. An asterisk (*) is used as the operand value to direct it across all ISR links, known SSCPs, or domains respectively.

NVC={ YES | NO }

Indicates whether the CNM record is sent across NetView Connect ISR links (NetView Connect does not support some CNM records supported by your product region).

Examples: &CNMCONT

```
&CNMCONT

&CNMCONT LINKNAME=NMP
```

```
&CNMCONT LOCAL
```

Notes:

When control is returned to the NCL procedure following a &CNMCONT statement the &RETCODE is set to one of the following:

0

The record has been successfully delivered to the target CNMPROC or is on its way to the remote SNAMS focal point.

8

An error has been encountered (for example, the target name specified does not exist).

See also the ISR command description in the online help.

More information:

&CNMDEL (see page 251)

&CNMDEL

The &CNMDEL verb deletes a CNM record or stops ISR delivery of the record to a remote region.

This verb has the following format:

```
&CNMDEL [ LOCAL |
          REMOTE |
          LINKNAME=linkname |
          SSCPNAME=sscpname |
          DOMAIN=domain ]
```

The &CNMDEL verb is used only within the CNMPROC to delete the current CNM record or to stop its delivery across ISR links.

Operands:

LOCAL

Specifies that the current CNM record is to be deleted from the local CNMPROC but is to be sent to all remote CNMPROCs.

REMOTE

Stops the delivery of the current CNM record across any ISR links to remote CNMPROCs.

LINKNAME=*linkname* | SSCPNAME=*sscpname* | DOMAIN=*domain*

Stops the delivery of the current CNM record to the CNMPROC running in the specified system. These operands are mutually exclusive.

Examples: &CNMDEL

```
&CNMDEL LINKNAME=NMP

&CNMDEL LOCAL
```

Notes:

When control is returned to the NCL procedure following a &CNMDEL statement the &RETCODE is set to one of the following.

0

The record has been successfully delivered to the target CNMPROC.

8

An error has been encountered (for example, the target name specified does not exist).

See also the ISR command description in the online help.

More information:

&CNMCONT (see page 250)

&CNMPARSE

The &CNMPARSE verb parses the MDO data supplied into user variables.

This verb has the following format:

```
&CNMPARSE MDO=source
        [ TYPE={ RU | SV | CV } ]
        [ MVNUM=nn ]
        [ PREFIX=prefix ]
```

This verb is typically used from an NCL procedure to format the CNM data into user variables as in an &CNMREAD TYPE=VECTOR operation.

Operands:

MDO=*source*

(Mandatory) Specifies the name of the MDO that contains the data to be formatted. The value can be a stem name or a compound MDO name.

TYPE={ RU | SV | CV }

(Optional) If omitted, the type is determined from the supplied MDO, which is assumed to be mapped by the $CNM system map. The MDO data is then formatted in the fashion of the &CNMREAD TYPE=VECTOR operation.

TYPE=RU

Indicates that the supplied data is an entire CNM RU and is to be formatted in the fashion of the &CNMREAD TYPE=VECTOR operation.

TYPE=SV

Indicates that the data to be formatted contains one or more concatenated subvectors. Each subvector comprises a 1-byte length field (*l*) and a 1-byte identifier field (*x*), followed by *l*-2 bytes of data. That is, the length field contains the length of the subvector (inclusive of the length byte and identifier byte).

The vectorization process parses the data into its constituent subvectors, placing each into variables prefixed by the value specified in the PREFIX= operand and with names which reflect their contents. The variables created are of the formats &*prefix*SV*xxnn*, &*prefixSVxxnn*X, and &*prefix*SVLIST, as in the &CNMREAD TYPE=VECTOR function.

TYPE=CV

Partitions SNA control vectors. Each control vector comprises a 1-byte type field (*x*) and a 1-byte length field (*l*), followed by *l* bytes of data. Thus the length of the control vector is *l*+2, that is, the length field does not include the type or length bytes.

The vectorization process parses the data into its constituent vectors, placing each into variables prefixed by the value specified in the PREFIX= operand and with names which reflect their contents. The variables created are of the format &*prefix*CV*xxnn*, where *xx* is the hexadecimal value of the control vector type, and *nn* is its occurrence number within the given data.

MVNUM=*nn*

(NMVT records only) Specifies the number of the Major Vector within the NMVT record to be formatted. If the specified Major Vector does not exist, no variables are created.

PREFIX=*prefix*

(Optional) Specifies a prefix for the variables created during the vectorization process.

Default: $CNM

Limits: 1 to 4 characters long and conformance to the standard NCL variable naming conventions

Examples: &CNMPARSE

```
&CNMPARSE MDO=NEWSREC.NMVT.ALERT TYPE=SV PREFIX=ALRT
&ASSIGN MDO=CNM MAP=$CNM
&ASSIGN MDO=CNM.type DATA=mdsmu
&ASSIGN MDO=CNM.mdsmu FROM VARS=CNMREC*
&CNMPARSE MDO=CNM
```

Notes:

When control is returned to the NCL procedure following an &CNMPARSE statement, &RETCODE is set to one of the following.

0

Request is successful.

4

No data is supplied.

8

Parse failed.

For &RETCODE 8, &ZMDORC and &ZMDOFDBK may be set if a mapping error causes the failure.

&CNMREAD

The &CNMREAD verb makes the next CNM record available to an NCL procedure.

This verb has the following format:

```
&CNMREAD [ WAIT={ nnnn | 0 } ]
         [ TYPE={ SEGMENT | VECTOR | BOTH | RESP } ]
         [ { ARGS [ RANGE=(start,end) ] |
             VARS=prefix* [ RANGE=(start,end) ] |
             VARS={ name | (name,name,...,name) } |
             MDO=mdoname [ MAP=mapname ] } ]
```

The &CNMREAD verb is used within the CNMPROC procedure to request delivery of the next CNM record. Such a record can arrive unsolicited, or a user procedure can have solicited the record. If no CNM record is immediately available, processing of the procedure is suspended and resumes when the next CNM record arrives, unless the WAIT operand is used. Multiple &CNMREAD statements are present within CNMPROC.

&CNMREAD can also be used from standard NCL user procedures that, having sent CNM data, process any results returned. Having issued the &CNMSEND, &CNMREAD is used to read back any records that were solicited or, particularly if no data is returned, merely the response to the sent CNM request.

&CNMREAD processing always sets the &RETCODE system variable and can set the &SYSMSG variable as described in later notes. If variables are created, &ZVARCNT is set.

Operands:

WAIT={ *nnnn* | 0 }

Specifies a time-out value of *nnnn* seconds. If this time interval expires before data satisfying the &CNMREAD request arrives, control is returned to the NCL procedure at the next statement and the &RETCODE system variable is set to 12. If a value of 0 is specified, control is returned immediately even if no data is available.

The WAIT operand is invalid when the MDO operand is specified.

Limits: 0 through 9999

TYPE=SEGMENT

Specifies that the record is delivered in character format hexadecimal, with each two characters representing one byte and is divided into 256-character sections. Each of these sections is placed into the next variable specified on the &CNMREAD statement. The first 256 characters (128 bytes) are placed into the first variable, the next 256 characters into the second variable, and so on.

TYPE=VECTOR

Specifies that vectorization is used to divide the CNM RU at strategic boundaries and place each section into a user variable. All user variable names commence with the prefix $CNM. When this option is used all existing variables with names &$CNM*xxxx* are deleted. The CNM RU is then examined and new variables created depending upon the RU type and contents as follows:

&$CNMDLVR

Contains the 8-byte Deliver RU, or is null if no Deliver RU is present.

&$CNMNSRU

Contains the embedded Network Services RU (NSRU). For an NMVT RU, it contains only the first 12 bytes, while for a RECFMS Type 0 it contains only the first 20 bytes of the RU. The remainder of these RUs consists of various CNM subvectors which are available in separate variables. For all other RU types, this variable contains the entire NSRU. However, if the NSRU exceeds 128 bytes in actual length, the variable contains only the first 128 bytes expanded, or 256 hexadecimal characters.

&$CNMNSRUX

Contains the portion (if any) of the NSRU which exceeds 128 bytes (the first 128 bytes being found in character-format hexadecimal in &$CNMNSRU).

&$CNMSV*xxnn*, &$CNMSV*xxnn*X, &$CNMSVLIST, and &$CNMMVLIST

Contain the CNM subvector information for the NMVT and RECFMS Type 0 RUs. Each CNM subvector in the RU is placed in a user variable where the name reflects the contents. Variable names are of the form &$CNMSV*xxnn*, where *xx* is the hexadecimal subvector type, and *nn* is the occurrence number of the vector type in the RU. If a subvector length exceeds 128 bytes then the portion which exceeds 128 bytes is placed in a variable named &$CNMSV*xxnn*X.

Consider the subvector X'07040000000001'. The length is X'07', the subvector type is X'04', and the character string '07040000000001' would be placed in the variable &$CNMSV0401. If the subvector X'030400' followed, it would be placed in &$CNMSV0402. When all the subvector information has been formatted, the variable &$CNMSVLIST contains a list of the subvector types found and their order. For example, if it contained 040491 this means that 2 type 04 subvectors were found followed by a type 91.

&$CNMHLU, &$CNMHTPU, &$CNMHPU, &$CNMHLNK, and &$CNMHSPU

Contain hierarchy information concerning the originating device. Each variable that is set contains the eight-character name of the resource and *one* of the following four-character type codes:

- DEV for a device
- CTRL for a PU or control unit
- LINE for a telecommunications line on an NCP
- CHAN for a local channel link
- LKST for a link station
- NCP for an NCP PU
- CPU for a host

If the NSRU originated from an LU, then &$CNMHLU is set; otherwise, &$CNMHTPU contains the target PU information. &$CNMHPU contains the name and type of the controlling PU (which may be the same as &$CNMHTPU), &$CNMHLNK contains the PU's link information, and &$CNMHSPU contains the PU's boundary function PU information. These variables are only set where information is available.

&$CNMNV*nn*

Contain externally sourced data pertinent to the NSRU. Your products set these variables under certain conditions. &$CNMNV01 is set if the Network Tracking System (NTS) feature is installed. This vector contains the LU name and session partner (if known) for NSRUs associated with logical units (for example, 3274 alerts and RTM data). &$CNMNV02 can also be set when NTS is installed and contains NTS RTM objective response time information.

&$CNMMVTYPE

Contains the major vector type (that is, X'0080', X'0025') of the MV currently vectored.

&$CNMMV*nn*

Contain the major vector type of the *n*th MV. *nn* is the number of the MV.

&$CNMMVLEN

Contains the hexadecimal length of the MV currently vectored.

TYPE=BOTH

Segments the CNM data into the variables nominated on the &CNMREAD statement and sets the user variables as described for a TYPE=VECTOR read.

TYPE=RESP

Specifies that only the response to the CNM data sent using a &CNMSEND statement is returned. Any outstanding CNM RU data is still available to be read with a subsequent &CNMREAD statement.

ARGS [RANGE=(*start*,*end*)]

Nominates the user-specified variables to hold the tokenized CNM record if segmentation is requested. The ARGS operand indicates that the system generates the token names automatically as &1, &2 through to &*n*, according to the range specified by the RANGE operand.

Tokenization is performed from left to right of the record. If more variables are specified than are required, the excess variables are set to null. If too few variables are specified to contain the data, some information is lost.

VARS=*prefix [RANGE=(*start*,*end*)]**

Nominates the user-specified variables to hold the tokenized CNM record if segmentation is requested. The VARS=*prefix** operand specifies that the system generates the token names automatically as &*prefix*1, &*prefix*2 through to &*prefixn*, according to the range specified by the RANGE operand. The prefix is one to four characters in length and must adhere to the standard NCL variable naming conventions. Tokenization then proceeds as described for the ARGS operand.

VARS={ *name* | (*name*,*name*,...,*name*) }

Nominates the user-specified variables to hold the tokenized CNM record if segmentation is requested. The VARS=name operand indicates a single specific token name only. Alternatively, a list of specific token names is provided, each name separated by a comma and the entire list enclosed in parentheses. Tokenization then proceeds as described for the ARGS operand.

MDO=*mdoname* [MAP=*mapname*]

Formats that the incoming data into an MDO with the name *mdoname*. If the incoming data is mapped, then the received map name is used to connect to Mapping Services Mapping Support. If the incoming data is not mapped, then the requester is responsible to connect to Mapping Services Mapping Support if necessary. MAP=*mapname* nominates the map name which defines the mapping of the data object.

The distributed $CNM map defines the MDO received on an &CNMREAD verb.

Examples: &CNMREAD

```
&CNMREAD TYPE=BOTH VARS=(A,B,C,D,E,F,G,H,I,J)

&CNMREAD ARGS RANGE=(1,12)

&CNMREAD TYPE=VECTOR

&CNMREAD TYPE=RESP
```

Notes:

When control is returned to the NCL procedure following an &CNMREAD statement the &RETCODE (return code) variable is set to one of the following:

0

For an &CNMREAD request other than TYPE=RESP, this return code indicates that data has arrived and is now placed in the variables as described for the &CNMREAD statement.

For an &CNMREAD TYPE=RESP request, this return code indicates a positive response was received following an &CNMSEND request.

4

Indicates that no data is outstanding. This return code is expected when an &CNMREAD request is issued in the following circumstances:

- Previous &CNMREAD statements have processed all reply data solicited using &CNMSEND.
- No reply data was returned following an &CNMSEND statement and a previous &CNMREAD TYPE=RESP has been issued.
- No &CNMSEND has been issued which specified that the results be returned to the soliciting user, or no &CNMSEND has been issued since the last &CNMCLEAR statement.

8

Indicates that some error, such as a negative response, has been encountered. The &SYSMSG variable is set and contains text explaining the actual error. An &CNMCLEAR is not required following such an error as an implicit clear is performed when the &CNMREAD completes.

12

Is set when the WAIT=*nnnn* option is used to indicate that the time interval specified expired before any data satisfying the particular request arrived.

An &CNMREAD request issued from CNMPROC always returns with the &RETCODE value set to 0 unless the WAIT=*nnnn* option was also specified. In this case an &RETCODE value of 12 is possible. The use of an &CNMREAD TYPE=RESP request from CNMPROC serves no useful purpose, as control is returned immediately with &RETCODE set to 0. While you are testing and developing a CNMPROC procedure, you can terminate the current version and invoke a new copy. Use the SYSPARMS CNMPROC=FLUSH command, which is designed to force termination of CNMPROC, followed by a SYSPARMS CNMPROC=*procname* command to restart CNMPROC.

For a user procedure, a reply to a &CNMSEND satisfies the &CNMREAD request for data. If multiple RUs are returned from a single solicitation, then each can be read in sequence by subsequent &CNMREAD statements. After the last reply RU is read a subsequent &CNMREAD returns with &RETCODE set to 4 (unless further &CNMSEND statements have been issued). Similarly, if multiple &CNMSEND solicitations are performed before issuing any &CNMREAD statements, each &CNMREAD request for data is satisfied by a reply RU until all RUs from all solicitations have been processed. All reply RUs are presented, in the order of their arrival from VTAM, for the first such request before any are returned for the next request. After all RUs from all requests have been processed in this manner a subsequent &CNMREAD completes with &RETCODE set to 4.

While reply RUs for a particular solicitation are available (that is, have not yet been processed by an &CNMREAD request), an &CNMREAD TYPE=RESP returns the response for that solicitation, even when subsequent &CNMSENDs are issued (that is, further requests have queued response information). After all reply RUs are processed for a solicitation, or where the &CNMSEND request did not solicit and an &CNMREAD TYPE=RESP has processed the response, a subsequent &CNMREAD TYPE=RESP returns the response from the next &CNMSEND statement.

Where an &CNMREAD request for data follows a single &CNMSEND request that does not solicit data, the &CNMREAD completes with an &RETCODE value of 4. However, if any subsequent &CNMSEND statement that did solicit data was issued before the &CNMREAD statement, data from the first such request would satisfy the &CNMREAD. In either case, the response information to those intervening requests which did not satisfy a &CNMREAD is lost, unless an error is detected in any such request. Detection of an error condition always results in &RETCODE being set to 8 and the &SYSMSG variable set, regardless of the type of &CNMREAD issued.

You are responsible to maintain synchronization between data sent and received across the CNM interface. The &CNMCLEAR NCL statement can assist in this respect.

If the TYPE=VECTOR or TYPE=BOTH operands are specified, the &$CNMSV* variables created (except for &$CNMSVLIST) have the Modified Data tag set. This setting allows the CNM information in them to be processed using the &ASSIGN verb or the &ZMODFLD system variable. For more information, see the descriptions of &ASSIGN and &ZMODFLD in this reference.

See also the distributed CNMPROC, $NWCNMPR.

More information:

&CNMCLEAR (see page 249)
&CNMSEND (see page 261)

&CNMSEND

The &CNMSEND verb sends the data supplied across the CNM interface.

This verb has the following format:

```
&CNMSEND [ READ={ USER | CNM | BOTH } ]
         [ LINKNAME=linkname | SSCPNAME=sscpname | DOMAIN=domain ]
         [ &var &var .... &var |
           MDO=mdoname |
           VARS=... |
           ARGS [ RANGE=(start,end) ] ]
```

The &CNMSEND verb is used from a user procedure to request NEWS to forward the CNM RU data supplied across the CNM interface to VTAM. The data is supplied in one or more NCL variables that contain the entire record to send to VTAM.

&CNMSEND processing always sets the &RETCODE system variable and may set the &SYSMSG variable as described in the later notes.

Operands:

READ={ USER | CNM | BOTH }

Specifies where to process any reply RUs solicited by this &CNMSEND request.

READ=USER

Specifies that only the soliciting user receives the results. The &CNMREAD statement does not need to be in the same NCL procedure as the &CNMSEND statement. However, any data solicited becomes unavailable after a &CNMCLEAR is issued or the user returns to the Primary Menu. This option is useful when data is solicited for inspection by the user only, avoiding NEWS database logging (and hence the possible loss of existing database records).

READ=CNM

Specifies that any data solicited is not returned to the soliciting user but undergoes standard CNM delivery, which normally involves forwarding to CNMPROC. A subsequent &CNMREAD issued by this user completes with a no data found condition. This option is most useful for unattended solicitation where results are returned for normal analysis by the NEWS system.

READ=BOTH

Enables a user to process the returned results and also provides standard CNM delivery.

Default: BOTH

LINKNAME=*linkname* | SSCPNAME=*sscpname* | DOMAIN=*domain*

Directs the record to a remote system for processing. These operands are mutually exclusive. NEWS uses the Inter-System Routing (ISR) feature to communicate with other systems running on remote hosts, and therefore can route any CNM request to a remote host for processing. The link, SSCP, or domain ID uniquely identifies the destination host. For remote processing to be possible, the INMC link to the remote host must be active and the NEWS facilities in ISR must be enabled for solicited message flows. If the destination name is that of the local system, the local CNM interface is used.

Note: For more information about the ISR command, see the online help.

&var &var .... &var

Are user-specified variables that contain the CNM record to forward to VTAM across the CNM interface. Each variable contains valid hexadecimal characters that, when concatenated, form the entire CNM RU.

Where a Network Services RU (NS RU) is sent to VTAM embedded in a Forward RU, the variables must include the Forward RU with the target resource and PU vectors appended. In addition, all relevant RU byte values and length fields must be appropriately set, with the exception of the 12 bits in the CNM header comprising the procedure correlator identifier (PRID). Where NEWS recognizes from the supplied RU data that this RU is a solicitation request, it automatically generates a PRID and inserts the value in the CNM RU. The PRID, echoed by the resource in any reply RUs, is used to correlate reply data with the &CNMSEND request.

MDO=*mdoname* | VARS=... | ARGS [RANGE=(*start*,*end*)]

Specifies the data to place in the alert queued to CNMPROC.

If the MDO= operand is used, the data is located as formatted in the MDO.

Specifying VARS=... or ARGS results in a $NCL MDO being built and delivered in the $MSG MDO, containing the named variables or arguments.

The MDO, VARS, and ARGS operands are mutually exclusive.

RANGE=(*start*,*end*) is specified with ARGS, to denote an argument range.

Examples: &CNMSEND

```
&CNMSEND &1
&CNMSEND READ=USER &REQMSRU
&CNMSEND READ=CNM &LDRU1 &LDRU2
```

Notes:

When control is returned to the NCL procedure following an &CNMSEND statement the &RETCODE (return code) variable is set to one of the following values:

0

The data supplied has been accepted without error by VTAM. However, VTAM or the target PU can still reject the CNM RU for various reasons, and a &CNMREAD is necessary to determine whether the request was executed successfully.

8

An error has been encountered (for example, the NEWS ACB was closed). The &SYSMSG variable is set and contains text explaining the actual error. An &CNMCLEAR is not required following such an error, as an implicit clear is performed when the &CNMSEND is complete. A subsequent &CNMREAD completes with a no data found condition.

More information:

&CNMREAD (see page 255)
&CNMCLEAR (see page 249)

&CNMVECTR

The &CNMVECTR verb vectors the data supplied into user variables.

This verb has the following format:

```
&CNMVECTR [ MVNUM=nn ]
          [ TYPE= { RU | SV | CV } ]
          [ VARS=prefix* ] &var &var .... &var
```

The &CNMVECTR verb is used from a user procedure to format the segmented CNM RU data into user variables as in an &CNMREAD TYPE=VECTOR operation.

Operands:

MVNUM=*nn*

Specifies which of the major vectors is to be formatted. Some NMVT records contain more than one embedded major vectors. If the specified major vector does not exist, no variables are created.

TYPE= { RU | SV | CV }

Specifies the type of formatting that is required.

TYPE=RU

Specifies that the supplied data is an entire CNM RU and is to be formatted in the fashion of the &CNMREAD TYPE=VECTOR operation.

TYPE=SV

Specifies that the data to be formatted is one or more concatenated subvectors. These vectors have the format *ln…. l* is the length of the subvector (inclusive of length byte and identifier byte), and *n* is the 1-byte subvector identifier. The vectorization process divides the data into its constituent subvectors and places them in the variables *&prefix*SV*nnoo*.

TYPE=CV

Partitions SNA control vectors. These vectors have the format *nl…. n* is the control vector type, and *l* is the 1-byte field containing the length of the data in the control vector (this length excludes the type byte and length byte). The vectorization process divides the data into its constituent vectors, and places them in the variables &*prefix*CV*nnoo*.

VARS=*prefix**

Specifies a prefix other than $CNM for the variables created during the vectorization process. The prefix is one to four characters and must adhere to the standard NCL variable naming conventions.

&var &var .... &var

Specifies the variables that contain the character-format hexadecimal data to process.

Examples: &CNMVECTR

```
&CNMVECTR &1 &2 &3 &4 &5 &6 &7 &8 &9

&CNMVECTR TYPE=SV VARS=VECT* +
0E91030D0D0056FE0FFE03FE0300
```

More information:

&CNMREAD (see page 255)

&CONCAT

The &CONCAT built-in function returns a string that is the concatenation of the supplied data.

This built-in function has the following format:

```
&CONCAT { var | const var | const ... var | const }
```

&CONCAT provides a means of concatenating or joining multiple variables or constants, to form a single variable to a maximum length of 256 characters.

&CONCAT is a built-in function and must be used to the right of an assignment statement. Multiple variables or constants is concatenated with a single &CONCAT statement.

Operands:

var

Specifies a system or user variable.

const

Specifies a constant value.

Examples: &CONCAT

```
&FLOOR = 03
&BLDG = HQ
&LOCATION = &CONCAT &BLDG &FLOOR
              -* &LOCATION is set to HQ03
&SRCHKEY = &CONCAT 55 LOCN &LOCATION
              -* &SRCHKEY is set to 55LOCNHQ03
```

Notes:

The total sum of the concatenated variables or constants cannot exceed the maximum size for one variable-any data for concatenation which exceeds 256 characters is truncated.

If &CONTROL DBCS or DBCSN or DBCSP is in effect, &CONCAT is sensitive to the presence of DBCS data (see page 1198).

&CONTROL

The &CONTROL verb sets NCL procedure control characteristics.

This verb has the following format:

```
&CONTROL [ ALIGNLc | ALIGNRc | NOALIGN ]
         [ CMD | NOCMD  ]
         [ CMDSEP | NOCMDSEP ]
         [ CONT | NOCONT ]
         [ DBCS | DBCSN | DBCSP | NODBCS ]
         [ DUPCHK | NODUPCHK ]
         [ ENDMSG | NOENDMSG  ]
         [ FINDRC | NOFINDRC ]
         [ FLDCTL | NOFLDCTL ]
         [ IFCASE | NOIFCASE ]
         [ INTEGER | REAL  ]
         [ INTLOG | NOINTLOG  ]
         [ KEYXTR | NOKEYXTR  ]
         [ LABEL | NOLABEL  ]
         [ LOOPCHK | NOLOOPCHK ]
         [ MDOCHK | NOMDOCHK ]
         [ PAKEYS | NOPAKEYS ]
         [ PANELID | NOPANELID ]
         [ PANELRC | NOPANELRC ]
         [ PFKMAP | NOPFKMAP ]
         [ PFKSTD | PFKALL | NOPFK ]
         [ RECCHK | NORECCHK ]
         [ RESCAN | RESCAN1 | NORESCAN ]
         [ RNGLIM | NORNGLIM ]
         [ SAVE | NOSAVE  ]
         [ SHAREW | NOSHAREW  ]
         [ SHRVARS | SHRVARS=* | ( [ *, ] pref,...,pref ) |
           NOSHRVARS | NOSHRVARS=* | ( [ *, ] pref,...,pref) ]
         [ SUB | NOSUB ]
         [ TRACE | NOTRACE | TRACELOG | TRACELAB | TRACEALL ]
         [ UCASE | NOUCASE ]
         [ USRCMD | NOUSRCMD ]
         [ VARSEG | NOVARSEG ]
```

Specifying &CONTROL with no operands returns all CONTROL variables to their default values (see note on SHAREW operand). The &CONTROL statement defines all processing options available for modifying NCL execution characteristics.

Each option available on the &CONTROL statement has two possible settings: ON or OFF. The &CONTROL options are represented by keywords, which typically have the following forms:

value or NO*value*

NO*value* represents the OFF setting for *value*.

Every option has a default setting which is in effect when a process is invoked. Procedures within the NCL process can execute &CONTROL statements anywhere in the logic to set or reset any of the &CONTROL options. The combination of &CONTROL option settings in force at any instant for a process are called the process's &CONTROL environment.

If &CONTROL SAVE is specified or defaulted, then whenever a procedure calls another procedure using the EXEC command, the current &CONTROL environment is remembered and restored when the called procedure ends and returns control. This prevents the calling procedure from being affected if any &CONTROL options are changed by the procedure it calls or by any lower-level procedures.

&CONTROL does not affect the logic of a procedure or have any direct effect on variables. The verb changes the execution characteristics of subsequent statements in the procedure.

&CONTROL statements are processed inline, so most operands that are coded on an &CONTROL statement is specified as variables. The value or the variable specified is substituted according to its value at the time the statement is executed.

By setting the value of a variable to the name of one of the &CONTROL operands you can dynamically generate &CONTROL environments.

The &CONTROL CONT and NOCONT operands cannot be the subject of dynamic substitution because they govern the processing of statement continuations and are processed only when a procedure is loaded for execution.

Operands:

ALIGNL*c* | ALIGNR*c* | NOALIGN

ALIGNL*c*

Specifies that substituted values are aligned on the left side of the variable being replaced. The length of the variable name is preserved, and, if necessary, the fill character *c* is used to pad the substituted value to the right. If the fill character *c* is omitted, a blank is assumed.

This option lets you align tabular output. The length of the variable name being replaced determines the point of alignment. Therefore, use variables with similar length names if alignment is required. If substitution data exceeds the length of the variable, alignment cannot be performed and the option is ignored. If the variable value is null at the time of substitution, padding occurs for the full length of the variable name.

ALIGNR*c*

Specifies that substituted values are aligned to the right-hand side of the variable being replaced. The length of the variable name is preserved, and, if necessary, the fill character *c* is used to pad the substituted value to the left. If the fill character *c* is omitted, a blank is assumed. If the variable value is null at the time of substitution, padding occurs over the full length of the variable name. This option lets you align numeric tables.

NOALIGN

Specifies that the process of variable substitution is performed for the entire length of the variable being substituted. If the substitution data is shorter than the variable being replaced, characters to the right of the variable are shifted to the left.

Default: NOALIGN

CMD | NOCMD

CMD

Specifies that all commands executed from within a procedure (with the exception of the PAGE, CLEAR, and K commands) are to be echoed to the user terminal and logged on the activity log.

NOCMD

Specifies that all commands, until the end of the procedure or until an &CONTROL CMD statement, are *not* to be echoed to the user terminal. Commands are still logged to the activity log. Alternatively, use the suppression character (-) on each command individually. The suppression character also suppresses logging the command to the activity log.

Default: CMD

CMDSEP | NOCMDSEP

CMDSEP

Scans command strings executed from NCL procedures for the command concatenation character (;) and splits the commands accordingly.

NOCMDSEP

Specifies no scanning for the command concatenation character (;) and treats a command string as one command, irrespective of any command concatenation characters within the text of the command. Use this option where the command being executed can contain a semi-colon within the text of the command itself (for example, NSBRO issuing a broadcast where the broadcast text may contain a semi-colon).

Default: CMDSEP

CONT | NOCONT

CONT

Concatenates procedure statements if the last character of a statement that is not blank is a plus sign (+). This option makes it possible for a single statement to exceed the length of a procedure record. CONT cannot be dynamically substituted because the CONT and NOCONT operands are resolved during the NCL procedure load process.

NOCONT

Specifies that statements are not concatenated and treats any trailing plus signs as part of the statement in which they occur. This option is necessary for procedures that use commands, such as EQUATE, where a trailing plus sign may imply a trailing blank.

The &CONTROL CONT and NOCONT operands cannot be the subject of dynamic substitution because they govern the processing of statement continuations and are processed only when a procedure is loaded for execution. For example, you cannot use the following construction:

```
&A = CONT
&CONTROL &A
```

All other &CONTROL options are dynamically substituted.

Default: CONT

DBCS | DBCSN | DBCSP | NODBCS

Specifies the presence or absence of Double Byte Character Set (DBCS) data. Manipulating DBCS strings requires special care to preserve its integrity. The NCL language provides extensive support for DBCS data manipulation. This support is available in product regions executing with the SYSPARMS DBCS operand set to YES, IBM, or FUJITSU.

A string of data can contain a mixture of single byte (SBCS) and double byte data. Typically, special characters known as shift characters delineate DBCS data. A shift out character is used to mark the start of DBCS data, and a shift in character marks the return to SBCS data.

DBCS

Specifies that NCL is to be sensitive to the presence of DBCS data. When this operand is in effect, NCL ensures that DBCS or mixed DBCS/SBCS strings are not corrupted during string manipulation.

The DBCS operand causes the counting of shift characters when NCL calculates offsets and lengths in mixed strings. This option is useful when preparing data for display on IBM terminals such as the 5550, where shift characters display as blank one-byte fields.

DBCS adds some processing overhead to NCL. Use it only when the presence of DBCS data is expected. This operand is ignored if your product region is running with the SYSPARMS DBCS operand set to NO.

DBCSN

Specifies that NCL is to be sensitive to the presence of DBCS data. When this operand is in effect, NCL ensures that DBCS or mixed DBCS/SBCS strings are not corrupted during string manipulation.

The DBCSN operand causes shift characters not to be counted when NCL calculates offsets and lengths in mixed strings. This option is useful when preparing data for display on Fujitsu or Hitachi terminals, where shift characters do not take a screen position.

DBCSN adds some processing overhead to NCL. Use it only when the presence of DBCS data is expected. This operand is ignored if your product region is running with the SYSPARMS DBCS operand set to NO.

DBCSP

Specifies that NCL is to be sensitive to the presence of DBCS data. When this operand is in effect, NCL ensures that DBCS or mixed DBCS/SBCS strings are not corrupted during string manipulation.

When calculating offsets and lengths in mixed strings, the DBCSP operand causes NCL to count or not count shift characters, depending on the processing environment of the NCL procedure. If the procedure is executing on behalf of a user logged on from a Hitachi 560/20 or Fujitsu DBCS terminal, shift characters are not included in offset and length calculations. If the NCL procedure is executing on behalf of a user logged on from an IBM DBCS terminal or a terminal that does not support DBCS data, the shift characters are included in offset and length calculations. If the NCL procedure is executing in a background region (for example, BSYS), shift characters are included in offset and length calculations if your product region is executing under an IBM operating system, but not if your product region is operating under a Fujitsu or Hitachi operating system.

DBCSP simplifies coding where you want to execute the NCL procedure on various operating systems or by various terminal types. The option adds some processing overhead to NCL. Use this operand only when the presence of DBCS data is expected. This operand is ignored if your product region is running with the SYSPARMS DBCS operand set to NO.

NODBCS

Specifies that NCL processing does not include any special consideration for DBCS data. If mixed DBCS/SBCS strings are present, NCL's string manipulation functions could corrupt the data.

DUPCHK | NODUPCHK

DUPCHK

Specifies that a check is made for duplicate labels whenever an &GOTO statement is executed. The occurrence of a duplicate label terminates the procedure.

NODUPCHK

Specifies that no check for duplicate labels is performed when an &GOTO statement is executed. Control resumes at the first occurrence of the specified label following the &GOTO statement. If not found before the end of the procedure, the search for the specified label continues from the top of the procedure. If the label is missing, an error occurs (unless NOLABEL is in effect). Specifying this option can speed up processing of large procedures. If duplicate labels exist, unpredictable results can occur.

NODUPCHK is typically specified only after a procedure has been thoroughly tested.

Default: DUPCHK

ENDMSG | NOENDMSG

ENDMSG

Specifies that a completion message N03906 is issued to signal the completion of a procedure.

NOENDMSG

Specifies the suppression of the N03906 completion message unless an error occurs while processing the procedure, or a FLUSH or END command terminates the procedure.

Default: ENDMSG

FINDRC | NOFINDRC

FINDRC

Specifies that an attempt to execute a nested procedure that does not exist in the procedure library returns a value of 100 in the &RETCODE system variable, rather than fail the requesting procedure. This option allows the requesting procedure to detect requests to nest to nonexistent procedures.

NOFINDRC

Specifies that an attempt to execute a nonexistent nested procedure fails the requesting procedure.

Default: NOFINDRC

FLDCTL | NOFLDCTL

FLDCTL

Specifies that &ZMODFLD processing is performed for panels displayed by the current procedure. See the &ZMODFLD variable for details of this facility. The FLDCTL control option is required before &ZMODFLD is made available.

NOFLDCTL

Specifies that &ZMODFLD processing is not performed for panels displayed by the current procedure.

Default: NOFLDCTL

IFCASE | NOIFCASE

IFCASE

Specifies that the operands on an &IF statement are converted to uppercase before comparison. The conversion occurs only for the purposes of comparison, and does not change the value of the operands. IFCASE has no effect if the SYSPARMS DBCS=YES option is in force.

If variable uppercase translation is performed automatically on assignment, the effect of the &CONTROL UCASE option can override the intended effect of IFCASE. See the UCASE operand description.

NOIFCASE

Specifies that the operands on an &IF statement are compared without case translation.

Default: IFCASE

INTEGER | REAL

Specifies the precision of arithmetic functions that are used.

INTEGER specifies integer arithmetic, where integer operands in an arithmetic expression yield only integer results.

If an arithmetic expression contains real numbers that contain a decimal point or decimal fraction, then the expression is evaluated as if &CONTROL REAL were in effect.

&CONTROL INTEGER also controls the manner in which variables are compared. If &CONTROL INTEGER is in effect, the &IF statement performs a numeric comparison of two variables if both contain integer numbers. Otherwise, it performs a string comparison.

If &CONTROL REAL is in effect, then &IF performs a numeric comparison if both variables contain integers, real numbers, or the results of real number arithmetic calculations. If a real number calculation is performed and &CONTROL REAL is not in effect when a comparison (&IF) is executed, then a variable containing the result of a real number calculation, or a real number, is not treated as an integer, and a string comparison occurs.

Default: INTEGER

Note: For more information about integer and real number arithmetic, see the *Network Control Language Programming Guide*.

INTLOG | NOINTLOG

INTLOG

Specifies that all commands issued by &INTCMD and all responses resulting from those commands are logged to the activity log (but not echoed to the user terminal). Because of the concealed way in which &INTCMD operates, you may want to log the activity for certain commands. Commands processed by &INTCMD and associated responses is identified on the activity log with **nclid** in the node name column.

NOINTLOG

Specifies that commands issued with &INTCMD, and any associated responses to those commands, are not logged to the activity log. If the NOINTLOG operand is specified, the lines on the log showing **nclid** in the node name column is suppressed. Using &INTCLEAR to discard outstanding responses associated with &INTCMD processing can disown messages and force them to be logged. In these cases, the NOINTLOG operand is ignored and the outstanding messages are logged. To avoid this, structure your procedure to process all outstanding messages.

Default: INTLOG

KEYXTR | NOKEYXTR

KEYXTR

Specifies that the key portion of the record is extracted from the body of the record when it is placed in the &FILEKEY system variable when processing a UDB which has a key that does not start in the first position of the record (non-RKP 0). A field separator is inserted to separate the data that existed on either side of the extracted key.

NOKEYXTR

Specifies that when processing a UDB that has a key that does not commence in the first position of the record (non-RKP 0), the key portion of the record is not extracted from the main body of the record when it is placed in the &FILEKEY system variable. This can help when processing UDBs using alternate indices, as the relative position of variables having data assigned into them does not change. It may therefore be possible to use common processing for both base and alternate index processing. In such a case, no field separator is inserted.

Default: KEYXTR

Note: For more information about using this facility, see the *Network Control Language Programming Guide*.

LABEL | NOLABEL

LABEL

Specifies that the label which is the subject of an &GOTO or &GOSUB must exist. If an &GOTO or &GOSUB is attempted to a label which has not been defined, the procedure terminates with an error.

NOLABEL

Specifies that an &GOTO or &GOSUB to an undefined label does not terminate the procedure but instead passes control to the statement following the &GOTO or &GOSUB. Using NOLABEL can significantly simplify procedure writing. It eliminates the use of multiple &IF statements to determine the course of action to be taken and permits immediate branching to the appropriate point. If the label to which the branch is attempted is not defined, the statement following the &GOTO or &GOSUB is executed. This can have major performance advantages over conventional methods.

Where possible, the &CONTROL NOLABEL technique should be used where a large number of branch conditions are possible, for example in a procedure that is written to process VTAM messages.

In this case, each message commences with a message number (in the format IST*nnnn*) which is associated with the type of processing required for that message. The message number can therefore be used as the label to be the target of an &GOTO.

In the following example, the &INTREAD statement is being used to request that the next VTAM message be presented to the procedure following the issuing of the command via the &INTCMD statement:

```
&CONTROL NOLABEL NODUPCHK
&INTCMD D LU1 .READ
      &INTREAD ARGS      &GOTO .&1
      -* Unexpected messages will drop past the &GOTO
       &WRITE ALARM=YES DATA=UNEXPECTED VTAM MSG &1 &2 +
            &3 &4 &5 &6
      &GOTO .READ
      -* Processing for expected messages follows
.ISTnnnI
.ISTnnnI
 :
         perform processing
 : &GOTO .READ
      -* End of output message from VTAM comes here
.IST314I
&END
```

Default: LABEL

LOOPCHK | NOLOOPCHK

Determines whether an NCL procedure is terminated in error due to excessive looping.

LOOPCHK

Specifies that the procedure is flushed if the &LOOPCTL setting is exceeded.

NOLOOPCHK

Specifies that a procedure is not flushed.

Default: NOLOOPCHK

MDOCHK | NOMDOCHK

Determines whether an NCL procedure is terminated in error if it encounters a Mapping Services error.

MDOCHK

Specifies that the procedure is flushed if, on an &ASSIGN statement, the &ZMDORC is set to 8 or higher.

NOMDOCHK

Specifies that the user is responsible for handling all Mapping Services errors.

Default: NOMDOCHK

PAKEYS | NOPAKEYS

PAKEYS

Specifies that the procedure intends to process terminal PA keys. If the user presses a PA key, control is returned to the procedure following a full-screen panel display, and the &INKEY variable is set to the value of the PA key pressed, which may be PA1, PA2, or PA3. If IPANULL=YES is specified on the #OPT statement on the panel definition, all input screen variables are nulled and internal validation bypassed. If IPANULL=NO is specified, all input variables retain their contents unmodified, as at the time the panel was displayed. No data is entered when using PA keys, and any data entered into input fields is lost. If two windows are open and both are eligible to receive PA key notification, it is indeterminate which window will receive such notification. PAKEYS applies to full-screen processing only.

NOPAKEYS

Specifies that the procedure is not interested in processing PA keys from the terminal. Pressing a PA key simply results in the terminal keyboard being unlocked.

Default: NOPAKEYS

PANELID | NOPANELID

PANELID

Specifies that the name of the current panel is to be displayed in the upper left hand corner of the panel. The panel name is displayed only if the panel contains some data on the first line and no field characters in the first 12 positions of the first line. The panel name overwrites any data in these first 12 positions. PANELID applies to full-screen processing only.

NOPANELID

Specifies that the current panel name is not to be displayed in the first 12 positions of the first line of the panel. NOPANELID applies to full-screen processing only.

Default: NOPANELID

PANELRC | NOPANELRC

PANELRC

Specifies that a procedure using an &PANEL statement to display a full-screen panel is designed to accept return codes from Panel Services that indicate specific processing conditions. The return code are available in the &RETCODE system variable on return from the &PANEL statement.

When processing with PANELRC, it is possible for the procedure to obtain notification of special situations that can assist in performing more advanced processing. For example, when using PANELRC it is possible to obtain the name of a field that has failed internal validation and to provide appropriate help information, or to determine that a requested panel does not exist and to avoid termination of the procedure. PANELRC applies to full-screen processing only.

Note: For more information about the PANELRC operand, see the *Network Control Language Programming Guide*.

Possible return codes are:

- 0—One or more fields on the panel have been entered or modified. All validation that was to be performed by Panel Services (for example, EDIT=NUM) has been completed without errors. Used in conjunction with return code 4 a procedure can determine if displayed data has been modified and therefore warrants validation and perhaps updating on a file. If an asynchronous panel operation was being performed, return code 0 means that the display will not be performed because input is available from the preceding display of the same panel. The procedure should process the input received before re-displaying the panel.
- 4—No fields on the panel have been modified since it was last displayed. It is the responsibility of the panel designer to allow for the situation where a panel is redisplayed, perhaps to flag a field in error, and this return code is then used to determine if any fields have been modified. This return code applies only to the last iteration with the terminal operator. If an asynchronous panel operation was being performed, return code 4 means that the panel display will not occur because input is available from the earlier display of the panel.

- 8—Panel Services field validation detected an error. The &SYSFLD variable contains the name of the input field in error and &SYSMSG contains the text of the error message that describes the error condition (for example, FIELD NOT NUMERIC). The procedure can take any appropriate action, including modifying the message text in &SYSMSG or displaying help information. If the panel has been designed to take advantage of #OPT statement ERRFLD processing facilities and the ERRFLD operand is specified as ERRFLD=&SYSFLD then a re-display of the panel will perform normal error processing and position the cursor to the field in error. This return code is ideal for escaping from a panel where one or more fields have been specified as REQUIRED=YES.
- 12—For synchronous &PANEL statements, 12 indicates that the INWAIT timer period has expired. No operator input has been made. The &INKEY system variable will be set to null. For asynchronous operations, 12 indicates that the panel display request has been accepted and the panel has been scheduled for display. If the same panel was displayed earlier and is being refreshed or updated, this indicates that no input was received from the earlier display.
- 16—The requested panel could not be found on the panels data set or a syntax error occurred preventing the panel display. The &SYSMSG system variable contains an explanation of the error.

NOPANELRC

Specifies that a procedure using an &PANEL statement to display a full-screen panel is not designed to accept return codes from Panel Services. Panel Services is to perform all processing. NOPANELRC applies to full-screen processing only.

When NOPANELRC is in effect the following actions corresponding to the return codes provided by PANELRC will occur:

- Fields changed—Must be determined by procedure.
- No fields changed—Must be determined by procedure.
- Field in error—Internal validation errors are processed by Panel Services. Redisplay of panels for error messages is automatic.
- INWAIT expired—Must be determined by procedure. &INKEY variable is set to null.
- Invalid panel name—NCL procedure terminates with an appropriate error message.

Default: NOPANELRC

PFKMAP | NOPFKMAP

PFKEYMAP

Specifies that the &INKEY variable (which is set to the value of the key used to input from a full-screen panel, for example, F1, F13) are to be set to a value in the range F1 to F12 when input results from the use of F13 to F24. Keys F13 to F24 are be mapped as F1 to F12. This can simplify the processing performed within the procedure as it need not cater for keys F13 to F24.

NOPFKMAP

Specifies that mapping of function keys is not to be performed. Keys F13 to F24 are presented unchanged and the NCL procedure must cater for them.

Default: NOPFKMAP

PFKSTD | PFKALL | NOPFK

PFKSTD

Specifies that non-system function keys, plus F3/4 and F15/16, are to be passed to the procedure for processing. Function keys F2/14 and F9/21are still processed by the system as SPLIT and SWAP keys. PFKSTD applies to full-screen processing only.

PFKALL

Specifies that all function keys are to be passed to the procedure for processing. Unless appropriate support is included in the NCL procedure, this option nullifies the use of split-screen operation. PFKALL applies to full-screen processing only.

NOPFK

Specifies that only non-system function keys are to be passed to the procedure for processing, following input from a full-screen panel. &INKEY is set to the value of the key pressed. System function keys are: F3/4, F15/16, F2/14, and F9/21. The use of F3/4 or F15/16 terminates (flushes) the procedure if this option is in effect. NOPFK applies to full-screen processing only.

Default: NOPFK

RECCHK | NORECCHK

RECCHK

Specifies that NCL is to test for recursive processing when invoking new nesting levels. NCL prevents a procedure of the same name as that already in use at a higher level from being invoked, as it could result in a recursive processing loop.

NORECCHK

Specifies that NCL is not to check for potential recursive processing when invoking new NCL nesting levels. When this option is used it is the responsibility of the procedure to ensure that recursive processing is not possible. Failure to adhere to this requirement may adversely effect other parts of the system.

Default: RECCHK

RESCAN | RESCAN1 | NORESCAN

RESCAN

Specifies that if NCL encounters an ampersand (&) within the value of a variable during substitution, then it treats it as an embedded variable name and not as data. The embedded variable name is subject to iterative resubstitution to a maximum of 16 times. For example, in the construction:

```
&CONTROL NORESCAN
&A = &CONCAT & TIME (&A value is &TIME)
&WRITE DATA=&A
```

&A is displayed as having a value of the character string &TIME.

In the construction:

```
&CONTROL RESCAN
&A = &CONCAT & TIME (&A value is &TIME)
&WRITE DATA=&A
```

&A is displayed as having a value of the character string 12.00.00, assuming the time is midday.

&CONTROL RESCAN is useful for resolving the value of variables passed to a nested or started NCL procedure when the parameters passed contain the names of other variables.

RESCAN1

Specifies that if, during substitution, NCL encounters an ampersand (&) within the value of a variable, then it treats it as an embedded variable name, not as data. There is no iterative resubstitution.

NORESCAN

Specifies that if, during substitution, NCL encounters an ampersand (&) within the value of a variable, it is to be treated as data, not as an embedded variable that is to be resubstituted.

Default: NORESCAN

RNGLIM | NORNGLIM

Specifying RNGLIM sets a limit of 64 on the operational range of the &ASSIGN (see page 206) statement when creating or modifying groups of variables. This is used to prevent logic errors (causing excessive numbers of variables to be created) in procedures that use &ASSIGN. RNGLIM does not restrict the range of variables that is deleted by &ASSIGN.

If very large numbers of variables are to be processed in one &ASSIGN operation, then use NORNGLIM to remove the limit of 64. NORNGLIM specifies that no range restriction is imposed on the ability of the &ASSIGN statement to generate groups of variables.

Default: NORNGLIM

SAVE | NOSAVE

SAVE

Specifies that when a nested procedure is invoked the current &CONTROL values are to be saved and automatically restored on return from the nested level. This ensures that any changes made to &CONTROL options by the nested level do not impact the current level. Thus, changes made to &CONTROL in the lower level impact only that level and other levels that it may invoke, and the saved values are restored on return.

NOSAVE

Specifies that &CONTROL values are not to be saved prior to invoking a nested level. Thus, changes to &CONTROL that are made in lower nesting levels remain in effect on return to this level. If NOSAVE is specified, it is the procedure's responsibility to ensure that changes to &CONTROL do not affect subsequent processing.

Default: SAVE

SHAREW | NOSHAREW

SHAREW

Specifies that the issuing procedure is prepared to allow other NCL processes, executing in the same NCL region processing environment associated with the same window, to issue &PANEL statements to take over the window display area.

If a procedure executes a &CONTROL SHAREW statement, any other process may obtain control of the display area. No indication is provided if any other process does obtain control of the window display area.

For this reason, &CONTROL with no operands does not reset SHAREW.

&CONTROL SHAREW status becomes effective when the issuing process obtains the right to ownership of the window.

If INWAIT=0 is specified on the panel definition, control does not return to the issuing procedure until the panel has been displayed.

NOSHAREW

Specifies that the issuing procedure is not prepared to allow any other process in the same NCL region to obtain control of the window display area by issuing &PANEL statements. If the procedure that executes an &CONTROL NOSHAREW statement does not actually have control of the window, then the status has no effect until the process obtains control of the window.

NOSHAREW is used when executing a process that needs to guarantee that no other process overwrites its information display.

Default: SHAREW

SHRVARS | SHRVARS=* | (*pref*,...,*pref*) | (*,*pref*,...,*pref*) | NOSHRVARS | NOSHRVARS=* | *(pref*,...,*pref*) | (*,*pref*,...,*pref*)

SHRVARS

Specifies that all user variables that exist when another nested procedure is invoked are to be shared with the nested procedure. When the nested procedure ends, all variables, including any new ones created by the nested level, are returned to the higher level. Variables deleted by the nested procedure are not reinstated on return.

The sharing of variables offers great flexibility in the design of modular procedures, as no limits are placed on the amount of data passed between the levels. The SHRVARS option only affects the next immediate level. If it in turn invokes another procedure, variables are not shared unless explicitly requested by another SHRVARS statement at that level.

SHRVARS does not affect the use of variables as parameters on an EXEC command. However, care must be taken when sharing variables such as &1, &2, and so on, since parameter variables specified on EXEC cause creation of variables &1, &2, and so on for the new procedure, which override the &1 and &2 variables of the invoking procedure.

&RETURN may be used to return variables that supplement any variables implicitly shared between procedures as a result of SHRVARS operation. This is ideal for returning an error message from a nested procedure, for example in the &SYSMSG variable.

SHRVARS=* | (*pref*,…,*pref*) | (*,*pref*,…,*pref*)

Specifies that sharing of variables is allowed between procedures in the same NCL process, but specification of prefix allows sharing only of generic ranges of variables. For example:

```
&CONTROL SHRVARS=(AB,DE)
```

allows sharing of only those variables that have names beginning with &AB or &DE. Other variables used by the sharing procedures are private. A maximum of 16 prefixes is specified. If multiple SHRVARS (or NOSHRVARS) statements are processed in a procedure the most recent takes effect and overrides any earlier specification.

SHRVARS=* means that a procedure is to share all generic prefixes that it was passed when it was invoked with any procedure that it in turn invokes. This allows different procedure nesting levels to share the same range of variables without explicit knowledge of the generic prefixes.

SHRVARS=(*,*pref*,…,*pref*) indicates that in addition to sharing all generic prefixes that it was passed when invoked the procedure wishes to share the further prefixes defined by *pref* with any procedure that it in turn invokes.

The SHRVARS option allows sharing of variables between procedures within the same NCL process. It does not imply any sharing of variables between a process and any other process that it may invoke using a START command, since the START command invokes a new NCL process.

NOSHRVARS

Specifies that variable sharing is not to be performed. In this case, data is passed on the EXEC command and is presented to the nested procedure in the variables &1, &2, and so on.

NOSHRVARS=* | (*pref*,…,*pref*) | (*,*pref*,…,*pref*)

Specifies that variable sharing is to occur between the current procedure in an NCL process and any that it executes, except for the variables that have names starting with the nominated generic prefixes. For example, a procedure that issues the statement:

```
&CONTROL NOSHRVARS=(WK,TST)
```

shares all variables except those beginning with &WK and &TST. This option allows procedures to retain private groups of variables which are not shared by (or returned by) nested procedures. A maximum of 16 prefixes is defined. If multiple NOSHRVARS statements are processed within a procedure, the most recent takes effect.

NOSHRVARS=* means that a procedure wishes to operate with the same exception list with any procedure it invokes as it was passed when it was invoked. This allows nesting levels to share (or exclude) the same range of variables without explicit knowledge of the generic prefixes involved.

NOSHRVARS=(*,*pref*,…,*pref*) indicates that, in addition to the generic prefix exception list that it was passed when it was invoked, a procedure wishes not to share the additional generic prefixes as defined by *pref*.

Default: NOSHRVARS

SUB | NOSUB

SUB

Specifies that all procedure statements, with the exception of comment lines that start with the suppression character (-*), are to be scanned, and that variable substitution is to be performed.

NOSUB

Specifies that variable substitution is no longer to be performed. &IF statements, &GOTO statements, and commands are still to be executed, but without prior substitution of variables. Substitution resumes when the next &CONTROL SUB statement is encountered during processing. The substitution process scans all lines for variables that commence with an ampersand (&). Unrecognized variables, which do not currently have an assigned value, are regarded as nulls and are eliminated from the command string. Thus, if comments or commands are to contain words that include an ampersand, the NOSUB control variable must be used to stop the substitution process from eliminating the potential variable from the string.

Default: SUB

TRACE | NOTRACE | TRACELOG | TRACELAB | TRACEALL

TRACE

Specifies that statements are to be displayed after variable substitution and prior to execution. Message N03802 is issued with the number of the statement being executed and the substituted text of the statement. Tracing is used during the development of procedures to ensure their correct operation. If tracing is active when an &PANEL statement is issued, any panel control statements in the panel definition are traced after variable substitution has been completed. Statements are displayed at the operator's terminal only, and not written to the activity log. For example:

```
&CONTROL TRACE NOINTLOG ALIGNR
```

Tracing can generate a large amount of output and should be used at selective points within the procedure. The system limits the maximum number of trace messages that is generated by an NCL procedure. This limit is set by the SYSPARMS NCLTRMAX command and defaults to 100. When this limit is reached, tracing is automatically terminated. Trace messages are not sent to the system log unless the TRACELOG operand is used.

NOTRACE

Cancels a previous &CONTROL TRACE, TRACEALL, or TRACELOG statement.

TRACELOG

Specifies that tracing of NCL statements is written to the system log and that the statements are not displayed at the terminal of the operator.

TRACELAB

Specifies that only procedure labels are traced. TRACELAB requires either TRACE or TRACELOG to start tracing. NOTRACE resets the TRACELAB option. A label trace includes details of the statement number, procedure name, and the NCL ID for labels only. TRACELAB is used to provide a summary trace to analyze overall procedure flow, perhaps across multiple procedures.

TRACEALL

Specifies the provision of expanded trace information. Standard trace information is restricted, to limit the amount of data displayed. TRACEALL requests that, in addition to the standard information, the N03802 trace message is to include the name of the processing procedure and its NCL ID.

Default: NOTRACE

UCASE | NOUCASE

UCASE

Specifies that, during assignment processing, the lowercase characters 'a' to 'z' are converted to uppercase when they are placed in the receiving variable. The equals symbol (=) indicates an assignment statement and implies movement of data into the target variable to the left of the statement.

When operating in a system in which the SYSPARMS DBCS=YES option is set, the UCASE and NOUCASE operands are ignored.

NOUCASE

Specifies that, during assignment processing, data is not converted to uppercase.

Default: UCASE

USRCMD | NOUSRCMD

Affect REPL equates only.

USRCMD

Specifies that REPL equates replace commands issued within the procedure, if a matching REPL equate exists.

USRCMD specifies that EQUATE processing is enabled for commands strings within this procedure. If a string has a corresponding EQUATEd text value, that value is used as a replacement for the EQUATEd command string.

NOUSRCMD

Specifies that REPL equates do *not* replace commands issued within the procedure, even if a matching REPL equate exists.

NOUSRCMD specifies that EQUATE processing is disabled for command strings within this procedure, even if a corresponding EQUATE exists.

Default: NOUSRCMD

VARSEG | NOVARSEG

VARSEG

Specifies that multiword variables that are passed to a nested procedure or that are received using an &INTREAD are to be segmented into individual variables on entry to the nested level or on completion of the &INTREAD. Each word of a multiword variable is separated and placed in the next sequential variable (for example, &1, &2, …, and &*n*).

NOVARSEG

Specifies that multiword variables that are passed to a nested procedure or that are received using an &INTREAD are not to be segmented. Variables remain unchanged on entry to the nested procedure or on completion of the &INTREAD. An alternative to the passing of data on the EXEC command is to use the SHRVARS operand to share all variables between the two procedures.

If variables containing hexadecimal data contain embedded blanks (X'40'), the variables are segmented regardless of the VARSEG or NOVARSEG operand. To avoid segmentation, code the following statement:

```
&CONTROL NOVARSEG &A=&FILEKEY -EXEC PROCB &A
```

instead of:

```
&CONTROL NOVARSEG -EXEC PROCB &FILEKEY
```

Null characters (X'00') are converted to blanks (X'40') by this process.

Default: VARSEG

More information:

DBCS Support in NCL (see page 1196)

&DATECONV

The &DATECONV built-in function does either of the following:

- Converts one date format to another.

  `&DATECONV DATE`*x* *date* `[ = ] DATE`*y* `[` *variation* `]`

- Subtracts two dates.

  `&DATECONV DATE`*x* *date*`1 - DATE`*y* *date*`2`

&DATECONV must be used to the right of an assignment statement.

You can use &DATECONV to reformat a date from one NCL date form to another NCL date form. In addition, you can optionally adjust the date by a number of days.

You can also use &DATECONV to subtract two dates.

DATE*x* formats are shown in the following table:

DATE*x*	Format	DATE*x*	Format
x = 1	YY.DDD	*x* = 9	*nnnnnn* (days since 1/1/0001)
2	DAY DD-MON-YYYY	10	YYYYMMDDHHMMSS*p*HHMM
3	DD-MON-YYYY	11	YYYYMMDDHHMMSS.FFFFFF*p*HHMM
4	DD/MM/YY	12	DD/MM/YYYY
5	MM/DD/YY	13	YYYY/MM/DD
6	YY/MM/DD	14	MM/DD/YYYY
7	YYMMDD	16	YYYY.DDD
8	YYYYMMDD	17	YYYYDDD

For date 10, *p* in the format is plus (+) or minus (-).

For date 11, YYYYMMDDHHMMSS is local time. *p* is plus (+) where local time is ahead of GMT; otherwise, *p* is minus (-). HHMM is the offset for GMT.

Operands:

For reformatting a date from one NCL date form to another NCL date form, this built-in function has the following format:

`&`*result* `= &DATECONV DATE`*x* `&`*indate* `[ = ] DATE`*y* `[` *variation* `]`

DATE*x*

Indicates the date format for the input date (that is, the date to convert) where *x* is a number. The format of the date is the same as the format of the date returned by the corresponding &DATE*n* system variable. See the previous table for the actual formats supported.

date1

A date string with format DATE*x*, specifying the input date.

DATEy

The date format required after the conversion. This format must be one of the NCL supported date formats &DATE1 to &DATE17 (excluding &DATE15).

variation

Specifies the number of days to vary the converted date, with respect to the original date. This variation is positive or negative, with NO space allowed anywhere from the sign to the end of the value. The variation can also be used to add or subtract a value, using the following format:

```
{ + | - }dddd.hhmmss.ffffff
```

This format must begin with a sign, followed by the number of days (any integer including 0 up to 2147483647 is acceptable), then optionally a period followed by hours, minutes and seconds, and optionally fractions of a second to microseconds. This operand is useful when using DATE10 and DATE11 formats to determine the exact date and time for some future or past interval.

For subtracting two dates, this built-in function has the following format:

```
&result = &DATECONV DATEx date1 - DATEy date2
```

DATE*x*

The format of the first date, where *x* is a number. The format of the date is the same as the format of the date returned by the corresponding &DATE*n* system variable. This format must be one of the NCL supported date formats &DATE1 to &DATE17 (excluding &DATE15). See the previous table for the actual formats supported.

date1

A date string with format DATE*x*, specifying the first date.

DATE*y*

The format of the second date format, where *y* is a number. The format of the date is the same as the format of the date returned by the corresponding &DATE*n* system variable. This format must be one of the NCL supported date formats &DATE1 to &DATE17 (excluding &DATE15). See the previous table for the actual formats supported.

date2

A date string with format DATE*x*, specifying the second date.

The value returned is *date1* - *date2*, where the sum or difference is one of the following forms:

- { + | - } *dddd*
- { + | - } *ddddhhmmss*
- { + | - } *dddd.hhmmss*
- { + | - } *dddd.hhmmss.ffff*
- { + | - } *ddddhhmmssffffff*

The form where *.hhmmss* is present is returned only if either input date is DATE10, while *.hhmmss.ffffff* is returned only if either input date is DATE11. *dddd* is an integer with a maximum absolute value of 2,147,483,647.

If you specify an invalid source date format or value, or if the resultant date exceeds the maximum date supported (that is, 31st December 2099) the target variable is set to nulls.

Examples:

```
&TODAY = &DATE1 &YESTERDAY = &DATECONV DATE1 &TODAY DATE2 -1
```

converts the system date of today (format YY.DDD), to the system date of yesterday (format DAY DD-MMM-YYYY)

```
&UPDATED = 90.360 &UPDATED = &DATECONV DATE1 &UPDATED DATE2
```

converts &UPDATED to WED 26-DEC-1990

```
&EXPIRES = 32.360 &EXPIRES = &DATECONV DATE1 &EXPIRES DATE2
```

converts &EXPIRES to SAT 25-DEC-2032

Notes:

&DATECONV lets you present date formats consistently for individual users, or systemwide. NCL procedures can be written using a standard internal date format. This format can then be tailored for system users depending upon other parameters such as language codes.

When converting from date formats with a two-digit year (YY) to a format with a four-digit year, the century is assumed as follows:

- If the year number (YY) for the original source variable is less than 50, the century is calculated as 20.
- If the original year (YY) is greater than or equal to 50, the century is calculated as 19.

When adjustment by a specific variation crosses the century boundary, the adjustment is performed first and the century of the resultant *yy* value is determined as described previously.

When the target date is DATE11, the output is normalized to be GMT with the system zone offset.

In general, the input and output dates are assumed to be local dates and times. If the input date is either DATE10 or DATE 11, the resultant date is calculated by taking into account the difference between the input time zone and the local time zone.

For example, if the local time zone is +1000:

```
&DATECONV DATE10 19920630143622+0000 DATE2
```

produces a result of WED 01-JUL-1992.

More information:

&DATEn (see page 824)

&DEC

The &DEC built-in function returns the decimal equivalent of a hexadecimal value.

```
&DEC hexadecimal value
```

&DEC must be used to the right of an assignment statement.

Examples: &DEC

```
&NUM = &DEC FFFF          -* &NUM is set to 65535
&NAMELEN = &DEC &HEXLEN   -* &NAMELEN is set to a decimal
                          -* length
                          -* for example, if &HEXLEN is 10
                          -* then &NAMELEN is set to 16
```

Notes:

- Hexadecimal values in the range 0 to 7FFFFFFF are returned as positive integers 0 to 2,147,483,647.
- Hexadecimal values FFFFFFFF to 80000001 are returned as -1 to -2,147,483,647.
- Values outside these ranges or invalid hexadecimal characters cause an NCL error.

More information:

&HEX (see page 361)

&DECODE

The &DECODE verb decodes all or part of an MDO, creating logical ASN.1 components from a serial byte string.

This verb has the following format:

```
&DECODE MDO=targetmdo
      [ { [ TRANSLATE={ NO | ISO | DEC | ASCII } ]
            SRC_CHARSET=name
          [ TGT_CHARSET=name ]
          [ SINGLE_SUB=xx ]
          [ DOUBLE_SUB=xxxx ] } ]
      [ TRANSFER={ NO | BER } ]
      [ FROM MDO=sourcemdo ]
```

An NCL procedure can decode all or part of an MDO from a serial byte stream, after transmission using the &DECODE verb.

Operands:

MDO=*targetmdo*

(Mandatory) It identifies the target component for the decode operation. The target MDO name is:

- A stem name (for example, MDO=ROSE) indicating that the target is an entire MDO
- A compound name (for example, MDO=CMIP.GETARG) indicating that the target is a component within the MDO

In both cases, the MDO must exist and be mapped, or the request fails.

TRANSLATE={ NO | ISO | DEC | ASCII }

(Optional) Specifies the translation of character strings. If TRANSLATE=NO is specified, or defaulted, no character set translation occurs. Otherwise, character strings are translated using the character set specified by this operand.

During translation, all character codes are assumed to be from the indicated character set and are translated to their EBCDIC equivalent. Source graphic characters that do not have an equivalent translation are translated to blanks. Control characters are translated to nulls. When used with a transfer syntax, translation takes place after decoding. When used without a transfer syntax, no source MDO is specified and the target MDO is modified in place.

Specification of this operand precludes the use of SRC_CHARSET, TGT_CHARSET, SINGLE_SUB, and DOUBLE_SUB.

SRC_CHARSET=*name*

(Optional) Names the source character set. If this operand is specified, the TRANSLATE operand cannot be specified.

Use SRC_CHARSET instead of TRANSLATE to specify translation using the Advanced Translation Facility (ATF).

TGT_CHARSET=*name*

(Optional) Names the target character set. If this operand is specified, the TRANSLATE operand cannot be specified.

TGT_CHARSET defaults to SOLVE. However, you can specify any other character set name to indicate that you want to translate to that character set.

SINGLE_SUB=*xx*

(Optional) Specifies an overriding target one-byte substitution character. The value is in two hexadecimal characters. For example, to replace all untranslatable characters with an EBCDIC question mark, code SINGLE_SUB=3F (assuming TGT_CHARSET=ASCII).

DOUBLE_SUB=*xxxx*

(Optional) Specifies an overriding target two-byte substitution character. The value is in four hexadecimal characters.

TRANSFER={ NO | BER }

(Optional) Specifies the use of transfer syntax when decoding. If NO is specified, or defaulted, no decoding takes place. TRANSFER=BER indicates the use of Basic Encoding Rules as the transfer syntax.

When a transfer syntax is specified, the FROM operand is required to indicate the *sourcemdo*, which is treated as a serial byte stream. Decoding takes place in accordance with the definition of the target MDO, and the result is placed into the target MDO. If translation was also requested (by specifying the TRANSLATE operand), translation of the target component character strings occurs after decoding.

If the TRANSFER operand is omitted, or TRANSFER=NO is specified, no decoding takes place, and the FROM keyword and source MDO cannot be specified.

FROM MDO=*sourcemdo*

This operand is mandatory and identifies the source component for the decode operation. The source MDO name is:

- A stem name (for example, MDO=ROSE) indicating that the source is an entire MDO
- A compound name (for example, MDO=CMIP.GETARG) indicating that the source is a component within the MDO

Return Codes:

&RETCODE	&ZFDBK	Meaning
0	0	All data successfully decoded
4	0	Input was empty-no data was decoded
	1	Unexpected trailing octets were ignored
	2	End-of-contents octets were assumed
8	1	Data invalid, could not be decoded
	2	Mandatory component missing
	3	Expected tag not found
	4	Invalid length encoding
	5	Invalid tag encoding
	6	Invalid contents encoding
	7	Incorrect multiple tagging
	8	Incorrect recurring tag (within a SEQUENCE OF or SET OF type)
	9	Tag value exceeds implementation limit
	10	Length value exceeds implementation limit
	11	Data value exceeds implementation limit
12	0	Undefined MDO name referenced

For &RETCODE 4, the &SYSMSG variable contains a warning message describing the condition.

For &RETCODE 8, the &SYSMSG variable contains an error message describing the component in error, the position in the data stream where the error was encountered, and the actual error condition.

Notes:

- When communicating with other open systems, it is necessary to encode data for transmission in a manner understood by both parties. This means that both systems need only agree on the format of the transmitted data, and not how that data is processed or kept locally in each system.

  A transfer syntax describes the format of the transmitted data. The transfer syntax formally defines the rules for converting an abstract data structure as defined by the abstract syntax into a serial data stream.

  Mapping Services uses Abstract Syntax Notation One (ASN.1), defined by ISO 8824 as the abstract syntax used to describe data structures within your product region. It also uses Basic Encoding Rules (BER) defined by ISO 8825, as the transfer syntax used to serialize data for transmission.

  Using the &ENCODE verb, an NCL procedure can encode all, or part of an MDO, into a serial byte stream, ready for transmission. Often, this process must take place in stages, as some ASN.1 structures require that various substructures are already encoded elsewhere before the structure itself is encoded. To cater for this requirement, encoding takes place from one MDO to another. When all encoding is complete the final MDO is transmitted as a serial byte stream.

  The &DECODE verb provides the reverse process, creating the logical ASN.1 components from a serial byte stream.

  In addition to the serialization of data for transmission, the &ENCODE and &DECODE verbs provide the ability to translate between the ASN.1 defined character sets into the local format, EBCDIC.

- Use of some operands implies the use of Advanced Translation Facility (ATF) for translation.

Example1:

```
&DECODE MDO=CMIP.GETARG TRANSLATE=DEC

&DECODE MDO=ROSE TRANSLATE=ISO TRANSFER=NO
```

In these examples, components present in *targetmdo*, and defined as ASN.1 character strings, are translated from the indicated character set to their EBCDIC equivalent.

Example 2:

```
&DECODE MDO=CMIP.GETARG TRANSFER=BER FROM +
        MDO=ROSE.RDIVAPDU

&DECODE MDO=ROSE TRANSLATE=NO TRANSFER=BER FROM +
        MDO=BYTESTR

&DECODE MDO=ROSE TRANSLATE=DEC TRANSFER=BER FROM +
        MDO=BYTESTR
```

These examples show decoding of a serial byte stream into *targetmdo*, with character string translation from the indicated character set, if any, to EBCDIC.

Example 3:

```
&DECODE MDO=BUD SRC_CHARSET=UNICODE
```

This example translates character data in the MDO BUD from Unicode to EBCDIC.

More information:

Code Page Selection (see page 1189)
&ENCODE (see page 309)

&DELAY

The &DELAY verb interrupts processing of a procedure for the specified number of seconds.

This verb has the following format:

```
&DELAY nnnn [ .nn ]
```

The verb lets you provide a pause during procedure execution, perhaps to allow time for another event to complete.

Operands:

nnnn.nn

The number of seconds for which processing is to be suspended. This number can range from 0 to 9999.99 seconds (2 hours, 46 minutes 39.99 second maximum).

A delay period of 0 seconds specified for the &DELAY statement creates a pseudo-wait where this NCL procedure allows control to be passed to other NCL tasks. Execution will be resumed as soon as the workload allows.

Examples: &DELAY

```
&DELAY 15  -*Suspend the procedure for 15 seconds

&DELAY 0.5 -*Suspend the procedure for half a second
```

Notes:

- You can interrupt a procedure that has suspended processing with an &DELAY statement, by issuing a GO or FLUSH command:
- GO causes processing to resume immediately after the &DELAY statement.
- FLUSH causes the entire procedure to terminate.

See Also:

- The &INTCMD (see page 370) and &INTREAD (see page 375) verb descriptions for synchronized event processing.
- The AT and EVERY command descriptions, in the Online Help, for time-controlled execution.

&DO

&DO groups a sequence of NCL statements together to form a logical program function block.

The group is delimited by a &DOEND statement.

An &DO group is usually executed after an &IF or &ELSE statement, but is coded anywhere in a procedure to help structure your code. There is no limit to the nesting levels of &DO constructions.

The main advantage of &DO grouping is to allow the use of structured programming techniques and thereby minimize or eliminate the use of multiple &GOTO statements.

Examples: &DO

```
&IF &DAY = WED &THEN +
 &DO
 -*
 -* Other NCL statements
 -*
 &DOEND
&ELSE +
 &DO
 -*
 -* Other NCL statements
 -*
 &DOEND
```

&DO groups must be terminated by an &DOEND statement, that is, &DO and &DOEND statements must be paired. Unbalanced pairs cause syntax errors at load time.

More information:

&DOWHILE (see page 306)
&DOUNTIL (see page 304)

&DOEND

&DOEND signifies the logical end of a group of statements that starts with &DO, &DOWHILE, or &DOUNTIL.

```
&DOEND
```

All &DO, &DOWHILE, or &DOUNTIL statements must be paired with a corresponding &DOEND statement. Unbalanced occurrences cause syntax errors.

Examples: &DOEND

```
&IF &DAY = SAT &THEN +
 &DO
 -*
 -* Other NCL statements
 -*
 &DOEND
&ELSE +
&DO   &I = 1
  &DOWHILE &I LE 100
 -*
 -* Other NCL statements
 -*
 &I = &I + 1
 &DOEND
&DOEND
```

&DO groups must be terminated by an &DOEND statement, that is, &DO and &DOEND statements must be paired. Unbalanced pairs cause syntax errors at load time.

More information:

&DOWHILE (see page 306)
&DOUNTIL (see page 304)

&DOM

The &DOM verb issues an MVS DOM to erase a non-roll delete WTO.

This verb has the following format:

```
&DOM ID=domid
```

The verb lets an NCL procedure issue an MVS DOM (delete-operator-message). Typically, this would be sometime after the procedure had issued an &WTO verb to send an NRD message for display on system consoles. The &WTO verb returns the DOMID in &ZDOMID. By saving this value in a user variable, the &DOM verb can use it later.

The &DOM verb is not supported on z/VM.

Operands:

ID=*domid*

Specifies the DOMID of the message to delete. *domid* must contain eight hexadecimal digits. The value returned in &ZDOMID after an &WTO is in the correct format, as is the &AOMDOMID system variable value when a WTO or WTOR message is current in an AOMPROC.

The eight hexadecimal digits consist of a two-digit system ID and a six-digit message ID.

Examples: &DOM

```
&WTO DESC=1 DATA=SEVERE MESSAGE......
&SAVEID = &ZDOMID
... do something else ....
&DOM ID=&SAVEID
&AOMREAD SET
&IF .&AOMMSGID = .id-that-MVS-never-deletes &THEN +
   &DOM ID=&AOMDOMID  -* we delete it.....
```

Notes:

When using &DOM, consider the following recommendations:

- Issue &DOM only with a valid DOMID. An invalid format DOMID (not eight hexadecimal digits) terminates the NCL procedure. If the DOMID appears valid, but the actual number is not valid, the wrong message may be deleted.
- Delete messages only for a good reason. Indiscriminate deletion of critical messages can lead to severe operational problems.
- The &DOM verb can also be used to delete NRD messages the system issues, which are not deleted by the system itself.

More information:

&WTO (see page 706)
&AOMDOMID (see page 762)

&DOUNTIL

&DOUNTIL builds a conditional loop with a test at the bottom.

```
&DOUNTIL expression [ AND | OR expression .... ]
```

The &DOUNTIL loop is executed repetitively until the conditions specified in the expressions become true. When the test succeeds, execution continues past the &DOEND statement paired with the &DOUNTIL statement.

Operands:

expression

This expression acts as the test for the &DOUNTIL condition. The expression is evaluated at the bottom of the loop:

- If not true the &DOUNTIL loop re-executes.
- If the expression is true, execution continues past the associated &DOEND statement.

Compounded expressions can be used, joined with AND or OR operators, but parentheses cannot be used.

Examples: &DOUNTIL

```
&DOUNTIL &A = 10 OR &B GT &A
   &B = &B + 2
   &A = &A + 1
&DOEND
```

This simple loop is repeated until &A reaches a value of 10, or until &B reaches a value greater than &A, whichever comes first.

```
&GOSUB .GETMSGS
&MSG0 = &STR the following messages were received
&CNT = 0
&DOUNTIL &CNT GE &MSGCNT
  &WRITE DATA = &MSG&CNT
  &CNT = &CNT + 1
&DOEND
&WRITE DATA=**End of messages**
```

This example shows a routine for writing a stream of messages set up by the .GETMSGS subroutine.

Note: The loop always executes once, therefore only the title and the end messages will be written if &MSGCNT=0.

Notes:

&DOUNTIL groups must be terminated by an &DOEND statement, that is, &DOUNTIL and &DOEND statements must be paired. Unbalanced pairs cause syntax errors at load time.

&LOOPCTL is provided to control runaway looping. Iterations of loops during &DOWHILE and &DOUNTIL processing are included in &LOOPCTL calculations.

More information:

&DOWHILE (see page 306)
&DO (see page 301)

&DOWHILE

&DOWHILE builds a conditional loop with the test at the top.

```
&DOWHILE expression [ AND | OR expression .... ]
```

The &DOWHILE loop is executed repetitively while the conditions specified in the expressions are true.

If initially false, the loop is not executed.

When the test fails, execution continues past the &DOEND statement paired with the &DOWHILE statement.

Operands:

expression

This expression acts as the test for the &DOWHILE condition and is evaluated at the top of the loop:

- If true the &DOWHILE loop executes.
- If the expression is false, execution continues past the associated &DOEND statement.

Compounded expressions is used, joined with AND, or OR operators, but parentheses cannot be used.

Examples: &DOWHILE

```
&DOWHILE &A << 10  AND &A >> &B
  &B = &B + 2
  &A = &A + 1
&DOEND
```

This simple loop is repeated while &A is less than 10, and while &A is greater than &B.

```
&LINECNT = 1
&GOSUB .GETDATALINE
&DOWHILE &LINECNT <= 20 AND &RETCODE EQ 0
   &LINE&LINECNT = &DATALINE
   &LINECNT = &LINECNT + 1
   &GOSUB .GETDATALINE
&DOEND
&IF &LINECNT <= 20 &THEN +
   &LINE&LINECNT = &STR **END**
```

This example sets up the variables &LINE1 to &LINE20 to data returned from the .GETDATALINE subroutine (which can read records from a file, for example). The loop is terminated by either reaching the end of the data available (that is, the routine .GETDATALINE returns a non-zero return code) or when all the variables have been set (that is, when all the variables have been set (that is, when &LINECNT reaches a value of 21). These variables can then be displayed on a panel.

Notes:

&DOWHILE groups must be terminated by an &DOEND statement, that is, &DO and &DOEND statements must be paired. Unbalanced pairs cause syntax errors at load time.

&LOOPCTL is provided to control runaway looping. Iterations of loops during &DOWHILE and &DOUNTIL processing are included in &LOOPCTL calculations.

More information:

&DO (see page 301)
&DOUNTIL (see page 304)

&ELSE

&ELSE provides an alternative logic path after &IF where the preceding &IF condition is false.

```
&ELSE { command | statement }
```

The &IF statement specifies a course of action when the condition being tested is true. By coding an &ELSE statement with an &IF statement, an explicit course of action is specified whenever the &IF condition is false.

Where &ELSE is used to specify a false condition logic option, no &GOTO statement is required after a true condition.

&ELSE must be coded as a statement separate from the preceding &IF statement. If &ELSE is coded on an &IF statement, it causes a syntax error.

Operands:

command* or *statement

This specifies the command or statement which is to be executed when the &IF condition is false. The command or statement must be coded in the same statement as &ELSE.

Examples:

```
&IF &STATUS = ACTIVE &THEN +
 &DO
  -*
  -* True Logic
  -*
 &DOEND
&ELSE +
 &DO
  -*
  -* False Logic
  -*
 &DOEND
.CONT
```

In this example, true logic is executed if &STATUS = ACTIVE. On completing true logic, processing resumes at the label .CONT. If &STATUS is not equal to ACTIVE, the false logic after the &ELSE statement executes. When this completes, processing also resumes at the .CONT label.

Note: No &GOTO statement is required to direct the logic flow.

More information:

&DO (see page 301)
&IF (see page 364)

&ENCODE

The &ENCODE verb encodes all or part of an MDO into a serial byte stream, ready for transmission.

This verb has the following format:

```
&ENCODE MDO=sourcemdo
      [ { TRANSLATE={ NO | ISO | DEC | ASCII } |
          TGT_CHARSET=name SRC_CHARSET=name
        [ SINGLE_SUB=xx ]
        [ DOUBLE_SUB=xxxx ] } ]
      [ CHECK | TRANSFER={ NO | BER } ]
      [ INTO MDO=targetmdo ]
```

An NCL procedure can encode all, or part, of an MDO into a serial byte stream, ready for transmission using the &ENCODE verb.

Operands:

MDO=*sourcemdo*

(Mandatory) Identifies the source component for the encoding operation. *sourcemdo* name is:

- A stem name, for example, MDO=ROSE (indicates that the source is an entire MDO)
- A compound name, for example, MDO=CMIP.GETARG (indicates that the source is a component within the MDO)

TRANSLATE={ NO | ISO | DEC | ASCII }

Specifies the translation of character strings. If TRANSLATE=NO is specified, or defaulted, then no character set translation occurs.

During translation, all character codes are assumed to be in EBCDIC and are translated to the indicated character set. Source graphic characters that do not have a defined translation are translated to blanks. Control characters are translated to nulls. When used with a transfer, syntax translation takes place before encoding, but does not affect *sourcemdo*, the encoded results being placed into *targetmdo*. When used without a transfer syntax and without the CHECK keyword, the source MDO is modified in place.

TGT_CHARSET=*name*

Names the target character set. If this operand is specified, the TRANSLATE operand cannot be specified.

Use TGT_CHARSET instead of TRANSLATE to specify translation using the Advanced Translation Facility (ATF).

SRC_CHARSET=*name*

Names the source character set. If this operand is specified, the TRANSLATE operand cannot be specified.

SRC_CHARSET defaults to SOLVE. However, you can specify any other character set name to indicate that you want to translate from that character set.

SINGLE_SUB=*xx*

(Optional) Specifies an overriding target one-byte substitution character. The value is in two hexadecimal characters. For example, to replace all untranslatable characters with an EBCDIC question mark, code SINGLE_SUB=3F (assuming TGT_CHARSET=ASCII).

This operand is valid only if TGT_CHARSET is specified.

DOUBLE_SUB=*xxxx*

(Optional) Specifies an overriding target two-byte substitution character. The value is in four hexadecimal characters.

This operand is valid only if TGT_CHARSET is specified.

CHECK | TRANSFER={ NO | BER }

The CHECK operand specifies that *sourcemdo* is checked for syntax. No translation or data transfer actually takes place, but the specified component is examined to determine whether it conforms to the abstract syntax definition. All subcomponents are examined. Missing mandatory components or invalid data causes the process to terminate.

The TRANSFER operand specifies the use of transfer syntax when encoding. If NO is specified (or defaulted), then no encoding takes place. TRANSFER=BER indicates the use of Basic Encoding Rules as the transfer syntax.

When the TRANSFER operand indicates use of a transfer syntax, the INTO keyword is required to indicate the *targetmdo*. Encoding takes place in accordance with the definition of *sourcemdo*, and the resultant data stream is placed into *targetmdo*. If translation was also requested (by specifying the TRANSLATE operand), translation of source component character strings occurs before encoding, but *sourcemdo* is unaffected.

If the TRANSFER operand is omitted, or TRANSFER=NO is specified, no encoding takes place, and the INTO keyword and *targetmdo* cannot be specified.

- A stem name (for example, MDO=ROSE) indicating that the target is an entire MDO
- A compound name (for example, MDO=CMIP.GETARG) indicating that the target is a component within the MDO

INTO MDO=*targetmdo*

Specifies the target component for the encode operation. The operand is required when the TRANSFER operand is used, but is otherwise invalid.

If a compound name is used, the MDO must exist and be mapped, or the request fails. If only a stem name is provided, the request creates the MDO if it does not exist. The MDO can be connected to a map on a subsequent NCL statement.

Return Codes:

&RETCODE	&ZFDBK	Meaning
0	0	All data successfully encoded
4	0	Input was empty-no data encoded
8	1	Data invalid-could not be encoded
	2	Mandatory component missing
12	0	Undefined MDO name referenced

For &RETCODE 4, the &SYSMSG variable contains a warning message describing the condition.

For &RETCODE 8, the &SYSMSG variable contains an error message describing the component in error, and the actual error condition.

Notes:

- When communicating with other open systems, it is necessary to encode data for transmission in a manner understood by both parties. This means that both systems need only agree on the format of the transmitted data, and not how that data is processed or kept locally in each system.

  A transfer syntax describes the format of the transmitted data. The transfer syntax formally defines the rules for converting an abstract data structure as defined by the abstract syntax into a serial data stream.

  Mapping Services uses Abstract Syntax Notation One (ASN.1), defined by ISO 8824 as the abstract syntax used to describe data structures within your product region. It also uses Basic Encoding Rules (BER) defined by ISO 8825, as the transfer syntax used to serialize data for transmission.

  Using the &ENCODE verb, an NCL procedure can encode all, or part of an MDO, into a serial byte stream, ready for transmission. Often, this process must take place in stages, as some ASN.1 structures require that various substructures are already encoded elsewhere before the structure itself is encoded. To cater for this requirement, encoding takes place from one MDO to another. When all encoding is complete the final MDO is transmitted as a serial byte stream.

  The &DECODE verb provides the reverse process, creating the logical ASN.1 components from a serial byte stream.

  In addition to the serialization of data for transmission, the &ENCODE and &DECODE verbs provide the ability to translate between the ASN.1 defined character sets into the local format, EBCDIC.

- The Advanced Translation Facility (ATF) only translates some data types.

Important differences exist between ATF and the &ENCODE/&DECODE verbs translation facility.

ENCODE could optionally translate from EBCDIC to ASCII, ISO, or DEC.

The differences between the ENCODE TRANSLATE=ASCII operation and the ATF translation from EBCDIC to ASCII are:

- The ENCODE substitution character is X'20', whereas the ATF substitution character is X'1A'.
- ENCODE translates most control characters to X'20', not to their ASCII equivalents.
- ENCODE translates X'B0' in EBCDIC to X'5E' in ASCII. This character is the ASCII caret (^). However, B0 in EBCDIC is not assigned.

The differences between the DECODE TRANSLATE=ASCII operation and the ATF translation from ASCII to EBCDIC are:

- The DECODE substitution character is X'40', whereas the ATF substitution character is X'3F'.
- DECODE translates most ASCII control characters to X'00' (but other undefined characters get translated to X'40').
- DECODE absorbs the following ASCII characters (that is, no output character is generated): X'0F', X'8E', X'8F'.

Do not replace the use of ASCII translation in ENCODE/DECODE with ATF directly.

Example 1:

```
&ENCODE MDO=ROSE CHECK
```

This example validates that an MDO conforms to its defined abstract syntax definition.

Example 2:

```
&ENCODE MDO=CMIP.GETARG TRANSLATE=ISO

&ENCODE MDO=ROSE TRANSLATE=DEC TRANSFER=NO
```

In these examples, components present in *sourcemdo*, and defined as ASN.1 character strings, are translated from EBCDIC to the indicated character set.

Example 3:

```
&ENCODE MDO=CMIP.GETARG TRANSFER=BER INTO +
      MDO=ROSE.ROIVAPDU

&ENCODE MDO=ROSE TRANSLATE=NO TRANSFER=BER INTO +
      MDO=BYTESTR

&ENCODE MDO=ROSE TRANSLATE=ASCII TRANSFER=BER INTO +
      MDO=BYTESTR
```

In these examples, *sourcemdo* is encoded into a serial byte stream, with character string translation from EBCDIC to the indicated character set, if any.

Example 4:

```
&ENCODE MDO=IN TGT_CHARSET=ASCII
```

This example translates character data in the MDO IN into ASCII.

Example 5:

```
&ENCODE MDO=IN TGT_CHARSET=JIS7 TRANSFER=BER INTO MDO=OUT
```

This example translates character data in the MDO IN into JIS7, a standard for single/double-byte character data, and applies ISO BER encoding too. The result is placed into the MDO OUT.

More information:

Code Page Selection (see page 1189)
&DECODE (see page 294)

&END

&END terminates the current nesting level to resume processing at a higher level. Optionally, a return code is passed back to the higher nesting level, and becomes available in the &RETCODE system variable.

```
&END [ return code ]
```

Operands:

return code

This is a numeric value from 0 to 99. If specified, this value becomes available in the &RETCODE system variable for testing by a higher nesting level (if one exists).

Examples: &END

```
&WRITE ALARM=YES DATA=ENTER LU NAME
&PAUSE ARGS
&IF .&1 = .  &THEN +
   &END 4
...
```

On return to a higher nesting level, the procedure can test the return code, for example:

```
&IF &RETCODE = 4 &THEN +
   &WRITE ALARM=YES DATA=LU NAME OMITTED
```

Notes:

If the return code is not specified on an &END statement, then any value set previously in another procedure level remains intact and becomes available across multiple nesting levels.

The &RETURN statement is used to return to a higher nesting level, passing any specified variables.

The &CONTROL SHRVARS statement is used to share variables between nested procedure levels.

More information:

&RETCODE (see page 861)
&CONTROL (see page 266)
&RETCODE (see page 573)
&RETURN (see page 576)
&EXIT (see page 319)

&ENDAFTER

&ENDAFTER terminates the current nesting level after executing the command following &ENDAFTER.

```
&ENDAFTER { command | statement }
```

&ENDAFTER is used to simplify procedure coding. Many procedures perform one command per operator entry, or similar. Each command might then require an &END statement, or an &IF must issue an &GOTO to branch to a point to perform a single function.

&ENDAFTER lets you execute a single command in conjunction with an &IF statement (if required), followed by automatic termination of the current nesting level.

This allows &GOTO statements to be eliminated from a procedure.

Operands:

command* | *statement

Specifies the text of the command or statement to be executed. Processing of the current nesting level finishes after this command or statement is executed.

Examples: &ENDAFTER

```
&IF .&OPTION = .  &THEN +
   &ENDAFTER -EXEC DFLTPROF
&IF .&OPTION = .TAPE &THEN +
   &ENDAFTER -EXEC TAPEPROF
&ENDAFTER -PROFILE ENV=PRIMARY UNSOL=NO
```

Notes:

If an EXEC of another procedure or nesting level is performed as the function of an &ENDAFTER statement, the new nesting level is executed before processing at the current nesting level is completed.

The following verbs are illegal on an &ENDAFTER statement:

```
&DO
&DOUNTIL
&DOWHILE
&ELSE
&ENDAFTER
&IF
&GOSUB
&GOTO
```

More information:

&IF (see page 364)

&EVENT

The &EVENT verb signals an event occurrence.

This verb has the following format:

```
&EVENT [ NAME=event name ]
       [ TYPE={ APPLICATION | SERVICEABILITY | UTILIZATION | CONFIGURATION |
                ACCESS | PROCEDURAL } ]
       [ SCOPE={ SYSTEM | REGION } ]
       [ OBJECT=object ]
       [ RESOURCE=resource ]
       [ REFERENCE=event reference code ]
       [ ROUTCDE=route code ]
       [ DATA=data | MDO=stem | VARS=... | ARGS [ RANGE=(start,end) ] ]
```

To signal listeners who are profiled for the declared event. The &EVENT statement causes message N00102 to be queued to the response queue of any processes that have an active event profile which matches the attributes of the &EVENT statement operands. The event listener should retrieve the N00102 message with an &INTREAD statement.

Operands:

NAME=*event name*

Specifies a mandatory 1- to 32-byte event identifier used to provide information about the event source. Names is composed of any valid NCL variable name characters and full stop (.) or underscore (_) characters. Names beginning with a dollar sign ($) are reserved for internal use.

TYPE=*event type*

Event type is a high-level event category which provides an efficient event profile filter. Valid event types are as follows:

APPLICATION

User-defined (this is the default).

SERVICEABILITY

Faults, errors, availability, degradation, recovery.

UTILIZATION

Statistics, raw performance, and accounting data, RTM.

CONFIGURATION

Object definition, relationship notifies.

ACCESS

Security alarms.

PROCEDURAL

Scheduling, process control.

SCOPE={ SYSTEM | REGION }

The default event scope of SYSTEM means that an event is delivered to all listeners in the Management Services domain. Scope of REGION is used to limit event notification to processes within the user's region.

OBJECT=*object*

A 1- to 32-byte object classification of the event resource. For example, LU, PU, or SESSION are SNA object categories.

RESOURCE=*resource*

A single resource instance, or a resource instance pair separated by commas. A resource pair is specified when a relationship exists between event resources (for example, a session pair). Each resource name is 1 to 64 bytes long.

REFERENCE=*event reference code*

Specifies a 1- to 32-byte event code such as message number or error code.

ROUTCDE=*route code*

Specifies a list of numbers from 1 to 128 which represent the event route code. Each number is represented by a bit. To qualify for event notification, receivers must have at least one corresponding route code set in the event profile.

DATA=*data* | MDO=*stem* | VARS=... | ARGS [RANGE=(*start*,*end*)]

Specifies optional data or mapped object to be passed to event receivers of the N00102 message. If DATA=data is specified , the data is part of the N00102 text. If MDO=*stem* is specified, the mapped object is available in the $INT MDO received by the &INTREAD verb (or directly in the MDO specified on the &INTREAD MDO=*mdo* operand).

Specifying VARS=... or ARGS results in a $NCL MDO being built and delivered in the $MSG MDO, containing the named variables or arguments.

Note: The DATA, MDO, VARS, and ARGS operands are mutually exclusive.

RANGE=(*start*,*end*) is specified with ARGS, to denote an argument range.

Notes:

Event classification generally depends on how much subjective rule based interpretation has been performed. For example an event containing raw performance data should be classified as a utilization event. A rule based application receiving this event might interpret it as a performance problem and generate a secondary serviceability event. This might then be passed on to another application which generates procedural events.

&EVENT resets the &RETCODE variable to zero. If at least one listener is profiled for the event, &ZFDBK is set to 00. If no listeners were notified of this event, &ZFDBK is set to 04.

Examples: & EVENT

```
&EVENT NAME=SNA_SESSION_COMPLETION +
       TYPE=CONFIGURATION +
       OBJECT=SESSION +
       REF=N01208 +
       RESOURCE=(&APPL,&LUNAME)+
       DATA=&VTAMTEXT
```

More information:

Sample Code (see page 1171)

&EXIT

The &EXIT verb terminates the current nesting level to resume processing at a higher level. Optionally, a return code is passed back to the higher nesting level, and becomes available in the &RETCODE system variable.

This verb has the following format:

```
&EXIT [ return code ]
```

Operands:

return code

This is a numeric value from 0 to 99. If specified, this value becomes available in the &RETCODE system variable for testing by a higher nesting level (if one exists).

Examples: &EXIT

```
&WRITE ALARM=YES DATA=ENTER LU NAME
&PAUSE ARGS
&IF .&1 = .  &THEN +
   &EXIT 4
.
.
.
```

On return to a higher nesting level, the procedure can test the return code, for example:

```
&IF &RETCODE = 4 &THEN +
   &WRITE ALARM=YES DATA=LU NAME OMITTED
```

Notes:

If the return code is not specified on an &EXIT statement, then any value set previously in another procedure level remains intact and becomes available across multiple nesting levels.

The &RETURN statement is used to return to a higher nesting level, passing any specified variables.

The &CONTROL SHRVARS statement is used to share variables between nested procedure levels.

More information:

&RETCODE (see page 861)
&CONTROL (see page 266)
&RETCODE (see page 573)
&RETURN (see page 576)
&END (see page 314)

&FILE

The &FILE verb connects, disconnects, switches, accesses, modifies, and deletes file records.

This verb has the following format:

```
&FILE { ADD | PUT } ID=fileid
                  [ KEY=key | KEYVAR=keyvar ]
                  [ OPT={ TRUNCATE | NOTRUNCATE } ]
                  [ { ARGS | VARS=prefix* } [ RANGE=(start,end) ] |
                      VARS={ var | ( var1,var2,...,varn ) } |
                      DATA=data | MDO=mdoname ]
                  [ PRTCNTL=opt | (opt1,opt2 [,opt3,opt4 ] )  ]

&FILE CLOSE [ OPT=ALL | ID=fileid ]

&FILE DEL ID=fileid
        [ KEY=key | KEYVAR=keyvar ]
        [ GENLEN=nn ]
        [ OPT={ KEQALL | KGEALL } ]

&FILE GET ID=fileid
        [ KEY=key | KEYVAR=keyvar ]
        [ OPT=type ]
        [ GENLEN=genlen ]
        [ { ARGS | VARS=prefix* } [ RANGE=(start,end) ] |
            VARS={ var | (var1,var2,...,varn) } |
            MDO=mdoname [ MAP=mapname ] ]

&FILE OPEN ID=fileid
         [ FORMAT={ MAPPED | UNMAPPED | UNMAPPED-DBCS | DELIMITED } ]
         [ MAP={ $NCL | mapname } ]
         [ KEYPAD={ BLANK | NULL } ]
         [ KEYEXTR={ YES | NO } ]
         [ DATA=exitdata ]

&FILE SET ID=fileid
        [ KEY=key | KEYVAR=keyvar ]
        [ GENLEN=nn ]
        [ FORMAT={MAPPED | UNMAPPED | DELIMITED } ]
        [ MAP={ $NCL | mapname } ]
```

The use of the ADD/PUT, CLOSE, DEL, GET, OPEN, and SET options of the &FILE verb is described on the following pages.

Example 1

This example reads the USERFILE and produces a listing of all users starting with the user ID specified in variable &STARTUSER.

```
&FILE OPEN ID=USERFILE FORMAT=DELIMITED
&IF &FILERC GT 8 &THEN +
   &GOTO .ERRMSG
&FILE SET ID=USERFILE KEY=&STARTUSER
&WRITE DATA=POS  USER ID USER NAME +
            DEPT PHONE NO
&L = &ASISTR&COUNT = 1
&DOUNTIL &FILERC GT 0
   &WRITE DATA=&L
   &FILE GET ID=USERFILE OPT=KGE VARS=(NAME,DEPT,PHONE)
   &IF &FILERC EQ 4 &THEN +
    &DO
     &L = &OVERLAY &L &COUNT 1 3 ALIGNR0
     &L = &OVERLAY &L &FILEKEY 7 8 ALIGNL
     &L = &OVERLAY &L &NAME 17 20 ALIGNL
     &L = &OVERLAY &L &DEPT 40 4 ALIGNL
     &L = &OVERLAY &L &PHONE 47 9 ALIGNL
     &COUNT = &COUNT + 1
    &DOEND
&DOEND
&WRITE DATA=** END OF LISTING **
&IF &FILERC NE 4 &THEN +
   &GOTO .ERRMSG
&FILE CLOSE ID=USERFILE
&END 0
.ERRMSG
&WRITE DATA=FILE NOT AVAILABLE OR ERROR OCCURRED +
            ACCESSING FILE +
           (FILERC=&FILERC, VSAMFDBK=&VSAMFDBK)
   &END 4
```

Example 2

This example updates the USERFILE file. The user ID to be updated is specified in variable &UPDUSERID. If the user details record is not found, then a new record is added to the file. The authority of the user is checked when the file is opened.

```
&FILE OPEN ID=USERFILE FORMAT=DELIMITED
&IF &FILERC GT 8 &THEN +
   &GOTO .ERRMSG
&ELSE +
   &IF &FILERC LT 4 &THEN +
      &DO
       &WRITE DATA=NOT AUTHORIZED TO UPDATE FILE
       &END 4
      &DOEND
&FILE GET ID=USERFILE KEYVAR=UPDUSERI +
            VARS=(NAME,DEPT,PHONE)
&IF &FILERC EQ 0 &THEN +
   &DO
      &NAME = &NEWNAME
      &DEPT = XXXX
      &PHONE = 1234567
      &FILE ADD ID=USERFILE VARS=(NAME,DEPT,PHONE)
      &FUNC = ADDED
   &DOEND
&IF &FILERC NE 0 &THEN +
   &GOTO .ERRMSG
&WRITE DATA=USER &UPUSERID WAS SUCCESSFULLY PROCESSED +
           ACTION TAKEN WAS &FUNC
&FILE CLOSE ID=USERFILE
&END0
.ERRMSG
   &WRITE DATA=USER FILE NOT AVAILABLE OR ERROR PROCESSING +
               FILE (FILERC=&FILERC, VSAMFDBK=&VSAMFDBK)
   &END 4
```

Example 3

This example deletes a record from the USERFILE file. The user ID to be deleted is specified in variable &DELUSERID. The authority of the user is checked when the file is opened.

```
&FILE OPEN ID=USERFILE FORMAT=DELIMITED
&IF &FILERC GT 8 &THEN +
   &GOTO .ERRMSG
&ELSE +
   &IF &FILERC NE 8 &THEN +
      &DO
          &WRITE DATA=NOT AUTHORIZED TO DELETE RECORDS +
                      FROM FILE
      &DOEND
&FILE GET ID=USERFILE KEYVAR=DELUSERID +
            VARS=(NAME,DEPT,PHONE)
&IF &FILERC EQ 0 &THEN +
   &FILE DEL ID=USERFILE &ELSE +
   &IF &FILERC EQ 4 THEN +
      &DO
          &WRITE DATA=USER &DELUSERID NOT FOUND ON FILE
          &END 4
      &DOEND
&IF &FILERC NE 0 &THEN +
   &GOTO .ERRMSG
      &WRITE DATA=USER &UPDUSERID WAS SUCCESSFULLY PROCESSED +
                  ACTION TAKEN WAS DELETE
&FILE CLOSE ID=USERFILE
&END 0
.ERRMSG
   &WRITE DATA=USER FILE NOT AVAILABLE OR ERROR +
               PROCESSING +
               FILE (FILERC=&FILERC, VSAMFDBK=&VSAMFDBK)
   &END 4
```

&FILE ADD

The &FILE ADD verb adds a file record.

This verb has the following format:

```
&FILE ADD ID=fileid
        [ KEY=key | KEYVAR=keyvar ]
        [ OPT={ TRUNCATE | NOTRUNCATE } ]
        [ { ARGS | VARS=prefix* } [ RANGE=(start,end) ] |
            VARS={ var | ( var1,var2,...,varn )|
            DATA=data | MDO=mdoname ]
        [ PRTCNTL=opt | (opt1,opt2 [ ,opt3,opt4 ] ) ]
```

Operands:

ID=*fileid*

Identifies the file.

KEY=*key*

Specifies the full key value to set for the file. The key value (after substitution) is in one of several forms:

- An unquoted character string (no embedded blanks), for example:

  ```
  01SMITH
  ```

  For unquoted strings, the key consists of displayable characters only, and the first blank delimits it.

- A quoted character string, for example:

  ```
  '01SMITH J.'   “03JONES PETER A”    “LU A0123BC2”C 'A. B. O''MALLEY'C
  ```

  The usual quote rules apply. Either single quotes (') or double quotes (”) are acceptable. Quotes must be paired and of the same type. A double quote is treated as a single quotation character when included within a string.

- A quoted hexadecimal character string, for example:

  ```
  '0012001F'X
  ```

  ```
  “0123456789ABCDEF”X
  ```

If there are no characters following the closing quote, a character string is assumed. If a single character follows the closing quote then it must be either 'C' (designating a character string), or 'X' (designating a hexadecimal string). For quoted hexadecimal strings, all characters must be valid hexadecimal characters (that is, 0 through 9 and A through F).

The user is responsible to understand the file key structure and determine which form of key designation is most appropriate.

Note: The &ZQUOTE built-in function can assist when building quoted strings, and that the KEYVAR operand provides a suitable alternative.

KEYVAR=*keyvar*

Provides an alternative way to nominate the full key value. *keyvar* is the name of a user or system variable, the contents of which are taken unchanged as the key value for the file set operation. The ampersand (&) is not required—if the & is coded, then the contents of the variable specified are assumed to contain the name of the variable containing the key.

If both the KEY and KEYVAR operands are omitted, the current file key value is used.

OPT={ TRUNCATE | NOTRUNCATE }

Specifies whether to allow truncation of data on an &FILE ADD. If NOTRUNCATE is specified (or defaulted), the procedure ends, with an error message, if the data exceeds the maximum record length of the file. When TRUNCATE is specified, a file return code of 1 is set if truncation occurs.

{ ARGS | VARS=*prefix } [RANGE (*start*,*end*)]**

Nominates the NCL variables for the ADD operation. NCL variables are validly used as the source of the ADD operation when the file format is DELIMITED or UNMAPPED, and when the format is MAPPED and MAP=$NCL. How the variable appears on the file depends upon the current processing mode (format). If the format is UNMAPPED, then the variables are simply concatenated to form the record. If the format is DELIMITED, the variables are placed on the file separated by X'FF' separators.

VARS={ *var* | (*var*1,*var*2,...,*varn*) }

Specifies the list of variables to write to the file. NCL variables are validly used as the source of the ADD operation when the file format is DELIMITED or UNMAPPED, and when the format is MAPPED and MAP=$NCL.

DATA=*data*

Data is allowed for DELIMITED or UNMAPPED format files only, and specifies a string of data for the file record. Substitution occurs on the specified string of data, and then the data is placed on the record as is.

MDO=*mdoname*

Nominates the MDO for the ADD operation. An MDO is the source of the data record on an ADD operation only when the file format is MAPPED.

PRTCNTL=*opt* | (*opt*1,*opt*2 [,*opt*3,*opt*4])

If FILE ADD writes to OS/VS SYSOUT data sets, use PRTCNTL to specify carriage control options to control print formatting. Print format categories are as follows:

- Paper movement—SKIP0, SKIP1, SKIP2, SKIP3, NEWPAGE, and DATA
- Underscoring—USCORE1 and USCORE2
- Text alignment—LEFT, RIGHT, and CENTER
- Bold print—BOLD

One option from each of these categories is specified.

Note: Multiple options are specified, but they must be enclosed in parentheses and separated by commas. For example:

```
PRTCNTL=(NEWPAGE,USCORE1,CENTER,BOLD)
```

SKIP0

Does not advance the paper before writing the record.

SKIP1

Advances the paper one line before writing the record. When writing records to OS/VS SYSOUT data sets, this value is the default if the PRTCNTL operand is omitted.

SKIP2

Advances the paper two lines before writing the record.

SKIP3

Advances the paper three lines before writing the record.

NEWPAGE

Skips to a new page before writing the record.

DATA

Indicates that the first text character contains the carriage control character for controlling the print options for printing this record. If machine control characters are used (instead of ANSI control characters), DATA is the only carriage control option allowed.

USCORE1

Underlines the text for this record (excluding blanks between words).

USCORE2

Underlines the text for this record including blanks between words.

LEFT

Aligns the text for this record to the left of the paper.

RIGHT

Aligns the text for this record to the right of the paper.

CENTER

Centers the text for this record. The record width, and therefore the location of the center, is determined from the logical record length of the file. The length is specified in the LRECL operand of the ALLOC command (for z/OS or MSP).

BOLD

Prints the text in bold form. Bold is achieved by overprinting the same characters.

Return Codes:

On completion of the operation, the &FILERC system variable is set as follows:

0

record added successfully

1

record added, truncation has occurred

4

record exists

8

error during processing of ADD request

16

NCL or Mapping Services processing error

Check &SYSMSG for details. The &VSAMFDBK system variable contains the VSAM completion code. For &FILERC=8, the &VSAMFDBK system variable is tested to determine the cause of the error. These error codes are explained in detail in your VSAM documentation. The &VSAMFDBK variable is always returned as a two-character value.

The &SYSMSG variable can also be set to contain error message details.

For mapped format files, the &ZMDORC and &ZMDOFDBK system variables are also set.

&FILE CLOSE

The &FILE CLOSE verb disconnects from one or more files.

This verb has the following format:

```
&FILE CLOSE [ OPT=ALL | ID=fileid ]
```

Operands:

OPT=ALL

Requests that all files currently open to this NCL process be closed.

ID=*fileid*

(Mandatory unless OPT=ALL) Identifies a single file to be closed.

Return Codes:

The &FILERC system variable is not set by this operation.

More information:

&FILE (see page 321)

&FILE DEL

The &FILE DEL verb deletes file records.

This verb has the following format:

```
&FILE DEL ID=fileid
        [ KEY=key | KEYVAR=keyvar ]
        [ GENLEN=nn ]
        [ OPT={ KEQALL | KGEALL } ]
```

Operands:

ID=*fileid*

(Mandatory) Identifies the file being deleted and sets this file to be the current *fileid*.

KEY=*key*

Nominates the full key value to be set for the file. The key value (after substitution) is in one of several forms:

- An unquoted character string (no imbedded blanks). For example:

  ```
  01SMITH
  ```

  For unquoted strings the key should consist of displayable characters only, and is delimited by the first blank.

- A quoted character string, for example:

  ```
  '01SMITH J.'   “03JONES PETER A”    “LU A0123BC2”C 'A. B. O''MALLEY'C
  ```

  The usual quote rules apply. Either single quotes (') or double quotes (”) are acceptable. Quotes must be paired and of the same type. A double quote is treated as a single quotation character when included within a string.

- A quoted hexadecimal character string. for example:

  ```
  '0012001F'X
  ```

  ```
  “0123456789ABCDEF”X
  ```

If there are no characters following the closing quote, a character string is assumed. If a single character follows the closing quote then it must be either 'C' (designating a character string), or 'X' (designating a hexadecimal string). For quoted hexadecimal strings, all characters must be valid hexadecimal characters (that is, 0-9, A-F).

It is the user's responsibility to understand the file key structure and determine which form of key designation is most appropriate.

Note: The &ZQUOTE built-in function can assist when building quoted strings, and that the KEYVAR operand provides a suitable alternative.

KEYVAR=*keyvar*

Provides an alternative way to nominate the full key value. *keyvar* is the name of a user or system variable, the contents of which are taken unchanged as the key value for the file set operation. There is no need to code the ampersand (&)—if the ampersand is coded, then the contents of the variable specified are assumed to contain the name of the variable containing the key.

If both KEY and KEYVAR are omitted, the current file key value is assumed.

GENLEN=*nn*

This operand is used to set the desired generic key length independently of the key value. If the length of the key provided is shorter than the value of GENLEN, then the length of the key is the effective GENLEN. This length is used for a generic delete (KEQALL) request.

OPT={KEQALL | KGEALL}

Either one of these operands is used to specify the delete option:

KEQALL

Specifies that a series or generic set of records is to be deleted. A partial key is specified by the &FILE DEL statement. All records that start with this partial key are deleted. After processing, the system variable &FILERCNT contains the number of records deleted by this statement.

KGEALL

Specifies that all records starting from the already-named partial key to the end of the UDB, are to be deleted. After processing, the system variable &FILERCNT contains the number of records deleted by this statement.

Return Codes:

On completion of the operation, the &FILERC system variable is set as follows:

0

Records deleted successfully

4

Record not found

8

Error during processing of delete request

16

NCL or Mapping Services processing error. Check &SYSMSG for details.

Check &SYSMSG for details. The &VSAMFDBK system variable contains the VSAM completion code. For &FILERC=8, the &VSAMFDBK system variable is tested to determine the cause of the error. These error codes are explained in detail in your VSAM documentation. The &VSAMFDBK variable is always returned as a 2-character value.

The &SYSMSG variable can also be set to contain error message details.

More information:

&FILE (see page 321)

&FILE GET

The &FILE GET verb accesses a file record or a sequence of file records.

This verb has the following format:

```
&FILE GET ID=fileid
        [ KEY=key | KEYVAR=keyvar ]
        [ OPT=type ]
        [ GENLEN=nn ]
      { { ARGS | VARS=prefix* } [ RANGE (start,end) ] |
        VARS={ var | (var1,var2,...,varn) |
        MDO=mdoname [ MAP=mapname ] }
```

Operands:

ID=*fileid*

Identifies the file being accessed and sets this file to be the current *fileid*. This operand is mandatory.

KEY=*key*

Nominates the full key value to be set for the file. The key value (after substitution) is in one of several forms:

- An unquoted character string (no imbedded blanks). For example:
- 01SMITH

  For unquoted strings the key should consist of displayable characters only, and is delimited by the first blank.

- A quoted character string, for example:

  ```
  '01SMITH J.'   “03JONES PETER A”    “LU A0123BC2”C 'A. B. O''MALLEY'C
  ```

  The usual quote rules apply. Either single quotes (') or double quotes (”) are acceptable. Quotes must be paired and of the same type. A double quote is treated as a single quotation character when included within a string.

- A quoted hexadecimal character string. for example:

  ```
  '0012001F'X
  ```

  ```
  “0123456789ABCDEF”X
  ```

If there are no characters following the closing quote, a character string is assumed. If a single character follows the closing quote then it must be either 'C' (designating a character string), or 'X' (designating a hexadecimal string). For quoted hexadecimal strings, all characters must be valid hexadecimal characters (that is, 0 to 9, A to F).

It is the user's responsibility to understand the file key structure and determine which form of key designation is most appropriate.

Note: The &ZQUOTE built-in function can assist when building quoted strings, and that the KEYVAR operand provides a suitable alternative.

KEYVAR=*keyvar*

Provides an alternative way to nominate the full key value. *keyvar* is the name of a user or system variable, the contents of which are taken unchanged as the key value for the file set operation. There is no need to code the ampersand (&)—if it is coded, the contents of the variable specified are assumed to contain the name of the variable containing the key.

Note: If both KEY and KEYVAR are omitted, the current file key value is assumed. The KEY and KEYVAR operands are mutually exclusive. It is possible to specify the OPT= option in conjunction with KEY or KEYVAR if it is a generic option, for example, OPT=KEQ, OPT=KLE, and so on. The KEY operand is only used once. Once generic position is obtained, the key operand is ignored.

OPT=*type*

Specifies a retrieval option that determines the search argument for the record to be read.

The following options are available for the OPT operand: KGE, KEQ, KGT, KEL, KLE, KLT, FWD, BWD, SEQ, SAVE, UPD, and END.

KGE

Specifies that a record key greater than or equal to the specified partial key is returned. Using KGE permits generic retrieval where a partial key is supplied-any records with a key (or part key) equal to or greater than that specified are returned.

KEQ

Specifies that a record key equal to the specified partial key is returned. Using KEQ permits generic retrieval where a partial key is supplied. Any records with a partial key equal to that specified are returned, starting with the lowest full key value and continuing forwards to the highest full key value.

After a successful retrieval by KGE or KEQ, the full key for the returned record is determined from the &FILEKEY system variable. The first &FILE GET that uses the KGE or KEQ operand establishes the starting point within the file from which records are to be retrieved. Successive &FILE GET statements specifying the KGE or KEQ operand continue returning records with a higher key. The generic retrieval is terminated by specifying the KEY= operand on an &FILE GET, &FILE SET, &FILE PUT or &FILE ADD statement, or by using the &FILE GET OPT=END operand.

KGT

Specifies that a record key greater than the specified partial key is returned. KGT is normally used to process a group of records where individual records are being updated. Updating a record during a generic retrieval process normally interrupts the generic retrieval and using KGT can avoid this. If no generic retrieval is in progress, KGT processing ensures that a record greater than the current &FILEKEY is returned. If a generic retrieval is in progress, the next key is returned. The procedure normally positions at a specific record using a KEQ call, and then continues processing using KGT calls.

Alternatively, the procedure could set a partial key lower than required and let the KGT call return the next highest record. Following successful retrieval, the full key of the record returned is determined from the &FILEKEY system variable. KGT differs from KGE processing in that a KGT call is not impacted by other file processes that interrupt a generic process, such as updates and deletes.

Generic retrieval must be terminated by specifying the KEY= operand on an &FILE GET, &FILE SET, &FILE PUT, &FILE ADD statement or by using the &FILE GET OPT=END operand.

KEL

Specifies that a record key equal to the specified partial key is returned. Using KEL permits generic retrieval where a partial key is supplied. Any records with a partial key equal to that specified are returned, starting from the highest full key value and continuing backwards to the lowest full key value.

KLE

Specifies that a record key less than or equal to the specified partial key is returned. Using KLE permits generic retrieval where a partial key is supplied-the highest record with a partial key equal to or less than that specified is returned.

After a successful retrieval by KEL or KLE, the full key for the returned record is determined from the &FILEKEY system variable. The first &FILE GET that uses the KEL or KLE operand establishes the starting point within the file from which records are to be retrieved. The first record returned is the one with the highest key that matches the partial key supplied. Successive &FILE GET statements specifying the KEL or KLE operand continue returning records with a lower key. The generic retrieval is terminated by specifying the KEY= operand on an &FILE GET, &FILE SET, &FILE PUT or a &FILE ADD statement or by using the &FILE GET OPT=END operand.

KLT

Specifies that a record key less than the specified partial key is returned. KLE is normally used when processing a group of records in backwards mode where individual records are being updated. Updating a record during a generic retrieval process normally interrupts the generic retrieval and using KLT can avoid this. If no generic retrieval is in progress, KLT processing ensures that a record less than the current &FILEKEY is returned. If a generic retrieval is in progress, the next lower key is returned.

The procedure normally positions at a specific record using a KEL call, and then continues processing using KLT calls. Alternatively, the procedure could set a partial key higher than required and let the KLT call return the next lowest record.

Following successful retrieval, the full key of the record returned is determined from the &FILEKEY system variable. KLT differs from KLE processing in that a KLT call is not impacted by other file processes that interrupt a generic process, such as updates and deletes. Generic retrieval must be terminated by specifying the KEY= operand on an &FILE GET, &FILE SET, &FILE PUT, &FILE ADD statement or by using the &FILE GET OPT=END operand.

FWD

Specifies that sequential retrieval is performed in a forward direction (that is, ascending keys for a KSDS). If no preceding &FILE GET statement has established a position within the file, retrieval starts with the lowest keyed record and subsequent &FILE GET statements return records in ascending key order until the highest keyed record has been returned.

BWD

Specifies that sequential retrieval is performed in a backward direction (that is, descending keys for a KSDS). If no preceding &FILE GET statement has established a position within the file, retrieval starts with the highest keyed record and subsequent &FILE GET statements return records in descending key order until the lowest keyed record has been returned.

To retrieve records from a specific point, first issue an &FILE GET statement to retrieve either a specific key or partial key and follow this with a series of &FILE GET FWD, or &FILE GET BWD statements to process the required record range. No &FILE GET KEY statement is required to commence sequential processing. Therefore it is not necessary to know in advance any of the keys on the data set. An &FILE GET using the KEY= operand is required if positioning to a specific point using a preliminary call with one of the generic retrieval options.

An &FILE GET FWD can directly follow an &FILE GET BWD (or KEL, KLE, KLT option) to reverse the processing direction of the file. Alternatively, an &FILE GET BWD can directly follow an &FILE GET FWD(or KEQ, KGE, KGT option) to reverse the processing direction of the file (see also the &FILE GET SEQ option next).

&FILE GET can also be used to process an ESDS VSAM file.

SEQ

Specifies that sequential retrieval is performed. Sequential retrieval is an efficient way of retrieving large numbers of records. If no preceding &FILE GET statement has established a start position within the file, the &FILE GET SEQ is equivalent to &FILE GET FWD. Otherwise, sequential retrieval starts from the current file position and in the current file direction. This means that an &FILE GET SEQ following an &FILE GET FWD (or KEQ, KGE, KGT option) retrieves the records following, whereas if it follows an &FILE GET BWD (or KEL, KLE, KLT option) it retrieves the previous records.

After a successful retrieval, the full key for the returned record is determined from the &FILEKEY system variable. Sequential retrieval is terminated by one of the following methods:

- Specifying the KEY= operand on an &FILE GET or &FILE SET
- Issuing an &FILE PUT or &FILE ADD statement
- Using the &FILE GET OPT=END operand
- Using an &FILE GET statement with an option other than SEQ

OPT=END is used to explicitly terminate sequential processing.

SAVE

Specifies that the current generic retrieval option is saved. This option releases VSAM positioning in the file but remembers the last partial key set by an &FILE GET KEY statement and the last full key read using &FILE GET. Since generic file retrieval involves retaining various VSAM resources, it is not good practice to hold your position in a VSAM file when an extended delay is possible (for example, whilst waiting for input from a terminal). By issuing an &FILE GET SAVE, the VSAM resources required for the generic retrieval are released; when the next generic retrieval request is received the next record is returned as though no interruption had occurred. That is, the record returned is the one that would have been returned anyway had the &FILE GET SAVE not been issued.

Since the only retrieval options on ESDS files are sequential (that is, FWD, BWD, and SEQ) and therefore hold VSAM position, the SAVE option is used effectively when browsing such files to remember the current position if a delay is expected.

UPD

Specifies that the record nominated by the KEY= operand from the preceding &FILE statement is retrieved for updating.

Using this option gives you exclusive use of the specified record, if it is not currently in use elsewhere. This ensures that no other retrieval of the same record is permitted before the record is replaced with an &FILE PUT statement.

Where multiple users can simultaneously perform record updating, use this option to ensure that no overlap in update processing is possible, and that the record is being processed uniquely by this requester.

When using this option, you are responsible for ensuring that the record is used exclusively. If another user is already processing the same record when you issue this request, it will fail with the appropriate return codes and you must then retry at a later time. If exclusive use is not obtained, the &FILERC system variable will be set to 8 and the &VSAMFDBK system variable will contain a value of 14.

Important! Procedures using this technique must not perform processing that results in excessive delays (such as prompting the terminal operator for input). Remember that other users cannot use a record while you have exclusive use of it.

END

Using this operand terminates a generic or sequential retrieval operation, and resets the current file position. This option should be used if a generic retrieval operation is to be halted and another generic retrieval operation performed with a different key or partial key. This operand can also be used to release exclusive control of a record obtained by the UPD operand, without needing to actually update the record. Using the &FILE CLOSE statement carries an implied &FILE GET OPT=END operation.

Note: &FILE GET OPT=END forces the flushing of all deferred I/O buffers, thereby committing any deferred update activity and ending any generic retrieval environment.

GENLEN=*nn*

Used to set the desired generic key length independently of the key value. This length is used for a generic GET request.

{ ARGS | VARS=*prefix } [RANGE (*start*,*end*)] |**
VARS={ *var* | (*var*1,*var*2,...,*varn*) } |*MDO=stem* |MAP=*mapname*

(Mandatory) One of the options must be specified.

{ ARGS | VARS=*prefix } [RANGE (*start*,*end*)]**

Nominates the NCL variables for the read operation. NCL variables are validly used as the target of the GET operation when the file format is DELIMITED or UNMAPPED, and when the format is MAPPED and MAP=$NCL.

VARS={ *var* | (*var*1,*var*2,...,*varn*) }

Specifies the list of variables to be read from the file. NCL variables are validly used as the target of the GET operation when the file format is DELIMITED or UNMAPPED, and when the format is MAPPED and MAP=$NCL. It is possible to subscript variables in the list with a data length, and use an asterisk (*) as a place holder. For example:

```
VARS=(a(10),b(5),*,c)
```

MDO=*stem*

Nominates the MDO for the read operation. An MDO is the target of the GET operation only when the file format is MAPPED.

MAP=*mapname*

Valid only if FORMAT=MAPPED is specified or defaulted on the &FILE OPEN statement for this file. It provides the default Mapping Services map name to be used to interpret the file contents. The map name specified should be registered within the Mapping Services Data Dictionary or the data read from the file will be effectively unmapped. The default name $NCL is a special case that allows NCL tokens to be used on PUT and GET statements instead of the MDO operand. The tokens are placed within a record structure that provides data transparency but is not compatible with the DELIMITED format files.

Return Codes:

0

record retrieved successfully, key in &FILEKEY variable

4

record not found (or end of data)

8

error during processing of get request

16

NCL or Mapping Services processing error, check &SYSMSG for details

The &VSAMFDBK system variable contains the VSAM completion code. For &FILERC=8, the &VSAMFDBK system variable is tested to determine the cause of the error. These error codes are explained in detail in your VSAM documentation. The &VSAMFDBK variable is always returned as a 2-character value.

The &SYSMSG variable can also be set to containing error message details.

For MAPPED format files, the &ZMDORC and &ZMDOFDBK system variables are set.

The &ZVARCNT is set by the verb to indicate how many variables were set by the get operation.

More information:

&FILE (see page 321)

&FILE OPEN

The &FILE OPEN verb connects to a file.

This verb has the following format:

```
&FILE OPEN ID=fileid
           [ FORMAT={ MAPPED | UNMAPPED | UMMAPPED-DBCS | DELIMITED } ]
           [ MAP={ $NCL | mapname } ]
           [ KEYPAD={ BLANK | NULL } ]
           [ KEYEXTR={ YES | NO } ]
           [ DATA=exitdata ]
```

Operands:

ID=*fileid*

(Mandatory) Identifies the file being opened. *fileid* must have previously been made accessible to NCL by the UDBCTL statement.

FORMAT={ MAPPED | UNMAPPED | UNMAPPED-DBCS | DELIMITED }

Specifies the file format processing requirement. The default is MAPPED which generally requires the use of Mapping Services operands on PUT and GET statements. A file record is processed as a complete entity by reading it into, or writing it from, an MDO. Mapping Services is used to reference, by symbolic name, individual fields within the record. For the special case where the map name is $NCL, the record is read into or written from standard tokens. Mapping Services is used to format or access the MDO containing the tokens. This provides data transparency by allowing tokens containing any data to be placed in file records for subsequent NCL access.

By processing a file in UNMAPPED format, any file records is processed using NCL tokens (but not Mapping Services). When reading a record the contents are segmented and placed into the tokens specified. When writing a record, all tokens are concatenated to form the actual record. No examination or translation of the token data takes place in either case.

Specifying FORMAT=UNMAPPED-DBCS is the same as FORMAT=UNMAPPED. However, if SYSPARMS DBCS=FUJITSU is in effect, DBCS translation between IBM DBCS and FUJITSU DBCS occurs.

FORMAT=DELIMITED provides processing that is used when reading into or writing from NCL tokens only. When reading a record, the contents are scanned for the delimiter character (X'FF') and each delimited section of the record placed into a separate token. However, no examination of the data takes place on writing so the use of this format with transparent data can cause unpredictable results.

MAP={ $NCL | *mapname* }

Valid only if FORMAT=MAPPED is specified or defaulted, and specifies the name of the Mapping Services map used to interpret the file contents.

The map name specified must be registered within the Mapping Services Data Dictionary, or the data read from the file is effectively unmapped. The default name $NCL is a special case that allows NCL tokens to be used on PUT and GET statements instead of the MDO operand. The tokens are placed within a record structure that provides data transparency but is not compatible with the DELIMITED format files.

KEYPAD={ BLANK | NULL }

Nominates the padding character for short keys as either blanks (X'40') or nulls (X'00'). For most character-oriented or name-oriented keys, the default BLANK is satisfactory. However, if the file keys contain characters below X'40', then padding using NULL is recommended.

KEYEXTR={ YES | NO }

Specifies whether to extract the file key from the record as part of the data. When the default KEYEXTR=YES is specified or defaulted, the key is removed from ('squashed out of') the record by a GET operation, or inserted by a PUT operation. Hence processing proceeds as though the record and the key are separate entities.

When KEYEXTR=NO is specified, the key remains as part of the data on a GET operation but is still separately accessible through the &FILEKEY system variable. On a PUT operation, the key portion of the data record is assumed to be present but is ignored, and the current contents of the &FILEKEY variable are used to overlay the key within the data.

When processing using the &FILE verb, the &CONTROL KEYXTR setting is ignored. Key extraction is determined on the &FILE OPEN by the KEYEXTR operand.

DATA=*exitdata*

Identifies any additional data to pass to a user exit on the &FILE OPEN. If used, this must be the last operand, and all data following the DATA= keyword is passed to the NCL &FILE validation exit (NCLEX01) without inspection.

Return Codes:

On completion of the operation, the &FILERC system variable is set as follows (certain return codes only apply if set by the validation exit NCLEX01, where this is in effect):

0

Restrict access to read only

4

Read with update ability, without delete

8

Read and update with delete ability

12

No access allowed

16

Specified *fileid* is not available (see &SYSMSG for details)

Return codes 0, 4, and 12 apply only if the validation exit NCLEX01 is in effect. Return codes 8 and 16 is set regardless of the exit being in effect.

Note: If no exit is in effect and the specified file ID is available for processing, no restriction to access applies and a value of 8 is set in &FILERC.

Failure of the procedure to limit processing within the bounds set by the processing exit, as indicated by the return code in &FILERC, results in termination of the procedure at the point at which an unauthorized function is attempted.

The use of a logical file identity as assigned by the &FILE OPEN statement makes it possible to use a single command (the UDBCTL command) to control the migration of all NCL procedures from one physical data set to another. The UDBCTL command associates a physical file with a logical name. These logical names are then valid for &FILE statements and provide a connection between the physical data set and the processing NCL procedure. This approach allows an installation to move all processing of NCL procedures onto another data set to free the previous data set for off-line processing. The approach also isolates procedures from any knowledge of real data set names and therefore makes JCL changes transparent to NCL procedures.

Each new file ID specified on an &FILE OPEN statement allocates sufficient storage to process subsequent requests associated with that file. The &FILE CLOSE statement is used to release file processing connections and any related storage. Termination of the NCL procedure also frees any associated storage.

The return code is tested to determine the cause of the error.

Important! Exercise great care while processing system data sets such as VFS. This type of processing must be performed using the UNMAPPED mode and requires an excellent understanding of the formats of the records in these data sets. Invalid processing of these data sets can cause unpredictable results and result in abnormal termination of the system.

Notes:

- The open options provide default processing options for the file.
- The old &FILEID verb was used both to open files, and to switch between files by setting the current *fileid* for implied action by other verbs, such as &FILEGET. The mandatory ID= operand on the &FILE verb enables several files to be processed simultaneously without confusion.

More information:

&FILE (see page 321)

&FILE PUT

The &FILE PUT verb puts a file record.

This verb has the following format:

```
&FILE PUT ID=fileid
        [ KEY=key | KEYVAR=keyvar ]
        [ OPT={ TRUNCATE | NOTRUNCATE } ]
        [ { ARGS | VARS=prefix* }
        [ RANGE=(start,end) ] | VARS={ var | ( var1,var2,...,varn ) } |
          DATA=data | MDO=mdoname ]
        [ PRTCNTL=opt | (opt1,opt2 [,opt3,opt4 ] )  ]
```

Operands:

ID=*fileid*

(Mandatory) Identifies the file being actioned.

KEY=*key*

Nominates the full key value to be set for the file. The key value (after substitution) is in one of several forms:

- An unquoted character string (no imbedded blanks). For example:

  ```
  01SMITH
  ```

  For unquoted strings the key should consist of displayable characters only, and is delimited by the first blank.

- A quoted character string, for example:

  ```
  '01SMITH J.'   “03JONES PETER A”    “LU A0123BC2”C 'A. B. O''MALLEY'C
  ```

  The usual quote rules apply. Either single quotes (') or double quotes (”) are acceptable. Quotes must be paired and of the same type. A double quote is treated as a single quotation character when included within a string.

- A quoted hexadecimal character string. for example:

  ```
  '0012001F'X
  ```

  ```
  “0123456789ABCDEF”X
  ```

If there are no characters following the closing quote, a character string is assumed. If a single character follows the closing quote then it must be either 'C' (designating a character string), or 'X' (designating a hexadecimal string). For quoted hexadecimal strings, all characters must be valid hexadecimal characters (that is, 0 to 9, A to F).

It is the user's responsibility to understand the file key structure and determine which form of key designation is most appropriate.

Note: The &ZQUOTE built-in function can assist when building quoted strings, and that the KEYVAR operand provides a suitable alternative.

KEYVAR=*keyvar*

Provides an alternative way to nominate the full key value. *keyvar* is the name of a user or system variable, the contents of which are taken unchanged as the key value for the file set operation. There is no need to code the ampersand (&)—if the & is coded, then the contents of the variable specified are assumed to contain the name of the variable containing the key.

Note: If both the KEY and KEYVAR operands are omitted, the current file key value is used.

OPT={ TRUNCATE | NOTRUNCATE }

Indicates whether or not to allow truncation of data on an &FILE PUT. If NOTRUNCATE is specified (or defaulted), the procedure will end with an error message, if the data exceeds the maximum record length of the file. When TRUNCATE is specified, a file return code of 1 is set if truncation occurs.

{ ARGS | VARS=*prefix } [RANGE (*start*,*end*)]**

Nominates the NCL variables for the PUT operation. NCL variables are validly used as the source of the PUT operation when the file format is DELIMITED or UNMAPPED, and when the format is MAPPED and MAP=$NCL. How the variable appears on the file depends upon the current processing mode (format). If the format is UNMAPPED, then the variables are simply concatenated to form the record. If the format is DELIMITED, the variables are placed on the file separated by X'FF' separators.

VARS={ *var* | (*var*1,*var*2,...,*varn*) }

Specifies the list of variables to be written to the file. NCL variables are validly used as the source of the PUT operation when the file format is DELIMITED or UNMAPPED, and when the format is MAPPED and MAP=$NCL.

DATA=*data*

Data is allowed for DELIMITED or UNMAPPED format files only, and specifies a string of data to be placed within the file record. Substitution occurs on the specified string of data, and then the data is placed on the record as is.

MDO=*mdoname*

Nominates the MDO for the PUT operation. An MDO is the source of the data record on a PUT operation only when the file format is MAPPED.

PRTCNTL=*opt* | (*opt*1,*opt*2 [,*opt*3,*opt*4])

If FILE PUT writes to OS/VS SYSOUT data sets, use PRTCNTL to specify carriage control options to control print formatting. Print format categories are as follows:

- Paper movement-SKIP0, SKIP1, SKIP2, SKIP3, NEWPAGE, and DATA
- Underscoring-USCORE1 and USCORE2
- Text alignment-LEFT, RIGHT, and CENTER
- Bold print-BOLD

One option from each of these categories is specified.

Multiple options is specified, but they must be enclosed in parentheses and separated by commas. For example:

```
PRTCNTL=(NEWPAGE,USCORE1,CENTER,BOLD)
```

SKIP0

Does not advance the paper before writing the record.

SKIP1

Advances the paper one line before writing the record. When writing records to OS/VS SYSOUT data sets this is the default value if the PRTCNTL operand is omitted.

SKIP2

Advances the paper two lines before writing the record.

SKIP3

Advances the paper three lines before writing the record.

NEWPAGE

Skips to a new page before writing the record.

DATA

Indicates that the first text character contains the carriage control character to be used for controlling the print options for printing this record. If machine control characters are used (instead of ANSI control characters), DATA is the only carriage control option allowed.

USCORE1

Underlines the text for this record (excluding blanks between words).

USCORE2

Underlines the text for this record (including blanks between words).

LEFT

Aligns the text for this record to the left of the paper.

RIGHT

Aligns the text for this record to the right of the paper.

CENTER

Centers the text for this record. The record width, and therefore the location of the center, is determined from the logical record length of the file, which is specified in the LRECL operand of the ALLOC command (for z/OS or MSP).

BOLD

Prints the text in bold form. This is achieved by overstriking the same characters.

Return Codes:

On completion of the operation, the &FILERC system variable is set as follows:

0

Record added successfully

1

Record added, truncation has occurred

8

Error during processing of PUT request

16

NCL or Mapping Services processing error

Check &SYSMSG for details. The &VSAMFDBK system variable contains the VSAM completion code. For &FILERC=8, the &VSAMFDBK system variable is tested to determine the cause of the error. These error codes are explained in detail in your VSAM documentation. The &VSAMFDBK variable always returned as a 2-character value.

The &SYSMSG variable can also be set to contain error message details.

For mapped format files, the &ZMDORC and &ZMDOFDBK system variables are also set.

More information:

&FILE (see page 321)

&FILE SET

The &FILE SET verb specifies or switches file records.

This verb has the following format:

```
&FILE SET ID=fileid
        [ KEY=key | KEYVAR=keyvar ]
        [ GENLEN=nn ]
        [ FORMAT={ MAPPED | UNMAPPED | DELIMITED } ]
        [ MAP={ $NCL | mapname } ]
```

Operands:

ID=*fileid*

Identifies the file.

KEY=*key*

Specifies the full key value to set for the file. The key value (after substitution) is in one of several forms:

- An unquoted character string (no embedded blanks), for example:

  ```
  01SMITH
  ```

  For unquoted strings, the key consists of displayable characters only, and the first blank delimits it.

- A quoted character string, for example:

  ```
  '01SMITH J.'   “03JONES PETER A”    “LU A0123BC2”C 'A. B. O''MALLEY'C
  ```

  The usual quote rules apply. Either single quotes (') or double quotes (”) are acceptable. Quotes must be paired and of the same type. A double quote is treated as a single quotation character when included within a string.

- A quoted hexadecimal character string, for example:

  ```
  '0012001F'X
  ```

  ```
  “0123456789ABCDEF”X
  ```

If there are no characters following the closing quote, a character string is assumed. If a single character follows the closing quote then it must be either 'C' (designating a character string), or 'X' (designating a hexadecimal string). For quoted hexadecimal strings, all characters must be valid hexadecimal characters (that is, 0 through 9 and A through F).

The user is responsible to understand the file key structure and determine which form of key designation is most appropriate.

Note: The &ZQUOTE built-in function can assist when building quoted strings, and that the KEYVAR operand provides a suitable alternative.

KEYVAR=*keyvar*

Provides an alternative way to nominate the full key value. *keyvar* is the name of a user or system variable, the contents of which are taken unchanged as the key value for the file set operation. The ampersand (&) is not required—if the & is coded, then the contents of the variable specified are assumed to contain the name of the variable containing the key.

If both the KEY and KEYVAR operands are omitted, the current file key value is used.

GENLEN=*nn*

Allows the generic key length for operations such as generic read, and generic delete, to be set independently of the actual key value. For example, by setting the key to 01SMITH, but GENLEN=2, &FILE GET OPT=KEQ begins reading from the key but terminates if any key encountered does not begin with 01. If the length of the key provided is shorter than the value of GENLEN, then the length of the key is the effective GENLEN.

FORMAT={ MAPPED | UNMAPPED | DELIMITED }

Allows the switching of the file processing format. The options are as described for the &FILE OPEN statement. No default applies to the &FILE SET statement. If the FORMAT operand is omitted, there is no change to the processing format.

MAP={ $NCL | *mapname* }

Valid only if FORMAT=MAPPED is specified explicitly on the &FILE SET or if the current processing is for format MAPPED. The meaning is as described for the &FILE OPEN statement.

Return Codes:

On completion of the operation, the &FILERC system variable is set as follows:

0

Record added successfully

16

NCL or Mapping Services processing error (see &SYSMSG for details)

More information:

&FILE (see page 321)

&FLUSH

Terminates all nesting levels within an NCL process.

```
&FLUSH
```

Flushes all nesting levels and terminates the process.

Examples:

```
&WRITE ALARM=YES DATA=ENTER STOP TO TERMINATE ALL +
   PROCESSING
&PAUSE ARGS
&IF .&1 EQ .STOP &THEN +  -* TERMINATE PROCESSING
   &FLUSH
  .
  .
  .
```

Notes:

- Use the &END (see page 314) statement to terminate only the current nesting level and resume processing at a higher level.
- Use the &QEXIT (see page 571) statement to terminate all nesting levels and the window from which the procedures were invoked (perhaps logging a user off if only one window is active).

&FNDSTR

Returns a numeric value indicating the presence of a search string within a supplied text.

```
&FNDSTR string text
```

&FNDSTR is used to search one or more variables to determine if a specified string exists within any of those variables.

Use &FNDSTR to search text to determine if a specified string exists within it. If the specified string is found within the first word of the text, the offset into that word is returned. This value can subsequently be used in an &SUBSTR statement if required.

If the specified string is found in a word other than the first, the value 999 is returned. If the specified string is not found, the value 0 is returned.

Operands:

string

The search string-if this string contains blanks, it must be specified in the &FNDSTR statement as a single variable. Specifying string with multiple variables is not valid. See example below.

text

The text string that is to be searched. If the search argument is found within the first word of the text, the offset to the string in the word is returned. If it is found in other than the first variable, a value of 999 is used.

Examples: &FNDSTR

```
&CHECK = &FNDSTR FAILED &MSGTEXT
&IF &CHECK NE 0 &THEN +
   &WRITE ALARM=YES DATA=REQUEST FAILED
&SRCH = &FNDSTR N10503 &MSG1 &MSG2 &MSG3
```

Notes:

- If multiple variables are searched and the search argument is found in other than the first variable, a value of 999 is returned. The procedure must then use multiple searches if the precise offset within a variable for other than the first position, is required.
- If the search argument consists of a variable made up of multiple words with imbedded blanks, it must exist entirely within a variable in the search text. An occurrence of the string spanning multiple words does not qualify for a found condition.
- If &CONTROL DBCS or DBCSN or DBCSP is in effect, &FNDSTR is sensitive to the presence of DBCS data.

More information:

&FNDSTR (see page 1198)
&SELSTR (see page 601)
&REMSTR (see page 572)

&GOSUB

Branches to a sub-routine within the procedure.

```
&GOSUB .label [ .limlabel [ EQ | NE | GT | LT | GE | LE ] ]
```

The &GOSUB verb lets you structure a procedure to take advantage of common processing routines, called sub-routines. &GOSUB changes the current processing location within a procedure. Control transfers to the statement beginning with the specified target label. On completion of the sub-routine an &RETSUB statement resumes processing at the statement following the &GOSUB statement.

The label, after variable substitution, must begin with a period and is from 1 to 12 characters long (not including the period).

The target label can precede or follow the statement containing the &GOSUB.

Operands:

.label

A label within the procedure.

.limlabel

A label in the procedure that delimits the extent of the search for .label. If *.limlabel* is found before the required label .label is located, the search terminates with a 'label not found' condition. Use *.limlabel* to improve the performance of procedures where a large number of unexpected labels normally discarded by the &CONTROL NOLABEL are received. When used with the search scope (see next operand), *.limlabel* can assist with table processing.

EQ | NE | GT | LT | GE | LE

An optional search scope used to qualify the selection of a target label that satisfies the search for .label up to the range set by *.limlabel*. If this operand is required, then *.limlabel* must be specified. In cases where *.limlabel* is not required, enter it with the same value as *.label*.

If this operand is omitted, *.limlabel* must exactly equal a procedure label to terminate the search.

EQ

Specifies that the first label generically equal to .label will satisfy the search.

NE

Specifies that the first label generically not equal to .label will satisfy the search.

GT

Specifies that the first label greater than .label will satisfy the search.

LT

Specifies that the first label less than .label will satisfy the search.

GE

Specifies that the first label greater than or equal to .label will satisfy the search.

LE

Specifies that the first label less than or equal to .label will satisfy the search.

Examples: &GOSUB

```
.MAINLINE
   &GOSUB .INITIALIZE
   &DOUNTIL &RETCODE NE 0
   &GOSUB .PROCESS
   &DOEND
   .

.INITIALIZE
   -*
   -* Initialization logic
   -*
   &RETSUB

.PROCESS
   -*
   -* Processing logic
   -*
   &IF &REQUEST = EXIT &THEN +
   &RETSUB 4
   &RETSUB 0
   &PANEL CMDENTRY
   &GOSUB .P$&COMMAND .PEND
   &RETSUB

.P$DISPLAY
   -*
   -* Display logic
   -*
   &RETSUB

.P$LIST
   -*
   -* List logic
   -*
   &RETSUB
.PEND
```

Notes:

- Using the &CONTROL NODUPCHK option can improve the processing of &GOSUB statements. This option eliminates duplicate label checking and lets you use a faster search algorithm. It should only be used in procedures that have been thoroughly tested, as its use in a procedure containing duplicate labels can lead to unpredictable results.
- The target label on an &GOSUB statement is a variable. Substitution is performed before attempting to transfer control to the target label.
- Missing labels result in an error unless &CONTROL NOLABEL is used. This returns control to the statement after the &GOSUB statement, if the label does not exist.
- Using label variables with &CONTROL NOLABEL is an easy and efficient way to isolate a target processing routine, instead of sifting data using multiple &IF statements.
- The system maintains a loop control counter to stop inadvertent runaway loops within a procedure. This counter (&LOOPCTL) is set to 1000 for each nesting level entered. It is decremented by 1 for each &GOTO or &GOSUB executed. If the counter reaches 0 the procedure terminates with an error message. (You can reset the loop control counter using the &LOOPCTL statement if a larger value is required.)

&GOTO

Branches to another statement within the procedure.

```
&GOTO .label [ .limlabel [ EQ | NE | GT| LT| GE | LE ] ]
```

The &GOTO verb lets you change the current processing location within a procedure-control transfers to the statement beginning with the specified target label.

After variable substitution, the label must begin with a period and is from 1 to 12 characters long (not including the period).

The target label can precede or follow the statement containing the &GOTO.

Note: Where possible, using &DO, &DOWHILE, and &DOUNTIL to construct do-loops is preferable to using &IF ... &GOTO logic.

Operands:

.label

A label within the procedure.

.limlabel

A label in the procedure that delimits the extent of the search for .label. If *.limlabel* is found before the required label .label is located, the search terminates with a 'label not found' condition. Use *.limlabel* to improve the performance of procedures where a large number of unexpected labels normally discarded by the &CONTROL NOLABEL facility are received. When used with the search scope (see next operand), *.limlabel* can assist with table processing.

EQ | NE | GT | LT | GE | LE

An optional search scope used to qualify the selection of a target label that satisfies the search for .label up to the range set by *.limlabel*. If this operand is required, then *.limlabel* must be specified. In cases where *.limlabel* is not required enter it with the same value as .label. In both cases the operator must be separated from the value being tested with a space .

If this operand is omitted, *.limlabel* must exactly equal a procedure label to terminate the search.

EQ

Specifies that the first label generically equal to .label will satisfy the search.

NE

Specifies that the first label generically not equal to .label will satisfy the search.

GT

Specifies that the first label greater than .label will satisfy the search.

LT

Specifies that the first label less than .label will satisfy the search.

GE

Specifies that the first label greater than or equal to .label will satisfy the search.

LE

Specifies that the first label less than or equal to .label will satisfy the search.

Examples: &GOTO

Notes: More examples are supplied in the system distribution library.

```
& CONTROL NOLABEL
&GOTO .&1  -* YES and NO are only valid options
           -* Control returns here if label .&1 is
           -* undefined.
&ENDAFTER &WRITE ALARM=YES DATA='YES' OR 'NO' MUST BE +
                                  ENTERED
.YES
   -*
   -* NCL statements
   -*
.NO
   -*
   -* NCL statements
   -*
.LOOP
   &INTREAD ARGS
   &GOTO .&1 .ENDTABLE   -* .ENDTABLE is LIMLABEL
   &GOTO .LOOP           -* Skip the message if label
                         -* not found
.IST970I
.IST355I
   &GOTO .LOOP           -* Process the next message
.IST346I
   -*
   -* Processing logic for IST346I
   -*
&GOTO .LOOP              -* Process the next message
.ENDTABLE                -* Defines end of table
```

Notes:

- The use of &GOTO is minimized by using &DO, &DOUNTIL and &DOWHILE statements.
- If &GOTO is used, labels must be unique within the current procedure nesting level (unless &CONTROL NODUPCHK is active). Any attempt to branch to a duplicated label results in an error.
- Using the &CONTROL NODUPCHK option can improve &GOTO statement processing. This option eliminates duplicate label checking and lets you use a faster search algorithm. It should only be used in procedures that have been thoroughly tested, as its use in a procedure containing duplicate labels can lead to unpredictable results.
- The target label on an &GOTO statement is a variable. Substitution is performed before attempting to transfer control to the target label.
- Missing labels will result in an error unless &CONTROL NOLABEL is used. This returns control to the statement after the &GOTO statement, if the label does not exist.
- Using label variables with &CONTROL NOLABEL is an easy and efficient way to isolate a target processing routine, instead of sifting data using multiple &IF statements.
- Label variables substituted into null variables are regarded as 'not found', and processing resumes at the next statement without causing an error. This also applies to invalid labels.
- The system maintains a loop control counter to stop inadvertent runaway loops within a procedure. This counter (&LOOPCTL) is set to 1000 for each nesting level entered. It is decremented by 1 for each &GOTO or &GOSUB executed. If the counter reaches 0 the procedure terminates with an error message. (You can reset the loop control counter using the &LOOPCTL statement if a larger value is required.)

More information:

&LOOPCTL (see page 405)
&CONTROL (see page 266)
&DO (see page 301)
&DOWHILE (see page 306)
&DOUNTIL (see page 304)

&HEX

Returns the hexadecimal equivalent of a decimal number.

```
&HEX decimalnumber
```

&HEX provides a means of converting a decimal number to its hexadecimal equivalent. &HEX is a built-in function and must be used to the right of an assignment statement.

Operands:

decimalnumber

Examples: &HEX

```
&NUM = &HEX 66635    -* &NUM is set to FFFF
&A = &HEX &1
&NUM = &HEX 00123    -* &NUM is set to 7B
&NUM = &HEX -2088976 -* &NUM is set to FFE01FF0
```

Notes:

- The maximum decimal value that is processed is 2147483647 (2,147,483,647), which returns the value 7FFFFFFF. The minimum decimal value that is processed is -2147483648 (-2,147,483,648), which returns the value 80000000.
- Up to 15 digits (maximum), is accepted. That is, 000002147483647 is valid.
- An invalid decimal value, or one too large, results in a procedure error.

More information:

&DEC (see page 294)

&HEXEXP

Returns a hexadecimal string that is the expansion of the supplied text according to EBCDIC encoding.

```
&HEXEXP text
```

&HEXEXP provides a means of converting a character string to its hexadecimal equivalent.

&HEXEXP is a built-in function and must be used to the right of an assignment statement.

Operands:

text

The character string for conversion, which is regarded as starting one blank after the &HEXEXP keyword.

Examples:

```
&1 = &HEXEXP ABCD  -* &1 will be set to C1C2C3C4

&Z = &HEXEXP 3C    -* &Z will be set to F3C3
```

Note:

&HEXPACK (see page 363) provides the reverse facility. The maximum length of a character string that is expanded is 128 bytes.

&HEXPACK

Returns a character string that is the EBCDIC encoding of the supplied hexadecimal values.

```
&HEXPACK hexstring1 [ hexstring2 .... hexstringn ]
```

&HEXPACK is a built-in function and must be used to the right of an assignment statement.

Operands:

***hexstring1* [*hexstring2 .... hexstringn*]**

A series of hexadecimal strings for conversion. Each string must contain only valid hexadecimal characters (0 to 9, A to F). If there is an odd number of characters within a string, that string is padded on the left with a zero before conversion.

The converted strings are concatenated into a single value. A string can contain hexadecimal representations of non-displayable characters for conversion to their hexadecimal equivalent. This lets you assign true hexadecimal values into variables.

Examples: &HEXPACK

```
&1 = &HEXPACK C1C2C3C4  -* &1 is set to ABCD

&Z = &HEXPACK 03FF 0D25  -* &Z is set to X'03FF0D25'
```

Notes:

The total sum of the concatenated variables or constants cannot exceed the maximum size for one variable.

- &HEXPACK is used in the following ways:
- To allow true hexadecimal values to be passed to &CALL programs
- On &SECCALL EXIT variables
- For sending to LU1 devices using &WRITE
- Invalid hexadecimal characters will result in a procedure error.

More information:

&WRITE (see page 695)
&HEXEXP (see page 362)

&IF

The &IF verb conditionally executes a command, statement, or DO group.

This verb has the following format:

```
&IF logical expression
[ { AND | OR } logical expression ] ...
    &THEN command or statement
```

&IF defines a logical expression and tests the truth of that expression, or defines multiple logical expressions connected by AND or OR operators and tests the truth of those expressions. If the result is true, the command or statement following the &THEN is executed.

If the result is false the next statement in the procedure sequence is executed. See the &ELSE statement description for an account of the support for the conditional execution of a command or statement associated with the false condition arising from &IF.

&IF supplies logical operators that are used to test the values of individual bit settings in hexadecimal variables.

See the &DO, &DOWHILE, and &DOUNTIL statement descriptions for an explanation of how to group commands and statements for execution in association with both the true condition, and the &ELSE false condition.

Operands:

logical expression

An expression in the form:

*value*1 *relational-operator value*2

value1 and *value2* are either variables or constants, and *relational-operator* is one of the following logical operators:

- EQ or = (equal)
- NE or ¬= (not equal)
- LT or < (less than)
- LE or <= (less than or equal)
- GT or > (greater than)
- GE or >= (greater than or equal)
- NG or ¬> (not greater than)
- NL or ¬< (not less than)
- BO (bits on)
- BZ (bits off)

- BM (bits mixed on and off)

Blanks must separate the logical operator from the values being tested.

Before the comparison, both values are translated to uppercase (by default) and all leading and trailing blanks are stripped. If either value is a series of blanks, it is treated as a single blank value. These changes apply only to the comparison operation and do not change either value for subsequent processing. (You can use the &CONTROL IFCASE option to suppress uppercase translation before the comparison.)

If operating in a system in which SYSPARMS DBCS=YES is active, no translation into uppercase is made.

When the logical operator indicates a bit test function, value1 and value2 are in hexadecimal format. Both values are regarded as the character representation of a hexadecimal byte; for example, 01 is treated as representing the hexadecimal byte X'01'. When performing bit tests, *value1* must be at least two bytes long. The first two characters are combined to represent the target hexadecimal byte, which is tested against the two-byte value of *value2*.

value2 must be two bytes only, in the range 00 to FF. The bits set in *value2* are tested against the byte represented by *value1*, according to the logical operator specified.

The operator BO signifies that all bits tested must be set for the expression to be true. BZ signifies that all bits tested must be set off for the expression to be true. BM specifies that if the tested bits are a mixture of on and off, the expression is true.

AND

Connects a series of logical expressions into a logical expression group. Each individual expression in the group must be true for the group as a whole to be considered true. AND takes precedence over OR.

OR

Connects a series of logical expression groups. The overall true/false analysis of the statement is determined by the true/false status of the individual groups, reading from left to right of the &IF statement. If any one group is found to be true, then the statement result is true. If all groups are found to be false then the statement result is false.

&THEN

Separates the right most logical expression or logical expression group from the command or statement that is to be executed if the statement is true. &THEN must have at least one blank on either side. &THEN is mandatory.

command or statement

Specifies the command or statement for execution if the statement is found to be true. The command or statement must be coded on the same statement as &IF.

Examples: & IF

```
&IF &DAY EQ WED &THEN +
   -START PROC2
&IF .&1 = .  &THEN +
   &WRITE DATA=ERROR, OPERAND OMITTED
&IF &HEXFLAG BO 02 &THEN +
   &GOTO .BITON
&IF &HEXFLAG BZ 03 &THEN +
   &GOTO .BITSOFF
&IF .&1 EQ .LU AND .&2 NE . &THEN +
   &GOTO .ACTLU
&ELSE +
   &GOTO .ERRORMSG
```

Notes:

Variable substitution is performed before processing the &IF statement. If a variable has a null or undefined value when substitution is performed, it is eliminated from the statement. For this reason, take care when testing variables if it is possible for no value to be assigned. For example, if an operator is required to enter a variable when a procedure is invoked, the procedure must then test that a variable has been entered or omitted. For the following statement:

```
&IF &1 EQ YES &THEN ....and so on
```

&1 is expected to be set to a value entered by the operator. A syntax error results if a value is not entered. After syntax substitution and before executing the &IF, the statement would appear as:

```
&IF EQ YES &THEN ....and so on
```

The variable &1 has been eliminated as it has a null value. This is easily overcome by prefixing the same constant to either side of the expression:

```
&IF .&1 EQ .YES &THEN ....and so on
```

In this case, if &1 is not set to a value, the resulting expression before executing the &IF would appear as:

```
&IF . EQ .YES &THEN ....and so on
```

No syntax error occurs, as the statement is still syntactically correct. If the &IF test is false, the &ELSE statement is used to indicate an alternative processing course. A subsequent &ELSE statement is optional. If the &IF test is true, the &ELSE action is ignored and processing resumes at the statement after completing the &ELSE process.

Note: &ELSE cannot be coded on the same statement as the associated &IF. &THEN must be coded on the same statement as the associated &IF.

More information:

&ELSE (see page 308)
&DO (see page 301)
&DOWHILE (see page 306)
&DOUNTIL (see page 304)
&GOTO (see page 357)

&INTCLEAR

Clears messages queued to a dependent processing environment.

```
&INTCLEAR [ TYPE=ALL | REQ | RESP | ANY ]
```

&INTCLEAR is used to discard outstanding messages queued to a dependent processing environment by previous &INTCMD processing, or queued by INTQ commands issued by other NCL processes which have directed messages to this process.

When an &INTCMD statement is executed, the associated command is executed in the dependent processing environment for the process that executes the &INTCMD statement. The results of the command are queued and returned line-by-line to the procedure when each subsequent &INTREAD statement is processed. During the analysis of the results of the command, the logic of the procedure can determine that another &INTCMD is to be issued and that remaining results held from the previous &INTCMD are no longer required. &INTCLEAR is used to discard these outstanding results so that another &INTCMD statement is issued.

If the &INTCLEAR statement were not used, the results of the second &INTCMD would be queued in order behind the original outstanding results and would be presented in that order.

&INTCLEAR eliminates the need to process all of the results from a command.

Operands:

TYPE=ALL

(Default) This option clears the dependent processing environment of the issuing process and any other dependent environments below it. This means that if the process had at some stage issued &INTCMD START to invoke an asynchronous, but dependent, NCL process, that process is terminated by &INTCLEAR TYPE=ALL.

Unsolicited message receipt and other PROFILE command options are reset.

TYPE=REQ

This option clears all currently queued REQUEST messages such as those generated by INTQ TYPE=REQ commands from other processes that have sent messages to the issuing process.

TYPE=RESP

This option clears all RESPONSE messages such as those generated by INTQ TYPE=RESP commands, commands issued by &INTCMD statements, or from other NCL command processes executing in the issuing process's dependent processing environment.

TYPE=ANY

This option clears all currently queued RESPONSE and REQUEST messages. This format is equivalent to both TYPE=REQ and TYPE=RESP.

Examples: &INTCLEAR

```
&INTCLEAR TYPE=REQ
```

Discards all messages sent to us from other processes issuing INTQ or &WRITE TYPE=REQ commands.

```
&INTCLEAR
```

Discards all messages queued to our dependent processing environment and terminates all dependent processes and profiles.

```
&INTCLEAR TYPE=RESP
```

Discards all messages generated by dependent processes and queued to the internal response queue for processing by &INTREAD. Does not terminate any dependent processes.

Notes:

If an &INTCLEAR statement is processed after &INTCMD executes a command which has not yet completed, the final results for that command are logged and then discarded. To limit the volume of messages flowing to the log, use the &INTCLEAR TYPE=RESP statement to discard unwanted messages from the dependent response queue.

&INTCMD

Schedules a command for execution in the issuing process's dependent processing environment.

`&INTCMD` *`command text`*

Executes a command in an NCL process's dependent processing environment and returns command results to the procedure for analysis.

Every NCL process has a dependent processing environment associated with it where it can execute commands or other NCL processes. &INTCMD is used to execute commands and processes in this dependent environment.

When an &INTCMD statement is processed, the command statement is executed in the issuing process's dependent processing environment. The command is processed in the normal manner, but all responses resulting from the command are returned to the originating process's dependent response queue, and not the user's terminal. The issuing process can then use the &INTREAD statement to retrieve queued responses one by one.

The result is a procedure that can correlate the results of commands it issues to react intelligently to the results of the command.

The process &INTCMD invokes, executes independently of the initiating procedure. If the initiating process ends before the dependent process has completed, the dependent process is flushed automatically. It is your responsibility to ensure that the initiating procedure synchronizes with the &INTCMD process by using &INTREAD statements issued in the initiating process.

Operands:

command text

Any upper and lower case command text.

Examples: &INTCMD

```
&INTCMD D TERM1
&INTCMD D NCPSTOR,ID=NCP1,LEN=4,ADDR=&1
&INTCMD MAISEND IMS /DIS A
&INTCMD START MONITOR
&INTCMD PROFILE MONMSG=Y
```

Notes:

Certain product commands do not apply when executed within an &INTCMD environment, and return the message:

```
N10104 COMMAND CANNOT BE SOURCED FROM &INTCMD STATEMENT.
```

These commands include the CLEAR, K, PAGE, END, FLUSH, GO, SPLIT, SWAP, and FSPROC.

If another NCL process is invoked by an &INTCMD statement (for example, &INTCMD START PROCNAME), the started process executes in the originating process's dependent processing environment as a dependent process.

MAI-OC command results issued from an &INTCMD statement are returned to your dependent response queue and is accessed by &INTREAD statements. However, if the dependent processing environment is removed (by using &INTCLEAR, or by terminating the procedure) from an OCS primary processing environment, the MAI-OC session is retained and further output is delivered to the OCS display. NCL processes invoked with &INTCMD (that is, executing in the originating process's dependent processing environment) have their own dependent processing environments. These procedures can therefore use &INTCMD without restriction to execute their own commands and dependent processes. If the highest level process terminates, all of its dependent levels terminate.

The NCL process can use &INTCMD PROFILE commands to control the behavior of the dependent environment, much the same way as an OCS operator. For example, &INTCMD PROFILE MONMSG=Y is used to receive monitor class messages on the dependent environments response queue.

Using internal command environments is not limited to standard procedures. &INTCMD is used within reserved procedures such as PPOPROC and MSGPROC, or from within the NMINIT and NMREADY procedures executed during system initialization.

Commands executed using &INTCMD are not echoed to your terminal; nor are the results of the commands - these are queued for processing by &INTREAD. However, all command results processed with &INTREAD are logged (by default) to the activity log to provide an audit trail of events. The messages appear in the log with **nclid** in the log's node name column. (The &CONTROL NOINTLOG statement is used to suppress logging of these messages to the activity log.)

The &INTCLEAR statement is used to flush any messages queued for &INTREAD processing. For example, when VTAM command responses arrive much later after the recipient's details have been removed by an &INTCLEAR statement. The message is directed to the activity log regardless of the setting of the &CONTROL statement. This also occurs if a procedure terminates before the messages or responses which it initiates, return.

Note: For more information about NCL execution, processing regions, and environments see the *Network Control Language Programming Guide*.

More information:

&INTREAD (see page 375)
&LOGCONT (see page 396)
&INTREPL (see page 382)
&INTCLEAR (see page 368)

&INTCONT

Propagates a message read using &INTREAD to the next higher processing environment.

```
&INTCONT [ COLOR=color | COLOUR=colour]
         [ HLIGHT=highlight ]
         [ INTENS={ HIGH | NORMAL } ]
         [ ALARM= { YES | NO } ]
         [ NRD={ NO | OPER } ]
```

After an &INTREAD statement, a procedure can propagate the received message to the terminal (if the executing process is not a dependent of a higher level process), or to the dependent response queue of the next higher level process (if the executing process is running in a dependent processing environment).

When an &INTCMD statement is processed, the command associated with it is executed in the issuing process's dependent processing environment. The command is processed in the normal manner but all command responses are returned to the originating process's dependent response queue and not the user's terminal. The issuing process can then use the &INTREAD statement to retrieve queued responses one by one.

&INTCONT allows the messages received by a process executing as a dependent to be passed up the hierarchy, without having to be the subject of explicit &WRITE statements. Optionally, some attributes of the message is altered.

Operands:

COLOR=*color* | COLOUR=*colour*

If the color attribute of the message received from &INTREAD is to be changed, specify the new color by using this operand. The existing color attribute of the message is tested by examination of the &ZMCOLOUR message profile variable set after execution of &INTREAD.

HLIGHT=*highlight*

If the display highlighting attribute of the message received from &INTREAD must be changed, use this operand to specify the new attribute. The existing highlighting attribute for the message is tested by examining the &ZMHLIGHT message profile variable set after executing &INTREAD.

INTENS={ HIGH | NORMAL }

If the INTENS (intensity) attribute of the message received from &INTREAD must be changed, use this operand to specify the new attribute. The existing INTENS attribute for the message is tested by examining the &ZMINTENS message profile variable set after executing &INTREAD.

ALARM={ YES | NO }

If the ALARM attribute of the message received from &INTREAD must be changed, use this operand to specify the new attribute. The existing ALARM attribute for the message is tested by examining the &ZMALARM message profile variable set after executing &INTREAD.

NRD={ NO | OPER }

If the non-roll delete attribute for the message received from &INTREAD must be changed, use this operand to specify the new attribute. The existing NRD attribute of the message is tested by examining the &ZMNRD message profile variable set after executing &INTREAD. NRD=YES is specified but is ignored; full non-roll delete with DOM correlation can only be set by the message originator, for example, &WRITE.

Examples: &INTCONT

```
&IF &ZMALARM = YES AND &INTYPE = RESP &THEN +
   &INTCONT ALARM=NO COLOR=RED
&IF &ZMREQSRC = USER &THEN +
   &INTCONT HLIGHT=REVERSE
```

Notes:

&INTCONT only propagates messages that are received after executing &INTREAD TYPE=RESP, or after &INTREAD TYPE=ANY is satisfied by a message from the procedure's dependent response queue. Messages satisfying &INTREAD TYPE=REQ cannot be propagated by &INTCONT. The procedure will terminate in error if you attempt to use &INTCONT to propagate a request message.

&INTCONT operates on the previous $INT MDO retrieved by an &INTREAD. Values in the $INT MDO is modified before the &INTCONT is issued, and will be reflected in the message that is propagated. Care should be taken when modifying attributes that affect message flow control. Modification of the $INT.SOURCE.TIME, $INT.DOMID or $INT.PREFIX can cause unpredictable results.

The message profile variable &ZINTYPE is set to a value of RESP if &INTCONT is used to propagate the message.

Note: For more information, see the *Network Control Language Programming Guide.*

More information:

&INTREAD (see page 375)
&INTCLEAR (see page 368)

&INTREAD

The &INTREAD verb retrieves the next message queued from the dependent processing environment of the issuing process.

This verb has the following format:

```
&INTREAD { VARS=prefix* [ RANGE=( start, end ) ] |
           VARS={ name | ( name, name, ..., name ) } |
           ARGS [ RANGE=(start, end) ] |
           STRING=( name, name, ..., name ) |
           SET  MDO=mdoname | SET }
         [ WAIT={ YES | NO | nnnn.nn } ]
         [ TYPE={ ANY | REQ | RESP } ]
         [ PRIORITY={ REQ | RESP } ]
         [ INPUT={ CHAR | HEX | HEXEXP } ]
```

The verb returns the next result line from a command executed by a previous &INTCMD statement, a message delivered by INTQ command, unsolicited messages, or event notifications.

Commands issued by an &INTCMD statement execute in the dependent processing environment of the issuing process, and results return to the dependent response queue for that dependent environment. The originating NCL process then uses the &INTREAD statement to retrieve command results line-by-line, from the response queue.

Other processes can send messages to your procedure using the &WRITE verb or INTQ commands.

This verb lets you write powerful logical procedures which correlate command results and react intelligently. On completing &INTREAD:

- System variable &ZVARCNT totals the variables created or modified by the operation.
- System variable &ZINTYPE indicates the dependent queue type where the message was obtained.
- The &INTREAD Message Profile is set and the $INT mapped object is available.

  Note: For more information about the message profile variables set for &INTREAD, see the *Network Control Language Programming Guide*.
- System variable &ZFDBK is set as follows on &INTREAD completion:

  0

  Message has been received.

  4

  Wait time has expired.

  8

  Response queue limit reached, messages discarded (the default is 32 K).
- &ZMDORC and &ZMDOFDBK are also set to indicate the result of MDO assignment that took place in making the $INT MDO available or in assigning the *mdoname* specified on the MDO operand.

Operands:

VARS=

Tokenizes the message into the nominated variables before control is returned to the procedure. Each word of the command output line is tokenized into the nominated variables from left to right. If insufficient variables are provided, data is lost. Excess variables are set to null. Formats for the operands associated with VARS= are:

prefix*

Generates the variables automatically during the tokenization process with variable names of '*prefix1* … *prefix2'*, and so on. The RANGE= operand is specified to indicate a starting and ending suffix number. *prefix** cannot be used with other variable names.

name

The name for a variable, excluding the ampersand (&).

***name*(*n*)**

As name, but *n* denotes the length of the data that is put into the variable.

***(*n*)**

Denotes a skip operation, where *n* represents the number of units to skip during the tokenization process. On VARS= statements, *n* denotes 'skip this number of words'. An asterisk (*) by itself is the same as *(1).

ARGS

Denotes that the line of text retrieved is tokenized and placed word-by-word into automatically generated variables with the form &1 through &*n*, depending on how many are required to hold the text. The RANGE= operand is coded to designate a start number and an end number to delimit the number of variables generated.

STRING=

Specifies that no tokenization is to be performed. The entire text of the command line is treated as a single string and returned to the procedure in the nominated variables. Formats for the operands associated with STRING are:

name

Specifies the name of a user-specified variable, excluding the leading &, in which to put the string text. Text is put into each variable up to the maximum length of each variable.

***name*(*n*)**

Specifies the name of a user-specified variable, excluding the leading &, in which to put the string text. Text is put into each variable for the length specified by *n*.

***(*n*)**

Denotes a skip operation, where *n* represents the number of units to skip during the tokenization process. On STRING statements, *n* denotes 'skip this number of characters'. An asterisk (*) by itself is the same as *(1).

SET

Specifies that no tokenization of the message is to be performed, but that the &INTREAD statement is to return only the message profile of the next message.

If SET is not specified, instructions must be coded on the &INTREAD statement specifying the tokenization requirements for the message text by using the other &INTREAD operands. SET is mutually exclusive with the INPUT operand.

MDO=*mdoname*

Specifies that, if a user MDO is embedded in the message, it is assigned to a local MDO with the name specified. An MDO might be present if the message was generated by an &EVENT MDO= or &WRITE MDO= statement. If no MDO is available, &ZVARCNT is set to zero, otherwise it is set to 1. All message profile variables are still set.

WAIT={ YES | NO | *nnnn.nn* }

WAIT=YES (the default) specifies that if no message is immediately available on the dependent response queue when &INTREAD is issued, the procedure waits until the next message arrives.

WAIT=NO specifies that control returns to the procedure immediately. If no message is available, all specified variables are set to null.

WAIT=*nnnn.nn* specifies the procedure can wait up to *nnnn.nn* seconds for a message to arrive. If no message is immediately available when the &INTREAD is issued (maximum is 9999.99 seconds). After *nnnn.nn* seconds without a message, control returns to the procedure with all specified variables set to null and &ZFDBK set to 4. If WAIT=0 is coded, control returns immediately. If WAIT=*nnnn.nn* is not used, it is your responsibility to ensure that the procedure is structured so that an &INTREAD is not issued in the case where messages cannot arrive. Otherwise, the NCL process waits indefinitely.

TYPE={ ANY | REQ | RESP }

TYPE=ANY returns the next message of any type, either from the dependent response queue or the dependent request queue. If messages are available on both queues, the first message from the dependent response queue is returned in preference.

TYPE=REQ indicates that &INTREAD is to return the first message from the dependent request queue of the issuing process.

TYPE=RESP (the default) indicates that &INTREAD is to return the first message from the dependent response queue of the issuing process.

PRIORITY={ REQ | RESP }

This operand is used with TYPE=ANY, to return the message if data is available on both the response and request queues. The default is PRIORITY=RESP. PRIORITY=REQ is used to select request messages in preference to responses.

INPUT={ CHAR | HEX | HEXEXP }

Specifies the format for the data returned in the variables created by the &INTREAD operation. The default is standard character data.

INPUT=HEX indicates that data in the variables is pure hexadecimal (and therefore not directly processable by NCL).

INPUT=HEXEXP indicates that data in the variables is hexadecimal data represented as expanded hexadecimal. For example, a hexadecimal byte with a value of 0A appears in a variable as two characters 0A.

HEX and HEXEXP support provide data transparency across &INTREAD operations.

Examples: &INTREAD

```
&INTREAD WAIT=5 ARGS RANGE=(20,80)
```

This requests the first message from the dependent response queue. The message is tokenized into variables &20 up to a maximum of &80 (&ZVARCNT is set to total how many variables were created). If no message is immediately available, control returns after 5 seconds.

```
&INTREAD VARS=(*(3),A(2),B(3),C,D,E,F)
```

This example reads the next message from the dependent response queue and tokenizes it into individual words:

- *(3) indicates that the first three words are ignored
- Two characters of the next word are placed in the variable &A.
- Three characters of the next word are placed in the variable &B
- The next four words are placed in variables &C, &D, &E and &F respectively.

Notes:

If a procedure is suspended waiting for an &INTREAD, use END or FLUSH commands to force termination.

When a message is received as a result of &INTREAD processing, a mapped data object, containing all attributes associated with the message, is made available to the NCL procedure in an object named $INT. The $INT object is an envelope for a user-defined, mapped data object, which is reassigned from the $INT.USERMDO component or assigned automatically if the MDO= operand was specified. The map name for the user MDO is available in the $INT.MAPNAME component or the &ZMAPNAME profile variable. If no user MDO is present when the MDO= operand is specified, a null unmapped MDO is assigned. Message profile variables and the $INT MDO will always be available after a successful &INTREAD, whether an embedded MDO is available or not. The $INT MDO is mapped by $MSG.

&INTCLEAR is used to clear outstanding messages queued to the dependent processing environment.

If &ZFDBK is set to 8 on &INTREAD completion, it means that the response queue was congested at some time with unprocessed messages that exceeded 32 KB. One or more messages might have been lost. This condition is unlikely to occur and represents a surge of messages being queued by dependent processing region activity faster than the &INTREAD process can process them. If this condition occurs, establish the cause of the large number of messages.

&INTREAD is sensitive to the current setting of the &CONTROL VARSEG option at the time the &INTREAD statement is issued. This option determines the processing method for the data delivered to the &INTREAD through INTQ commands.

For example (where a procedure using INTQ passes data to another procedure):

```
&A = &STR SHOW SESSINTQ ID=&NCLID &A
&INTREAD TYPE=REQ ARGS
```

If &CONTROL VARSEG (default) is in effect when the second procedure issues the &INTREAD, it receives two variables. &1 contains the word SHOW, and &2 the word SESS, even though the words were originally contained within one variable. However, if &CONTROL NOVARSEG is in effect, the second procedure receives a single variable containing SHOW SESS.

A response message is propagated to the current processes owning environment, using either the &INTCONT or &INTREPL statement.

Note: For more information about NCL processing environments and dependent processing environments, see the *Network Control Language Programming Guide*. See also the INTQ command description in the Online Help.

More information:

Sample Code (see page 1171)
&INTCMD (see page 370)
&INTCONT (see page 373)
&INTREPL (see page 382)
&INTCLEAR (see page 368)

&INTREPL

Propagates a message read via &INTREAD to the next higher processing environment, and optionally changes its message text.

```
&INTREPL [ COLOR=color | COLOUR=colour ]
         [ HLIGHT=highlight | HLITE=highlight ]
         [ INTENS={ HIGH | NORMAL } ]
         [ ALARM={ YES | NO } ]
         [ SCAN={ YES | NO }
         [ NRD={ NO | OPER } ]
         [ DATA=replacement text ]
```

After an &INTREAD statement, a procedure might choose to propagate the entire received message to the terminal (provided the executing process is not the dependent of a higher level process), or to the dependent response queue of the next higher level process (if the executing process is running in a dependent processing environment).

When an &INTCMD statement is processed, the command associated with the statement is executed in the issuing process's dependent processing environment. The command is processed in the normal manner but all command responses are returned to the originating process's dependent response queue, and not to the user's terminal. The issuing process can then use the &INTREAD statement to retrieve queued responses one-by-one.

&INTREPL allows messages received by a process executing as a dependent, to be passed up the hierarchy without having to use explicit &WRITE statements—the message text is changed before propagation occurs.

Operands:

COLOR=*color* | COLOUR=*colour*

If the color attribute of the message received from &INTREAD must be changed, use this operand to specify the new color. The existing color attribute of the message is tested by examining the &ZMCOLOUR message profile variable set after executing &INTREAD.

HLIGHT=*highlight* | HLITE=*highlight*

If the display highlighting attribute of the message received from &INTREAD must be changed, use this operand to specify the new attribute. The existing highlighting attribute for the message is tested by examining the &ZMHLIGHT message profile variable set after executing &INTREAD.

INTENS={ HIGH | NORMAL }

If the INTENS (intensity) attribute of the message received from &INTREAD must be changed, use this operand to specify the new attribute. The existing INTENS attribute of the message is tested by examining the &ZMINTENS message profile variable set after executing &INTREAD.

ALARM={ YES | NO }

If the ALARM attribute of the message received from &INTREAD must be changed, use this operand to specify the new attribute. The existing ALARM attribute of the message is tested by examining the &ZMALARM message profile variable set after executing &INTREAD.

SCAN={ YES | NO }

Indicates that the text of the message does or does not include the highlighting character, (@). Specify SCAN=YES if your replacement text includes this character to highlight individual words in the message.

NRD={ NO | OPER }

If the non-roll delete attribute of the message received from &INTREAD must be changed, use this operand to specify the new attribute. The existing NRD attribute of the message is tested by examining the &ZMNRD message profile variable set after executing &INTREAD. NRD=YES is specified but is ignored; full non-roll delete with DOM correlation can only be set by the message originator, for example &WRITE.

DATA=*replacement text*

Specifies the text for the message that is to replace the text received after the &INTREAD operation. The entire text must be coded. DATA can only be specified as the last keyword on the statement, because the data string is regarded as being everything right of the DATA= keyword, to the end of the statement.

Examples: &INTREPL

```
&IF &ZMALARM = YES &THEN +
   &INTREPL COLOR=RED INTENS=HIGH +
            DATA=*IMPORTANT* &ZMTEXT
```

Notes:

&INTREPL only propagates messages received by executing &INTREAD TYPE=RESP, or after &INTREAD TYPE=ANY is satisfied by a message from the procedure's dependent response queue. Messages that satisfy &INTREAD TYPE=REQ cannot be propagated by &INTREPL. The procedure terminates in error if you attempt to use &INTREPL to propagate a request message. The message profile variable &ZINTYPE is set to a value of RESP if &INTREPL is used to propagate the message.

Attributes of the message can also be changed by modifying the $INT by modifying the $INT MDO before issuing the &INTREPL or &INTCONT. Some care should be taken when modifying message attributes in this way, as some attributes affect processing control. Also, some attributes will be reset by the system during message re-delivery.

Note: For more information, see the *Network Control Language Programming Guide*.

&INVSTR

Returns a string that is the inverted form of the supplied text.

```
&INVSTR text
```

&INVSTR is a built-in function and must be used to the right of an assignment statement.

&INVSTR is used in an assignment statement to reverse the data order. For example, inverting the data 123456 assigns the nominated variable the data 654321.

Operands:

text

The data to be inverted and placed in the nominated variable. A maximum of data that is specified is the maximum length of the variable. The data is reversed and placed into the target variable of the assignment statement.

Examples: &INVSTR

```
&A = &INVSTR ABCDEF  -* &A will be set to FEDCBA

&1 = &INVSTR &1 &2
```

Notes:

&INVSTR is a simple way of stripping a number of characters from the end of a variable length string: use &INVSTR to invert the string, then &SUBSTR to strip the required number of characters from the front of the string. A second &INVSTR statement then returns the string to its original order.

&INVSTR is useful when constructing keys for file processing from data entered from a full-screen panel if the data entered must be reordered to construct a suitable key.

&LBLSTR

Returns a string with leading blanks deleted.

&LBLSTR *text*

&LBLSTR is a built-in function and must be used to the right of an assignment statement.

User variables can contain leading blanks, whether entered by an operator during full-screen processing or created by other built-in functions such as &SUBSTR and &ASISTR.

&LBLSTR removes any leading blanks from the data and assigns the data remaining into the target variable of the assignment statement.

If the data consists entirely of blanks, then the target variable is set to null.

Operands:

text

Data or a variable with data from which leading blanks are to be removed.

Examples: &LBLSTR

```
&1 = &LBLSTR &INPUT -* If &INPUT = '  ABC', then
                    -* &1 is set to 'ABC'
```

Note:

The &NBLSTR function is used to remove both leading and trailing blanks. The &TBLSTR function removes only trailing blanks.

&LENGTH

Returns a number indicating the length of the supplied variable or constant.

```
&LENGTH { variable | constant }
```

&LENGTH is a built-in function and must be used to the right of an assignment statement. The character length value of the target variable is returned in the variable to the left of the assignment statement.

Operands:

variable* | *constant

A variable or constant.

Examples: &LENGTH

```
&A = ABCDEF       -* assigns a value to &A

&LEN = &LENGTH &A -* &LEN is assigned a value of 6
```

Notes:

&LENGTH is used to determine the length of a string before attempting to use &SUBSTR to extract the right hand part of that string. The &SETLENG statement is used to set a variable to a specific length.

If &CONTROL DBCS or DBCSN or DBCSP is in effect, &LENGTH is sensitive to the presence of DBCS data.

More information:

&LENGTH (see page 1200)

&LOCK

Obtains or releases access to a logical resource.

```
&LOCK PNAME=resource name
    [ TYPE={ EXCL | SHR | TEST | FREE } ]
    [ ALTER={ YES | NO } ]
    [ MNAME=resource name ]
    [ WAIT={ NO | YES | nnnn } ]
    [ TEXT={ text | var | userid procedure-name time } ]
```

&LOCK controls concurrent access to a particular logical resource by different NCL processes. &LOCK lets a process nominate a resource by name (for example, a file ID) for either exclusive or shared access to that resource. This status is altered during processing when ALTER =YES is specified. A procedure that needs to access the resource can use &LOCK to find out whether it is entitled to use it, or whether the current process or another process already has exclusive use.

&LOCK is used to synchronize the activities of different procedures that access, update, or reference common information.

On completion, &RETCODE is set to indicate the result of the operation.

Operands:

PNAME=*resource name*

(mandatory) Specifies the primary name for the resource targeted by this &LOCK action. The name is any string of 1 to 16 characters.

TYPE={ EXCL | SHR | TEST | FREE }

TYPE=EXCL indicates a request for exclusive control of the resource, as identified by the PNAME (primary name) and optional MNAME (secondary name) operands.

TYPE=SHR indicates a request for shared access to the nominated resource.

TYPE=TEST indicates a request to find out whether the nominated resource is already the subject of an &LOCK operation by this or another process.

TYPE=FREE indicates a request to release the shared or exclusive access to the nominated resource by this process.

ALTER={ YES | NO }

Specifies whether or not an NCL procedure can request that the lock type and/or lock text be changed during processing. ALTER=YES allows the type and text to be changed, and ALTER=NO, the default, specifies that no change is allowed.

For the rules that apply when changing the lock type and text, see Altering the Lock Type During Processing.

MNAME=*resource name*

Specifies the minor name for the resource. This operand is optional. The name is any string of 1 to 256 characters.

WAIT={ NO | YES | *nnnn* }

WAIT=NO indicates that the request for shared or exclusive control is to lapse immediately if access to the nominated resource cannot be obtained.

When WAIT=YES is specified for TYPE=SHR or EXCL, it indicates that if the request for access to the resource cannot be granted immediately, the &LOCK function will wait indefinitely until the request is granted, or the request fails. For TYPE=TEST, if the resource is not currently held by any user, the request will wait indefinitely until the next request of any type is made by another process.

WAIT=*nnnn* indicates that if the request for exclusive or shared control of the resource cannot be granted immediately, then the &LOCK function will wait until the request is granted, for up to *nnnn* seconds maximum (the time value can range from 1 to 9999 seconds). For TYPE=TEST, if the resource is not currently held by any user, the request will wait for the specified period or until the next request of any type is made by another process.

TEXT=*text* | *var* | userid procedure-name time

Specifies a maximum of 24 characters of free form text to be associated with this access request. The value of the text is returned in the variable &1 for certain &LOCK operations. If no text is supplied on the &LOCK statement, the default is the current user ID, suffixed by the name of the base NCL procedure, suffixed by the time. The text assigned to the request is composed of text, or variables, or a mixture of both, but cannot contain blanks. If blanks are required in the text, assign the required value to a variable using the &STR function, then use the variable on the TEXT= operand.

Return Codes:

TYPE=EXCL and TYPE=SHR requests complete with &RETCODE set as follows:

0

Access to the resource is granted. For an EXCL request no other processes can gain access to the resource until it is explicitly FREEd, or the process terminates.

4

The request has been ignored because this process has control of the resource already. &1 is set to the text of the existing lock. &ZFDBK is set as follows:

0

Lock is already held shared

4

Lock is already held exclusively

8

A lock conflict has been detected

8

Access to the resource is denied. Another process has control of the resource. The variable &1 contains the text of the oldest request that obtained control of the resource. &ZFDBK is set as follows:

0

Lock is held shared (TYPE=EXCL requests only)

4

Lock is held exclusively

When WAIT=YES or WAIT=*nnnn* is specified:

12

The request is denied. Access to the resource is not possible. A PURGE command has been issued for the waiting lock request or the time limit has expired. The variable &1 contains the text of the oldest request that obtained control of the resource. &ZFDBK is set as follows:

0

Lock is held shared (TYPE=EXCL requests only)

4

Lock is held exclusively

16

The request is rejected. &SYSMSG is set to the text describing the error condition.

TYPE=TEST requests complete with &RETCODE set as follows:

0

No other procedure has shared or exclusive control of the resource, or is testing the resource.

4

The request will fail because this process already has control of the resource. &1 is set to the text of the existing lock. &ZFDBK is set as follows:

0

Lock is already held shared

4

Lock is already held exclusively

8

Lock conflict has been detected

8

Another process controls the resource. Variable &1 contains the text of the oldest request that obtained control of the resource. &ZFDBK is set as follows:

0

Lock is held shared

4

Lock is held exclusively

8

A TEST request has been made against the resource

TYPE=FREE requests complete with &RETCODE set as follows:

0

Shared or exclusive control of the resource has been released by this process. Variable &1 contains the text of the request just released. &ZFDBK is set as follows:

0

Lock is held shared

4

Lock is held exclusively

4

The request is ignored. This procedure has no &LOCK control, either shared or exclusive, of the nominated resource.

When ALTER=YES is specified, requests complete with &RETCODE set as follows:

0

Shared or exclusive control of the resource has been granted. Variable &1 contains the text of the request just released. &ZFDBK is set as follows:

0

Lock is held shared

4

Lock is held exclusively

8

The request is denied. &ZFDBK is set as follows:

0

Lock is held shared

4

Lock is held exclusively &1 is set to the text of the lock that prevented the request from succeeding. The text of the requesting NCL procedure's original lock request remains unchanged, and there is no alteration to the original lock ownership (that is, the NCL procedure still owns the lock in shared status as if the &LOCK ALTER=YES was not issued).

Examples: &LOCK

```
&TODAY = MONDAY
&DTIME = NIGHT
&LOCK TYPE=EXCL PNAME=&TODAY MNAME=&DTIME
   .
   . -* process
   .
&LOCK TYPE=FREE PNAME=&TODAY MNAME=&DTIME
&LOCK TYPE=SHR PNAME=CUSTFILEMNAME=CUSTREC
&LOCKTEXT = &STR &0 write custfile
&LOCK TYPE=EXCL ALTER=YES PNAME=CUSTFILE MNAME=CUSTREC WAIT=20 +
      TEXT=LOCKTEXT
```

Notes:

The resource referred to in connection with the &LOCK statement is not a real entity such as a file or a variable. A resource in this context is a name to which one or more NCL procedures refer. The name is made up of a primary name (PNAME), and optionally a minor name (MNAME), for example, CONFIG.ABC and CONFIG.XYZ (where CONFIG is the primary name and ABC and XYZ are minor names).

If two procedures agree to use the resource CONFIG.ABC to control their access to a UDB, each would request exclusive use of the resource CONFIG.ABC. While one was granted access, the other would fail to obtain access and could not use the UDB. It must be noted that &LOCK resources provide protection only if all procedures that need to synchronize access to a resource are written using &LOCK against a mutually-agreed resource.

Termination of the process releases all locks that are still held.

The &LOCK TYPE=TEST WAIT=YES construction is used to create a semaphore for signalling between procedures.

Note: For more information, see the *Network Control Language Programming Guide*. See also the PURGE and SHOW NCLLOCKS commands in the Online Help.

Altering the Lock Type During Processing

A process can alter the status of its resource lock during processing, if required, by using the ALTER=YES operand on the &LOCK verb. Altering the status from exclusive to shared is always possible, but altering the lock status from shared to exclusive is more complicated.

When the status is altered from exclusive to shared during processing, any other lock requests that are waiting for shared access to the resource become valid for shared ownership. They are granted shared access to the resource immediately, causing the requesting procedures to resume execution. The text of the lock is replaced by the text specified on the &LOCK verb.

When altering the status from shared to exclusive, the request can only be satisfied if there are no other procedures that have shared ownership of the resource. Also, if the resource is the primary resource, the upgrade request can only be satisfied if there are no other minor resources (with shared or exclusive status) with the same primary name. If any other shared requests for the lock arrive before the status is altered to exclusive, these new shared requests are given precedence over the change to exclusive, and are granted shared ownership of the lock. If the request is successful, the procedure will own the lock exclusively, and the text of the lock will be replaced by the text specified on the &LOCK verb.

As with normal shared and exclusive requests, the WAIT operand plays an important part in determining the success or failure of a lock status alteration request. However, the waiting period is only significant for a request to change from shared to exclusive, as the request to change from exclusive to shared is always satisfied immediately.

When WAIT=NO is specified, the request must be satisfied immediately or it will fail.

When WAIT=*nnnn* or WAIT=YES is specified, the requesting process can wait for the specified period of time for the change to be successful. This can happen when other procedures release locks.

When the status type on the &LOCK request is unchanged from the current status (that is, the procedure holds a shared lock then issues a shared lock request with the ALTER=YES operand) the request is treated as a request to alter the lock text only. This request is always successful.

&LOGCONT

Resumes normal processing of a message delivered to LOGPROC.

```
&LOGCONT [ USER=userid ]
         [ NODE=luname ]
```

Used within the LOGPROC procedure to request that a message previously delivered for processing by &LOGREAD be returned for standard log processing.

Operands:

USER=*userid*

If specified, indicates the new user ID to which this log message is to be attributed.

NODE=*luname*

If specified, indicates the new terminal name to which this log message is to be attributed.

Examples: &LOGCONT

```
&LOGCONT USER=CHIEFOP
```

Notes:

- &LOGCONTs issued when no message is currently being processed by LOGPROC are ignored.
- An &LOGCONT need not be issued if another &LOGREAD is to be issued. If a message is being processed and another &LOGREAD is issued without an intervening &LOGDEL or &LOGREPL, an implied &LOGCONT is performed and the message returned for standard processing before the next &LOGREAD is satisfied.
- &LOGCONT is used to free a message for delivery even though the LOGPROC procedure continues processing before issuing another &LOGREAD.

More information:

&LOGREAD (see page 400)
&LOGDEL (see page 397)
&LOGREPL (see page 404)

&LOGDEL

Requests deletion of the log record currently being processed by LOGPROC.

```
&LOGDEL
```

Used with the LOGPROC procedure to delete the log message. An &LOGDEL statement must only be used after an earlier &LOGREAD statement—the record it returns is the record that will be deleted by the &LOGDEL statement.

Examples: &LOGDEL

```
&LOGREAD ARGS
&IF &4 EQ N13505 &THEN +
   &LOGDEL
```

Any attempt to use &LOGDEL in other than the designated LOGPROC procedure results in an error.

More information:

&LOGREAD (see page 400)

&LOGON

Passes control of this terminal to another application.

```
&LOGON [ LOGMODE=logmode ]
       [ APPL= { applname | * } ]
       { panelname | * }
       [ string ]
```

An NCL procedure operating under EASINET control can request that control of a terminal be passed to another application. The &LOGON statement initiates passing the terminal to another application.

Operands:

LOGMODE=*logmode*

This operand is optional but must be coded as the first operand if used. It is the name of an VTAM logmode entry used when processing this request. If not specified, the logmode used is the default from either the APPL definition specified by the APPL= operand, or from the nominated DEFLOGON entry. If logmode is specified and the target application is this product region, the terminal is disconnected and reacquired using the specified logmode table entry.

APPL={ *applname* | * }

The name for the required application. This operand is optional, and (if used) designates a specific target application. When this operand is coded, it must be coded after the LOGMODE operand (if present), and before any other operands. If APPL= is used to designate a specific application, the string operand is optional. If APPL=* is coded it indicates that the target application is this product region, in which case the terminal is disconnected and immediately reacquired.

***panelname* | ***

(Mandatory) The name for a panel for display before passing control to the target application. This panel normally informs the terminal operator that the logon request has been accepted and is being processed.

This operand is ignored for LU1 terminals, but must still be supplied. For LU1 terminals, any text assigned to the &SYSMSG variable before the &LOGON is executed is written to the terminal if the logon request is accepted.

If you specify *, no panel is displayed before passing the terminal to the target application.

string

The meaning of this operand depends on the presence of the APPL operand.

If APPL is coded on the &LOGON statement, the string operand is optional and treated as user data text (maximum length 168 bytes) to be passed to the target application specified by the APPL= operand within the logon data. If user data is supplied in variables, the contents of the variables could be hexadecimal data, which is passed without change to the target application.

If APPL is not coded, string is mandatory and represents text that is compared against the logon paths defined in the DEFLOGON command. This lets the &LOGON statement select target applications indirectly, through the DEFLOGON mechanism, rather than explicitly by the APPL= definition.

Examples: &LOGON

```
&LOGON REQACCEPT IMS
&LOGON LOGONOK  &INKEY USERID1/PASSWORD
&LOGON APPL=&SELECTION OKPANEL &USERDATA
&LOGON LOGMODE=MODEL4 OKPANEL TSO
```

Notes:

After issuing an &LOGON statement, the NCL procedure will not regain control unless the terminal cannot be passed to the target application. In this case, processing resumes with the next statement after the &LOGON statement. The &SYSMSG variable will contain an error message describing the reason for the failure.

Any attempt to use the &LOGON statement from other than an EASINET procedure results in an error.

Note: See also the $EASINET procedure in the distribution library.

&LOGREAD

Requests that the next log message be made available to LOGPROC.

```
&LOGREAD { VARS= prefix* [ RANGE=( start,end ) ]|
           VARS={ name | ( name,name, ..., name ) } |
           STRING=( name, name, ..., name) } |
           ARGS [ RANGE=( start,end ) ] |
           MDO=mdoname | SET }
```

The system-level LOGPROC procedure can intercept and process messages directed to the activity log.

The &LOGREAD statement is reserved for use by the LOGPROC NCL procedure and requests that processing of the LOGPROC procedure be suspended until the next log message is available.

After the &LOGREAD statement, the LOGPROC procedure can analyze the log message and initiate any special processing required.

Optionally, the LOGPROC procedure can delete messages from the log using the &LOGDEL statement. The &LOGREAD statement is designed to tokenize the incoming log messages into variables in a variety of forms to simplify analysis and interpretation of the messages by the LOGPROC logic.

On completion of &LOGREAD the system variable &ZVARCNT is set to the number of variables created or modified by the operation.

The profile of the message received by &LOGREAD is set in a suite of reserved system variables. The message profile (which includes attributes such as color, highlighting, and source information) provides a complete description of all log message attributes, in addition to the message text. The message attributes can also be accessed using the $LOG MDO, which is always available after a successful &LOGREAD. The $LOG MDO is mapped by the $MSG map.

Note: For more information about the &LOGREAD message profile, see the *Network Control Language Programming Guide*.

Operands:

SET

Specifies that no tokenization of the log message is to be performed, but that the &LOGREAD statement is to return only the message profile for the next log message.

If SET is not specified, operands must be coded on the &LOGREAD statement specifying the tokenization requirements for the message text using other &LOGREAD operands.

VARS=

Specifies that the message is to be tokenized into the nominated variables before control returns to the procedure. Each word of the log record line is tokenized into the nominated variables from left to right. If insufficient variables are provided, some data will be lost. Excess variables are set to null. The formats of the operands that is coded with VARS= are described below:

prefix*

Denotes that variables are generated automatically during the tokenization process, and that variable names will be *prefix1* … *prefix2* and so on. (The RANGE= operand is specified to indicate a starting and ending suffix number). *prefix** cannot be used with other variable names.

name

The name of a variable, excluding the ampersand (&).

***name*(*n*)**

As name, but *n* denotes the length of the data to be placed in the variable.

***(*n*)**

Denotes a skip operation, where *n* represents the number of units to be skipped during the tokenization process. On VARS= statements, *n* denotes 'skip this number of words'. An asterisk (*) by itself is the same as *(1).

STRING=

Specifies that no tokenization is to be performed. The entire text of the command line is treated as a single string and returned to the procedure in the nominated variables. The formats of the operands associated with STRING are:

name

User-specified variables, excluding the leading &, into which the string text is put. Text is placed into each variable up to the maximum length of that variable.

***name*(*n*)**

User-specified variables, excluding the leading ampersand (&), into which the string text is put. Text is placed into each variable for the length *n*.

***(*n*)**

Denotes a skip operation, where n represents the number of units to be skipped during the tokenization process. On STRING statements, *n* denotes 'skip n characters'. An asterisk (*) by itself is the same as *(1).

ARGS

Denotes that the line of text retrieved is tokenized and placed word-by-word into automatically generated variables of the form &1 through &*n*, depending on how many are required to hold the text.

The RANGE= operand is coded to designate start and end values to delimit the number of variables generated.

MDO=*mdoname*

Specifies the object name to which an MDO is to be assigned if an MDO is present in the log message.

MDOs are user data structures, which is passed to NCL processes with &EVENT or &WRITE statements. They will be propagated to the system log if they are present in an eligible log message.

Examples: &LOGREAD

```
&LOGREAD ARGS RANGE=(20,80)
```

The next log message to arrive is tokenized into variables from &20 up to &80 maximum. &ZVARCNT is set to indicate how many variables were created.

```
&LOGREAD VARS=(*(3),A(2),B(3),C,D,E,F)
```

The next log message is tokenized into individual words. *(3) indicates that the first 3 words are ignored, 2 characters of the next word are placed in the variable &A, three characters of the next word are placed in the variable &B and the next 4 words are placed into variables &C, &D, &E, and &F respectively.

```
&LOGREAD VARS=ABC* RANGE=(1,50)
```

The next log message is tokenized and placed word-by-word into a series of automatically generated variables with the form ABC1, ABC2, ..., ABC50, and so on. Variables are generated, up to the limit specified by the RANGE operand.

```
&LOGREAD STRING=(A,B(2),*(5),C(3))
```

Reads the next log message as a single text string. The first 256 bytes are placed in &A, the next 2 characters are placed in &B, the next 5 characters are ignored, and the next 3 characters are placed in &C.

Notes:

Log records always include the time the record was generated (in format HH.MM.SS.TH), with the user ID for the user who generated the record, and the TERMINAL where that user is logged on.

If STRING is not specified, these fields are tokenized as follows:

- First variable is assigned the time.
- Second variable is assigned the user ID.
- Third variable is assigned the terminal.

These three fields are always present.

If no USERID or TERMINAL is associated with a particular message (such as for commands executed during system initialization), either or both of these fields are set to a single minus (-) character. Therefore, the NCL procedure can assume that the fourth variable will contain the first word of the actual text of the log message.

For commands executed from an NCL process's dependent processing environment (via &INTCMD), the terminal field appears as **nclid**, where *nclid* is the NCL process identifier.

After a successful &LOGREAD, a $LOG MDO will always be available to LOGPROC. $LOG is mapped by $MSG and contains all attributes associated with a message. In addition, if the message was an envelope for a user MDO, the user MDO is referenced directly from $LOG.USERMDO or automatically assigned from the MDO= operand on the &LOGREAD. The mapname for the user MDO is available in $LOG.MAPNAME or the &ZMAPNAME profile variable.

&ZMDORC and &ZMDOFDBK are set.

In the activity log certain characters are used to highlight specific classes of messages. For example, the plus sign (+) prefixes commands, and an equal sign (=) prefixes VTAM PPO messages. These special indicators appear in the physical printed log only, and are not returned in the tokens that result from an &LOGREAD statement.

An example of using LOGPROC to intercept particular messages arises with File Transmission Services (FTS). A LOGPROC procedure is written to monitor when transmissions finish. On detecting the completion of certain transmissions (determined by their request name or data set name), LOGPROC can dynamically generate job control language (JCL) to process a newly-arrived data set and automatically submit it to the internal reader.

An example of a utility to perform this is supplied with the system as UTIL0005.

&LOGREAD can also be used to split the activity log into different categories and write records to data sets where they are available for on-line browsing. An example of a LOGPROC to perform this, and a supplementary procedure to perform on-line log browsing is supplied with the system.

Any attempt to use &LOGREAD in other than the designated LOGPROC procedure results in an error.

Note: For more information about the LOGPROC procedure, see the *Reference Guide*. See also LOGPROC message profile variables.

More information:

&LOGDEL (see page 397)
&LOGCONT (see page 396)
&LOGREPL (see page 404)

&LOGREPL

Replaces the text of the last log message delivered to LOGPROC.

```
&LOGREPL [ USER=userid ]
         [ NODE=luname ]
           DATA=message text
```

Used within the LOGPROC procedure to change the text for a message delivered for processing by &LOGREAD, and returns the message for standard log processing.

Operands:

USER=*userid*

If specified, indicates the new user ID to which this log message is to be attributed.

NODE=*luname*

If specified, indicates the new terminal name to which this log message is to be attributed.

DATA=*message text*

The new message text to be assigned to the message. DATA can only be specified as the last keyword on the statement since the data string is regarded as being everything following the DATA= keyword, to the end of the statement.

Examples: &LOGREPL

```
&LOGREPL USER=CHIEFOP DATA=**MESSAGE TEXT SUPPRESSED** &TIME
```

Note:

&LOGREPLs issued when no message is currently being processed by LOGPROC are ignored.

More information:

&LOGREAD (see page 400)
&LOGDEL (see page 397)
&LOGCONT (see page 396)

&LOOPCTL

The &LOOPCTL verb sets a new runaway loop control limit when &CONTROL LOOPCHK is in effect. If the default, &CONTROL NOLOOPCHK, is in effect, the &LOOPCTL is ignored.

When a new procedure nesting level begins, &LOOPCTL is initialized to 1000. This value decrements by 1 for each &GOTO, &GOSUB, or &DOEND processed for a &DOWHILE, or &DOUNTIL statement. If the &LOOPCTL reaches 0, the procedure is terminated for being a potential runaway loop. &LOOPCTL is set to a new value, avoiding premature termination of a procedure.

This verb has the following format:

```
&LOOPCTL number
```

Operands:

number

Specifies a numeric value in the range 1 through 10,000.

Examples: &LOOPCTL

```
&LOOPCTL 2000
&IF &LOOPCTL < 200 &THEN +
 &LOOPCTL 700
```

Notes:

&LOOPCTL (see page 837) is also a system variable that returns the current loop control counter value.

Certain system functions imply a reset of the current loop counter. For example, the successful displaying of a panel using the &PANEL statement. In such a case where a panel is displayed awaiting operator input, the current loop counter is reset to 1000. This feature eliminates the need to reset the counter using &LOOPCTL in procedures that perform long processing runs, but which are not looping.

Resetting &LOOPCTL within PPOPROC, MSGPROC, CNMPROC, and LOGPROC should not be necessary. Although these are decremented as is usual, &LOOPCTL is automatically reset when each &PPOREAD, &MSGREAD, &CNMREAD, &LOGREAD, and &INTREAD is issued. Thus, it is only necessary to reset it within a procedure where an abnormally high amount of processing is being performed for a single message.

More information:

&GOTO (see page 357)
&GOSUB (see page 354)
&DOEND (see page 302)
&CONTROL (see page 266)

&MAICMD

Specifies an MAI primary command.

```
&MAICMD command-string
```

Operands:

command-string

The following values is specified in this field:

END

Requests termination of MAI as the primary environment processor. When the MAI menu process terminates then control will be passed to a new primary environment-either the primary menu or OCS.

JF, JR, or J

Specifies that a jump is to be performed when the MAI Primary menu terminates.

&MAICONT

Sends the current data stream to the terminal and/or the application.

```
&MAICONT [ CONT | PLU | SLU | BOTH ]
         [ VIEW={ WAIT | OPT } ]
```

Requests that the current data streams be sent on to one or other or both of the PLU (application) or SLU (terminal).

Operands:

CONT

Indicates that the data stream should flow in the current direction.

If the last data stream received via &MAIREAD was received from the PLU, that data stream is forwarded to the SLU. If the last data stream was received from the SLU, that data stream (or one built up using &MAIPUT or modified using &MAIINKEY) is forwarded to the PLU.

PLU

Indicates that the data stream should be sent to the PLU. If the last data stream received via &MAIREAD was received from the SLU, that data stream (or one built up using &MAIPUT or modified using &MAIINKEY) is forwarded to the PLU. If the last data stream was received from the PLU, one or more &MAIPUT statements must have been issued to build up a data stream, which is then sent back to the PLU, otherwise the statement is ignored.

SLU

Indicates that the data stream should be sent to the SLU (terminal). The last data stream received via &MAIREAD must have been received from the PLU (application); otherwise the statement is ignored.

BOTH

Applies if the last data stream received via &MAIREAD was from the PLU, and indicates that the data stream should be sent to the SLU, and that a data stream built up using one or more &MAIPUT statements is to be sent to the PLU.

If no &MAIPUTs have been issued, &MAICONT BOTH functions like &MAICONT SLU. If the last data stream received was from the SLU, &MAICONT BOTH functions like &MAICONT PLU.

VIEW=WAIT

Indicates that data sent to the SLU must be displayed before the procedure resumes. The session cannot have screen ownership at the time the &MAICONT is issued-the user could be viewing another MAI session. VIEW=WAIT suspends execution of the procedure until the user jumps back to this session.

VIEW=OPT

Indicates that data sent to the SLU (terminal) is to be displayed.

At the time when the &MAICONT is issued, the session cannot have screen ownership, as the user might be viewing another MAI session. In this case, VIEW=OPT results in the data stream being discarded, and never being displayed at the terminal. However, if the session does own the screen at the time &MAICONT is issued, the data is displayed normally.

Examples: &MAICONT

```
&MAICONT BOTH VIEW=OPT
&MAICONT
&MAICONT SLU VIEW=WAIT
```

Notes:

If no &MAICONT is issued between two successive &MAIREAD statements, an &MAICONT CONT VIEW=OPT is implied.

An &MAICONT issued when there is no data outstanding is normally ignored. Data is outstanding from the time an &MAIREAD is satisfied until an &MAICONT, &MAIDEL or another &MAIREAD is issued. However, in the case of a script procedure started using a .S session skip command or session cleanup, an &MAICONT PLU is allowed before the first &MAIREAD. An &MAICONT PLU is also allowed after the time interval specified on an &MAIREAD WAIT= verb expires.

An &MAICONT that requests data to be sent to the PLU is ignored if the current session state does not allow input. For example, if the keyboard of an SNA terminal is locked. Such an &MAICONT should not be issued unless the procedure knows the &MAICONT will be successful. The &MAIUNLCK system variable is provided so that the procedure can wait for a data stream that unlocks the keyboard if necessary.

An &MAICONT PLU is often used to automatically reply to the application. For example, if the application prompts for a user ID, the script procedure can set up a reply data stream using &MAIPUTs, and then send that reply using &MAICONT PLU. Alternatively, an &MAICONT BOTH could be used. This would send the reply just as &MAICONT PLU does, but also send the prompt from the application on to the terminal. In this way, the terminal user sees the output from the application as it occurs. The VIEW operand governs whether the procedure proceeds even if the user is viewing a different session.

With certain applications some care must be taken to ensure screen image integrity. Take the ISPF product as an example. ISPF usually only sends data to the terminal that has changed since the last send. If a script procedure does not send each output from ISPF on to the terminal, an eventual send could result in only a partial screen image. This problem is solved by using the VIEW=WAIT option to ensure that each data stream is sent to the terminal. Alternatively, if output is not to be sent to the terminal, the procedure can request to show the complete screen image again. Using ISPF, this would be accomplished by pressing the PA2 key (using &MAIINKEY and &MAIPUT), and then sending the resulting complete screen image on to the terminal.

If the script procedure has issued an &PANEL statement without a subsequent &PANELEND, then either an explicit or implicit &MAICONT SLU will result in an &PANELEND to allow the data to be sent to the terminal. This does not apply if VIEW=OPT is coded.

More information:

&MAIPUT (see page 418)
&MAIINKEY (see page 416)
&MAIREAD (see page 419)

&MAICURSA

Sets up the cursor address when building a data stream to be sent to the application.

```
&MAICURSA [ ROW=n ] [ COL=n ]
```

Sets the cursor address to be present in a data stream to be sent to the PLU (application). The data stream is built up using one or more &MAIPUT statements.

Operands:

ROW=*n*

Indicates the cursor screen row position. If the ROW operand is not present, row 1 is assumed.

COL=*n*

Indicates the cursor screen column position. If the COL operand is not present, column 1 is assumed.

Examples: &MAICURSA

```
&MAICURSA ROW=2 COL=34
```

Notes:

The supplied row and column values must be within the dimensions of the current screen, otherwise an error results.

The &MAICURSA statement only has effect when a data stream built using &MAIPUT(s) is sent to the application. If no &MAICURSA statement is issued, row 1, column 1 is assumed.

It is often not necessary to set a cursor address when building a data stream. Under most circumstances, the application is not sensitive to cursor position at time of input. However, &MAICURSA is provided for those times when you wish to set a specific cursor position.

More information:

&MAIPUT (see page 418)

&MAIDEL

Signifies that a data stream is not to be delivered.

```
&MAIDEL
```

Requests that the current data stream be deleted, that is, not delivered to either the PLU (application) or the SLU (terminal).

Notes:

&MAIDEL is most often used when an automatic reply is generated to the application, but the last data stream received from it is not to be sent to the terminal. For example, a script can perform automatic logon to an application, and place the user in an initial transaction, but not let the user see anything displayed on the terminal while this takes place.

An &MAIDEL issued when there is no data outstanding is ignored. Data is outstanding from the time an &MAIREAD is satisfied until an &MAICONT, &MAIDEL, or another &MAIREAD is issued.

More information:

&MAICONT (see page 407)
&MAIREAD (see page 419)

&MAIDSFMT

Places the entire current data stream into variables.

```
&MAIDSFMT { VARS=prefix* [ RANGE=(start,end) ] |
            VARS={ name | ( name, name, ..., name) } |
            STRING &name &name ... &name |
            ARGS [ RANGE=(start,end) ] }
```

Requests that the current data stream be made available to the procedure by placing it into the variables specified. The data stream is placed into the variables in character format hexadecimal, with each two characters representing one hexadecimal byte. For instance, if a single byte of the data stream contains X'1D', it will be represented in a variable as two characters: 1D.

On completion of &MAIDSFMT the system variable &ZVARCNT is set to the count of variables created or modified by the operation.

Operands:

VARS=

Specifies that the data stream is to be tokenized into the nominated variables before control is returned to the procedure. Tokenization is performed from left to right of the data stream. If necessary, the data stream is broken into sections whose length is the maximum allowed for a single variable or the amount specified. If insufficient variables are provided, some data will not be available to the procedure. The format of the operands associated with VARS= is as follows:

prefix*

Denotes that variables are to be generated automatically during the tokenization process, and that the variable names will be *prefix1 prefix2* ... and so on. The RANGE= operand is specified to indicate a starting and ending suffix number. Default is RANGE=(1,64). *prefix** cannot be used in conjunction with other variable names.

name

The name of a variable, excluding the &.

***name*(*n*)**

The same as name, but *n* denotes the number of bytes of the data stream that are to be placed in the variable.

***(*n*)**

Denotes a skip operation, where *n* represents the number of bytes of the data stream to be skipped during the tokenization process. Specifying an asterisk by itself is the same as *(1).

STRING

STRING on an &MAIDSFMT statement functions in a similar way to VARS. The data stream is tokenized into the nominated variables before control is returned to the procedure. Tokenization is performed from left to right of the data stream. If necessary, the data stream is broken into sections whose length is the maximum allowed for a single variable, or the amount specified. If insufficient variables are provided, some data will not be available to the procedure. Excess variables will be set to a null value. The formats of the operands associated with STRING are as follows:

&name

The name of a variable including the leading &.

***&name*(*n*)**

As *name*, but *n* denotes the number of bytes of the data stream that are to be placed in the variable.

***(*n*)**

Denotes a skip operation, where *n* represents the number of bytes of the data stream to be skipped during the tokenization process. Specifying an asterisk by itself is the same as *(1).

ARGS

Denotes that the data stream is to be placed into automatically generated variables of the form &1 through &*n*, depending on how many are required to hold the text. The RANGE= operand is coded to designate a start number and, optionally, an end number, which delimits the number of variables to be generated. The default is RANGE=(1,64).

Examples: &MAIDSFMT

```
&MAIDSFMT ARGS RAN GE=(20,30)
```

Requests that the data stream be placed into the variables &20 through &50. &ZVARCNT will be set to indicate how many variables were created.

```
&MAIDSFMT VARS=(*(3),A(2),B(3),C,D,E,F)
```

*(3) indicates that the first 3 bytes of the data stream are ignored, the next 2 bytes are then to be placed in the variable &A, the next three are placed in the variable &B and the rest of the data stream is placed into the variables &C, &D, &F and &F respectively, with each of these variables containing the maximum possible number of characters.

```
&MAIDSFMT VARS=ABC* RANGE=(1,20)
```

The data stream is placed into variables &ABC1, &ABC2, and so on, up to &ABC20.

Notes:

&MAIDSFMT is useful to interrogate a data stream more fully than would be possible using &MAIFIND. Data streams could also be archived, for example, for future investigation.

The data stream obtained from the PLU by &MAIDSFMT is used after some modification on an &MAIREPL verb. &MAIREPL totally replaces a data stream received from the PLU.

If there is no data outstanding when an &MAIDSFMT is issued, the current screen image is returned. This allows a script started part way through a session to determine the format and content of the current screen image. The current screen image is also returned if &MAIDSFMT is used in the MAI help procedure. Data is outstanding from the time an &MAIREAD is satisfied until an &MAICONT, &MAIDEL, or another &MAIREAD is issued.

More information:

&MAIFIND (see page 414)
&MAIREAD (see page 419)
&MAICONT (see page 407)
&MAIREPL (see page 421)
&MAIDEL (see page 411)

&MAIFIND

Determines whether a data stream contains a given string.

```
&MAIFIND [ CHAR | HEX ] string
```

Determines whether the data stream currently being processed (from either the application or the terminal) contains the specified string of data.

Operands:

CHAR

Specifies that the string is provided in character format.

HEX

Specifies that the string is provided in character-format hexadecimal. The string must only contain valid hexadecimal characters (0 to 9, A to Z), and must contain an even number of characters.

string

The string for which to search. The string starts at the first non-blank character after any operand, and ends at the last non-blank character on the statement. A variable or variables is specified as the string. For hexadecimal specification, embedded blanks in the string are eliminated.

Examples: &MAIFIND

```
&MAIFIND ENTER LOGON
&MAIFIND CHAR ENTER PASSWORD
&MAIFIND HEX D9C5C1C4E8401D
```

Return Codes:

On completion the &RETCODE system variable is set as follows:

0

The string is found

4

The data stream does not contain the specified string

Notes:

An &MAIFIND issued when there is no data outstanding is ignored. Data is outstanding from the time an &MAIREAD is satisfied until an &MAICONT, &MAIDEL, or another &MAIREAD is issued.

The data stream searched for the string is the one last received by an &MAIREAD.

The data stream search is not case sensitive. The string and the data stream are always converted to uppercase before the search is performed.

More information:

&MAICONT (see page 407)
&MAIREAD (see page 419)
&MAIDEL (see page 411)

&MAIINKEY

Sets the attention key that is to be simulated in a generated data stream.

```
&MAIINKEY key
```

Sets the value of the attention key to be simulated when generating a data stream to be sent to the PLU (application).

Operands:

key

Substitute one of the following values to represent the key to be simulated:

ATTN

ATTN key

CLEAR

CLEAR key

ENTER

Enter key

PA1 to PA3

Program Attention Key 1 to 3

PF1 to PF24

Function key 1 to 24

Leading zeroes on function key values need not be specified.

Examples: &MAIINKEY

```
&MAIINKEY PF3
```

Notes:

&MAIINKEY is used to change the value of the key used to enter data at the terminal. If the script procedure receives data from the SLU and issues an &MAIINKEY before issuing &MAICONT PLU (or equivalent), the key used is changed in the data stream sent to the application.

&MAIINKEY defaults to ENTER when data from the PLU is received.

More information:

&MAICONT (see page 407)
&MAIREAD (see page 419)
&MAIDEL (see page 411)

&MAIPUT

The &MAIPUT verb builds a data stream for the PLU (application), either as an automatic reply to data received from the PLU or to replace a data stream received from the SLU (terminal) before sending it on to the PLU.

&MAIPUT fills in the input fields on the panel that is the current logical screen image. Multiple &MAIPUT statements are issued to build up the data stream. Each one supplies the contents of one input field. &MAIPUT statements are used to reply to PLU data, or to change the input from the terminal (SLU) before sending it on to the application using &MAICONT.

This verb has the following format:

```
&MAIPUT [ ROW=n ]
        [ COL=n ]
          data
```

Operands:

ROW=*n*

Indicates the screen row position of the input field. If ROW is not specified but COL is, ROW=1 is assumed.

COL=*n*

Indicates the screen column position of the input field. If COL is not specified but ROW is, COL=1 is assumed.

data

Specifies the data to be contained in the input field. The data starts at the first non-blank character after any operand, and ends at the last non-blank character on the statement. A variable or variables is specified as the data. If no data is specified, but ROW=, COL=, or both are specified, a null input field is generated, the equivalent of pressing the EOF key in an input field.

If data is specified, but both ROW= and COL= are not specified, an unformatted field is generated, consisting of only data with no Set Buffer Address sequence. This form of input results from a screen with no set input fields defined, where the whole screen is available for input.

If data, and ROW=, COL=, or both are specified, a field with a Set Buffer Address sequence followed by the data is generated.

Examples: &MAIPUT

```
&MAIPUT ROW=24 COL=6 SHOW USERS
&MAIPUT TRN1
&MAIPUT ROW=2 COL=1

&MAIPUT ROW=1 COL=1 SPF 2
&MAICONT PLU
```

Notes:

The supplied row and column values must be within the dimensions of the current screen, otherwise an error results.

An &MAICONT PLU statement (or equivalent) is used to send the generated data stream to the application. The &MAIINKEY and &MAICURSA statements are used to provide a value for the attention key to be generated and a cursor address at time of input.

&MAIPUT with no operands is used to prepare a data stream with no modified fields. In this case, you can use &MAIINKEY to specify the key to press and &MAICURSA to specify the cursor address.

An &MAIPUT issued when there is no data outstanding is normally ignored. Data is outstanding from the time an &MAIREAD is satisfied until an &MAICONT, &MAIDEL or another &MAIREAD is issued. However, in the case of a script procedure started using a .S session skip command or session cleanup, an &MAIPUT is allowed before the first &MAIREAD. An &MAIPUT PLU is also allowed after the time interval specified on an &MAIREAD WAIT= verb expires.

More information:

&MAICONT (see page 407)
&MAIREAD (see page 419)
&MAIDEL (see page 411)
&MAICURSA (see page 410)
&MAIINKEY (see page 416)

&MAIREAD

Waits for the next data stream.

```
&MAIREAD [ ANY | PLU | SLU ] [ WAIT=nnnn ]
```

Waits for the arrival of the next data stream from the PLU (application), SLU (terminal) or either, or for a specified time interval to expire.

Operands:

ANY

Suspends execution of the NCL procedure until the next data stream arrives from either the PLU or the SLU.

PLU

Suspends execution of the NCL procedure until the next data stream arrives from the PLU (application).

SLU

Suspends execution of the NCL procedure until the next data stream arrives from the SLU (terminal).

nnnn

Specifies a time interval in seconds after which the procedure will regain control if no data stream of the requested type arrives. Range is 1 to 9999.

Examples: &MAIREAD

```
&MAIREAD
&MAIREAD ANY
&MAIREAD PLU
&MAIREAD SLU WAIT=60
```

Notes:

Following the receipt of a data stream, it is searched for a particular string using &MAIFIND, sent onward using &MAICONT, or deleted using &MAIDEL.

If two &MAIREAD statements are issued without an intervening &MAICONT or &MAIDEL, an implied &MAICONT CONT VIEW=OPT is performed.

If an &MAIREAD statement specifies the PLU operand only, terminal input will bypass the procedure and be sent on to the application. Conversely, if an &MAIREAD statement specifies the SLU operand only, output from the application will bypass the procedure and be sent on to the terminal.

If any specified time interval expires before the requested data stream arrives, &MAIFRLU will be a null value; otherwise, it contains either PLU or SLU according to the source of the data stream.

More information:

&MAICONT (see page 407)
&MAIFIND (see page 414)
&MAIDEL (see page 411)
&MAIFRLU (see page 844)

&MAIREPL

Replaces a data stream destined for the terminal.

```
&MAIREPL hexadecimal data
```

Allows the script NCL procedure to provide a replacement data stream to be sent to the terminal.

Operands:

hexadecimal data

The complete data stream to be sent to the terminal, expressed in character format hexadecimal, with each two characters representing one hexadecimal byte. For instance, if a single byte of the data stream is to contain X'1D', two characters must be specified on the &MAIREPL statement, 1D. Blanks contained on the statement are eliminated, allowing the data stream to be provided from multiple variables.

Examples: &MAIREPL

```
&MAIREPL F1C21140401D60C140

&MAIREPL &1 &2 &3 &4 &5
```

Notes:

An &MAIREPL can only be used to replace a data stream received from the PLU (application). It is ignored if no PLU data stream is currently being processed.

An &MAICONT statement or another &MAIREAD statement is required to actually send the data stream on to the terminal.

More information:

&MAICONT (see page 407)
&MAIREAD (see page 419)

&MAISADD

Adds a new session definition, based on user variables.

```
&MAISADD [ PREFIX={ MAI | prefix} ]
         [ FIELDS={ name | ( name1,name2,...,namen ) } ]
```

Used to add a new session definition with the attributes specified in the FIELDS operand.

Operands:

PREFIX={ MAI | *prefix* }

Specifies the prefix for the user variables to be accessed for the session details. The default is MAI. The user variable names are formed by concatenating the PREFIX operand value with the field names specified on the FIELDS operand.

FIELDS={ *name* | *name*1,*name*2,...,*namen* }

Specifies the field names of the user variables to be accessed. The user variable names are formed by concatenating the PREFIX operand value with the field names specified on this operand. The following fields can be specified on this operand:

NODE

Specific virtual terminal

Null or 1- to 8-character name

LOGON

Logon string

1 to 50 characters

SESS

Session identifier

1 to 8 characters

FJMP1/2

Jump forward key

Null or F*nn*/PF*nn*/PA*n*/ATTN

RJMP1/2

Jump reverse key

Null or F*nn*/PF*nn*/PA*n*/ATTN

MENU1/2

Jump to menu key

Null or F*nn*/PF*nn*/PA*n*/ATTN

SISM

Screen Image Services menu key

Null or F*nn*/PF*nn*/PA*n*/ATTN

SISP

Screen print key

Null or F*nn*/PF*nn*/PA*n*/ATTN

SWAP1/2

Swap to other window key

Null or F*nn*/PF*nn*/PA*n*/ATTN

REST

Restart option

Y or N

WAIT

Wait option

Y or N

FJUMP

Fast jump option

Y or N

COMP

Compression option

Y or N

LMODE

Specific logmode

Null or 1- to 8-character name

SCTPR

Script name and option

Null or 1- to 8-character name

Return Codes:

On completion, the &RETCODE system variable is set as follows:

0

The function completed successfully.

4

An error in the attributes was detected. &SYSMSG is set and the NCL variable is updated with the ERR attribute.

&MAISCMD

Specifies an MAI session command against the current session.

```
&MAISCMD session-command
```

The current session is set by the &MAISGET verb. The values are processed in the order of the in-storage MAI selection list when the MAI procedure terminates.

Operands:

session-command

Valid session commands are:

S

The command starts the session, if not already started, and selects it for display if no other has been previously selected.

A

Activates the session if not already active.

Note: It is not selected for display.

B

Positions the session as the last in the selection list of sessions.

T

Positions the session as the first in the selection list of sessions.

C, CU,

Cancels the active session. The session failure is reported to the associated CC, CF application as either an unconditional, conditional or forced disconnection.

H

Marks the session (if active) as hidden. It will not be selected via nonspecific session jumping.

SL

Marks the session (if active) as hidden until output arrives. Upon arrival of any data stream from the associated application, the hidden status is reset and the session is then eligible for selection by nonspecific session jumping.

M

Modifies the session definition. The session is active or inactive. If the session is active certain attributes cannot be changed: for example, node-name, script procedure.

U

Updates the session definition on external storage. A full-screen function is invoked to display the current stored definition. Changes made are not reflected in the current in-storage definition. The session definition is deleted by this function.

P

Purges the definition from the in-storage session definition list. The session is no longer displayed by the MAI menu. (However, unless it is also deleted via a stored definition update, it appears when MAI is re-initialized.)

L

Logs on to an inactive session displaying the session characteristics (for modification or confirmation) before starting the session.

R

Repeats a session definition. The current session definition is used as a model to build a new session definition that is placed in the session list immediately after the one selected. The session ID of the new session definition is set to blanks.

&MAISGET

The &MAISGET verb retrieves details of the specified session into user variables.

Use the verb to access the attributes of session list entries or the defaults. You can also use the verb to mark a session as the current session for &MAISCMD.

This verb has the following format:

```
&MAISGET { ID=sessid | RELNUM=nn | DEFAULTS }
         [ PREFIX={ MAI | prefix} ]
         [ ATTR={ ANY | LIST } ]
         [ FIELDS={ name | ( name1,name2,...,namen ) } ]
         [ PAD ] }
```

Operands:

ID=*sessid*

Specifies the one- to eight-character session identifier that uniquely identifies a defined MAI-FS session to access.

RELNUM=*nn*

Specifies the numerical position of a relative session within the user's session list to access.

DEFAULTS

Retrieves the default MAI-FS session characteristics.

PREFIX={ MAI | *prefix* }

(Optional) Specifies the prefix for the user variables to access. The user variable names are formed by concatenating the PREFIX operand value with the field names specified on the FIELDS operand.

Default: MAI

ATTR={ ANY | LIST }

(Optional) Specifies whether the attributes of the active session are returned, or those attributes of the session list entry only. ATTR=LIST returns the attribute values of the session list entry. If an active session is available, ATTR=ANY returns the attribute values of the session; otherwise, the attributes of the session list entry are returned.

FIELDS={ *name* | *name*1,*name*2,...,*namen* }

(Optional) Specifies the field names of the user variables to return. The user variable names are formed by concatenating the PREFIX operand value with the field names specified on this operand. The following table shows the fields that are specified on this operand:

Field Name	Description	Values Returned
DESC	Session description	String
DEFD	DEFLOGON description	String or N53D03 message
SSTAT	Session status	RUNNING, WAITING, ENDED, <->
SHIDE	Session visibility	HIDDEN, SLEEPING, null
SOUTW	Session output status	*OUTPUT*, null
SWNDO	Active window of session	1, 2,- (- meaning not active)
NODE	Specific virtual terminal	Null or 1- to 8-character name
LOGON	Logon string	1 through 50 characters
SESS	Session identifier	1 through 8 characters
FJMP1/2	Jump forward key	Null or F*nn*/PF*nn*/PA*n*/ATTN
RJMP1/2	Jump reverse key	Null or F*nn*/PF*nn*/PA*n*/ATTN
MENU1/2	Jump to menu key	Null or F*nn*/PF*nn*/PA*n*/ATTN
SISM	Screen Image Services menu key	Null or F*nn*/PF*nn*/PA*n*/ATTN
SISP	Screen print key	Null or F*nn*/PF*nn*/PA*n*/ATTN
SWAP1/2	Swap to other window key	Null or F*nn*/PF*nn*/PA*n*/ATTN
REST	Restart option	Y or N
WAIT	Wait option	Y or N
FJUMP	Fast jump option	Y or N
COMP	Compression option	Y or N
LMODE	Specific logmode	Null or 1- to 8-character name
SCTPR	Script name and option	Null or 1- to 8-character name

PAD

(Optional) Returns the fields with their default maximum length.

Return Codes:

On completion, the &RETCODE system variable is set as follows:

0

The function completed successfully and the variables are set.

4

An error in the request was detected. &SYSMSG is set to indicate the error condition. No user variables are updated.

Note:

When the verb accesses a session list entry, the session is marked as the current session. A subsequent &MAISCMD verb affects this session.

&MAISPUT

The &MAISPUT verb updates MAI session list entries, or the defaults.

This verb has the following format:

```
&MAISPUT { ID=sessid | RELNUM=nn | DEFAULTS }
         [ PREFIX={ MAI | prefix} ]
         [ FIELDS={ name | ( name1,name2,...,namen ) } ]
```

Operands:

ID=*sessid*

Specifies the one- to eight-character session identifier that uniquely identifies a defined MAI-FS session to update.

RELNUM=*nn*

Specifies the numerical position of a relative session within the defined list to update.

DEFAULTS

Updates the default session details.

PREFIX={ MAI | *prefix* }

Specifies the prefix for the user variables to update. The user variable names are formed by concatenating the PREFIX operand value with the field names specified on the FIELDS operand.

Default: MAI

FIELDS={ *name* | *name*1,*name*2,...,*namen* }

Specifies the field names of the user variables to update. The user variable names are formed by concatenating the PREFIX operand value with the field names specified on this operand. The following table shows the fields that are specified on this operand:

Field Name	Description	Value
NODE	Specific virtual terminal	Null or 1-to 8-character name
LOGON	Logon string	1 through 50 characters
SESS	Session identifier	1 through 8 characters
FJMP1/2	Jump forward key	Null or F*nn*/PF*nn*/PA*n*/ATTN
RJMP1/2	Jump reverse key	Null or F*nn*/PF*nn*/PA*n*/ATTN
MENU1/2	Jump to menu key	Null or F*nn*/PF*nn*/PA*n*/ATTN
SISM	Seen Image Services menu key	Null or F*nn*/PF*nn*/PA*n*/ATTN

Field Name	Description	Value
SISP	Screen print key	Null or F*nn*/PF*nn*/PA*n*/ATTN
SWAP1/2	Swap to other window key	Null or F*nn*/PF*nn*/PA*n*/ATTN
REST	Restart option	Y or N
WAIT	Wait option	Y or N
FJUMP	Fast jump option	Y or N
COMP	Compression option	Y or N
LMODE	Specific logmode	Null or 1-to 8-character name
SCTPR	Script name and option	Null or 1-to 8-character name

Return Codes:

On completion, the &RETCODE system variable is set as follows:

0

The function completed successfully.

4

An error in the attributes was detected. &SYSMSG is set and the NCL variable is updated with the ERR attribute.

&MASKCHK

Returns a string that indicates the matching of a value against a nominated wildcard mask.

```
&MASKCHK mask data [ wildcard | * ]
```

&MASKCHK is a built-in function and must be used to the right of an assignment statement.

&MASKCHK is used to perform generic pattern matching. The mask is a string of characters containing wildcard characters, by default asterisks (*), which imply that any character is accepted in positions occupied by wildcard characters, when testing the data string.

This provides a rapid way of determining if the target data is within an acceptable range, or matches specific selection criteria.

Testing the mask against data is performed on a character by character basis moving from left to right. After this comparison, one of the following values is returned:

EQ

The data matches the supplied mask and is the same length as the mask.

EQL

The data matches the supplied mask, but is longer than the mask. This implies the supplied mask contains a wildcard character as its last character.

EQS

The data matches the supplied mask, but is shorter than the mask.

NE

The data does not match the supplied mask.

Operands:

mask

A selection mask from 1 to 256 characters in length containing one or more wildcard characters. By default the wildcard character is an asterisk (*), but an alternative character is specified. A wildcard character within mask implies that any value in that data position is accepted. If the last character of mask is a wildcard character, then no further data checking is performed and any trailing characters are accepted.

data

1 to 256 characters of target data to be scanned to see if it matches the selection criteria established by mask.

***wildcard* | ***

A single character regarded as a wildcard character. By default an asterisk is used. Multiple occurrences of this character, in any position of mask is acceptable.

Examples: &MASKCHK

```
&A = &MASKCHK IST* &MSGID
&IF &A = EQ &THEN +
   &GOSUB .PROCESSIST

&A = &MASKCHK ABC* ABCD   -* &A will be set to EQ
&A = &MASKCHK A*C* ABCD   -* &A will be set to EQ
&A = &MASKCHK %%C% A*CD % -* &A will be set to EQ
&A = &MASKCHK ABC* ABCDE  -* &A will be set to EQL
&A = &MASKCHK ABC* ABC    - * &A will be set to EQS
&A = &MASKCHK ABC* DEF    -* &A will be set to NE
```

Note:

When using &MASKCHK, a procedure must allow for when a value of EQ, EQL, or EQS is returned. In such cases the following approach is used:

```
&A = &MASKCHK P* &TERM &IF &A NE NE &THEN &WRITE Mask matched successfully
```

&MSGCONT

Resumes normal processing of a message delivered to MSGPROC.

```
&MSGCONT[ COLOR=color | COLOUR=colour ]
        [ HLIGHT=highlight | HLITE=highlight ]
        [ INTENS={ HIGH | NORMAL } ]
        [ ALARM={ YES | NO } ]
        [ NRD={ NO | OPER } ]
```

Used within a MSGPROC procedure to request that a message previously delivered for processing by an &MSGREAD be returned for standard message processing. Optional attributes of the message is modified.

Operands:

COLOR=*color* | COLOUR=*colour*

Specifies that the message color is to be changed to that selected. Valid values are:

RED GREEN BLUE TURQUOISE YELLOW PINK WHITE NONE

HLIGHT=*highlight* | HLITE=*highlight*

Specifies the message highlight option to be used. Valid values are:

REVERSE BLINK USCORE NONE

INTENS={ HIGH | NORMAL }

Specifies the message display intensity to be used.

ALARM={ YES | NO }

Specifies whether the terminal alarm is sounded when the message is delivered to an OCS screen.

NRD={ NO | OPER }

If the non-roll delete attribute of the message must be changed, use this operand to specify the new attribute value. NRD=YES is ignored; full non-roll delete with DOM correlation can only be set by the message originator, for example, &WRITE.

Examples: &MSGCONT

```
&IF &ZMMONMSG = YES &THEN +
   &MSGCONT COLOR=RED HLIGHT=REVERSE
```

Notes:

When an &MSGCONT verb is issued and no message is currently being processed by MSGPROC, it is ignored.

An &MSGCONT need not be issued if another &MSGREAD is to be issued. If a message is being processed and another &MSGREAD is issued without an intervening &MSGDEL or &MSGREPL, an implied &MSGCONT is performed and the message is returned for standard processing before the next &MSGREAD is satisfied.

&MSGCONT is used to free a message for delivery while the MSGPROC procedure continues processing before issuing another &MSGREAD.

More information:

&MSGREAD (see page 435)
&MSGDEL (see page 434)
&MSGREPL (see page 440)

&MSGDEL

Deletes the message currently being processed by MSGPROC.

```
&MSGDEL
```

Used within a MSGPROC procedure to request that the message previously delivered for processing by an &MSGREAD be deleted. Once deleted, the message is not available for further processing.

Examples: &MSGDEL

```
&IF &ZMSOLIC = NO &THEN +
   &MSGDEL
```

Note:

An &MSGDEL issued when no message is available is ignored.

More information:

&MSGREAD (see page 435)
&MSGCONT (see page 433)
&MSGREPL (see page 440)

&MSGREAD

Requests that the next message be made available to MSGPROC.

```
&MSGREAD { SET |  VARS=prefix* [ RANGE=(start,end) ] |
           VARS={ var | ( var1, var2, ..., varn ) } |
           STRING= ( name, name, ..., name) |
           ARGS  [ RANGE=(start,end) ] |
           MDO=mdoname }
         [ WAIT={ YES | NO | nnnn.nn } ]
         [ TYPE={ ALL  | SOLICIT | UNSOLICIT } ]
         [ DOM={ YES | NO } ]
```

Used within a MSGPROC procedure to request delivery of the next message. If no message is available immediately, procedure processing is suspended at this point and resumes when the next message to be sent to the user's terminal arrives.

Multiple &MSGREAD statements is present within a MSGPROC, to make the processing of group messages easier.

On completing &MSGREAD, system variable &ZVARCNT is set to the number of variables created or modified by the operation.

The profile of the message received by &MSGREAD is set in a suite of reserved system variables. The message profile (which includes attributes such as color, highlighting, and source information) provides a complete description of all the message attributes in addition to the message text.

When &MSGREAD completes, the system variable &ZFDBK is set as follows:

0

Message has been received.

4

Wait time has expired.

Operands:

SET

Specifies that no message tokenization is performed; the &MSGREAD statement returns only the message profile of the next message.

If SET is not specified, operands must be coded on the &MSGREAD statement specifying the tokenization requirements for the message text by using other &MSGREAD operands.

VARS=*prefix [RANGE=(*start*,*end*)] |**
VARS={ *var* | (*var1*, *var2*, ..., *varn*) }

Specifies the message is to be tokenized into variables from left to right before control is returned to the procedure. If insufficient variables are nominated, some data is lost. Excess variables are set to null. The formats of the operands that is coded with VARS= are described below:

prefix*

Denotes that variables are generated automatically during tokenization, with variable names *prefix1*, *prefix2*, and soon. (The RANGE= operand is specified to indicate a starting and ending suffix number). This format cannot be used with other variable names.

var

The name of a variable, excluding the ampersand (&).

***var*(*n*)**

As *var*, but *n* denotes the length for the data that is put into the variable.

***(*n*)**

Denotes a skip operation, where *n* is the number of units to be skipped during tokenization. On VARS= statements, *n* denotes 'skip n words'. An asterisk (*) by itself is the same as *(1).

STRING=

Specifies no tokenization is performed. The entire text of the message is treated as a single string and returned to the procedure in the nominated variables. Formats for operands associated with STRING are:

name

The user-specified variables, excluding the leading ampersand (&), where string text is to be placed. Text is put into each variable up to the maximum length for that variable.

***name*(*n*)**

User specified variables, excluding the leading ampersand *, where string text is put. Text will be placed into each variable for the length specified by *n*.

***(*n*)**

Denotes a skip operation, where n represents the number of units to be skipped during the tokenization process. On STRING statements, *n* denotes 'skip *n* characters'. An asterisk (*) by itself is the same as *(1).

ARGS [RANGE=(*start*, *end*)]

Denotes that the line of text retrieved is tokenized and placed word-by-word into automatically generated variables with the form &1 through &*n*, depending on how many are required to hold the text. The RANGE operand option is coded to designate a start and an end value to delimit the number of variables generated.

MDO=*mdoname*

Specifies that, if an embedded MDO is present in the message received by &MSGREAD, it will be assigned to an object with the name specified in *mdoname*.

WAIT={ YES | NO | *nnnn.nn* }

Specifies the action &MSGREAD processing takes if no message is available for processing immediately. Code WAIT=NO if you want control returned immediately to the statement after &MSGREAD when no incoming message is available. Code WAIT=*nnnn.nn* to specify the number of seconds up to which &MSGREAD waits for a message, before returning control to the procedure (maximum is 9999.99 seconds). Code WAIT=YES or omit the operand (it will default to YES), if &MSGREAD is to wait indefinitely for a message to arrive. Coding WAIT=0 has the same effect as WAIT=NO.

TYPE={ ALL | SOLICIT | UNSOLICIT }

Indicates whether &MSGREAD filters solicited or unsolicited messages. Default is ALL, which means that all messages satisfy &MSGREAD. Code SOLICIT if you want to receive solicited messages only.

DOM={ YES | NO }

Specifies whether receipt of a delete operator message (DOM) instruction can complete an &MSGREAD operation. If MSGPROC wants to know about the flow of DOMs to the user's window, code DOM=YES. When a DOM completes &MSGREAD, message profile system variables will indicate that a DOM (and not a real message) has been received.

Examples: &MSGREAD

```
&MSGREAD WAIT=5 ARGS RANGE=(20,80)
```

This requests the first message from the message queue, specifying that it is to be tokenized into variables &20 up to a maximum of &80 (&ZVARCNT is set to total how many variables were created). If no message is immediately available, control returns after 5 seconds.

```
&MSGREAD VARS=(*(3),A(2),B(3),C,D,E,F)
```

This example reads the next message from the message queue and tokenizes it into individual words. *(3) indicates that the first 3 words are ignored, 2 characters of the next word are placed in the variable &A, three characters of the next word are placed in the variable &B and the next 4 words are placed in variables &C, &D, &E and &F respectively.

```
&MSGREAD STRING=(A,B(2),*(5),C(3))
```

Reads the next message from the message queue as a single string of text. The first 256 bytes are placed in &A, the next 2 characters are placed in &B, the next 5 characters are ignored and the next 3 characters are placed in &C.

```
&MSGREAD MDO=BRUCE
```

Reads the next message from the message queue and, if the message contains a user MDO, assigns the embedded MDO into an object called BRUCE and mapped by &ZMAPNAME.

Notes:

After a successful &MSGREAD, a $MSG MDO will always be available to the MSGPROC. $MSG is mapped my $MSG, and contains all attributes of the message. If the message is an envelope for a user MDO, the user MDO will be automatically assigned the name specified on the MDO= operand. It is also available directly from the $MSG.USERMDO component and its map name is in $MSG.MAPNAME or &ZMAPNAME.

After an &MSGREAD a useful technique is to use an &GOTO statement for the routine that will process the message, using the first token of the message (normally the message number).

```
&CONTROL NOLABEL
.READ
&MSGREAD ARGS
GOTO .&1
&MSGCONT     -* Unexpected messages will be
&GOTO .READ  -* caught and returned for normal processing

.msg1
.msg2
-*
-* Processing
-*...
&GOTO .READ
```

While testing and developing a MSGPROC procedure you might need to terminate the current version and invoke an updated copy. MSGPROC is invoked on entering OCS Mode. Exiting OCS flushes the current MSGPROC. Subsequent re-entry to OCS loads the latest copy of the procedure (unless it was preloaded, when the preloaded copy is used).

The PROFILE MSGPROC=FLUSH command is used to terminate an active MSGPROC process. PROFILE MSGPROC=*name* can then be used to initiate a new MSGPROC process.

If MSGPROC terminates for any reason, standard message processing resumes.

MDO assignment which occurs during &MSGREAD processing results in the setting of &ZMDORC and &ZMDOFDBK system variables.

More information:

&MSGCONT (see page 433)
&MSGDEL (see page 434)
&MSGREPL (see page 440)

&MSGREPL

Replaces the text of a message delivered to MSGPROC.

```
&MSGREPL [ COLOR=color | COLOUR=colour ]
         [ HLIGHT=highlight | HLITE=highlight ]
         [ INTENS={ HIGH | NORMAL } ]
         [ ALARM={ YES | NO } ]
         [ SCAN={ YES | NO } ]
         [ NRD={ NO | OPER } ]
         [ DATA=replacement text ]
```

Used within the MSGPROC procedure to request that a message previously delivered for processing by an &MSGREAD be changed to the supplied text and returned for standard delivery.

Operands:

COLOR=*color* | COLOUR=*colour*

Specifies that the message color is to change to the selected color. Valid values are:

RED GREEN BLUE TURQUOISE YELLOW PINK WHITE

LIGHT=*highlight* | HLITE=*highlight*

Specifies the message highlight option is used. Valid values are:

REVERSE BLINK USCORE NONE

INTENS={ HIGH | NORMAL }

Specifies the message display intensity is used.

ALARM={ YES | NO }

Specifies whether the terminal alarm is sounded when a message is delivered.

SCAN={ YES | NO }

Specifies whether the text contains at sign (@) word highlighting characters for processing.

NRD={ NO | OPER }

If the non-roll delete attribute of the &MSGREAD message must be changed, use this operand to specify the new attribute. The existing NRD attribute of the message is tested by examining the &ZMNRD message profile variable set after executing &MSGREAD. NRD=YES is specified but is ignored; full non-roll delete with DOM correlation can only be set by the message originator, for example, &WRITE.

DATA=*replacement text*

The full text for the replacement message, including any message numbers required. Text is either upper or lower case.

The maximum length for replacement text is 256 characters.

If no text is supplied the request is treated as an &MSGDEL. DATA can only be specified as the last keyword on the statement since the data string is regarded as being everything right of the DATA= keyword, to the end of the statement.

Examples: &MSGREPL

```
&MSGREPL HLIGHT=BLINK DATA=NETWORK NODE &7 INACTIVE.
```

Notes:

If &MSGREPL is issued while no message is being processed, it is ignored.

After issuing &MSGREPL, the message is no longer available in its original format and &MSGCONT need not be issued to return the message for normal processing.

An &MSGREPL is followed immediately by an &MSGREAD to make the next message available.

More information:

&MSGREAD (see page 435)
&MSGDEL (see page 434)
&MSGCONT (see page 433)

&NBLSTR

Returns a string with leading and trailing blanks removed.

```
&NBLSTR string
```

&NBLSTR is a built-in function and must be used to the right of an assignment statement.

User variables may contain leading or trailing blanks entered by an operator during full-screen processing, or from other built-in functions such as &SUBSTR, &ASISTR, and &SETLENG.

&NBLSTR removes any leading and trailing blanks from data and returns this string.

If the data consists entirely of blanks, the target variable is set to null.

Operands:

string

Data or a variable with data from which both leading and trailing blanks are to be removed.

Examples: &NBLSTR

```
&CMD = &NBLSTR &COMMAND
&1 = &NBLSTR &1
```

To remove only leading blanks, use the &LBLSTR function. To remove only trailing blanks, use the &TBLSTR function.

More information:

&LBLSTR (see page 386)
&TBLSTR (see page 651)
&ASISTR (see page 205)

Free-form Syntax

Several of the NDB verbs, which are described on the following pages, use a special syntax, which is different from normal NCL syntax, to allow easy coding of data definitions and scan requests. The relevant verb descriptions indicate the part that uses this free-form syntax. The free-form part must always be coded after any fixed-form, standard NCL-syntax operands on the same statement.

The rules for free-form syntax are as follows:

- Blanks are only required to delimit adjacent words, except inside data values. Blanks are not required, but is specified, around or next to special characters (listed below) that act as delimiters.

  Blanks inside data values are significant, except that trailing blanks are never stored in character-format data.

  Notes: NCL variables with blanks in the value are regarded as a special case, and the blank is regarded as part of the data value. This is because blanks inside NCL variables are represented internally in a special way.

- The following special characters act as delimiters, unless enclosed in a quoted string. They have special meaning to the syntax:

  (

  Left bracket

)

  Right bracket

  :

  Colon (meaning range)

  =

  Equal sign

  p

  Not sign

  <

  Less than sign

  >

  Greater than sign

  &

  Ampersand (meaning AND)

  |

  Bar (meaning OR)

  ,

  Comma

  Real blank (not embedded in an NCL variable)

Certain combinations of these characters are treated as a single token for parsing. These combinations are p=, <=, and >=, meaning not equal, less than or equal, and greater than or equal respectively.

- Values is enclosed in quoted strings whenever the value might contain a special character, or a real blank.

  The quotes may be single (') or double(”). If the data value being quoted contains a single or double quote, you can quote the data with the other quote, or double up each occurrence of the quote character.

  For example, *'This''s a quoted value'* will be regarded as the value *This's a quoted value*.

  The &NDBQUOTE built-in function provides an easy way to automatically quote data when necessary.

  A data value can always be quoted, even date, hexadecimal, or numeric values. Quoting also prevents any possibility of the value being regarded as a keyword.

- The following words is used instead of special characters, to aid clarity. If surrounded by other words, ensure at least one blank separates them.

  EQ (=)

  NE (p=)

  LT (<)

  GT (>)

  LE (< or =)

  GE (> or =)

  AND (&)

  OR (|)

  TO (:)

  NOT (p)

- Several statements support a START/DATA/END construct, that allows free-form expressions to be constructed that are longer than a single NCL statement is allowed to be. These statements is coded in the following way if the free-form text will fit on one statement (with possible continuations):

  &NDB*xxxx dbname* [*operands*] [DATA] *free-form text*

  To overcome NCL statement length limitations, and also to allow the free-form text to be built piece-meal (for example, by indirect variable reference), the statements can also be coded as:

  &NDB*xxxx dbname* [*operands*] START

  &NDB*xxxx dbname* [DATA] *part-of-free-form-text*

  &NDB*xxxx dbname* END

  The free-form text is broken anywhere a blank is valid. Any number of intermediate statements is used to build the complete free-form text. The database is not called until the END statement is encountered.

  Any other operands must be coded on the &NDB*xxx* START statement.

  Note: The statements can be interspersed with other NCL statements, including statements referencing other or even the same database, and even statements building free-form text for the same database, as long as they are different statements. That is, you is concurrently building a multi-statement add and update for the same database, but not two different adds for the same database.

  To cancel a partially built statement,

  &NDB*xxx dbname* CANCEL

  This statement is valid even if no current START/END set is being built (thus it is used in general error routines).

&NDBADD

Adds a record to a NetMaster database (NDB).

```
&NDBADD dbname { [ DATA ] add-text | START | END | CANCEL |
                   FORMAT=fmtname [ FSCOPE={ PROCESS | GLOBAL } ] }
```

where *add-text* is:

```
fieldname = fieldvalue [ fieldname = fieldvalue ... ]
```

The &NDBADD statement allows an NCL procedure to add a new record to an NDB. The new record contains the fields listed on the &NDBADD statement, with the listed values. Following completion of the statement, the system variable &NDBRC will indicate success or failure, and, if successful, the system variable &NDBRID will have the record ID of the new record.

Operands:

dbname

The name of the NDB that you wish to add a record to is a required operand. The NDB named must have been previously opened by an &NDBOPEN statement.

DATA

Indicates that free-form text (see page 442) follows. This operand is optional, but it is recommended, as it prevents any ambiguous meaning of a field name or field value of DATA, START, END, or CANCEL.

START

Indicates the start of a multi-statement &NDBADD. The statement must end after the START keyword.

END

Indicates the end of a multi-statement &NDBADD. This statement will call the database, passing the concatenated &NDBADD DATA information.

CANCEL

Indicates an active &NDBADD START/END set is to be canceled. If there is no active &NDBADD START/END for this database, the statement is ignored.

FORMAT=*fmtname* [FSCOPE={ PROCESS | GLOBAL }]

FORMAT=*fmtname* specifies that the output format *fmtname*, defined on the &NDBFMT statement, is to be used.

The nominated format must exist in the nominated scope. PROCESS is the default and means a format defined by the current NCL process. GLOBAL indicates a format is to be found in the global format pool for the NDB.

If this operand is specified, the START, DATA, CANCEL, and END operands are not allowed.

fieldname = fieldvalue ...

Free-form text naming the fields to be given values in the new record, and the values to be placed in those fields.

There may be as many name = value pairs as desired, and they may be split across multiple statements as described in the front of the chapter, using START/DATA.../END.

If *fieldvalue* is a null variable (for example, &NULL =/ &NDBADD ... X = &NULL), the null variable will be passed to the database as a null indicator, indicating the relevant field is not present. This is not the same as present, with a null value (for example, 0 for a numeric field).

The following code sets FIELD1 to a value, FIELD2 to present, with a null value, and FIELD3 not present:

```
&FIELD1 = value      -* set to a value
&FIELD2 = &SETBLNK 1 -* set to a blank
&FIELD3 =            -* set null
&NDBADD MYNDB DATA FIELD1 = &FIELD1 FIELD2 =+
            &FIELD2 FIELD3 = &FIELD3
```

Note: The omission of a field name and its accompanying value also sets that field as 'not present' in the new record.

Examples: &NDBADD

The following example will add a record to the NDB called MYNDB, setting the field SURNAME to the value BLOGGS, and the field FIRSTNAME to the value FRED.

```
&NDBADD MYNDB DATA SURNAME='BLOGGS' FIRSTNAME='FRED'
```

The next example adds a new record to the NDB called MYNDB, and builds the list of fields across multiple statements. It illustrates how the free-form text is split at any point where a blank is valid.

```
&NDBADD MYNDB START
&NDBADD MYNDB DATA SURNAME = JONES
&NDBADD MYNDB DATA DOB =
&NDBADD MYNDB DATA 600101  FIRSTNAME = 'JOHN'
&NDBADD MYNDB END
```

Notes:

Errors encountered whilst processing the &NDBADD statement may cause the procedure to terminate, or may just be reflected in the &NDBRC system variable, depending on the setting of &NDBCTL ERROR option.

If the record is added successfully (&NDBRC is 0 after the single-statement &NDBADD, or after the &NDBADD END for a multiple-statement add), the system variable &NDBRID will contain the assigned record ID of the new record.

At least one *fieldname* = *fieldvalue* must be specified to successfully add a record (although the value may be the null indicator).

More information:

&NDBUPD (see page 513)
&NDBDEL (see page 462)

&NDBCLOSE

Signs off (disconnects) from an NDB database.

```
&NDBCLOSE dbname
```

The &NDBCLOSE statement is used to terminate the connection an NCL procedure has with an NDB. All current formats, sequences, scan lists an so on, are deleted. If further access to the database is desired, the procedure must issue an &NDBOPEN (see page 486) statement.

Operands:

dbname

The name of the NDB from which you want to sign off is a required operand. If not signed on to this NDB, an error response is given, that might cause the procedure to terminate, depending on the &NDBCTL ERROR= setting.

Examples: &NDBCLOSE

```
&NDBCLOSE MYNDB
```

This example signs off from the NDB called MYNDB. Any defined formats and sequences are deleted.

Notes:

An implicit &NDBCLOSE is done at NCL procedure termination for all open NDBs. (This is only at highest-level termination, not nested EXECs.)

Any active &NDB*xxx* START/DATA/END statements for this database are terminated, as if *&NDBxxx* CANCEL was specified.

If the procedure is not currently signed on the NDB, response 35 is returned, and the procedure is terminated if the current &NDBCTL ERROR setting is ABORT.

If the database was not started, or is in stopping status, and this is the last user signing off, the database stops, and may enter the LOCKED status.

Note: See also the NDB START and NDB STOP commands in the Online Help.

&NDBCTL

Alters NDB NCL processing characteristics.

```
&NDBCTL [ QUOTE={NO | YES }
        [ DATEFMT={ * | NO | 1 | DATE1 | 2 | DATE2 |
          3 | DATE3 | 4 | DATE4 | UK | 5 | DATE5 | US |
          6 | DATE6 | 7 | DATE7 | 8 | DATE8 | 9 | DATE9 |
          10 | DATE10 } ]
        [ TIMEZONE={ SYSTEM | USER | * | shhmm } ]
        [ ERROR={ ABORT | CONTINUE } ]
        [ TRACE={ NO | YES } ]
        [ MSG={ YES | NO | LOG } ]
        [ SCANDEBUG={ NO | YES } ]
```

The &NDBCTL statement allows an NCL procedure to control several processing options related to NDBs. For example, the procedure may retain control after a database error.

If the &NDBCTL statement is coded with no operands, the effect is to set all options to the default values underlined. If the &NDBCTL statement has any operands specified, only the specified operands are changed.

Operands:

QUOTE={ NO | YES }

Controls whether or not values for character, hexadecimal (FMT=HEX), hexadecimal number (FMT=NUM BASE=HEX), and date format data must be quoted.

QUOTE=NO, the default, means that these data types need not be quoted, and embedded blanks in NCL variables in free-format text are treated as part of the surrounding word. Also, &NDBQUOTE does not force-quote non-null data.

QUOTE=YES forces the quoting requirement for the listed data types, causes embedded blanks to be treated as real blanks, and causes &NDBQUOTE to force-quote all non-null data.

DATEFMT={ * | NO | 1 | DATE1 | 2 | DATE2 | 3 | DATE3 | 4 | DATE4 | UK | 5 | DATE5 | US | 6 | DATE6 | 7 | DATE7 | 8 | DATE8 | 9 | DATE9 | 10 | DATE10 }

Sets the expected input format and used output format for the DATE and CDATE fields.

DATEFMT=* (the default) means that date data will be accepted in the format that the current user is profiled with, either UK or US format, or, if blank or neither of these, the system language (either UK or US). Returned date data will also be in this format.

DATEFMT=NO, for DATE fields, means that date data is accepted in YYMMDD format only, and returned in this format. DATEFMT=NO for CDATE fields means that a 6-digit number is processed as YYMMDD (DATE7) and an 8-digit number is processed as YYYYMMDD (DATE8).

The formats for DATE1 through DATE10 are as follows:

- 1 or DATE1 = YY.DDD
- 2 or DATE2 = DAY DD-MON-YEAR
- 3 or DATE3 = DD-MON-YEAR
- 4 or DATE4 or UK = DD/MM/YY
- 5 or DATE5 or US = MM/DD/YY
- 6 or DATE6 = YY/MM/DD
- 7 or DATE7 = YYMMDD
- 8 or DATE8 = YYYYYMMDD
- 9 or DATE9 = *n* (number, base day)
- 10 or DATE10 = YYYYMMDD0000+0000

TIMEZONE={ SYSTEM | USER | * | *shhmm* }

Allows a Greenwich Mean Time (GMT) offset value to be nominated.

TIMEZONE=SYSTEM specifies that the system offset is used.

TIMEZONE=USER specifies that the user time zone is used if it is set, otherwise no time zone is used.

TIMEZONE=* specifies that the user time zone is used if it is set, otherwise the system offset is used.

TIMEZONE=*shhmm* specifies a signed GMT offset value, where *s* is the sign, *hh* is hours, and *mm* is minutes

Note: For a signed-on user, other than EASINET, a value is always set at signon. If the user does not have a time zone defined in UAMS, the system offset is used.

When a time zone offset is set using the TIMEZONE operand, any and all TIMESTAMP fields specified on an &NDBADD, &NDBUPD, or &NDBSCAN are adjusted using this offset. This allows the fields on the NDB to be normalized to GMT regardless of where a user is situated. Only TIMESTAMP fields is altered, as the date might need to be changed when the time is altered-the user's time zone is subtracted from the timestamp.

When retrieving fields, all TIMESTAMP fields are adjusted in reverse to convert from GMT time to the user's time; that is, the user time zone is added to the timestamp.

Passing the TIMESTAMP with a trailing Z, which signifies GMT, suppresses alteration of the fields.

SCANDEBUG output will show the GMT timestamp, followed by a Z.

ERROR={ ABORT | CONTINUE }

Sets the processing option related to NDB error handling.

ERROR=ABORT (the default) means that any database errors cause the procedure to terminate with an error message. Database errors are defined as any database request that gets a return code (in &NDBRC) greater than 29.

ERROR=CONTINUE allows the procedure to retain control after an error, with &NDBRC giving the error return code.

Syntax errors in the &NDB*xxx* statements themselves always cause the procedure to terminate (for example, &NDBCTL ERROR=*xxx* causes the procedure to terminate). However, syntax errors in free-form text are returned as a database error.

TRACE={ NO | YES }

Allows display of the tokenizing of free-form text when processing requests allow free-form text.

TRACE=NO (the default) prevents the display of each distinct token in the free-form text.

TRACE=YES will produce this message for each input token as the free-form text is parsed by the database manager:

```
N87710 TOKEN: token-value (maximum 1st 20 characters)
```

This is useful in identifying the exact point an error is occurring in a free-form text statement.

MSG={ YES | NO | LOG }

Controls the issuing of error messages by the database manager.

MSG=YES (the default) means that error messages are sent to the environment the NCL procedure is running in (typically, the OCS window). They will also be logged.

MSG=NO means that no messages are issued, except certain messages are always forced out (for example, database corruption messages and the trace message).

MSG=LOG means that the error messages will only be sent to the activity log.

SCANDEBUG={ NO | YES }

Controls whether &NDBSCAN statements generate debugging messages showing the parsed scan-Request, and messages showing the record counts passing each level of the scan.

SCANDEBUG=NO (the default) specifies that no debugging information is displayed. SCANDEBUG=YES means that debugging information is produced.

Note: Irrespective of the DATEFMT setting, date data can always be entered in YYMMDD format.

The delimiters '/' is any of the characters ' / \ . , : ; - _ '. Some of these require the date to be quoted on input. Returned dates, however, always use '/'.

Examples: &NDBCTL

```
&NDBCTL DATEFMT=NO ERROR=CONTINUE
```

This example sets the acceptable date format to YYMMDD only, and allows the procedure to stay in control after an error.

```
&NDBCTL
```

This example resets all &NDBCTL options to their default values.

Notes:

Syntax errors in the &NDBCTL statement always cause the NCL procedure to terminate with an error message.

The &NDBCTL values are saved/restored based on the setting of &CONTROL SAVE/NOSAVE.

More information:

&NDBRC (see page 853)
&NDBERRI (see page 852)
&DATEn (see page 824)
&CONTROL (see page 266)

&NDBDEF

The &NDBDEF verb adds, updates, or deletes field definitions to/from an NDB database. This feature is described for completeness. Use the NDB FIELD command (which provides a better way to perform these functions) instead.

This verb has the following format:

```
&NDBDEF dbname { [ ADD ] field-entry | UPDATE field-entry | DELETE field-entry }
```

where *field-entry* is (minimum acceptable abbreviations are in uppercase):

```
{ fieldname | ( fieldname
              [ { Fmt= | Format= } { Char | Num | Float | Hex | X | Date |
                                     Cdate | Time | Timestamp } ]
              [ Key={ No | Yes | Unique | Sequence } ]
              [ NULLField={ Yes | No } ]
              [ NULLValue={ Yes | No } ]
              [ Update={ Yes | No } ]
              [ Caps={ Yes | No | Search } ]
              [ Description=description ]
              [ { USER1= | U1= } value ]
              [ { USER2= | U2= } value ]
              [ { USER3= | U3= } value ]
              [ { USER4= | U4= } value ]
              [ NEWNAME=name ]
              [ BASE={ NONE | DECIMAL | HEX } ] ) }
```

Note: Specifying KEY=SEQUENCE changes the default of NULLFIELD=YES to NULLFIELD=NO, and UPDATE=YES to UPDATE=NO. Specifying NULLFIELD=YES or UPDATE=YES is invalid in this case. The default values shown apply for ADD only. For UPDATE, all operands that are not specified for the field name on the &NDBDEF statement, remain unchanged.

When an NDB is created (using the NDB CREATE command), there are no field definitions. Use the &NDBDEF ADD statement to insert at least one field definition into the database.

Field definitions can also be removed from an NDB. When field definitions are removed, any index tables associated with the field are removed, and the field data in any record becomes inaccessible. Whenever a record is updated, the deleted fields are removed from that record.

Operands:

dbname

Specifies the name of the NDB that you want to add or delete field definitions in is a required operand. An &NDBOPEN statement must have opened this NDB previously.

ADD

(Optional) Adds field definitions to the database.

At least one field definition must follow the ADD keyword.

UPDATE

Updates the listed fields.

At least one field definition must follow the UPDATE keyword.

The UPDATE option allows the following attributes to be updated at any time:

- DESC
- USER1 to 4
- UPDATE (except sequence key)
- KEY=Y to KEY=N
- KEY=U to KEY=Y/N

DELETE

Deletes the listed fields from the database.

At least one field definition must follow the DELETE keyword.

field-entry

Specifies the field to add or delete.

More than one field is added or deleted in one &NDBDEF statement, up to the maximum permissible NCL statement continuation limit.

fieldname is the name of the field. The name has the same format as an NCL variable name; that is, 1 through 12 characters alphanumeric, and, if the first character is numeric, the entire name must be numeric. Field names must be unique within an NDB. Do not precede the name with an ampersand (&), unless you want the actual field name to be the contents of the named variable.

The optional field operands for format, key, and so on, are ignored for a delete request (however, they must be valid).

If a field entry that follows the optional ADD keyword has no parenthesis around it, all the default values are used.

The optional operands are:

***Fmt*= | *Format*=**

Specifies the field format. Valid values are:

Char

Character data. Any character value may be provided. The data is stored internally as entered, with trailing blanks removed.

Num

Numeric data. Values provided must be numeric, range -2147483648 to 2147483647. The data is stored internally as a binary fullword. If keyed, the keys collate correctly (that is, -1 before 0 before 1 if reading sequentially).

Float

Floating-point data. The data is stored internally in IBM 8-byte normalized floating-point format. 15 significant digits and exponent ±70. If keyed, the keys collate on ascending numeric value.

Hex | X

Hexadecimal data. Values provided must be valid expanded-hexadecimal data. The data is stored internally in the hexadecimal-compressed format.

Note: Trailing zeros are significant, and the values ABCD and ABCD00 are regarded as different. If keyed, the keys collate on the binary value.

Date

Date data. Values must be a valid date, the input format depending on the &NDBCTL DATEFMT option. The data is stored internally as unsigned packed, 3 bytes in DDMMYY format. If keyed, the keys collate on ascending dates.

Cdate

Data is provided in one of several formats, controlled by the user, system language code, or both, and the current &NDBCTL DATEFMT setting. The data is stored internally as a 3-byte binary number being the number of days from 1/1/0001.

Time

Data is provided in HHMMSS.TTTTTT format (the decimal point and fraction is truncated or omitted). The data is stored internally as a 5-byte binary number, being the number of microseconds since midnight.

Timestamp

Data is provided in YYYYYMMDDHHMMSS.TTTTTT format. The data is stored internally as a concatenation of a 3-byte CDATE and 5-byte TIME.

Default: Char

Note: For the UPDATE keyword, the field format can only be changed if the NDB is empty.

Key=

The keying option. Valid values are:

No

The data is not keyed.

Yes

The data is keyed, with duplicate key values permitted.

Unique

The data is keyed, with duplicate key values not permitted.

Sequence

The data is keyed, with duplicate key values not permitted. In addition, the key is used as the data sequence key, similar to a VSAM primary key.

Default: No

Note: KEY=SEQUENCE forces NULLFIELD=NO and UPDATE=NO.

NULLField=

Specifies whether a field is absent in a record. Absent means not provided, and is not the same as present, with a null value (for example, all blank for a character field).

Yes

Means that the field is not required when a record is added, or the field can be set to null on update.

No

Means that the field must be present when a record is added, and the field cannot be set null on update.

Default: Yes

NULLValue=

Specifies whether a field can be added with, or updated to, the null value (all blank for character, 0 for numeric, blank for hexadecimal, and 000000 for date).

Yes

Means that the null value is acceptable as a value for this field.

No

Means that the null value is invalid for this field. An attempt to add a record with this field containing the null value, or to update the field to a null value causes an error.

Default: Yes

Update=

Specifies whether the field value can change on an update statement.

Yes

Means that the value is changed on an &NDBUPD statement.

No

Means that the field value cannot be changed on an &NDBUPD statement.

Note: The field and its value are specified on an &NDBUPD statement in this case, but the value must be identical to the current value.

Default: Yes

Caps=

Specifies the presence and preservation of lowercase data in FMT=CHAR fields. Specify this operand as CAPS=YES if coded on other field formats.

Yes

Lowercase data is folded to uppercase data in the stored value, and for the key value if the data is keyed. Key values on &NDBSEQ and &NDBGET, and search arguments on &NDBSCAN are also folded to uppercase.

No

Lowercase data is not folded to uppercase data in either the stored value, or the key value if the data is keyed. Key values on &NDBSEQ and &NDBGET, and search arguments on &NDBSCAN are not folded to uppercase.

Search

Lowercase data is not folded to uppercase in the stored value, but is folded to uppercase for the key value if the data is keyed. Key values on &NDBSEQ and &NDBGET, and search arguments on &NDBSCAN are folded to uppercase. Retrieved data is as originally entered (subject to &CONTROL UCASE/NOUCASE).

Default: Yes

Note: NDBs are language-specific. The LANG operand on the NDB CREATE command specifies the language code when the NDB is created. The uppercase translation table for that language code is used for search arguments and CAPS=YES fields when translating to uppercase.

Description=*description*

Allows an optional description of the field, up to 60 characters, to be stored with the field definition. This description is retrieved with the &NDBINFO statement. If there are special characters or blanks in the description, surround it with quotes.

USER1= | U1=

Allows the storage of an optional user-defined piece of information, for example, a formatting indicator. The format is like description, but limited to eight characters.

USER2= | U2=

USER3= | U3=

USER4= | U4=

Like USER1, USER2, USER3, and USER4 allow eight characters of extra, user-defined information to be stored.

NEWNAME=*name*

This optional operand is used to rename the field, for UPDATE only.

BASE={ NONE | DECIMAL | HEX }

Specifying NONE or DECIMAL for a numeric field results in an external representation as a signed decimal number (on output, a minus sign only is used, if necessary). Specification of HEX causes the external representation to be a hexadecimal string, with leading zeros suppressed on output. For negative numbers, the full seven hexadecimal digits are used (leading X'f').

For a field format other than NUMERIC, the BASE operand is ignored, but its syntax is checked.

Update a field definition to change the BASE operand—either by the &NDBDEF verb or the NDB FIELD command.

If the NDB is in load mode or newly created, the following attributes can also be updated: KEY (to/from anything except SEQ), and CAPS=N to CAPS=S and conversely. In addition, the field is renamed at any time.

The NDB ALTER command allows a field to be dynamically indexed without using this facility. However, if you want to alter many fields, using LOADMODE and a full rebuild of indexes would be faster than several passes, one per field.

Examples: &NDBDEF

This example adds four fields to the NDB named MYNDB:

```
&NDBDEF MYNDB (SURNAME FMT=C KEY=YES NULLFIELD=NO +
               UPDATE=NO)+
              (FIRSTNAME FMT=C KEY=N) +
              (DOB FMT=DATE DESC='Date of birth') +
              COMMENT
```

This example deletes two fields from the NDB named MYNDB:

```
&NDBDEF MYNDB DELETE DOB COMMENT
```

This example adds the field SUBURB to the NDB named MYNDB, as a non-keyed character field that can be updated:

```
&NAME = SUBURB
&NDBDEF MYNDB ADD &NAME
```

Notes:

If the database is to have a sequence key, it must be the first field ever defined, and the field definition cannot be deleted. There can only be one sequence key.

Currently there is no facility for making a field keyed, or removing the keys on a field, once it is defined.

The NDB RESET command removes all data from an NDB, but leaves the field definitions intact.

If there are any errors in the definitions, none of the definitions are added or deleted.

The NCLEX01 security interface can disable the &NDBDEF verb. If you disable the verb, the NDB FIELD command becomes the only way to alter NDB field definitions. This feature allows protection of NDB field definitions.

Note: See also the NDB CREATE and NDB RESET commands in the Online Help.

More information:

&NDBINFO (see page 478)

&NDBDEL

Deletes a record from an NDB database.

```
&NDBDEL dbname RID=n
```

The &NDBDEL statement allows an NCL procedure to delete a record from an NDB. The record to be deleted is identified by a record-ID (RID). This record ID is assigned to each record when it is added to the database by &NDBADD, and never changes. The system variable, &NDBRID, is set by several &NDB statements to indicate the RID of the current record.

Operands:

dbname

The name of the NDB that you wish to delete the record from is a required operand. This NDB must have been previously opened by an &NDBOPEN statement.

RID=*n*

This required operand is used to indicate the RID of the record to delete. n is the record ID, which is inserted by substitution, typically using &NDBRID, which has been set by, for example, a preceding &NDBGET statement.

Examples: &NDBDEL

```
&NDBGET MYNDB FIELD=SURNAME VALUE='BLOGGS' FORMAT +
 NO-FIELDS
&NDBDEL MYNDB RID=&NDBRID
```

This example uses &NDBGET to retrieve a record with field SURNAME equal to BLOGGS, and then will delete it.

```
&NDBSEQ MYNDB DEFINE SEQ=S1 FIELD=SUBURB VALUE=BRONX
&NDBGET MYNDB SEQ=S1 FORMAT NO-FIELDS
&DOWHILE &NDBRC = 0
   &NDBDEL MYNDB RID=&NDBRID
   &NDBGET MYNDB SEQ=S1 FORMAT NO-FIELDS
&DOEND
```

This example deletes all records (if any) in NDB MYNDB with field SUBURB equal to BRONX.

Notes:

Errors encountered whilst processing the &NDBDEL statement could cause the procedure to terminate, or may just be reflected in the &NDBRC system variable, depending on the setting of &NDBCTL ERROR option.

A successful delete sets &NDBRC to 0. If the RID does not exist in the database, &NDBRC is set to 1.

More information:

&NDBADD (see page 445)
&NDBGET (see page 472)
&NDBUPD (see page 513)

&NDBFMT

Defines a list of fields to be retrieved by an &NDBGET statement, or to be added or updated by the &NDBADD or &NDBUPD statements.

```
&NDBFMT dbname { [ DEFINE ]
                   FORMAT=formatname
                 [ FSCOPE={ PROCESS | GLOBAL } ]
                 [ USAGE={ OUTPUT | INPUT } ]
                 { START | [ DATA ] format-text } |
                  [ DATA ] format-text | END | CANCEL |
                    DELETE FORMAT= { formatname | * }
                 [ FSCOPE={ PROCESS | GLOBAL } ] }
```

format-text for an INPUT format is:

```
{ NO-FIELDS | ALL-FIELDS | KEY-FIELDS |
  FIELDS [ field-entry ] [ field-entry ] ... }
```

field-entry for an INPUT format is:

```
{ { name1 [ = { name2 | .RID } ] | pfix1* [ = pfix2* ] } |
 ( { name1 [ = { name2 | .RID } ] | pfix1* [ = pfix2* ] }
   [ LENGTH= { 0 | length } ]
   [ PAD= { ' ' | c | 'c' } ]
   [ JUSTIFY= { LEFT | RIGHT | CENTER } ]
   [ TRUNCATE ]
   [ NULLFIELD={ DELETE | NULLVALUE | PAD | NORETURN } ] ) |
   .LINK ( FROM=fieldname TO=keyedfieldname )
   [ ID=fmtid ]
   [ FROMID=fromfmtid ]
   [ NOFIND= { WARNING | n | IGNORE } ] )
 format-text }
```

format-text for an OUTPUT format is:

```
{ ALL-FIELDS | FIELDS [ field-entry ] [ field-entry ] ... }
```

field-entry for an OUTPUT format is:

```
[ ( ] { name1 [ =name2 ] | pfix1* [ =pfix2* ] }
[ [ TRUNCATE ] ) ]
```

The &NDBFMT statement allows an NCL procedure to predefine a list of fields to be returned when using &NDBGET. The advantage of predefinition is that it reduces the overhead of parsing and interpreting the format list on every &NDBGET. The parsing and interpretation is done, and the field list validated, just once, when the &NDBFMT statement is executed. The &NDBGET statement can then use the FORMAT=*format* name syntax to use the predefined format—the overheads is greatly reduced when reading a large number of records.

Operands:

dbname

The name of the NDB that the format is to apply to is a required operand. This NDB must have been previously opened by an &NDBOPEN statement.

DEFINE

An optional operand, indicating a new format definition follows, or the start of a multi-statement format definition follows.

FSCOPE={ PROCESS | GLOBAL }

Controls whether the format is PROCESS-level (this is the default) or GLOBAL (to the database).

A GLOBAL format has separate name space from any process. This means that global formats can have the same name as formats defined by any number of processes. The only way to create, reference, or delete a global format is by use of the FSCOPE=GLOBAL operand of the &NDBFMT, &NDBGET, &NDBADD, &NDBUPD or &NDBINFO verbs.

USAGE={ INPUT | OUTPUT }

Controls the usage of the format. INPUT is the default and means that the format will be used for input from the NDB (that is, the &NDBGET verb).

USAGE=OUTPUT means that this format is to be used for output operations to the NDB (that is, the &NDBADD and &NDBUPD verbs).

FORMAT=*formatname*

A required operand for a single-statement definition, or the start of a multi-statement format definition, providing the name of the new format. *formatname* must be a 1- to 12-character name, the first character alphabetic or national, and the rest alphanumeric or national. *formatname* must not be already defined for the nominated database, but the same format name is defined on several databases.

START

Indicates the start of a multi-statement &NDBFMT. The statement must end after the START keyword.

DATA

Indicates that free-form text (see page 442) follows. This operand is optional, but it is recommended, as it prevents any ambiguous meaning of a *name1* or *name2* value of DATA, START, END, or CANCEL.

END

Indicates the end of a multi-statement &NDBFMT. This statement will call the database management system, passing the concatenated &NDBFMT format information.

CANCEL

Indicates an active &NDBFMT START/END set is to be canceled. If there is no active &NDBFMT START/END for this database, the statement is ignored.

DELETE FORMAT= { *formatname* | * }

Indicates that the named format (*formatname*), or all formats (*) within the scope specified or defaulted, are to be deleted. Following execution of this statement, the relevant formats are no longer defined.

***format-text* (for an INPUT format)**

Free-form text describing the desired format. The first keyword in this text indicates the specific type of format, which is one of the following:

NO-FIELDS

Indicates that no data is wanted. An &NDBGET will only set &NDBRC to indicate the successful retrieval or otherwise, of the requested record.

ALL-FIELDS

Indicates that all fields defined in the database are to be returned, with fields not present in a given record being set to null. The fields will be returned in NCL variables of the same name.

KEY-FIELDS

Indicates that all fields defined in the database as being keyed (KEY=YES,UNIQUE, or SEQUENCE in the field definition) are to be returned, with key fields not present in a given record being set to null. The fields will be returned in NCL variables of the same name.

FIELDS

Indicates that a field list follows. The list will indicate the desired database fields to be returned, with optional renaming of the returned data.

Note: The field list is null, meaning the same as NO-FIELDS.

***field-entry* (for an INPUT format)**

When using the FIELDS option, each field-entry denotes an operation to be performed during an &NDBGET. The operation may be to assign values from the database records to NCL variables, or it may be to link to other records in this NDB using field values in the previous records as keys.

If only *name1* is present, it is both the name of the database field and the name of the NCL variable that will be set to its value. If both *name1* and *name2* are present, *name1* is the name of the NCL variable that will hold the returned value, and *name2* is the name of the field in the database.

If only *pfix1** is present, all of the fields whose names begin with *pfix1* will be assigned to NCL variables of the same name. If both *pfix1** and *pfix2** are present, *pfix1* is the NCL variable prefix that will be used for the variables containing the returned values from all of the NDB fields which begin with *pfix2*. *pfix1* is null, which means that the NCL variable names will be equivalent to the NDB field names, but without the prefix *pfix2*. In this case, if there is a NDB field called *pfix2*, it will not be assigned.

.RID

Indicates that the NDB record ID of the retrieved record will be assigned to the NCL variable name1.

LENGTH=*length*

Indicates the length of the NCL variable returned. Specifying 0 (the default) means that the variable will be set to the length of the corresponding NDB field. NDB fields shorter than that specified on the LENGTH option will be padded and justified as per the operands explained below. Longer fields will always be truncated on the right.

PAD= { ' ' | *c* | '*c*' }

Indicates the pad character to use when the length for this NCL variable is greater than the length of the NDB field. The default is space.

JUSTIFY={ LEFT | RIGHT | CENTER }

Indicates how the data from the NDB field will be justified to the NCL variable when the length of the NCL variable is greater than the NDB field.

LEFT, the default, indicates that the data will start in the first character position of the variable, the rightmost portion of the variable containing the specified pad character.

RIGHT indicates that the data will finish in the right most character position of the NCL variable, the leftmost part of the variable being filled with the specified pad character.

CENTER indicates that the data will be centered and the same amount of the specified pad character will be used on either side.

TRUNCATE

If generic prefixes are specified, then it is possible for NCL variables to exceed 12 characters. However, this will normally cause an error. If TRUNCATE is specified, variable names will be truncated to a maximum of 12 characters. No checking on duplicates will be performed.

NULLFIELD= { DELETE | NULLVALUE | PAD | NORETURN }

Determines the action to take on a receiving NCL variable when the NDB field is not present.

DELETE, the default, specifies that the variable is to be deleted.

NULLVALUE indicates that the variable will be assigned the correct null value for that field type - 0 for numeric fields, space for character fields, and 00/00/00 for date fields.

PAD indicates the variable will be assigned the value of the specified pad character.

NORETURN indicates that no action is to be performed on the variable if the corresponding field is null. This means that if the variable had a previous value in it, that value would remain there after the GET. NORETURN is very dangerous unless it is used with MODFLD=YES in the &NDBGET statement as it cannot be determined whether a field was modified or not by the &NDBGET.

.LINK

Gives the user the facility to access up to 21 NDB records with a single &NDBGET statement via a linked get. The maximum number of .LINK requests on an &NDBGET statement is 20 (one .LINK retrieves one record).

FROM=*fieldname*

Specifies the name of the field that will contain the source data for the linked get. This field is the field in the first (not .LINKed) record retrieved by the &NDBGET (if no FROMID specified), or in a previously linked record (if FROMID specified).

TO=*keyedfieldname*

Specifies the key field to use in the search for the linked get. The search is always like a GET OPT=KEQ. If more than one record is found matching the given key, the one with the lowest RID is returned.

ID=*formatid*

This is an optional 1- to 12-character name used to identify this particular link. This name is used in a subsequent link entry to identify this record, as opposed to the original record, as the source of the linked get.

FROMID=*fromformatid*

This is an optional 1- to 12-character name which must have appeared as the ID operand of a previous linked entry. If omitted, the key value specified in the FROM operand is taken from the primary record. Otherwise it is taken from the record obtained in the link operation identified by from format ID

NOFIND = { WARNING | *n* | IGNORE }

Specifies the action to take should there be no matching record when a linked get is performed.

WARNING, the default, means that the operation will terminate with a response code of 10, and all subsequent link operations are ignored.

Specifying n will cause a response code of n to be generated, n being within the range 10 to 19. Subsequent link operations will be ignored.

Specifying IGNORE will return with a response code of 0 and subsequent link operations will be attempted if possible.

***format-text* (for an OUTPUT format)**

ALL-FIELDS

Indicates that all fields defined in the database are to be added or updated. The fields will be retrieved from NCL variables of the same name.

FIELDS

Indicates that a field list follows. The list will indicate the desired database fields to be added or updated.

***field-entry* (for an OUTPUT format)**

If only *name1* is present, it the name of both the NCL variables and the name of the database field that will be set to its value. If both *name1* and *name2* are present, *name1* is the name of the field in the database, and *name2* is the name of the NCL variable containing its value.

If only *pfix1** is present, then all of the database fields whose names begin with *pfix1* will be set to values retrieved from NCL variables of the same name. If both *pfix1** and *pfix2** are present, *pfix2* is the NCL variable prefix that will be used for the variables to set values to the NDB fields which begin with *pfix1*.

Referring to the same NDB field name twice in an OUTPUT format will cause an error.

TRUNCATE

If an NCL variable name that is greater than 12 characters long is generated from the combination of the NCL generic prefix and the suffix from a generically found NDB field name, the NCL variable name will be truncated to 12 characters.

Examples: &NDBFMT

The following example defines a format for MYNDB, called F1, to return all fields in the database when used on an &NDBGET. Any fields not in a retrieved record cause the appropriate NCL variable to be set no null.

```
&NDBFMT MYNDB DEFINE FORMAT=F1 DATA ALL-FIELDS
```

The next example defines a new format for MYNDB, called F2, to return 3 fields when used on &NDBGET, and returns the value in the database field 'SURNAME' in the NCL variable &SNAM, and 'FIRSTNAME' in &FNAM.

```
&NDBFMT MYNDB DEFINE FORMAT=F2 START
&NDBFMT MYNDB DATA FIELDS           -* indicate field list
&NDBFMT MYNDB DATA SNAM = SURNAME   -* get surname back in
                                    -* &SNAM
&NDBFMT MYNDB DATA DOB              -* get B back in &DOB
&NDBFMT MYNDB DATA (FNAM=FIRSTNAME) -* get firstname back
                                    -* in &FNAM
&NDBFMT MYNDB END
```

The next example defines a new format for MYNDB, called F3, to return all the fields in the record prefixed by the letters 'ORD', and then to use the value in the field 'ORDCUST#' to perform a GET OPT=KEQ using the CUST# field as the key. If this get is successful, then all of the fields prefixed by CUST are returned from the second record.

```
&NDBFMT MYNDB DEFINE FORMAT=F3 DATA FIELDS ORD* +
       .LINK (FROM=ORDCUST# TO=CUST#) FIELDS CUST*
```

The next example defines a new format for MYNDB, called F4, to return the record number of the found record in the NCL variable &RECRD, and then to use the value in the field 'ORDCUST#' to perform a GET OPT=KEQ using the CUST# field as the key. If this get is successful, then all of the fields prefixed by CUST are returned from the second record. It then uses the value in the field 'ORDSTAT' from the original record to perform a GET OPT=KEQ using ORDSTATKEY as the key. If successful, &ORDSTATREC is set from the record found using ORDSTATKEY.

```
&NDBFMT MYNDB DEFINE FORMAT=F4 DATA FIELDS RECRD = .RID +
       .LINK (FROM=ORDCUST# TO=CUST#) FIELDS CUST* +
       .LINK (FROM=ORDSTAT TO=ORDSTATKEY) FIELDS ORDSTATREC
```

The next example defines a new format for MYNDB, called F5, to return all the fields in the record prefixed by the letters 'OLINE' to corresponding variable names beginning with 'L'.

```
&NDBFMT MYNDB DEFINE FORMAT=F5 DATA FIELDS L* = OLINE* +
      .LINK (FROM=OLINEITEM# TO=ITEM# ID=ITEM) +FIELDS ITEM* +
      .LINK (FROM=ITEMSUPP# TO=SUPPLIER# FROMID=ITEM) +FIELDS SUPP*
```

For example, OLINE1 goes to L1, OLINE2 goes to L2, and so on. Then the value in the field 'OLINEITEM#' is used to perform a GET OPT=KEQ using the ITEM# field as the key. If this get is successful, then all of the fields prefixed by ITEM are returned from the second record. It then uses the value in the field 'ITEMSUPP#' in the second record to perform a GET OPT=KEQ using the SUPPLIER# field as the key. If this get is successful, all of the fields prefixed by SUPP are returned from the third record.

Notes:

Errors encountered while processing the &NDBFMT statement may cause the procedure to terminate, or may just be reflected in the &NDBRC system variable, depending on the setting of &NDBCTL ERROR option.

All defined PROCESS-level formats for a given database are deleted when an &NDBCLOSE for that database is executed, and, as all open databases are implicitly closed when the highest-level procedure terminates, all defined PROCESS-level formats are deleted too.

The &NDBGET statement may specify a format description itself, but the START/DATA/END option is not available, thus the format description must fit onto a single statement.

Format names are private to an NCL procedure. Any number of users of an NDB may use the same format name, with no interference. If a format is specified on an &NDBGET for a key field histogram, it is ignored.

More information:

&NDBGET (see page 472)

&NDBGET

Retrieves a record from an NDB database. Histograms (statistical information) is retrieved for keyed fields and histogram sequences.

```
&NDBGET dbname { RID=n [ OPT= { KEQ | KGE | KGT | KLE | KLT } ] |
                 KEY=fieldname VALUE=value
                  [ OPT= {KEQ | KGE | KGT | KLE | KLT } ] |
                 FIELD=fieldname VALUE=value
                  [ OPT= { KEQ | KGE | KGT | KLE | KLT | GEN } ] |
                 SEQUENCE=seqname [ SKIP=n ]
                  [ DIR= { FWD | BWD } ] }
               [ MODFLD= { NO | YES } ]
               { FORMAT=formatname [ FSCOPE={ PROCESS | GLOBAL } ] |
                 FORMAT format-text }
```

The &NDBGET verb allows an NCL procedure to retrieve a record from an NDB. There are four basic retrieval methods:

- Get by record ID (RID). This method is useful when the procedure knows the RID of the desired record.
- Get by key-field value. This method allows a procedure to get a record that matches a particular keyed field value.
- Get a keyed field value and record count.
- Get from a predefined sequence, defined by &NDBSEQ or &NDBSCAN. This method allows sequential retrieval to be easily performed.

Operands:

dbname

This operand is mandatory. It specifies the name of the NDB that you want to retrieve the record from. The NDB specified must have been previously opened by an &NDBOPEN statement.

RID=*n*

Indicates a get by record ID (RID), and provides the RID of the desired record, or the comparison RID if not using the default OPT=KEQ.

OPT={ KEQ | KGE | KGT | KLE | KLT }

This optional operand indicates the relation to the passed RID that the retrieved record RID will have.

KEQ

Retrieves the record with the passed RID, if it exists.

KGE

Retrieves the record with the passed RID, if it exists. If none exists, the nearest RID greater than the passed RID is retrieved, if one exists.

KGT

Retrieves the record with the nearest RID greater than the passed RID, if one exists.

KLE

Retrieves the record with the passed RID, if it exists. If none exists, the nearest RID less than the passed RID is retrieved, if one exists.

KLT

Retrieves the record with the nearest RID less than the passed RID, if one exists.

KEY=*fieldname* VALUE=*value*

Indicates that a keyed field histogram is to be performed. *fieldname* must be keyed, and not a sequence key. Rather than return the associated record (as GET FIELD= does), instead the field value (as requested by the VALUE operand) is retrieved, and the count of records having that value is also returned.

The returned key field value is always returned in &NDBKEYVALUE. The returned record count is always returned in &NDBKEYCOUNT. The &NDBRID system variable is always set to 0.

The VALUE= operand provides the key value for a get by keyed field. The value is used to locate a record for the named field. Value must be a valid value for the type of the field. For example, for a numeric field, it must be a valid number. If the value contains special characters or embedded blanks, it must be enclosed in quotes.

Note: Embedded blanks in NCL variable values are a special case, and are treated as part of the value.

OPT= { KEQ | KGE | KGT | KLE | KLT | GEN }

Indicates the relation to the passed VALUE that the retrieved record key FIELD will have.

KEQ

Retrieves the record with the passed VALUE, if it exists.

KGE

Retrieves the record with the passed VALUE, if it exists. If none exists, the nearest VALUE greater than the passed VALUE is retrieved, if one exists.

KGT

Retrieves the record with the nearest VALUE greater than the passed VALUE, if one exists.

KLE

Retrieves the record with the passed VALUE, if it exists. If none exists, the nearest VALUE less than the passed VALUE is retrieved, if one exists.

KLT

Retrieves the record with the nearest VALUE less than the passed VALUE, if one exists.

GEN

(Character fields only) Retrieves the record with the lowest value generically equal to the passed VALUE, if one exists.

Note: Trailing blanks are significant in the passed value in this case.

FIELD=*fieldname*

Indicates a get by a keyed field, and provides the name of that field. If this operand is specified, the VALUE=value operand must also be coded. The named field must be a defined field on the database, and it must be keyed.

VALUE=*value*

Provides the key value for a get by keyed field. The value is used to locate a record with a value satisfying the OPT= relation for the named field. Value must be a valid value for the type of the field, for example, it must be a valid number for a numeric field. If the value contains special characters or embedded blanks, it must be enclosed in quotes.

Note: Embedded blanks in NCL variable values are a special case, and are treated as part of the value.

OPT= { KEQ | KGE | KGT | KLE | KLT | GEN }

Indicates the relation to the passed VALUE that the retrieved record key FIELD will have.

KEQ

Retrieves the record with the passed VALUE, if it exists.

KGE

Retrieves the record with the passed VALUE, if it exists. If none exists, the nearest VALUE greater than the passed VALUE is retrieved, if one exists.

KGT

Retrieves the record with the nearest VALUE greater than the passed VALUE, if one exists.

KLE

Retrieves the record with the passed VALUE, if it exists. If none exists, the nearest VALUE less than the passed VALUE is retrieved, if one exists.

KLT

Retrieves the record with the nearest VALUE less than the passed VALUE, if one exists.

GEN

(Character fields only) Retrieves the record with the lowest value generically equal to the passed VALUE, if one exists.

Note: Trailing blanks are significant in the passed value, in this case.

SEQUENCE=*seqname*

Indicates a get from a predefined sequence, as defined by an &NDBSEQ or &NDBSCAN statement. *seqname* is the name of the sequence to retrieve from. If the SKIP= and DIR= operands are not specified, they default to SKIP=+1 and DIR=FWD, thus giving standard, forward, sequential retrieval.

A key field histogram sequence is specified.

The returned key field value is always returned in &NDBKEYVALUE. The returned record count is always returned in &NDBKEYCOUNT. The &NDBRID system variable is always set to 0.

As for a direct get on a keyed field, the format is ignored and the returned fields are as described previously.

SKIP=*n*

Allows specification of a skip amount, that is, the number of records to skip over before retrieving one. SKIP=+1 is the default, causing a skip to the next record in the direction indicated by DIR. SKIP=0 will cause a reread of the last record obtained from this sequence. A negative skip amount reverses the direction specified by DIR.

Note: This is perfectly symmetrical; SKIP=-1, DIR=FWD is equivalent to SKIP=+1, DIR=BWD, and so on.

DIR= { FWD | BWD }

Allows specification of the direction of sequential retrieval. FWD means ascending values, BWD means descending values (assuming a positive skip amount).

FORMAT=*fmtname* [FSCOPE={ PROCESS | GLOBAL }]

FORMAT=*fmtname* specifies that the input format *fmtname*, defined on the &NDBFMT statement, is to be used.

The nominated format must exist in the nominated scope. PROCESS is the default and means a format defined by the current NCL process. GLOBAL indicates a format is to be found in the global format pool for the NDB.

If this operand is specified, the START, DATA, CANCEL, and END operands are not allowed.

MODFLD = { YES | NO }

If MODFLD=YES is specified, any NCL variables which previously had the modified field attribute set will have that attribute reset, provided the GET operation produces a zero return code. In addition, any fields which were modified as a result of the GET statement will have their modified attribute set. This includes variables set to pad characters as a result of the NULLFIELD option of the &NDBFMT.

The system variable &ZMODFLD is used to return the names of the modified fields and &ZVARCNT will be available as the number of modified fields.

This option is used with the NORETURN option (see the description of the &NDBFMT verb) for improved efficiency. If MODFLD=NO is specified or defaulted, no modified field attribute reset will occur, and the variables will not have their modified attributes set. The system variables &ZMODFLD and &ZVARCNT will be unchanged.

Examples: &NDBGET

The following example retrieves the record with the RID in &SAVERID, and returns all defined database fields to the procedure (assuming format F1 as defined in the &NDBFMT examples). If there is no record with that RID, &NDBRC will be set to 1, and no fields will be returned.

```
&NDBGET MYNDB RID=&SAVERID FORMAT=F1
```

The next example shows one way to sequentially read an entire database, in RID sequence. The first &NDBGET gets the record with the lowest RID on the database, and the second &NDBGET gets the next-highest, until the last record is read. There must not be any &NDB*xxx* statements in the process record section, as &NDBRID would lose its value. In this case, the value must be saved in a user variable. (See the next example for a better approach.)

```
&NDBGET MYNDB RID=0 OPT=KGE FORMAT ALL-FIELDS

&DOWHILE &NDBRC = 0
   ... process record
   &NDBGET MYNDB RID=&NDBRID OPT=KGT FORMAT ALL-FIELDS
&DOEND
```

The next example uses a defined sequence to read the entire database, but only returning every fifth record (this is useful for creating test files).

```
&NDBSEQ MYNDB DEFINE SEQUENCE=S1 RID
&NDBGET MYNDB SEQUENCE=S1 SKIP=5 FORMAT ALL-FIELDS
&DOWHILE &NDBRC = 0
   ... process record
   &NDBGET MYNDB SEQ=S1 SKIP=5 FORMAT ALL-FIELDS
&DOEND
```

The next example sequentially reads MYNDB forwards, backwards, and then forwards, and so on, forever. The sequence is defined with KEEP=YES, which means that it stays defined at each EOF. Whenever an EOF is reached, the skip is inverted (+1 - -1, -1 - +1), and the get loop restarted.

```
-* database shuttlecock.
&SKIP = +1 &NDBSEQ MYNDB DEFINE SEQ=S2 FIELD=SURNAME KEEP=YES
&DOWHILE A = A
   &NDBGET MYNDB SEQ=S2 SKIP=&SKIP FORMAT ALL-FIELDS
   &DOWHILE &NDBRC=0      ... process record
      &NDBGET MYNDB SEQ=S2 SKIP=&SKIP FORMAT ALL-FIELDS
   &DOEND
   &SKIP = 0 - &SKIP
&DOEND
```

Notes:

Errors encountered whilst processing the &NDBGET statement may cause the procedure to terminate, or may just be reflected in the &NDBRC system variable, depending on the setting of &NDBCTL ERROR option.

A record-not found condition on RID and FIELD gets will set &NDBRC to 1. An end-of-file condition on SEQ gets will set &NDBRC to 2.

The actual NCL variables set by a successful &NDBGET depend on the format used. An unsuccessful get never alters any NCL variables. Thus, when an end-of-file response is returned, the NCL variables retain the values set by the last successful &NDBGET.

The &NDBGET statement may specify a format description itself, but the START/DATA/END option is not available. Therefore, the format description must fit onto a single statement. It is more efficient to pre-define formats that are used more than once in a procedure.

When reading via a sequence, an EOF condition (&NDBRC = 2) will delete the sequence, unless it was defined with the KEEP=YES option (on &NDBSEQ or &NDBSCAN). If MODFLD=YES is specified, the order in which &ZMODFLD returns field names is undefined.

More information:

&NDBQUOTE (see page 490)
&NDBFMT (see page 464)
&NDBSEQ (see page 508)
&NDBSCAN (see page 492)

&NDBINFO

Retrieves information about an NDB database.

```
&NDBINFO dbname { DB | [ FIELD ] { NAME=fieldname | NUMBER=n } }
                [ FORMAT=formatname [ FSCOPE={ PROCESS | GLOBAL } ] ]
                [ FULL | SHORT ]
```

The &NDBINFO verb allows an NCL procedure to obtain information about an NDB, including information about the database itself, and information about the fields defined in it.

The information is returned in NCL variables.

Operands:

dbname

Specifies the name of the NDB that you wish to retrieve information about, and is a mandatory operand. The NDB must have been previously opened by an &NDBOPEN statement.

DB

Indicates that information about the database itself is to be returned. The information is returned in the following NCL variables:

- NDBDBNAME contains the database name, as coded on the &NDBINFO statement
- NDBDBVKL contains the VSAM key length of the database
- NDBDBVRL contains the VSAM maximum data length of the database
- NDBDBNFLDS contains the number of currently defined fields in the database
- NDBDBNRECS contains the number of records in the database
- NDBDBNRID contains the next RID the database will use on an &NDBADD statement
- NDBDBLANG contains the language code that describes the database's uppercase table If null, the standard (US-format) uppercase table is used
- NDBDBLOADMD indicates if the database is in load mode or not Values are NO and YES
- NDBDBRIDRU indicates the RID reuse status for the database Values are:
  - N/S—not supported
  - POSS—possible (supported but not enabled)
  - ENAB—enabled but not presently active
  - ACT—presently active
  - ACT/C—active with a KEYSTATS scan (collection) in progress
  - COLL—a KEYSTATS scan is in progress, but reuse is not currently active (no ranges available)
- NDBDBRIDNRU contains the number of reused RIDs handed out since the last KEYSTATS run
- NDBDBRIDNNW contains the number of new RIDs handed out since the last KEYSTATS run
- NDBDBIVERS contains the internal version of the database as 4 digits. RID reuse became available at 0510

[FIELD] NAME=*fieldname*

Indicates that information about the named field is to be returned. If the named field does not exist in the nominated NDB, then &NDBRC is set to 3. The information returned is listed below.

[FIELD] NUMBER=*n*

Specifies that information about relative field number n is to be returned. This number must be from 1 to the value returned by an &NDBINFO DB request in &NDBDBNFLDS. The information returned on the field requests is set into NCL variables as shown:

- &NDBFLDNAME contains the name of the field
- &NDBFLDFMT contains the field format, values CHAR, NUM, HEX, FLOAT, or DATE
- &NDBFLDKEY contains the field key option, values NO, YES, UNIQUE, or SEQUENCE
- &NDBFLDNULLF contains the NULLFIELD option, values YES or NO
- &NDBFLDNULLV contains the NULLVALUE option, values YES or NO
- &NDBFLDUPD contains the UPDATE option, values YES or NO
- &NDBFLDCAPS contains the CAPS option, values YES, NO, or SEARCH
- &NDBFLDDESC contains the field description, if one was defined; otherwise it contains one blank
- &NDBFLDUSER1 contains the USER1 information if present; otherwise, it contains one blank
- &NDBFLDUSER2 contains the USER2 information if present; otherwise, it contains one blank
- &NDBFLDUSER3 contains the USER3 information if present; otherwise, it contains one blank
- &NDBFLDUSER4 contains the USER4 information if present; otherwise, it contains one blank
- &NDBFLDMAXL contains the maximum length, in characters, that the field can hold. For CHAR fields, this is 255 if not keyed, and (VSAM keylen - 8) if keyed. For NUM fields, is 15. For HEX fields, this is 254 if not keyed, and ((VSAM keylen - 9) * 2) if keyed. For DATE fields, this is 8.
- &NDBFLDINTID contains a 4-digit hexadecimal string (representing a 2-byte value) that is the internal field identification inside the NDB
- &NDBFLDBASE contains the value DEC or HEX for numeric format fields and the value NONE for all other formats

- &NDBFLDKSTAT contains a string of blank-separated numbers as follows:

  dddddddd ff a b c d e f g h i j k l m n o p q r s

  where:

  - *dddddddd* is the date that key statistics were last collected for this field, in the format YYYYMMDD. If this number is all zeros, no key statistics have been collected for it.

    Note: Even non-keyed fields will have a non-zero date set after key statistics collection, but all other fields will be zero.

  - *ff* is two flag characters, each equal to Y (meaning YES) or N (meaning NO). The first is Y if a table overflow occurred while calculating the modal unique value occurrence (see the description of fields g and h following). The second flag is Y if a table overflow occurred while calculating the modal VSAM record occurrence (see the description of fields l and m following).
  - *a* is the number of unique values found for this key.
  - *b* is the total number of VSAM records that hold information about records having this key.
  - *c* is the number of times this key field actually exists in data records.
  - *d*, *e*, and *f* are the minimum (*d*), maximum (*e*), and average (*f*) number of records that have the same unique value. For example, for a unique key, *d*, *e*, and *f* are all 1, or zero if no records have the field present.
  - *g* is the modal number of records that have the same unique value; that is, the most often occurring count of same-numbers of each unique value.
  - *h* is the count for the modal value. For example, if the most often occurring count of a unique value is 35, then g is 35. If this count occurs 97 times, then *h* is 97. If the first of the two flags (*ff*) described is set to Y, this modal count (*h*) might not be correct, as the table used to keep counts/occurrences can overflow.
  - *i*, *j*, and *k* are the minimum (*i*), maximum (*j*), and average (*k*) numbers of VSAM records used to hold information for any unique value.
  - *l* is the modal number of VSAM records for any unique value; that is, the most often occurring count of same-numbers of VSAM records for each unique value.
  - *m* is the count for the modal value. For example, if the most often occurring count of VSAM records for a unique value is 27, then l is 27. If this count occurs 115 times, then m is 115. If the second of the two flags (*ff*) described is set to Y, this modal count (*m*) might not be correct, as the table used to keep counts/occurrences can overflow.

- *n*, *o*, *p*, *q*, *r*, and *s* are the minimum (*n*), maximum (*o*), average (*p*), modal (*q*), modal count (*r*), and all-equal (*s*) key lengths for this field. The key length statistics represent significant key field characters (NDBs use VSAM KSDSs, and the VSAM key length is fixed for each individual NDB). For a character field, this is the length less trailing blanks. For a HEX field, this is the exact length provided (trailing X'00's are significant).

  Numeric, date, and floating point fields are all fixed length-4, 3, and 8 respectively. In this case, all the values except modal count are that value (modal count equals the number of VSAM records. The modal value calculation cannot overflow for key length statistics.

  The all-equal key length is the maximum key length, in all cases (including fixed length fields), where all values found had the same prefix. It can range from zero to the maximum key length. (It is used by the scan optimizer in weighing generic and range requests.)

  The maximum number of records having any unique value is greater than one for a field defined as KEY=UNIQUE. This is because NDB ALTER OPT=BLDX can tolerate unique key violations during index build, allowing you to correct them later.

FORMAT=*formatname* [FSCOPE={ PROCESS | GLOBAL }]

FORMAT=*formatname* indicates that the format *formatname*, defined on the &NDBFMT statement, is to be used. The nominated format must exist in the nominated scope. PROCESS is the default, and indicates a format defined by the current NCL process. GLOBAL indicates that a format is to be found in the global format pool for the NDB.

If the format does not exist, a response code of 20 is returned. If the format exists, a set of variables is returned, as follows:

&NDBFMTUSAGE

Contains the value INPUT if this is an input format (that is, usable on an &NDBGET statement), or the value OUTPUT if this is an output format (for use with &NDBADD or &NDBUPD verbs).

&NDBFMT0

Contains the number of &NDBFMT*n* NCL variables returned.

&NDBFMT1 to &NDBFMT*n*

Contain strings of blank-separated values that describe either an individual field entry or a link to another record.

If the format is for input use, and the variable describes a return field, it is in the following format:

a b c d e f g h i

where:

- *a* is the return NCL variable name.
- *b* is the return format. If no editing was applied (justify, pad, and so on), this is the same as the database variable format (*d*). If editing was done, it is CHAR.
- *c* is the name of the NDB field that the data is coming from.
- *d* is the format of the NDB field: CHAR, NUM, HEX, DATE or FLOAT.
- *e* is the format length requested. It is zero if not specified.
- *f* is the format pad character. If not specified, or blank, or if the format length (*e*) is zero, the return value is a single dash character (-). Otherwise, it is the pad character quoted (for example, a per cent sign). If the pad character is a single quote, it is quoted using double quote characters.
- *g* is the justify option—LEFT, CENTER, or RIGHT. If the format length (*e*) is zero, *g* is returned as a single dash (-).
- *h* is the NULLFIELD return option—DELETE, NULLVAL, PAD, or NORETURN.
- *i* is a single dash (-) at present (reserved for a possible date format option).

If the format is for input use, and the variable describes a link to another record, it is in the following format:

a b c d e f

where:

- *a* is the literal .LINK, which is an invalid variable name, thus distinguishing this information from a variable description.
- *b* is the ID value assigned to this link, or a single dash (-) if none was specified.
- *c* is the from-ID value for this link, or a single dash if none was specified-meaning the from record is the primary retrieval record.
- *d* is the from NDB field name.
- *e* is the to NDB field name—it must be keyed.
- *f* is the no find option—it is the word IGNORE or a number from 10 to 19.

If the format is for output use, it is in the following format:

a b c d

where:

- *a* is the source NCL variable name from which data will be extracted.
- *b* is the target NDB field name that will be set.
- *c* is the target NDB field format (CHAR, and so on).
- *d* is additional information—currently, *TEMP is always a dash.

The order of returned information in these variables agrees with the original format definition in that field for a specific record or link definition. However, the order of field description entries is not necessarily the same as the original format, as generic or range field specifications are expanded out and all fields are returned in internal ID order within a record or link.

FULL | SHORT

The FULL option requests that all information about the field be returned. The SHORT option requests that the only information that is obtained without reading the NDB be returned. The following fields are not returned:

- &NDBFLDDESC
- &NDBFLDUSER1, 2, 3, and 4
- &NDBFLDKSTAT

Examples: &NDBINFO

The following example returns information about the NDB called MYNDB in NCL variables &NDBDB... (as described previously).

```
&NDBINFO MYNDB DB
```

The next example returns information about the field called 'SURNAME' in MYNDB. The returned information is in NCL variables &NDBFLD... (as described previously).

```
&NDBINFO MYNDB NAME=SURNAME
```

The following example returns information about all the fields defined in MYNDB.

```
&NDBINFO MYNDB DB
&I = 1
&DOWHILE &I LE &NDBDBNFLDS
   &NDBINFO MYNDB NUMBER=&I
   ... process field definition
   &I = &I + 1
&DOEND
```

Notes:

Errors encountered whilst processing the &NDBINFO statement may cause the procedure to terminate, or may just be reflected in the &NDBRC system variable, depending on the setting of &NDBCTL ERROR option.

The nominated NCL variables are not updated if a nonzero response is returned in &NDBRC (for example, named field not found).

The &NDBINFO statement makes it easy to write generalized NCL procedures to manipulate NDBs. The procedures can open any NDB and, by using &NDBINFO statements, build tables of control information for further processing.

In a similar way, a generalized database unload/reload utility is constructed.

Note: See also the NDB CREATE command description in the Online Help.

More information:

&NDBDEF (see page 455)

&NDBOPEN

Signs on (connects) to an NDB database. The NDB is opened in read-only mode and data is sent to a user open exit.

```
&NDBOPEN dbname
        [ EXCLUSIVE ]
        [ INPUT ]
        [ DATA userdata ]
```

The &NDBOPEN verb is used to initiate the connection an NCL procedure has with an NDB. It is similar to the initial &FILEID for a UDB. An &NDBOPEN statement must be executed before any other database-specific &NDB*xxx* statements are executed. If the database has already been opened by the procedure, an error condition may be indicated, depending on the setting of the &NDBCTL ERROR= options.

Operands:

dbname

Specifies the name of the NDB that you wish to sign on to, and is a mandatory operand. If already signed on to this NDB, an error response will be given, that may cause the procedure to terminate, depending on the &NDBCTL ERROR= setting.

If the database is not active, or started, this statement will cause it to be activated.

EXCLUSIVE

This optional operand indicates that the procedure wants exclusive access to the database. That is, the open will fail if there are any other signed on users, and, if successful, no other users will be permitted to sign on (&NDBOPEN) until this user signs off.

INPUT

An optional operand that indicates that this NCL process is not going to issue any update-type verbs (&NDBADD, &NDBDEL, &NDBDEF, or &NDBUPD).

Note: &NDBOPEN causes the database to be actually started, the database is started in read-only mode and all other users will also be restricted to read-only mode.

DATA *userdata*

An optional operand that allows you to specify data that is passed to the NCL user exit (as specified by the NCLEX01 SYSPARMS operand). This is only done if NDB is using the exit (NDBOPENX is set to YES), and if this &NDBOPEN actually establishes the path to the NDB (that is, the NDB is not already open by this process). The first 50 characters only are passed.

Examples:

The following example will sign on to the NDB called MYNDB. Other database statements referencing MYNDB can then be used.

```
&NDBOPEN MYNDB
```

The next example will sign on to MYNDB in exclusive mode, and prevent any other procedure from signing on.

```
&NDBOPEN MYNDB EXCLUSIVE
```

Notes:

An &NDBOPEN connects the entire procedure environment, that is, any upper or lower level nested EXECs.

The EXCLUSIVE option is useful for database backup and restore procedures to prevent concurrent update activity.

If the procedure is already currently signed on the NDB, response 34 will be returned, and the procedure will be terminated if the current &NDBCTL ERROR setting is ABORT. Aside from this case, all other error indications are returned to the procedure; that is, an implicit &NDBCTL ERROR=CONTINUE is in effect for an &NDBOPEN statement.

If a user exit is invoked (SYSPARMS NDBOPENX=YES), then if open is prevented by the exit, response code 40 is returned which may be dealt with as appropriate.

Notes: See also the NDB START command description in the Online Help.

More information:

&NDBCLOSE (see page 449)

&NDBPHON

Returns a phonetic value for a character string, typically a name.

```
&NDBPHON { SOUNDEX | USER } data
```

Operands:

SOUNDEX | USER

Controls the type of phonetic conversion to be performed.

SOUNDEX performs standard SOUNDEX encoding on the supplied data. See Knuth, Art of Computer Programming, Vol III, pp. 391-392.

USER drives the NDBPHON user exit. If there is no exit, a syntax error results. The user exit parameters are described below.

data

Specifies the source data to pass to the conversion (for example, a surname). The data is always converted to upper case before processing.

&NDBPHON Exit Call Details

NDB can invoke a user exit when the &NDBPHON built-in function is called with the USER option.

The exit that is called is determined by the setting of the NDBPHONX SYSPARM. If no exit is set, then a syntax error results.

When the NDBPHONX SYSPARM nominates an exit, the exit is loaded. Any previous exit is deleted. (NDBPHONX=NO will delete any old exit without loading a new one).

When an &NDBPHON USER built-in function is executed, and the user supplied data is not null, then this exit is invoked as follows:

- It is called under the main task for your product region and so must not issue O/S waits, because this will impact other processing.
- The user interface is as follows:
  - R1: Pointer to parameter list (described later).
  - R13: Standard 72-byte save area.
  - R14: Return address.
  - R15: Entry point.
  - AM: 31 in MVS/XA or MVS/ESA.
- Parameter list:

  A(PARM1)

  A(PARM2)

  A(PARM3)

  A(PARM4)

  A(PARM5)

  A(PARM6) with high bit set on

  - PARM1 is a fullword function code. 0 (decimal) is &NDBPHON USER. You should allow for function codes 4, 8, and 12 for the future and return r15=0 for them.
  - PARM2 is a fullword containing the source data length. It will contain a value from 1 to 256. This length excludes leading and trailing blanks.
  - The source data. This is a 256-character area. The source data is placed here, uppercased, and padded to 256 with blanks.
  - PARM4 is a fullword, initialized to 0, that the exit must update with the length of the returned phonetic translation. A length of 0 to 256 must be set.
  - PARM5 is a 256-character area, initialized to blanks, that must be set to the return phonetic translation.
  - PARM6 is a 256-byte work area, initialized to binary 0. This area is used by the exit as required.

The exit must return r15=0 and set PARM4 and PARM5 as appropriate if it can perform the conversion. If it returns r15 not 0, the NCL process is aborted.

Important! The exit is called from under the MAINTASK for your product region. It must not issue any O/S waits, because these will severely impact processing in your region.

A sample exit, PHONEX01, is provided. It illustrates the parameter list usage. It implements the same SOUNDEX algorithm as &NDBPHON SOUNDEX uses.

If the exit abends, your product region will abend.

&NDBQUOTE

Places quotes around data to protect special characters.

```
&NDBQUOTE data
```

The &NDBQUOTE built-in function allows an NCL procedure to protect data that contains special characters, that would otherwise cause premature truncation of the value, or syntax errors, when used on an &NDB*xxx* statement that accepts free-form text.

The rules that &NDBQUOTE uses are as follows:

- If the data contains any special characters, quoting is required.
- If the data commences with either a single (') or double (”) quote, quoting is required.
- If quoting is required, then, if the data contains no double quotes, place a double quote at each end. Otherwise, if the data contains no single quotes, place a single quote at each end. Otherwise, place a single quote at each end, and replace each embedded single quote by two single quotes.

The result of this operation is a single data entity that is preserved when processed by the free-form text parsing logic.

&NDBQUOTE is a built-in function and must be used to the right of an assignment statement.

Operands:

data

The data to be quoted. A null value is acceptable, and is reflected by a null value being returned by &NDBQUOTE.

Example: &NDBQUOTE

This example illustrates the rules &NDBQUOTE uses. The brackets are used to outline the new values.

```
&D1 = &STR AB C
&D2 = &STR AB & C
&D3 = &STR AB'C
&D4 = &STR A'B”C
&D5 = &STR 'AB C&
DQ1 = &NDBQUOTE &D1   -* &DQ1 = < AB C >
&DQ2 = &NDBQUOTE &D2  -* &DQ2 = < ”AB & C” >
&DQ3 = &NDBQUOTE &D3  -* &DQ3 = < AB'C >
&DQ4 = &NDBQUOTE &D4  -* &DQ4 = < 'A''B”C' >
&DQ5 = &NDBQUOTE &D5  -* &DQ5 = < ”'AB C” >
```

Notes:

Input data, for example, from a panel, should always be processed by &NDBQUOTE if it can contain special characters.

&NDBQUOTE will always handle the current list of special characters, so the NCL code will not need to be altered if a new release of NDB changes the list of special characters.

Although character format data may be up to 255 characters long, &NDBQUOTE overheads can reduce the effective length, due to the NCL restriction that no word can exceed 256 characters during substitution. For example, a 255 character string containing an ampersand (&) would require quoting, and thus become 257 characters long. This is too long for an NCL variable.

Should the data be too long to quote, the procedure will be terminated with an error message.

&NDBSCAN

Scans a NetMaster database (NDB) for all records matching a search argument.

```
&NDBSCAN dbname { [ SEQUENCE=result-list-name ]
                  [ KEEP={ YES | NO } ]
                  [ SORT=sort-expression ] ]
                  [ EXEC={ YES | NO } ]
                  [ OPTIMIZE={ NO | YES } ]
                  [ RETDEL={ NO | YES } ]
                  [ RETMSG={ NO | YES } ]
                  [ RETPOS={ NO | YES } ]
                  [ IOLIMIT=n ]
                  [ TIMELIMIT=n ]
                  [ STGLIMIT=n ]
                  [ RECLIMIT=n ]
                  { START | [ DATA ] S-EXP | END | CANCEL } }

S-EXP: [ SELECT * FROM ndbname [ correl-id ] WHERE ] EXP1

EXP1: EXP2 [ OR EXP2 ... ]

EXP2: EXP3 [ AND EXP3 ... ]

EXP3: [ NOT ... ] EXP4

EXP4: ( EXP1 ) | EXP5

EXP5: [ IGNORE { TRUE | FALSE } ]
        TEST1 | TEST2 | TEST3 | TEST4 | TEST5 | TEST6 | TEST7 | TEST8 | TEST9 | TEST10

TEST1: SEQUENCE sequence-name

TEST2: L-LIST PRESENT

TEST3: L-LIST ABSENT

TEST4: [ FIELDS ] fieldname IS [ NOT ] NULL

TEST5: [ FIELDS ] fieldname [ NOT ] BETWEEN value AND value

TEST6: EXISTS ( SUBSEL )

TEST7: [ FIELDS ] fieldname [ NOT ] IN { ( value [ , value ] ) | ( SUBSEL ) }

TEST8: L-LIST [ NOT ] LIKE R-LIST

TEST9: L-LIST  CONTAINS R-LIST

TEST10: L-LIST { = | ¬= | < | > | <= | >= }
               { R-LIST | [ ANY | ALL | SOME  ] ( SUBSEL ) }

SUBSEL: SELECT { fieldname [ : fieldname ] | prefix* }
```

```
                [ , ... ] FROM ndbname [ correl-id ] WHERE EXP1

L-LIST: [ ANY | ALL ] [ FIELDS ]
        { fieldname [ : fieldname ] | prefix* } [ ,  ... ]

R-LIST: [ ANY | ALL | SOME ]
        { [ VALUES ] { value { [ : value | GENERIC ] } } [ , ... ]|
            FIELDS [ ( correl-id ) ]
          { fieldname [ : fieldname ] | prefix* } [ , ... ]
          [ { PLUS | MINUS } number ] }
```

The &NDBSCAN statement is used to find all records in an NDB that pass a set of criteria, called a *scan-expression*.

- These criteria include such things as:
- A field or list of fields, that is equal to, not equal to(or other test) a given value, or list of values.
- A field or list of fields, that is equal to (or other test) another field, or list of fields, in the same record. For numeric or date format fields, an adjustment amount is specified.
- The presence or absence of a given field, or list of fields, in a record.
- The results of a previous &NDBSCAN, which is further filtered, combined with other scan results.
- Generic and range matches on values.

The resulting list of records can optionally be saved under a user-nominated result-list-name, for processing by &NDBGET.

To prevent a scan from using excessive system resources, you can optionally impose limits on the amount of these resources.

The fields referenced in the scan need not be keyed. The scan processing logic uses keys wherever possible, but will automatically switch to reading records whenever a non-keyed field is referenced. The only penalty is the number of I/Os, and the elapsed time.

Operands:

dbname

Specifies the name of the NDB that is to be scanned, and is mandatory. The NDB must have been previously opened by an &NDBOPEN statement.

SEQUENCE=*result-list-name*

An optional parameter, which provides a name for the result list. This name can then be used in an &NDBGET SEQUENCE=result-list-name statement to retrieve the records that passed the scan.

If this operand is omitted, no results are saved. The value of &NDBRC indicates the result of the scan, and the scan information variables are set. KEEP= and SORT= cannot be specified if this operand is omitted. The result-list-name must not already exist, either as a scan result list name or as the name of a SEQUENCE specified on an &NDBSEQ DEFINE statement.

If SEQUENCE is specified, the result-list-name is only saved if the scan completes and &NDBRC is set to zero. SEQUENCE is abbreviated to SEQ.

KEEP={ NO | YES }

This is an optional operand which is only valid if SEQUENCE is also specified. It indicates whether or not the result list is to be retained when an &NDBGET for the sequence returns an end-of-file (&NDBRC=2).

KEEP=NO indicates the result is to be deleted (this is the default). KEEP=YES indicates that the result list is not to be deleted, but is to be retained until explicitly deleted by &NDBSEQ DELETE, or by an &NDBCLOSE for this database.

SORT=*sort-expression*

This is an optional operand which is only valid if SEQUENCE is specified. It indicates the name of the field on which the final result will be sorted. Any field defined in the database is used, not just keyed fields.

The options for this operand are:

- omitted (no sort wanted)
- () (no sort wanted)
- *name* (entire field contents, ascending)
- *name*(*start*,*end*,*dir*)
- (*name1*(*start*,*end*,*dir*),*name2*(*start*,*end*,*dir*),...)

start is the start offset (* means 1). If *start* is omitted, 1 is assumed. It is *, or 1 for non-character fields.

end is the end position within the field. If *end* is omitted or specified as *, this means the end of the field. It must be omitted or * for non-character fields.

dir is the sort direction. It must be A (indicating ascending, the default) or D (indicating descending). Up to 7 sort fields is specified. If more than 1 is specified, the entire list must be enclosed in parentheses.

If the last sort field is descending, the RIDs of records with equal sort keys are also sorted descending. This is to ensure complete and correct up/down symmetry.

CAPS=SEARCH fields are uppercased when used as sort keys.

EXEC={ YES | NO }

This operand allows you to syntax check a scan request without executing it. If the scan expression (and the sort expression, if specified) has no errors, then the scan is executed, by default.

Specification of EXEC=NO means that no execution takes place. If there are no errors in the scan or sort expressions, &NDBRC will be set to zero.

OPTIMIZE={ NO | YES }

(Or OPTIMISE) This operand allows you to subset the optimization option in effect for the NDB for this scan.

You cannot turn on optimization if it is off at the NDB level.

RETDEL={ NO | YES }

This option (defaulting to NO for compatibility), if set to YES, will cause &NDBGET statements reading the resultant scan-built sequence to not skip over deleted records. Rather, the get will return an NDBRC of 1 (record not found), and the RID. This option will greatly simplify selection list processing of scan sequences.

RETMSG={ NO | YES }

RETMSG=YES specifies that all messages produced as a result of errors in the scan or sort expression, are to be returned in NCL variables with names &NDBMSG*n*, where n starts from 1.

This is completely independent of any &NDBCTL MSG= setting.

RETPOS={ NO | YES }

This option (defaulting to NO for compatibility), if set to YES, will cause &NDBGET statements reading the resultant scan-built sequence to set the new &NDBSQPOS system variable to the relative position in the scan sequence of the record just read. This will be a number from 1 to the number of records in the scan sequence.

If the RETDEL=YES option is also in effect, the &NDBSQPOS variable will be set when a delete record indication is returned (&NDBRC set to 1).

IOLIMIT=*n*

This operand allows the scan to be limited to processing a specified number of logical I/Os. A logical I/O corresponds to a VSAM request, not to physical disk I/O. A scan that exceeds this limit will be terminated with &NDBRC set to 5.

Omission of this operand, or coding it as IOLIMIT=0, or IOLIMIT=, causes the value of the SYSPARMS NDBDIOL setting to be used. If the value of n is greater than the SYSPARMS NDBMIOL setting, the SYSPARMS setting is used.

TIMELIMIT=*n*

This operand allows the scan to be limited to a specified elapsed time, expressed in seconds. A scan that exceeds this time will be terminated with &NDBRC set to 6.

Omission of this operand, or coding it as TIMELIMIT=0 or TIMELIMIT=, causes the value of the SYSPARMS NDBDTML setting to be used. If the value of n is greater than the SYSPARMS NDBMTML setting, the SYSPARMS setting is used.

STGLIMIT=*n*

This operand allows the scan to be limited to a specified amount of working storage, expressed in Kilobytes. A scan that exceeds this limit will be terminated with &NDBRC set to 7.

Omission of this operand, or coding it as STGLIMIT=0 or STGLIMIT=, causes the value of the SYSPARMS NDBDSTL setting to be used. If the value of n is greater than the SYSPARMS NDBMSTL setting, the SYSPARMS setting is used.

RECLIMIT=*n*

This operand allows the scan to be limited to a specified number of records that finally pass. If this limit is met or exceeded during the final phase of scan processing, the scan is terminated with &NDBRC set to 8.

Omission of this operand, or coding it as RECLIMIT=0 or RECLIMIT=, causes the value of the SYSPARMS NDBDRCL setting to be used. If the value of n is greater than the SYSPARMS NDBMRCL setting, the SYSPARMS setting is used.

RECLIMIT=1 is used if you only want to know whether any records at all have or have not passed the scan criteria, and you are not concerned with the exact number or specific IDs of any such records. &NBDSCAN RECLIMIT=1 will not set the &NDBRID variable.

By specifying RECLIMIT=1, the scan terminates with &NDBRC set to 8, and &NDBRID set to 0, when one passing record is found.

START

This operand indicates the start of a multi-statement &NDBSCAN request. The scan-expression will be built up by one or more &NDBSCAN DATA statements.

The SEQ, KEEP, SORT, and LIMIT operands must all be specified on the START statement. The START operand must be the last operand specified on the statement.

DATA

This operand indicates that a scan-expression (for a single-statement scan), or part of a scan-expression (for a START/DATA/END multi-statement scan), follows. This operand is omitted, but inclusion will prevent any syntax errors if the scan-expression contains any &NDBSCAN operands (for example, START or END).

END

This operand indicates the end of a multi-statement scan. The entire scan-expression is concatenated together and passed to the database for processing.

CANCEL

This operand indicates that a partially-built multi-statement scan is to be canceled. This operand is valid even if no current &NDBSCAN START/DATA/END set is active for this database.

Examples:

The following example will find all records with the field DOB present and less that JAN 1st,1960. The list of matching records is saved in the sequence called RESULT which can then be read using &NDBGET SEQ=RESULT.

```
&NDBSCAN MYNDB SEQ=RESULT SORT=SURNAME +
   DATA DOB LT 600101
```

The next example will find, in the database PROBLEMS, all records with the field DATEOCUR less than today (&DATE7 is the current date in YYMMDD format), that are either not closed (&DATECLOS ABSENT) or were closed later that 5 days after opening.

```
&NDBSCAN PROBLEMS SEQ=S1 SORT=DATEOCUR START
&NDBSCAN PROBLEMS DATA DATEOCUR LT &DATE7
&NDBSCAN PROBLEMS DATA AND (DATECLOS ABSENT
&NDBSCAN PROBLEMS DATA OR DATECLOS GT FIELD DATEOCUR PLUS 5)
&NDBSCAN PROBLEMS END
```

The next example will determine if there are any records on MYNDB with the field SURNAME equal to JONES. If SURNAME is keyed, an &NDBGET statement would work, and be more efficient. However, the &NDBSCAN statement works regardless of the keying or otherwise, and allows more complex expressions to be specified.

If there are no records with NAME=JONES, &NDBRC will be set to 4. If there is at least one record, the scan will terminate, and &NDBRC will be set to 8. &NDBRID will contain the RID of the first record that the scan found.

Note: No assumption is made about the number of other records that may have passed the scan.

```
&NDBSCAN MYNDB RECLIMIT=1 DATA SURNAME = JONES
```

The next example builds a list of all records in the PERSONEL NDB where all the fields starting with REVIEW are equal to POOR and all fields starting with ATTITUDE are also equal to POOR.

```
&NDBSCAN PERSONEL SEQ=SACKINGS START
&NDBSCAN PERSONEL DATA ALL REVIEW*= 'POOR' AND
&NDBSCAN PERSONEL DATA ALL ATTITUDE*= 'POOR'
&NDBSCAN PERSONEL END
```

Notes:

&NDBSCAN is an extremely powerful verb. The SYSPARMS command NDB*xxx* operands allow the setting of default and maximum limits for I/O, storage, and matching records, which you can use to control &NDBSCAN.

You need never be concerned whether fields are keyed or non-keyed when coding an &NDBSCAN statement. The only difference will be in the number of I/Os generated and the elapsed time.

&NDBCTL SCANDEBUG=YES displays the generated action table which shows whether record-level scanning is required.

&NDBSCAN requests run asynchronously from other database requests. The SYSPARMS NDBSUBMN and NDBSUBMX operands allow you to specify minimum and maximum subthreads, which handle scan requests, for any one NDB. Too many concurrent scan requests can impact other product region users. Too few concurrently allowed scan requests can cause a backlog of scans.

More information:

&NDBGET (see page 472)
&NDBSEQ (see page 508)

Comments on Syntax

Following are comments on each section of the scan syntax.

S-EXP

Is the initial part of the syntax. For compatibility with SQL, an optional SELECT clause is specified. If desired, the FROM sub-clause can specify an overriding correlation ID for this expression, that is used on subselects to form correlated queries. The NDB name is assumed in the following situations:

- *correl-id* is omitted.
- SELECT clause is omitted.

Note: *ndbname* must be the current NDB name.

Following the optional SELECT clause, a scan expression must be specified, represented by EXP1.

correl-id must be from 1 to 8 characters, in PDSNAME format. The ID does not need to be unique. It cannot be the value WHERE. The value '*' is also acceptable, meaning 'current NDB'.

EXP1

Is the top level of an expression or parenthesized part of an expression. The OR (|) connector connects any number of EXP2 sub-expressions.

EXP2

Is the level of expression or parenthesized part of a sub-expression that allows the connection of any number of EXP3 sub-expressions by use of the AND connector. An ampersand (&) can also represent AND.

Note: AND binds tighter than OR.

EXP3

Is the part of the syntax that shows the NOT connector used to negate parts of the expression. Two adjacent NOT connectors cancel each other out. A NOT sign (¬) represents NOT.

EXP4

Is the part of the syntax that shows a parenthesized expression used to override precedence rules. The depth of parenthesis nesting has no limit.

EXP5

Is the part of the syntax that shows that a test is one of ten varieties. These tests are described next.

The optional IGNORE clause allows you to ignore the test completely and treat it as true (IGNORE TRUE) or false (IGNORE FALSE). This feature is useful when the IGNORE clause is inserted dynamically into a complex scan expression.

Note: The IGNORE clause is only supported if the quoted data option is in effect.

TEST1

Feeds in the result list of a previous &NDBSCAN as a list of matching records.

TEST2

Tests whether a field (or list of fields) is PRESENT in a record. See L-LIST for specification of the field list.

TEST3

Tests whether a field (or list of fields) is ABSENT (that is, not present) in a record.

TEST4

Tests a field for presence (IS NOT NULL) or absence (IS NULL) in a record. It is the ANSI SQL version of the presently existing PRESENT and ABSENT operators. No field lists are supported, but the FIELDS keyword is supported.

TEST5

Tests a field for (not) being within the (inclusive) range of two values. It is the ANSI SQL version of the presently supported *field* = *value* : *value* syntax. Only a single field is supported (no lists or ranges). Similarly, no lists or ranges are allowed. The FIELDS keyword is supported.

TEST6

Tests for a nonempty set of records that pass a subselect. This test always evaluates to TRUE (at least one record) or FALSE (no records).

TEST7

Tests for set membership. The format where a list of values is supplied, is equivalent to the presently supported list of values for the equal (=) or not equal (¬=) operators. The format where a subselect is used is new. No field name lists are supported, but the FIELDS keyword is supported.

Note: If the quoted data option is not in effect, a subselect is not recognized here. The SELECT keyword could be a valid data value for a character field.

TEST8

Is the pattern match test, with NOT LIKE list capabilities. LIKE uses the ANSI SQL pattern match wildcards:

% (percent sign)

Matches 0 or more characters.

_ (underscore)

Matches exactly one present character.

Note: The R-LIST supports correlated field references and is supported for the [NOT] LIKE operators.

TEST9

Is the CONTAINS operator. The operator cannot have a subselect on the right.

Note: R-LIST correlated field references are acceptable.

TEST10

The standard relational operator syntax. NDB allows list of fields on the left, and lists of either fields or values on the right.

If a subselect is specified, then no left side lists are permitted.

Note: The SOME keyword is only permitted and recognized if the quoted data option is in effect.

SUBSEL

Is the nested selection syntax. The values of the nominated fields of the records that pass the scan expression are used in the containing scan expression. The EXISTS test is an exception, where the fact that at least one record passes the subselect is the only thing that matters. The nominated fields must be type-compatible with the other fields in the containing subselect expression.

The FROM clause is required, and an optional correlation ID (*correl-id*) is specified, overriding the default of the NDB name. Correlation IDs do not need to be unique, but should be.

L-LIST

Is a list of fields to test. Lists of fields can include field name ranges or generic prefixes. With the exception of the PRESENT and ABSENT operators, all listed fields must be of the same type. The new ANSI SQL compatible operators do not support lists of fields. (Although they do support the FIELDS keyword, so that an unambiguous test is defined if you have a *fieldname* of FIELDS).

Each range or generic specification must match at least one field on the NDB. Overlapping ranges or generic specifications are allowed. The duplicate fields are ignored and the resulting internal list has each field only once.

R-LIST

Is a list of fields or values to test. Lists of values can include value ranges or generic specifications (for the equal (=) or not equal (¬=) operators only). Lists of fields can include (for numeric and DATE format fields only) an optional adjustment value. Lists are not supported for the ANSI SQL operators.

If an optional *correl-id* is inserted before a field list, then, if the specified *correl-id* is not the same as the containing select/subselect assigned *correl-id*, the test is a correlated test. The *correl-id* applies to all fields in the following field list. The ID can only be specified once per R-LIST, immediately after the FIELDS keyword. In this case, no adjustment value is allowed (the PLUS/MINUS clause).

ANY is the default for all operators except the ALL NOT EQUAL (p=) operator, which defaults to ALL.

Scan Processing

A scan is processed in four phases:

1. The scan-expression is parsed, and an action table built.
2. The action table is processed, using keys wherever possible, to obtain a list of records that (might) pass. If nothing in the table is processed using keys, the whole database is regarded as passing phase 2.
3. The final result list is built. If the entire expression is evaluated using keys, and the SORT= operand was not specified, phase 3 merely builds the result list.

   If the expression involved non-keyed fields, or certain operators (for example CONTAINS), all records on the list from phase 2 are read, and processed against the action table, to determine whether they pass.

   If the request included a SORT field, phase 3 will build a sort list for all records that pass. This list contains the sort fields for each record that passes.
4. If SORT was requested, the sort is performed, and the final result list is built.

Note: Optimization of the request can cause some of phase 2, 3, or 4 to be bypassed. For example, if there were no key fields in the scan, but the SORT key was the SEQUENCE key, the entire file would be read in that order, obviating the need for the actual SORT phase.

If the scan completes, and at least one record is found, &NDBRC will be set to 0, and the NCL variables will be set.

If the scan terminates without finding any records, &NDBRC will be set to 4,and the NCL variables will be set.

If the scan terminates because it exceeds the I/O, TIME, STORAGE or RECORD limits set (explicitly on the statement, or implicitly by the SYSPARMS NDB*xxx* settings), response codes 5 (I/O), 6 (TIME), 7 (STORAGE), or 8 (RECORD) will be set, and the NCL variables will be set, as indicated on page 2-432. In this case, the system variable &NDBRID might or might not be set to the first RID that the scan passed (not necessarily the lowest valued sort key record, if SORT).

If SEQUENCE is specified, the result-list-name is only saved if the scan completes and &NDBRC is set to 0.

The following NCL variables are set by a scan that terminates successfully, or unsuccessfully with a warning response:

&NDBSCANNRECS

Contains the number of records that passed the scan. When the scan terminates with a limit exceeded warning, this value might be non-zero if the scan terminated in phase 3 or later. If it is non-zero, the system variable &NDBRID will contain the first RID that passed the scan.

&NDBSCANIOCNT

Contains the number of logical I/Os performed by the scan. If the scan terminated on an I/O limit, this count will be 1 greater than that limit.

&NDBSCANTIME

Contains the elapsed time taken by the scan, accurate to one hundredth of a second, in the format SECONDS.TH. If the scan terminated on a TIME limit, the value will be marginally greater than that limit.

&NDBSCANSTG

Contains the maximum working storage used by the scan, expressed in Kilobytes. This working storage is used to hold intermediate results lists, sort fields, and so on. If the scan terminated on a STORAGE limit, this value will be marginally greater that the limit.

Logic

When comparisons are being performed, special logic is used to prevent records that do not have a given field present passing a test. The logic is called three-valued logic.

This example illustrates three-valued logic: There are 100 records on a database, 10 records with the NAME field present, containing the value SMITH, 65 records with the NAME field present, containing some other value, and the remaining 25 records having no name fields, then:

- The test NAME = SMITH will match the 10 records which have SMITH in the NAME field.
- The test NAME NE SMITH will match the 65 records with the NAME field present, but not equal to SMITH.
- The remaining 25 records are the third value, that is, they are not 'equal to SMITH' and they are not 'not equal to SMITH' (because the NAME field is not in the record).

It is this three-valued logic that allows multiple-disjoint record types to share an NDB.

To explicitly match records with a given field PRESENT or ABSENT, operators of the same name are provided.

The null-field result propagates throughout the scan-expression. This means that the expression NOT (NAME EQ SMITH OR NAME NE SMITH) will not match all records with the NAME field absent. The result will always be no records. Only the ABSENT operator will match the records with the NAME field absent.

The &NDBCTL SCANDEBUG=YES option causes the action table to be displayed. The table shows the NULL-Field actions on each line, as well as the number of records passing each test.

Correlated Subselects

Subselects are implemented for both non-correlated and correlated queries.

A non-correlated query (select) is one where the search values for the tests in the query are either constant (for this scan-they could be supplied from user variables when the scan statement was executed, but they are constant for the duration of the statement), or are other field values in the same record (the previously supported field to field comparison).

In this case, each query (select) or subquery (subselect) is logically executed once only, and the result used as input to a higher-level query (select) or as the result of the scan.

A correlated query (select) is one where at least one field to field test is done in the expression, but the right-hand-side field is qualified by a correlation-id that is not the correlation-id of the current SELECT, but is a correlation-id of a parent (higher level) SELECT. In this case, each time the correlated subselect result is needed (it cannot be the primary select as it has no parent), the entire subselect (and possibly sub-subselects) will be reevaluated, using as test arguments the current values of the relevant fields of the currently considered parent record. A nested loop results.

For purposes of deciding on use of keys, a correlated test can use keys, as at the time the subselect is executed the supplied search values (parent field values) are constant.

The fields referenced in the scan need not be keyed. The scan processing logic will use keys wherever possible, but will automatically switch to reading records whenever a non-keyed field is referenced. The only penalty is the number of I/Os, and the elapsed time.

&NDBSEQ

Defines, deletes, or resets a sequential retrieval path for a NetMaster database (NDB). Histograms (statistical information) is retrieved for keyed fields.

```
&NDBSEQ dbname { DEFINE SEQUENCE=seqname { RID | FIELD=fieldname | KEY=fieldname }
                [ FROM=value ] [ TO=value ]
                [ VALUE=value ] [ GENERIC=value ]
                [ KEEP= { NO | YES } ] |
                   DELETE SEQUENCE= { seqname | * } |
                   RESET SEQUENCE=seqname
                [ REPOS=value | RID=n | RELPOS=n ] }
```

The &NDBSEQ statement allows an NCL procedure to define, delete, or reset a sequential access path to an NDB.

A sequential path is defined to be by RID, or by any key field defined in the database. The bounds of the path can also be defined.

An NCL procedure can have any number of currently defined sequences. Positioning is maintained independently for each.

The &NDBSCAN statement can also define a sequence, as the result from a scan. The &NDBSEQ statement is used to delete or reset an &NDBSCAN-defined sequence.

Operands:

dbname

Specifies the name of the NDB that you wish to define, delete, or reset a sequence in. This operand is mandatory. The NDB named must have been previously opened by an &NDBOPEN statement.

DEFINE

This operand indicates a new sequence is being defined. SEQUENCE=*seqname* must be coded.

SEQUENCE=*seqname*

This operand, for DEFINE, names the new sequence, and, for RESET, names an existing sequence that is to be reset. *seqname* is a 1- to 12-character name, the first being alphabetic or national, the remainder alphanumeric or national. Sequence names are unique to an NCL procedure. For DEFINE, *seqname* must not currently exist, including any scan result lists.

RID

This operand indicates that the sequence being defined is to be by record ID (RID). This parameter is mutually exclusive with the FIELD=*fieldname* parameter. The VALUE=*value* and GENERIC=*value* parameters cannot be used for a sequence by RID.

FIELD=*fieldname*

This parameter indicates that the sequence being defined is to be by the named key field.

KEY=*fieldname*

This option indicates that a keyed field histogram sequence is wanted. *fieldname* must be keyed, and not a sequence key. The resulting sequence is read using &NDBGET on the defined sequence. Rather than return the associated record (as gets on SEQ FIELD= does), the field value is retrieved, and the count of records having that value is also returned.

When retrieving, the FORMAT operand of GET is ignored. It must be specified to satisfy GET syntax, but (for example) FORMAT NO or FORMAT * is specified.

The returned key field value is always returned in &NDBKEYVALUE. The returned record count is always returned in &NDBKEYCOUNT. The &NDBRID system variable is always set to 0.

FROM=*value*

This parameter indicates the (inclusive) starting value of either RID, or the named key field, for the sequence. The provided value must be valid for the field format, or a valid number from 1 to 1 billion for RID. The value is quoted, if required.

If this parameter is coded, the GENERIC and VALUE parameters cannot be specified.

Omission of this parameter, as well as the GENERIC and VALUE parameters (for *FIELD=fieldname*) implies the lowest possible value for the field format (or RID).

TO=*value*

This parameter indicates the (inclusive) ending value of either RID, or the named key field, for the sequence. The provided value must be valid for the field format, or a valid number from 1 to 1 billion for RID. The value is quoted, if required.

If this parameter is coded, the GENERIC and VALUE parameters cannot be specified.

Omission of this parameter, as well as the GENERIC and VALUE parameters (for *FIELD=fieldname*) implies the highest possible value for the field format (or RID).

VALUE=*value*

This parameter indicates the FROM and TO values are to be the same value, as specified. This is useful when reading by a non-unique key, top obtain all records with a given equal key value.

Note: This is not the same as GENERIC=value. If this parameter is coded, the GENERIC, FROM, and TO parameters cannot be coded.

This parameter is invalid for a sequence by RID.

GENERIC=*value*

This parameter indicates a sequence of all records with the nominated key field generically equal to the passed key value.

If this parameter is coded, the VALUE, FROM, and TO parameters cannot be coded.

This parameter is invalid for a sequence by RID, or for a sequence by a NUM or DATE format field.

KEEP= { NO | YES }

This parameter indicates whether the defined sequence is to be kept when an &NDBGET returns an end-of-file condition (&NDBRC=2).

KEEP=NO (the default) indicates the sequence is to be deleted at EOF.

KEEP=YES indicates the sequence is to be retained until explicitly deleted, or until an &NDBCLOSE for the database. The &NDBSEQ RESET statement is useful in this case.

DELETE

This parameter indicates an existing sequence, or all defined sequences for this database, is to be deleted.

SEQUENCE={ *seqname* | * }

The name of the sequence to delete, if present, or all sequences for this database, if '*' is coded.

RESET

This parameter indicates an existing sequence is to be reset, which removes any current end-of-file, and, optionally, positioned to a particular place within the sequence.

REPOS=*value*

This optional parameter allows a sequence to be positioned to a particular place. The next &NDBGET will retrieve the record with the key field (or RID) equal to, or nearest to (depending on the direction), the REPOS value. This value must be in a suitable format for the defined sequence field, or RID.

For sequences created with &NDBSCAN, this parameter is only valid if the scan had

SORT=*fieldname* specified. In this case, *fieldname* sets the format of the REPOS value.

RID=*n*

This optional parameter allows a sequence built by &NDBSCAN only to be repositioned to a specific RID, if it is in the result list. If not, response 1 will be returned.

RELPOS=*n*

This optional operand allows a sequence to be positioned to relative record n. n must be a number in the range from 1 to the number of records in the sequence.

Examples: &NDBSEQ

The following example defines a sequence that is used to read the entire database sequentially, in RID order.

```
&NDBSEQ MYNDB DEFINE SEQUENCE=S1 RID
```

The next example defines a sequence that is used to read all records on the database with field SURNAME equal to the value SMITH.

```
&NDBSEQ MYNDB DEFINE SEQ=S2 FIELD=SURNAME VALUE='SMITH'
```

The next example defines a sequence that is used to read all records on the database with field SURNAME generically equal to the value SMITH. This would include SMITHSON, SMITHE, and SMITH-WADDINGTON, for example.

```
&NDBSEQ MYNDB DEFINE SEQ=S3 FIELD=SURNAME GENERIC='SMITH'
```

The next example could be used to reset the sequence S2, defined previously, if part-way through reading all the SMITH records, to allow restarting from the beginning (or end, if &NDBGET DIR=BWD is used).

Note: If an end-of-file response had been returned, the reset would fail, as the sequence definition did not specify KEEP=YES, and thus would have been deleted.

```
&NDBSEQ MYNDB RESET SEQ=S2
```

Notes:

Errors encountered whilst processing the &NDBSEQ statement may cause the procedure to terminate, or may just be reflected in the &NDBRC system variable, depending on the setting of &NDBCTL ERROR option.

A successful define, delete, or reset will set &NDBRC to 0.

More information:

&NDBGET (see page 472)
&NDBSCAN (see page 492)

Sequential Retrieval

Sequential database retrieval allows an NCL procedure to maintain any number of concurrent sequential retrieval paths into an NDB.

Each path is in one of the following four states:

- Outside the sequence range. This is the default for a newly defined sequence, or one that is RESET (with no REPOS=/RID=).
- Positioned on a record within the sequence range. The last record returned by an &NDBGET for the sequence is the current position, and is reread by &NDBGET SKIP=0.
- At back-EOF. An &NDBGET DIR=FWD has encountered an end-of-file condition and set back-EOF.

  Note: Any further forward gets will continue to receive the EOF response (assuming the sequence was defined with KEEP=YES. If not, further gets of any kind would receive an error response).
- At front-EOF. AN &NDBGET DIR=BWD has encountered an end-of-file condition and set front-EOF.

  Note: Any further backward gets will continue to get the EOF response, subject to the previous comments.

When at an EOF, a get in the opposite direction will reset the EOF and recommence the sequence.

The first &NDBBGET after a sequence is defined, or after a RESET with no REPOS=/RID= sets the correct end to start from (front if FWD, back if BWD).

An &NDBSEQ RESET REPOS=value with a value outside the defined range will set the relevant EOF, or position to the first (or last) record in the range, depending on the direction of the next &NDBGET.

Note: Sequences have an implicit forward direction of ascending key field or RID values. For this reason, the nominated TO value must not be less than the nominated FROM value. To retrieve records in descending sequence, the &NDBGET DIR=BWD option is used.

&NDBUPD

Updates a record in a NetMaster database (NDB).

```
&NDBUPD dbname
    { [ RID=rid ] [ DATA ] update-text |
        RID=rid START | END | CANCEL |
        RID=rid FORMAT=fmtname [ FSCOPE={ PROCESS | GLOBAL } ] }
```

update-text is:

```
fieldname = fieldvalue [ fieldname = fieldvalue ... ]
```

The &NDBUPD statement allows an NCL procedure to update an existing record in an NDB. The existing record will have the fields listed in the &NDBUPD statement set to the new values, and all other fields will be left as is. Following completion of the statement, the system variable &NDBRC will indicate success or failure if the update.

Operands:

dbname

The name of the NDB that you wish to update the record in is a mandatory operand. This NDB must have been previously opened by an &NDBOPEN statement.

RID=*rid*

A required parameter, if a single-statement update, or if the START statement of a multi-statement update. Rid is the Record ID of the record you want to update.

DATA

Indicates that free-form text (see page 442) follows. This operand is optional, but it is recommended, as it prevents any ambiguous meaning of a field name or field value of DATA, START, END, or CANCEL.

START

Indicates the start of a multi-statement &NDBUPD. The statement must end after the START keyword. The RID=*rid* parameter must be specified on this statement.

END

Indicates the end of a multi-statement &NDBUPD. This statement will call the database, passing the concatenated &NDBUPD DATA information.

CANCEL

Indicates an active &NDBUPD START/END set is to be canceled. If there is no active &NDBUPD START/END for this database, the statement is ignored.

FORMAT=*fmtname* [FSCOPE={ PROCESS | GLOBAL }]

FORMAT=*fmtname* specifies that output format *fmtname*, defined on the &NDBFMT statement, is to be used.

The nominated format must exist in the nominated scope. PROCESS is the default and means a format defined by the current NCL process. GLOBAL indicates a format is to be found in the global format pool for the NDB.

If this operand is specified, the START, DATA, CANCEL, and END operands are not allowed.

***fieldname* = *fieldvalue* ...**

Free-form text naming the fields to be updated in the nominated record, and the new values for those fields. There is as many *fieldname* = *fieldvalue* pairs as desired, and they may be split across multiple statements, using START/DATA.../END.

If *fieldvalue* is a null variable (for example, &NULL =/ &NDBADD ... X = &NULL), the null variable will be passed to the database as a null indicator, indicating the relevant field is to be set not present. This is not the same as present, with a null value (for example, 0 for a numeric field).

For example, the following sets FIELD1 to a value, FIELD2 to present, with a null value, and FIELD3 not present.

```
&FIELD1 = value              -* set to a value
&FIELD2 = &SETBLNK 1         -* set to a blank
&FIELD3 =                    -* set null
&NDBUPD MYNDB RID=&RID DATA FIELD1 = &FIELD1 +
                            FIELD2 = &FIELD2 +
                            FIELD3 = &FIELD3
```

Omission of a field name and its accompanying value means that the field stays as it was in that record.

Examples: &NDBUPD

The following example sets the value of field FIRSTNAME in the record with the RID in &SAVERID to JOHN. All other fields in the record are left as they were.

```
&NDBUPD MYNDB RID=&SAVERID DATA FIRSTNAME='JOHN'
```

The next example updates the first record in the NDB called MYNDB with field SURNAME equal to JONES, altering the values of fields DOB and FIRSTNAME. It illustrates how the free-form text is split at any point where a blank is valid.

```
&NDBGET MYNDB FIELD=SURNAME VALUE=JONES
&NDBUPD MYNDB RID=&NDBRID START
&NDBUPD MYNDB DATA DOB =
&NDBUPD MYNDB DATA FIRSTNAME = 'MARK'
&NDBUPD MYNDB END
```

Notes:

Errors encountered while processing the &NDBUPD statement can cause the procedure to terminate, or may just be reflected in the &NDBRC system variable, depending on the setting of &NDBCTL ERROR option.

If the record is updated successfully (&NDBRC is 0 after the single-statement &NDBUPD, or after the &NDBUPD END for a multiple-statement update), the system variable &NDBRID will contain the record id of the updated record.

At least one *fieldname* = *fieldvalue* must be specified to successfully update a record (although the value may be the null indicator).

More information:

&NDBADD (see page 445)
&NDBDEL (see page 462)

&NPFxCHK

Returns a value indicating the current user's NPF authorization for access to a resource.

```
&NPFDCHK resource [ resource2 .... resourcen ]

&NPFMCHK resource [ resource2 .... resourcen ]

&NPFVCHK resource [ resource2 .... resourcen ]
```

The Network Partitioning Facility (NPF) restricts the range of network resources a user may reference. The &NPFDCHK, &NPFMCHK, and &NPFVCHK statements are built-in functions that let you test a user's authority to reference a particular resource. Used with &INTCMD and &INTREAD, they let you edit multi-resource displays from VTAM to display only the resources to which that user is allowed access:

&NPFDCHK

Determines whether the user is entitled to use the resource nominated by a VTAM display command, for that procedure.

&NPFMCHK

Determines whether the user is entitled to receive messages relating to the resource nominated, for that procedure.

&NPFVCHK

Determines whether the user is entitled to use VTAM VARY and MODIFY commands in the resource nominated, for that procedure.

Each of these verbs is a built-in function and must be used to the right of an assignment statement.

If the user is authorized for the resource, a YES value is returned in the variable specified, to the left of the assignment statement.

If the user is not authorized, a NO value is returned.

Multiple resources is tested by a single statement-the user must be authorized for each of the specified resources. If the user is not authorized for one or more of the resources, a NO value is returned.

Operands:

resource

The VTAM network name for the resource to be tested. This can optionally be qualified by a network identifier; for example, NET1.TERM001.

Examples: &NPFVCHK

```
&AUTH = &NPFVCHK NODE1
&IF &AUTH EQ NO &THEN +
   &ENDAFTER &WRITE ALARM=YES DATA=NOT AUTHORIZED

= &CONTROL NOLABEL
&INTCMD D &NODE
.READ
   &INTREAD ARGS
   &GOTO .&1
   &WRITE ALARM=YES DAT=UNEXPECTED MESSAGE &1
   &END
.IST nnn I
   -*
   -* Processing
   -*
   &GOTO .READ
.IST nnn I
   &AUTH = &NPFDCHK &7
   &IF &AUTH = NO &THEN +
   &GOTO .READ
   &WRITE &1 &2 &3 &4 &5 &6 &7 &8 &9
   &GOTO .READ
.IST314I
 &END
```

Notes:

The maximum length resource name that is specified is 64 characters.

Using &NPF*x*CHK functions in procedures executed by a user with no NPF restrictions always returns a YES value.

The &RSCCHECK function can also be used to perform authorization verification.

&NPFDCHK is equivalent to &RSCCHECK $NMCMDD.

&NPFMCHK is equivalent to &RSCCHECK $NMMSG.

&NPFVCHK is equivalent to &RSCCHECK $NMCMD.

More information:

&RSCCHECK (see page 578)

&NRDDEL

Deletes one or more non-roll deletable messages.

```
&NRDDEL domid [ domid2 ...  domidn ]
```

Non-roll deletable (NRD) messages issued to OCS windows is hidden by the OCS operator (by placing the cursor on the NRD message and pressing Enter), but not deleted. These messages can only be deleted by the &NRDDEL statement executed within an NCL procedure.

NRD messages may be sourced externally (for example, from the AOM system component), or by using the &WRITE NRD=YES statement.

Operands:

domid

The delete operator message identifier designating the message that is to have its non-roll deletable status removed. The DOM ID for a message is obtained from system variable &ZDOMID (after an &WRITE NRD=YES statement generating a NRD message), or from &ZMDOMID (when a message is received by MSGPROC which has an NRD attribute).

Each DOM ID specified on the &NRDDEL statement refers to one message.

Examples: &NRDDEL

```
&NRDDEL &M1 &M2 &M3
```

Notes:

Messages hidden by an OCS operator pressing the Enter key are invisible only to that operator and may be redisplayed by using the NRDRET command.

Messages from an NCL procedure using &WRITE NRD=YES are automatically deleted when the procedure terminates - use &WRITE NRD=OPER to retain the messages you generate. These messages have the general operational characteristics of NRD messages but do not have an associated DOM ID.

&NUMEDIT

Returns a number in a specified format.

```
&NUMEDIT (xx,yy,zz) number
```

&NUMEDIT is a built-in function and must be used to the right of an assignment statement. After real number arithmetic calculations, the result is present in one or more variables held in a mathematical form. For example, 22.1 occurs as:

```
+.221000000000000E+02
```

&NUMEDIT lets you reformat this representation into a standard decimal number, so that it displays with the required number of decimal positions.

Operands:

xx

Specifies the number of characters reserved for the integer portion of the result (that is, the section of the number to the left of the decimal point). The range is 0 to 15, and, if this operand is not specified, it defaults to 5. If 0 is specified, it defaults to 1.

If *xx* is greater than the number of characters in the integer portion, then the variable is padded with blanks to the left of the leading digit. If *xx* is less than the number of digits in the integer portion of the number, then truncation does not occur. The entire integer portion is preserved. This might cause misalignment if you are displaying columns of figures.

yy

Specifies the number of significant decimal positions to be preserved; that is, the number of digits preserved to the right of the decimal point. The range is 0 to 15, and, if this operand is not specified, it defaults to 2. If 0 is specified, an integer value is returned.

If *yy* is greater than the number of significant decimal positions, then the variable is padded with zeros to the right of the last digit. If *yy* is less than the number of significant decimal positions, then the result is truncated or rounded (as indicated by the *zz* operand) to *yy* positions.

zz

Indicates whether the result is truncated or rounded and whether the exponent format is maintained. Valid values for *zz* is as follows: 0 The default, meaning truncation and no exponent required R Meaning rounding and no exponent required E Meaning truncation and exponent required ER Meaning rounding and exponent required

If you specify E, then the number is presented with a single digit to the left of the decimal point, then the number of significant decimal places defined by *yy*, and the exponent to the right of the least significant decimal position.

For example, suppose the variable &RESULT contains the real number 38.9 with the format:

```
+.389000000000000E+02
```

After executing the statement:

```
&A = &NUMEDIT (0,4,E) &RESULT the value of &A is 3.8900E+01
```

number

The variable with the real number or integer that is to be reformatted.

Real numbers are manipulated as base 16 floating point numbers, so there is no exact representation for some decimal values. This can cause an apparent loss of precision in the 15th significant digit if more than 15 decimal digits were supplied as input.

Examples: &NUMEDIT

```
&RESULT = ( 27.993 * 4.882 )  -* &RESULT is set to
                              -* +.136661826000000E+03
&A = &NUMEDIT (0,5,0) &RESULT -* &A becomes 136.66182
```

&OVERLAY

Returns a string that has been overlaid by a supplied string.

```
&OVERLAY target source
       [ start [ length [ ALIGNL [c] | ALIGNR [c] | ALIGNC [lr] ] ] ]
```

&OVERLAY is a built-in function and must be used to the right of an assignment statement.

&OVERLAY allows data from one string (source) to be used to replace data in a second (target) string.

Operands:

target

The constant value for the target string, or the name for a variable holding the target string.

source

The source string, or the name for the variable holding the source string, which is used to overlay the target string.

start

The location within the target string where the overlay process starts. If omitted, overlay starts at the first position in the target variable. If the start location lies beyond the end of the target variable, the target is padded with blanks up to the start location.

length

The string length for which the overlay process is to occur. If longer than the source variable, blanks are used to complete the overlay operation. If omitted, the whole of the source variable is used to overlay the target string.

ALIGNL*c*

If the overlay length exceeds the length of the source string, this option specifies that the source string is put to the left of the overlaid area, and the remainder of the overlaid area is to be padded to the right with the character *c*. If *c* is omitted, blanks are used as filler.

ALIGNR*c*

If the overlay length exceeds the length of the source string, this option specifies that the source string is placed to the right of the overlaid area and the remainder of the overlaid area is padded to the left with the character *c*. If *c* is omitted, blanks are used as filler.

ALIGNC*lr*

If the overlay length exceeds the length of the source string, this option specifies that the source string is placed at the center of the overlaid area and the remainder of the overlaid area is padded to left and right with the (left) character *l* and the (right) character *r*. If *lr* is omitted, blanks are used as filler.

Examples:

```
&A = AAAAA
&B = BBB &1 = &OVERLAY &A &B     -* &1 will be set to BBBAA
&1 = &OVERLAY &A &B 2            -* &1 will be set to ABBBA
&1 = &OVERLAY &A &B 2 1          -* &1 will be set to ABAAA
&1 = &OVERLAY &A &B 1 4          -* &1 will be set to BBB A
&1 = &OVERLAY &A &B 7 3          -* &1 will be set to AAAAA BBB
&C = ABCDEFGHIJK
&D = 111
&1 = &OVERLAY &C &D 1 7 ALIGNL0  -* &1 will be set to
                                 -* 1110000HIJK
&1 = &OVERLAY &C &D 1 7 ALIGNR0  -* &1 will be set to
                                 -* 0000111HIJK
&1 = &OVERLAY &C &D 1 7 ALIGNC<> -* &1 will be set to
                                 -* <<111>>HIJK
```

Notes:

Variable substitution is performed before processing the &OVERLAY statement. If a variable has a null value when substitution is performed, it is eliminated from the statement. For this reason, take care to ensure that all values are specified as desired.

&OVERLAY is used to set a repeated value. For example, to set a character string of 64 Xs:

```
&X64 = &OVERLAY X X 1 64 ALIGNRX
```

&OVERLAY is useful for string manipulation and the management of data in tabular formats.

If &CONTROL DBCS or DBCSN or DBCSP is in effect, &OVERLAY is sensitive to the presence of DBCS data (see page 1201).

&PANEL

Displays the specified full screen panel.

```
&PANEL [ NAME= ] panelname
       [ TYPE=SYNC | ASYNC ]
       [ CDELAY=YES | NO ]
       [ MODALL=NO | YES]
```

The &PANEL statement is used to request the display of a full-screen panel. If necessary, a full-screen environment is established for the NCL process.

The full-screen panel must have been previously defined in a panel library using the MODS : Panel Maintenance function. On catering for undefined panels, see Notes for this verb.

An &PANEL statement is used from any NCL procedure operating within a user processing region associated with a full-screen terminal.

Use of the &PANEL statement from a procedure not operating in a region that supports full-screen mode will result in an error.

Before the specified panel is displayed, it is scanned and system and user variables are substituted. Thus, user variables required by the panel must be created within the NCL procedure before issuing the &PANEL statement.

On completion of entry by the terminal operator, control will be returned to the NCL procedure statement following the &PANEL statement. Data entered by the terminal operator will be available in the user variables defined in the panel definition.

Operands:

NAME=*panelname*

The 1- to 12-character name of the panel. This panel must exist in a panel library accessible by the user.

TYPE=SYNC | ASYNC

Indicates whether this is a synchronous (SYNC) or asynchronous (ASYNC) panel request. Synchronous requests display the panel and cause the process to be suspended pending the arrival of input from the panel or expiry of a time-out period. Asynchronous requests result in immediate return of control to the process, with input arrival being notified in due course via &INTREAD.

CDELAY=YES | NO

This operand indicates whether or not contention delay is used for this panel. By default, when the panel is displayed on a terminal which is currently in input mode, panel services delays panel output for the period set by the SYSPARMS CDELAY operand. For CDELAY=NO, panel output overrides the terminal's input mode and the panel is displayed at once, if necessary, interrupting the operator.

MODALL=NO | YES

This operand indicates how input fields on the panel are to be treated when &CONTROL FLDCTL is in effect. By default, only input fields changed by the operator or set by the program (using &ASSIGN OPT=SETMOD) will be returned in the modified field list.

If MODALL=YES is specified, every input field on the displayed panel will be returned in the modified field list. The modified field list is accessed using &ASSIGN OPT=MODFLD, or using the &ZMODFLD system variable.

Examples: &PANEL

```
.HELP   &PANEL &HELPPANEL
&IF .&INKEY = .PF03 &THEN +
      &RETSUB
&IF .&INKEY = .PF07 &THEN +
      &HELPPANEL = HELP1
&IF .&INKEY = .PF08 &THEN +
      &HELPPANEL = HELP2
&GOTO .HELP
```

Notes:

An NCL process's use of Panel Services is represented by a full-screen environment. Issuing an &PANEL statement will automatically establish a full-screen environment if one does not already exist. You can use the FSPROC command to initiate NCL processes with a full-screen environment. In this way, the process's activation is synchronized with its bidding for window ownership. The full-screen environment exists until the NCL process terminates or is explicitly terminated by the &PANELEND statement.

Note: For more information about the facilities provided for the processing of full-screen panels, see the *Network Control Language Programming Guide*.

An attempt to display a panel that has not been defined or has been incorrectly defined will normally result in termination of the procedure with an appropriate error message. However, using &CONTROL PANELRC it is possible for the procedure to receive notification through the &RETCODE system variable that the required panel does not exist or cannot be displayed due to syntax errors. If &CONTROL PANELRC is in effect, the &RETCODE system variable will be set to 16 on return from the &PANEL statement and the &SYSMSG system variable will contain the text of the message that describes the error. The procedure can test the &RETCODE variable and adjust its processing accordingly.

&CONTROL PANELRC implies that the procedure has been designed to cater for the range of return codes that is returned from an &PANEL statement. See the &CONTROL statement PANELRC operand for full details.

Any NCL process which has access to a full-screen window may issue &PANEL at any time. When multiple NCL processes are executing in the same NCL processing environment (invoked by the START command) they can all bid for the user window to display panels. To prevent another process from taking the window, a process must use the &CONTROL NOSHAREW option to indicate that it is not prepared to give up control of the window.

When the procedure is prepared to relinquish ownership of the window it should issue a &PANELEND statement, which allows any other process that wants the window for a panel display to gain access to it. An NCL process using asynchronous panels may issue &CONTROL SHAREW to indicate it is prepared to share the window without terminating its full-screen environment.

If INWAIT=0 is specified on a panel definition the panel is displayed before the issuing process continues. The issuing process may therefore be suspended indefinitely waiting for control of the window.

If INWAIT is specified with a positive interval defined, the panel will always be displayed for that interval from the time that the panel wins control of the window, regardless of how long it takes to get control of the window.

When displaying a panel with an INWAIT interval specified, your product region will only wait for the INWAIT interval (at maximum) to get control of the window. After that time the procedure regains control since the INWAIT period is considered to have passed.

&PANELEND

Terminates the full-screen environment of the current process.

```
&PANELEND
```

The &PANELEND statement indicates that the issuing procedure no longer wishes to use Panel Services. The full-screen environment, if it exists, is terminated.

If the procedure had earlier executed &PANEL to display a screen panel, it can indicate by &PANELEND that it has finished with panel processing, if the process is actually the current window owner the window will be passed to the next process with an &PANEL request outstanding or returned to the primary environment owner, for example, OCS.

Note: A long running process may issue a panel when some event occurs. Once the operator has responded to the panel, the procedure can issue &PANELEND and continues its event monitoring role.

&PARSE

Provides generalized parsing functions for tokenizing data into variables.

```
&PARSE [ { DELIM={ c | 'cccccccc' | "cccccccc" } | SEGMENT }  ]
       { VARS=name |  VARS=( name, name1, ..., namen ) |
         VARS=prefix* [ RANGE=( start, end ) ] |
         ARGS  [ RANGE=( start, end ) ] }
       [ REMSTR=varname ]
       [ OPT={ option | ( option, ..., option ) } ]
       [ INPUT={ CHAR | HEX | HEXEXP } ]
       [ DATA=text ]
```

&PARSE is a verb that allows a data string to be split into sections identified by delimiters or segmented according to length. Each section is placed into a separate target variable. On completion of &PARSE, the system variable &ZVARCNT is set to the number of variables created or modified by the operation.

Operands:

DELIM={ *c* | '*cccccccc*' | "*cccccccc*" }

Specifies the delimiter character or a series of individual delimiter characters that will be used as the argument for the parsing process. Every occurrence of a DELIM character in the DATA string that is being parsed represents the end of a section of the data. The section, minus the delimiter character, is placed in the next target variable and the parsing process continues. Special case processing takes place when a blank is detected as the only delimiter character (see Notes following the examples).

If DELIM is omitted the default delimiter character is a blank.

The delimiter series may be 1 to 8 characters long. Series of 2 or more characters must be enclosed in single or double quotes.

SEGMENT

Specifies that there is no delimiter character but that the parsed string will be placed into the receiving variables in segments that correspond to the length of the individual variables. The length defaults to the maximum variable length unless overridden by length specifications in a variable list.

VARS=

Specifies the target variables that are to be assigned the parsed sections of the DATA. The format of the VARS operands may be:

name

The name of a variable, excluding the ampersand (&).

***name*(*n*)**

As name, but *n* denotes the length of the data to be placed in the variable. If necessary, the data is truncated.

***(*n*)**

Denotes a skip operation, where *n* represents the number of units to be skipped during the tokenization process. On VARS= statements, n denotes 'skip this number of words'. An asterisk (*) by itself is the same as *(1).

If SEGMENT is specified, then *n* denotes 'skip this number of characters'. An asterisk (*) by itself is the same as *(1).

prefix*

Denotes that variables are generated automatically during the parsing process, and that variable names will be *prefix1* .. *prefix2*, and so on. The RANGE= operand is specified to indicate a starting and ending suffix number. *prefix** cannot be used with other variable names.

ARGS

Specifies that &1 through &*n* are the variables to be assigned the parsed sections of data. The RANGE= operand may be coded to designate start and end numbers to delimit the number of variables generated.

REMSTR=*varname*

Nominates a variable that is to be assigned the remainder of the DATA string that is being parsed if insufficient target variables are specified to hold the entire parsed string. This option is mutually exclusive with the SEGMENT operand.

OPT={ *option* | (*option*,*option*) }

Specifies one or more additional options that are to apply to the results of the &PARSE statement. Supported options are:

ASIS

Which specifies that leading and trailing blanks are to be preserved when text is assigned into the target variables. If ASIS is not specified, then leading and trailing blanks are stripped from the parsed string sections as they are placed in the target variables.

NONULLS

Specifies the action to be taken if two consecutive delimiters are found within the DATA being parsed or if the data starts with a delimiter. In this case there is an implied null section, that is, a section of zero length. If NONULLS is specified, then the zero length section is ignored and no null variable is created. If NONULLS is omitted, then a null variable is created.

INPUT={ CHAR | HEX | HEXEXP }

Describes the expected data input and mode of processing. The default of character mode should be used for standard character data. HEX and HEXEXP should be used when the input contains non-character hexadecimal data. HEXEXP indicates that the hexadecimal data be expanded into display characters in the resulting target variables.

DATA=*text*

The string of data that is to be parsed. The string may be present in one or more variables or may be coded explicitly.

DATA= must be specified as the last keyword on the statement since the data string is regarded as being everything to the right of the DATA= keyword to the end of the statement.

Examples: &PARSE

```
&PARSE DELIM=, ARGS OPT=ASIS DATA=123,  456,789
```

results in:

```
&1 = 123
&2 =    456
&3 = 789
```

```
&PARSE DELIM=':,' VARS=(A,B,C,D) REMSTR=REST +
       DATA=aaa:bbb, ,ccc,ddd:eee
```

results in:

```
&A = aaa
&B = bbb
&C =      -* null value
&D = ccc
&REST = ddd:eee
```

```
&PARSE DELIM=',:' VARS=(A,B,C,D) REMSTR=REST +
       OPT=NONULLS DATA=aaa:bbb,  ,ccc,ddd:eee
```

results in:

```
&A = aaa
&B = bbb
&C = ccc
&D = ddd
&REST = eee
```

```
&PARSE ARGS DATA=this is a variable msg.
```

results in:

```
&1 = this
&2 = is
&3 = a
&4 = variable
&5 = msg.
```

```
&INPUT=AABBBCCCCCCC
&PARSE SEGMENT VARS=(A(2),B(3),C) DATA=&INPUT
```

results in:

```
&A = AA
&B = BBB
&C = CCCCCCC
```

Notes:

The parse process proceeds from left to right through the data, until either all data has been parsed or all target variables have been assigned a value. Where data remains to be parsed it may be optionally assigned to a variable as specified by the REMSTR operand.

Characters recognized as delimiter characters are never included in the result variables. The only exception is a variable nominated in the REMSTR operand.

Where blank has been specified, or has defaulted, as a delimiter, it is never recognized as a delimiter when found following a recognized delimiter. Additionally, once this condition has occurred, a blank is not recognized as a delimiter until a non-blank, non-delimiter character is next encountered.

If OPT=ASIS was not specified and a segment has been isolated which contains only blanks, a null value is assigned to the corresponding target variable, unless OPT=NONULLS has been specified.

More information:

&SELSTR (see page 601)
&REMSTR (see page 572)

&PAUSE

Suspends processing of an NCL process until the operator requests that processing continue or be terminated.

```
&PAUSE [ { VARS=prefix* [ RANGE=(start, end) ] |
           VARS={ name | (name, name, ..., name) } |
           STRING=( name, name, ..., name) |
           ARGS [ RANGE=( start, end ) ] [ user supplied text ] } ]
```

The &PAUSE statement allows an NCL process to suspend processing pending input from an OCS window. When using a non-full screen procedure, interaction with the OCS operator may be required. An example of this is a procedure that contains documentation for the operator describing some manual action that must be taken, such as calling a number for service. Once the specified action has been taken, the operator can enter a command to resume or to terminate processing:

- Use a GO command to resume processing. Optionally, the GO command can pass information to the procedure.
- Use a FLUSH command to terminate the NCL process.

Enter an &PAUSE statement in the procedure at the point where processing is to be suspended. The &PAUSE is specified either with or without user text. If no text is entered, the system will supply appropriate highlighted text defining the action the operator must take to continue or terminate processing of the procedure:

```
N04107 PROCEDURE xxxxxxxx NCLID=nnnnnn PAUSED.
```

The highlighted text (default or as supplied on the &PAUSE statement) is displayed as a non-roll delete message on the OCS window and will remain displayed until the procedure leaves its paused state.

On completing the &PAUSE, the system variable &ZVARCNT is set to the number of variables created or modified by the operation.

Operands:

VARS=

Specifies the variables that will contain the operator's response. The operator response is entered using a GO command followed by text. The response text (excluding the GO command keyword) will be tokenized into the nominated variables from left to right before control is returned to the procedure. If insufficient variables are provided, some data will not be available to the procedure. Excess variables will be set to a null value. The formats of the operands that may be coded with VARS are described below.

prefix*

Denotes that variables are generated automatically during the tokenization process, and that variable names will be *prefix1* … *prefix2*, and so on. (The RANGE= operand may be specified to indicate a starting and ending suffix number). *prefix** cannot be used with other variable names.

name

The name of a variable, excluding the ampersand (&).

***name*(*n*)**

As *name*, but *n* denotes the length of the data to be placed in the variable. If necessary, the data is truncated.

***(*n*)**

Denotes a skip operation, where n represents the number of units to be skipped during the tokenization process. On VARS= statements, n denotes 'skip this number of words'. The asterisk (*) by itself is the same as *(1).

STRING=

Specifies that no tokenization is to be performed. The entire text of the command line is treated as a single string and returned to the procedure in the nominated variables. The formats of the operands associated with STRING are:

name

User-specified variables, excluding the leading &, into which the string text is put. Text is placed into each variable up to the maximum length of that variable.

***name*(*n*)**

User-specified variables, excluding the leading &, into which the string text is put. Text is placed into each variable for the length specified *n*.

***(*n*)**

Denotes a skip operation, where n denotes 'skip this number of characters'. An asterisk (*) by itself is the same as *(1).

ARGS

Denotes that the line of text retrieved will be tokenized and placed word by word into automatically generated variables of the form &1 through &*n*, depending on how many are required to hold the text. The RANGE= operand may be coded to designate a start number and optionally an end number, which delimits the number of variables that will be generated.

user supplied text

Optional upper and lower case text describing the action the user must take. When specifying this form of the &PAUSE statement, the VARS or STRING operand cannot be specified. Entry of text on the GO command will be assigned as if ARGS was coded.

Note: When user-supplied text is not specified, system default message N04107 is written. When information concerning the reason the procedure paused, or operator action is required, it is the user's responsibility to ensure that an appropriate prompt is displayed, such as using &WRITE, before issuing the &PAUSE.

Examples:

```
&CMDLINE GO ID=&ZNCLID _YES  -* Preformat OCS reply for user
&WRITE DATA=ENTER “GO YES” TO CONTINUE, “GO NO” TO END.
&PAUSE ARGS
&IF .&1 EQ .NO &THEN +
 &END
-*
-* Processing
-*

&PAUSE STRING=(CMD)
ROUTE NMT &CMD

&PAUSE ARGS RANGE=(20,80)
```

Waits for the operator response, specifying that it is tokenized into variables &20 to a maximum of &80. &ZVARCNT is set to indicate how many variables were created.

```
&PAUSE VARS=(*(3),A(2),B(3),C,D,E,F)
```

Reads the operator response text and tokenizes the message into individual words; *(3) indicates that the first 3 words are ignored, 2 characters of the next word are placed in the variable &A, three characters of the next word are placed in the variable &B and the next 4 words are placed in variables &C, &D, &E and &F respectively.

```
&PAUSE VARS=ABC* RANGE=(1,50)
```

Reads the operator response text, tokenizes it and places the text word by word into a series of automatically generated variables of the form ABC1 ABC2 ... ABC50. As many variables as required are generated, to the limit specified by the RANGE operand.

```
&PAUSE STRING=(A,B(2),*(5),C(3))
```

Reads the operator response as a single string of text. The first 256 bytes are placed in &A, the next 2 characters are placed in &B, the next 5 characters are ignored and the next 3 characters are placed in &C.

Notes:

The status of all paused procedures in a user's NCL processing region is interrogated using the SHOW PAUSE command.

Multiple NCL processes may be executing in the same NCL processing environment and all may issue &PAUSE to await operator input. The operator must use the ID operand of the GO command to identify the process being referenced.

If an OCS operator exits while a procedure is in a paused state, the procedure will be flushed. The PROFILE EXIT command is used to prevent termination of OCS while NCL processes are active.

Procedures executing in a dependent processing environment can issue &PAUSE. In association with the GO command this provides a useful mechanism for process communication between hierarchically dependent NCL processes.

While a procedure remains paused in an OCS environment, the P mode indicator will be displayed to the left of the command line to warn the operator. This indicator will be cleared when all pause conditions are satisfied.

More information:

&INTCMD (see page 370)
&INTREAD (see page 375)

&PPI

Provides a facility for programs to exchange data. Each of the &PPI options shown in the syntax below is described individually following this generic description.

```
&PPI ALERT [ OPT={ NONE | ASIS | HEXPACK } ]
           [ DATA=data |
             VARS=prefix* [ RANGE=(start,end) ] |
             ARGS [ RANGE=(start,end) ] |
             VARS={ var | ( var1, var2, ..., varn ) } |
             MDO=mdoname ]

&PPI DEACTIVATE [ MAXQUEUE=n ]

&PPI DEFINE ID={ * | name } [ MAXQUEUE=n ]

&PPI RECEIVE [ WAIT={ YES | NO | n | NOTIFY } ]
             [ OPT={ NONE | ASIS | HEXEXP } ]
             [ VARS={ var | ( var1, var2, ..., varn ) } |
               VARS=prefix* [  RANGE=( start,end ) ] |
               ARGS [ RANGE=( start,end ) ] |
               STRING=( name, name, ..., name ) |
               MDO=mdoname [ MAP=map ] }

&PPI SEND TOID=name
        [ OPT={ NONE | ASIS | HEXPACK } ]
        [ DATA=data |
          VARS=prefix* [ RANGE=( start,end ) ] |
          ARGS [ RANGE=(start,end) ] |
          VARS={ var | ( var1, var2, ..., varn ) } |
          MDO=mdoname ]

&PPI STATUS [ ID=name ]
```

The Program-to-Program Interface (PPI) provides a general purpose facility to exchange data between programs written in any language. It also provides a facility for any program to forward a generic alert to your product region. No special authorization is required to use PPI and it does not depend on having your product region running.

This implementation of PPI is supported on z/OS, MSP, MSP/AE, MSP/EX, and VOS3. The PPI implementation can use Cross-Memory Services or Service request Block (SRB) scheduling, so that MSP is supported.

The NCL &PPI verb provides access to your product. This interface allows any programming language to freely exchange information.

Note: The API provided by the Program-to-Program Interface is described in IBM's *NetView Application Programming Guide: Program to Program Interface* (SC31-6093-0).

PPI services is provided by the SOLVE Subsystem Interface (SSI) or by the NetView Subsystem Interface.

&PPI Verb

All PPI facilities is accessed using the &PPI verb. The specific request is identified by a keyword immediately following the &PPI verb. These keywords generally correspond to the various functions described in the API.

The full set of &PPI requests is as follows:

```
&PPI ALERT
&PPI DEACTIVATE
&PPI DEFINE
&PPI RECEIVE
&PPI SEND
&PPI STATUS
```

Return Codes, System and User Variables

After each execution of the &PPI verb, the &RETCODE and &ZFDBK system variables are set to indicate the success, or otherwise, of the request. &ZFDBK is the PPI return code, and the &RETCODE is returned by the &PPI verb. The following table shows the correlation between the values in these variables.

Note: For more information about &RETCODE and &ZFDBK, see the *Network Control Language Programming Guide*.

The following table shows the correlation between the &ZFDBK and &RETCODE system variables.

&ZFDBK	&RETCODE	
00	00	The PPI request completed successfully
04	04	The specified receiver is not active—the data buffer or generic alert has been queued
10	00	The PPI facility is active and can be used
14	00	The receiver program is active
15	04	The receiver program is already inactive
16	08	The receiver program is already active
18	12	The receiver ECB is not zero
20	12	Invalid request type
22	12	The program issuing this request is not executing in primary addressing mode
23	08	The user program is not authorized
24	12	PPI is not active
25	12	The ASCB address is not correct
26	08	The receiver program is not defined
28	12	This product release does not support PPI
30	04	No data buffer in the receiver buffer queue
	20	&PPI RECEIVE WAIT=NOTIFY—no data buffers in queue, notify is queued
31	12	The receiver data buffer length is too short to receive the next data buffer
32	12	No storage is available
33	12	Invalid buffer length
35	08	The receiver buffer queue is full
36	12	Unable to establish ESTAE protection as requested
40	12	Invalid sender ID or receiver ID
90	12	A processing error has occurred

Other system variables are:

&ZPPI

Indicates whether this system appears to support PPI or not.

&ZPPINAME

Contains the PPI receiver ID that this NCL process is registered as.

Some &PPI functions set specific NCL user variables:

&PPISENDERID

Contains the PPI ID of the sender of a received message.

&PPIDATALEN

Contains the length of the actual received data in bytes.

Determining PPI or Receiver Status

The STATUS option of the &PPI verb allows an NCL process to determine the status of PPI itself (available or not), or the status of a PPI RECEIVER (by using the ID=name operand).

In either case, the process can examine the &RETCODE and &ZFDBK system variables after the request. If &RETCODE is 0, then PPI or the receiver is available/defined.

Defining the Process as a Registered PPI Receiver

By using the DEFINE option of the &PPI verb, an NCL process can register itself as a receiver. A 1 to 8 character name is supplied, which must be unique (that is, not presently defined to PPI or currently inactive). If your product region is providing PPI services, an alternative is to use the ID=* option, which causes PPI to provide a unique name. This option is useful when talking to globally named servers, as you need not worry about trying to find a unique name.

A process need not be defined to send data using the SEND and ALERT options. In this case, a sender ID of #*nclid* (7 characters) is used.

Sending a Generic Alert

One function of the PPI facility is the collection of generic alerts and forwarding to general CNM reporting (for example, NEWS). The &PPI ALERT verb allows any NCL process to send an alert to CNM. The alert must be formatted as an NMVT, including the NMVT header.

Sending Data to a Receiver

The &PPI SEND verb option allows any NCL process to send data to a nominated receiver. This receiver must be defined, but may be inactive (in which case data is queued unless the queue limit is reached).

The receiver may not be an NCL process at all, and may reside in another address space.

The data to be sent may be just a character string, hexadecimal data that is packed before sending, or an MDO object.

Receiving Data

An NCL process may receive data directed to its defined receiver ID using the RECEIVE option of the &PPI verb. That data may come from other NCL processes, including other product regions, or from other programs.

Standard parsing option, as on the other &*xxx*READ verbs, may be used. Alternatively, MDOs may be received.

The WAIT= operand allows the procedure to indicate whether or not it will wait if no data is available, and if none is available, how long. Alternatively, the process can use WAIT=NOTIFY to cause a message to be delivered to the dependent response queue when data arrives (thus allowing other work to be performed. When the notification arrives via &INTREAD, the process can reissue the &PPI RECEIVE).

Deactivating the Receiver ID

The &PPI Deactivate option allows an NCL process to disconnect itself from a defined PPI receiver ID. Optionally, a queue limit is specified, allowing data to be queued even though no receiver is present. The ID is reactivated by this or any other NCL process later.

If an NCL process that is defined to PPI terminates, an automatic deactivation occurs.

Uses of PPI

Since PPI is available to any environment, not just NCL, PPI provides a simple, powerful technique for access to a product from the outside. For example, an NCL process could provide a batch program with the ability to issue selected product commands and return the results of the command to it.

PPI also provides an alternative method of communication between two NCL procedures, with no data loss if your product region terminates. The data remains queued in the PPI server address space. The NCL procedures need not be active on the same product region.

Examples

These examples wait for requests to arrive on the PPI queue. Each request is then executed as a product command, and any messages received from the command are sent back through the PPI. The ID for the server is made up of CMD and the current domain ID; therefore, a different copy of this procedure could run on each product region in the system. If the procedure is stopped, requests are queued, and on the next invocation the queue is processed. To stop the server, you can either flush the procedure (interrupting the current request) or issue an INTQ command, putting the string STOP onto the procedure's dependent request queue.

Example 1:

The following example issues requests to the previous procedure. It shows how the communication takes place. Any program using the PPI (whether it is an NCL procedure on this or another region, or a program written in another language) can request information in the same way.

```
&IF &ZPPI NE YES &THEN +
    &ENDAFTER +
      &WRITE COLOR=RED DATA=PPI INTERFACE NOT AVAILABLE
&PPI DEFINE ID=CMD&ZOMID
&IF &RETCODE NE 0 &THEN +
   &ENDAFTER +
       &WRITE COLOR=RED DATA=INITIALIZE FAILED +
              RC=&RETCODE FDBK=&ZFDK
```

```
&DOUNTIL &RETCODE NE 0
   &PPI RECEIVE VARS=PPI* WAIT=NOTIFY
   &DOWHILE &RETCODE EQ 0     -* Process anything waiting
                              -* on the queue
      &WRITE COLOR=YELLOW LOG=YES TERM=YES +
             DATA=REQUEST RECEIVED FROM &PPISENDERID TO +
                  ISSUE COMMAND &PPI1 &PPI2 &PPI3 &PPI4 &PPI5 +
                  &PPI6 &PPI7 &PPI8 &PPI9
      &INTCMD &PPI1 &PPI2 &PPI3 &PPI4 &PPI5 &PPI6 &PPI7 +
              &PPI8 &PPI9
      &DOUNTIL &RETCODE NE 0 +
       INTREAD SET WAIT=5 TYPE=RESP
       &IF &ZFDBK = 0 &THEN +
        &PPI SEND TOID=&PPISENDERID DATA=&ZMTEXT
       &ELSE +
        &RETCODE 1
      &DOEND
      &IF &RETCODE NE 1 &THEN +
       &WRITE COLOR=RED DATA=PPI SEND TO +
        &ZPPISENDERID FAILED, RC=&RETCODE, FDBK=&ZFDBK
      &ELSE
        &PPI SEND TOID=&PPISENDERID DATA=END=CMD&ZDOMID
      &PPI RECEIVE VARS=PPI* WAIT=NOTIFY
   &DOEND
   &IF &RETCODE = 20 &THEN +  -* Wait for a request to
                              -* appear on queue
   &DO
      &DOUNTIL .&INTL1 EQ .N00101
       &INTREAD WAIT=YES TYPE=ANY VARS=INT*
       &IF .&ZINTYPE EQ .REQ AND .&INT1 EQ .STOP &THEN +
        &ENDAFTER +
         &WRITE COLOR=YELLOW +
                DATA=STOP REQUEST FROM &ZMREQID ACCEPTED
      &DOEND
      &RETCODE 0
   &DOEND
&DOEND
&WRITE COLOR=RED +
       DATA=PROGRAM SERVER CMD&ZDOMID FAILED, RC=&RETCODE,+
            ZFDBK=&ZFDBK
&END
```

Example 2:

This example sends the data passed on the EXEC or START command to the server shown in Example 1. The messages returned will then be written to the terminal.

```
&IF &ZPPI NE YES &THEN +
    &ENDAFTER +
       &WRITE COLOR=RED DATA=PPI NOT AVAILABLE ON THIS +
                                SYSTEM
&PP1 STATUS ID=CMD&ZDOMID
&IF &RETCODE NE 0 &THEN +
    &ENDAFTER +
       &WRITE COLOR=RED DATA=PROGRAM RECEIVER CMD&ZDOMID +
                                NOT AVAILABLE
&PPI DEFINE ID=*
&IF &RETCODE NE 0 &THEN +
    &ENDAFTER +
       &WRITE COLOR=RED DATA=DEFINE TO PPI FAILED
&WRITE COLOR=PINK DATA=SENDING COMMAND "&ALLPARMS" TO +
                        SERVER CMD&ZDOMID
&PPI SEND TOID=CMD&ZDOMID DATA=&ALLPARMS
&DOUNTIL .&MSGLN1 EQ .END-CMD&ZDOMID OR &RETCODE NE 0
    &PPI RECEIVE WAIT=YES +
                 STRING=(MSGLN1,MSGLN2,MSGLN3,MSGLN4)
    &IF &RETCODE EQ 0 AND .&MSGLN1 NE .END-CMD&ZDOMID +
      &THEN &WRITE COLOR=TURQUOISE DATA=&MSGLN1 &MSGLN2 +
                                          &MSGLN3 &MSGLN4
&DOEND
&PPI DEACTIVATE MAXQUEUE=0       -* Prevent others queuing
                                 -* to this ID.
&WRITE COLOR=PINK DATA=*** END OF MESSAGES ***
&END
```

&PPI ALERT

Sends a generic alert.

```
&PPI ALERT [ OPT={ NONE | ASIS | HEXPACK } ]
           [ DATA=data |
             VARS=prefix* [ RANGE=(start,end) ] |
             ARGS [ RANGE=(start,end) ] |
             VARS=(list) | MDO=stem ]
```

Operands:

ALERT

This option of the &PPI verb allows an NCL process to send a generic alert to the PPI ALERT receiver, NETVALRT. The alert must be in NMVT format, including the NMVT header. If the process is not defined using &PPI DEFINE, the sender ID used is #*nnnnnn*, where *nnnnnn* is the 6-digit (leading zeros) NCL ID.

OPT={ NONE | ASIS | HEXPACK }

An optional parameter that indicates how the supplied data is to be processed before sending. This operand is only valid when the data is specified using the DATA= operand.

NONE

The default, indicates that the data is to have trailing blanks stripped, but is to otherwise be left alone.

ASIS

Indicates that the data is to be completely left alone. This allows passing binary data in NCL tokens with no alteration.

HEXPACK

Indicates that the data supplied is character-format hexadecimal strings. Each string must contain an even number of hexadecimal characters. The strings are each packed to binary and then abutted together. The resulting binary data is sent as is. For generic alerts, this is the most useful format.

Note: The OPT operand is only valid when the DATA operand is specified.

DATA=*data*

This operand, if specified, delimits the start of the data to send as the generic alert. The data starts immediately after the DATA= keyword.

If the alert is successfully sent, and NETVALRT is active, &RETCODE will be set to 0. If the alert is queued and NETVALRT is inactive, &RETCODE will be set to 4. Other values indicate an error of some sort. &ZFDBK is inspected to determine the exact PPI return code.

VARS=*prefix [RANGE=(*start*,*end*)] |ARGS [RANGE=(*start*,*end*)] VARS=(*list*)**

The contents of the variables or arguments specified are concatenated to form the data which is sent as the generic alert.

MDO=*stem*

The data within this MDO is sent as the generic alert.

&PPI DEACTIVATE

Deactivates and disconnects an NCL process from PPI.

```
&PPI DEACTIVATE [ MAXQUEUE=n ]
```

Operands:

DEACTIVATE

The DEACTIVATE option of the &PPI verb allows a PPI-defined NCL process to deactivate the defined receiver ID. This action also disconnects the NCL process from PPI, allowing a new DEFINE, possibly with a different receiver ID.

The PPI queue limit may be altered when deactivating.

MAXQUEUE=*n*

An optional parameter, that allows alteration of the queue limit for the deactivated receiver. If not specified, the existing queue limit is maintained.

The allowable range is 0 to 9999.

Following a successful deactivate, &RETCODE will be 0. &ZFDBK will also be 0. &ZPPINAME will now be null, indicating that no PPI receiver ID is associated with the process.

If a DEFINEd NCL process does not issue &PPI DEACTIVATE before terminating, an implicit deactivation occurs, with the queue limit being maintained as it was.

More information:

Examples (see page 541)

&PPI DEFINE

Defines an NCL process to PPI.

```
&PPI DEFINE ID={ * | name } [ MAXQUEUE=n ]
```

Operands:

DEFINE

The DEFINE option of the &PPI verb allows an NCL process to register itself to PPI. It can optionally ask PPI to supply a unique receiver/sender ID. Once defined, the NCL process can use all other &PPI verb options. When sending data using &PPI SEND or &PPI ALERT, the defined name is used as the sender ID.

The DEFINE option can also be used by a defined process to alter the queue limit.

ID={ * | *name* }

A required parameter that supplies the name that the procedure is to be registered to PPI under. If name is used, then it must be a valid PPI sender/receiver ID, as described earlier in this section. If * is used, PPI will supply a unique name, if possible.

If the process is already defined, then the * option cannot be used, and the name the procedure is defined under must be supplied. This name is available in the &ZPPINAME system variable. This is done when dynamically altering the queue limit.

If the NCL process is executing in the product region that is connected to the SSI that owns PPI, then names starting with NETV or NETM is used. Otherwise, they cannot, and a PPI return code of 40 (in &ZFDBK; &RETCODE of 12) will be returned.

MAXQUEUE=*n*

An optional parameter, that allows setting of the defined receiver queue limit. For a new definition, the default value, if not specified, is 10.

The allowable range is 0 to 9999.

If altering the queue limit for an already-defined receiver, there is no default. Not specifying this parameter means that no change occurs.

Following a successful define, &RETCODE will be 0. &ZFDBK will also be 0. &ZPPINAME will contain the actual receiver ID allocated. This will equal the name supplied if a name was supplied, or will be the PPI-allocated name if an asterisk was used.

More information:

Examples (see page 541)

&PPI RECEIVE

Receives data from PPI.

```
&PPI RECEIVE [ WAIT={ YES | NO | n | NOTIFY } ]
             [ OPT={ NONE | ASIS | HEXEXP } ]
             { VARS=(list) |
               VARS=prefix*[ RANGE=(start,end) ] |
               ARGS [ RANGE=(start,end) ] | STRING=(list) |
               MDO=stem [ MAP=map ] }
```

Operands:

RECEIVE

The RECEIVE option of the &PPI verb allows a PPI-defined NCL process to receive the next available data buffer queued to that receiver ID. The data may be parsed into NCL variables in a variety of ways.

The NCL process need not be concerned with the length of the incoming data; this is handled automatically.

WAIT={ YES | NO | *n* | NOTIFY }

An optional parameter, that indicates what action to take if there is no data buffer immediately available to process.

WAIT=YES (the default) indicates that the process is to wait indefinitely for a buffer to arrive. The process may be flushed while waiting.

WAIT=NO indicates that the process is to immediately continue execution. In this case, &RETCODE will be 4, and &ZFDBK will be 30 (the PPI return code for no data).

WAIT=*n* indicates that the process is to wait the indicated number of seconds (from 0.01 to 9999.99). If no buffer arrives in that time, action is as for WAIT=NO. The process may be flushed while waiting.

WAIT=NOTIFY indicates that the process is to continue execution, and that a message is to be queued to the dependent processing environment, informing it when a data buffer arrives. This as akin to the way that &PANEL TYPE=ASYNC works. The message may be seen using &INTREAD. This allows the process to wait for several events simultaneously. If a new &PPI RECEIVE (of any type) is issued before the message has been queued, the notification is canceled (a message may already be queued, but not yet received using &INTREAD, so be careful). The notification message (N00101) has a type of PPI, and an event class of RECEIVE. If the receiver ECB is posted with a shutdown code (99), the event class will be SHUTDOWN.

If no data is immediately available, &RETCODE will be set to 20. indicating that a message will be queued to &INTREAD.

The process simply reissues &PPI RECEIVE after reading the notification message.

OPT={ NONE | ASIS | HEXEXP }

An optional parameter that controls how the incoming data is parsed.

NONE (the default) means that any unprintable data is translated to blanks when placing it into the nominated NCL tokens.

ASIS means that unprintable characters will be left alone when placing it into NCL tokens. This is typically useful only when using the STRING parse option, as the other options delimit on blanks (x'40'), which may actually be part of the hexadecimal data.

HEXEXP means that each string being placed into an NCL token is hexadecimal-expanded to character-format hexadecimal. This limits the amount of data that maybe placed into a token to 128 source bytes (for a maximum of 256 hexadecimal-expanded characters). Typically useful only with the STRING parse option.

VARS=(*list*) |

VARS=*prefix[RANGE=(*start*,*end*)] |**

ARGS [RANGE=(*start*,*end*)] |

STRING=(*list*)

One of these options is required, and sets the parsing option for the received data.

The input data is parsed using blanks as the separator, or broken up into string segments.

VARS=(*list*) parses the data on blanks into a list of variables. Each entry in the list must be a valid variable name, without the & (unless you wish to substitute the variable name itself). Each variable in turn receives the next blank-delimited piece of source data, truncated to 256 characters if required (128 before hexadecimal-expansion of using OPT=HEXEXP). A piece of source data may be skipped by specifying * as the variable name. Several may be skipped by specifying *(*n*). A variable may have the amount of data placed in it truncated by specifying *name*(*n*) for that entry.

VARS=*prefix** and ARGS (which is the same as VARS=*) parse as for VARS=(*list*) with the variable names being formed from the prefix, suffixed by the numbers specified in the RANGE= operand. All the target variables in the range are cleared before this parsing is performed.

STRING=(*list*) indicates that the data is to be placed into the nominated variables in order, delimiting only be length. If no subscripts are specified on the variables in the list, 256 is used. Part of the data may be skipped by using *(*n*) and each variable may have an explicit length specified after it.

Note: If you are using OPT=HEXEXP, these lengths refer to the length before hexadecimal-expansion, and in this case the maximum length allowed is 128.

Following a successful RECEIVE, &RETCODE will be 0. &ZVARCNT contains the count of the number of variables updated.

The sender ID of the data is stored in the NCL user variable &PPISENDERID. The actual byte length of the received data is stored in the user variable &PPIDATALEN.

MDO=*stem* [MAP=*map*]

The received data is assigned into the MDO. If MAP=*map* is specified, then the specified map is attached to the MDO.

More information:

Examples (see page 541)

&PPI SEND

Sends data to a PPI receiver.

```
&PPI SEND TOID=name
        [ OPT={ NONE | ASIS | HEXPACK } ]
        [ DATA=data |
          VARS=prefix* [ RANGE=(start,end) ] |
          ARGS [ RANGE=(start,end) ] |
          VARS=(list) |
          MDO=stem ]
```

Operands:

SEND

The SEND option of the &PPI verb allows an NCL process to send data to any defined PPI receiver. The receiver need not be another NCL process. If the sending process is not defined (via &PPI DEFINE), the sender ID used is #*nnnnnn*. *nnnnnn* is the 6-digit (leading zeros) NCL ID.

TOID=*name*

A required parameter, providing the PPI receiver ID to send to. The ID must be a valid PPI receiver ID (see earlier in this document). It must be defined, although not necessarily active.

OPT={ NONE | ASIS | HEXPACK }

An optional parameter that indicates how the supplied data is to be processed before sending. This operand is only valid when the data to send as the generic alert is specified using the DATA= operand.

NONE

The default, indicates that the data is to have trailing blanks stripped, but is to otherwise be left alone.

ASIS

Indicates that the data is to be completely left alone. This allows passing binary data in NCL tokens with no alteration.

HEXPACK

Indicates that the data supplied is character-format hexadecimal strings. Each string must contain an even number of hexadecimal characters. The strings are each packed to binary and then abutted together. The resulting binary data is sent as is.

DATA=*data*

This operand, if specified, delimits the start of the data to send as the generic alert. The data starts immediately after the DATA= keyword.

VARS=*prefix [RANGE=(*start*,*end*)] |ARGS= [RANGE=(*start*,*end*)]VARS=(*list*) |**

The contents of the variables or arguments specified are concatenated to form the data which is sent as the generic alert.

MDO=*stem*

The data within the MDO is sent as the generic alert.

Note:

If the data is successfully sent, and the target receiver is active, &RETCODE will be set to 0. If the data is queued and the receiver is inactive, &RETCODE will be set to 4. Other values indicate an error of some sort. &ZFDBK is inspected to determine the exact PPI return code.

More information:

Examples (see page 541)

&PPI STATUS

Determines PPI or receiver status.

```
&PPI STATUS
   [ ID=name ]
```

Operands:

STATUS

The STATUS option of the &PPI verb allows an NCL process to determine the status of PPI itself, or of a specific receiver.

ID=*name*

An optional parameter. If specified, indicates that a specific receiver ID is to be checked. &RETCODE will indicates whether the nominated receiver ID is active (0), inactive (4), or undefined (8).

If omitted, the status of PPI itself is returned. If PPI is active, &RETCODE will be 0. Other values indicate that PPI is not active.

Note:

&PPI status provides further information beyond the &ZPPI system variable, which simply indicates whether PPI may be usable in this environment.

More information:

Examples (see page 541)

&PPOALERT

Generates a simulated PPO message and specifies delivery options for a local or remote system.

```
&PPOALERT [ { [ LOCAL ] [ REMOTE ] [ PPOPROC ] } | ALL ]
              [ LINK={ linkname | * } | DOMAIN={ domainid | * } ]
              [ MSGID=nnnn ]
              [ MNAME=name ]
              [ PNAME=name ]
              [ MNETNAME=name ]
              [ PNETNAME=name ]
              [ ONETNAME=name ]
              [ ODOMAIN=name ]
              [ OSSCP=name ]
              [ OLINK=name ]
              [ LDOMAIN=name ]
              [ ALARM={ YES | NO } ]
              [ COLOR=color ]
              [ HLIGHT=highlight ]
              [ INTENS={ HIGH | NORMAL } ]
              [ NRD={ NO | OPER } ]
              [ SCAN={ YES | NO } ]
              [ { DATA | TEXT= } message-text ]
```

Simulates a PPO message on a LOCAL or REMOTE system. &PPOALERT may be used to signal an event on the local system, or one which is connected by an ISR link.

Operands:

LOCAL REMOTE PPOPROC | ALL

These operands correspond to the DELIVER options on the DEFMSG command. They determine the destinations for message delivery. For simulated VTAM messages where no options are specified, the delivery options in the DEFMSG table of the target system are used, otherwise these options will override the delivery options in the DEFMSG table. (For more information, see Notes for this verb.)

LINK={ *linkname* | * } | DOMAIN={ *domainid* | * }

Specifies the ISR-connected system on which the &PPOALERT is to be simulated. LINK=* generates an alert on all connected systems. DOMAIN=* generates an alert on all connected systems as well as the system issuing the alert.

MSGID=*nnnn*

A 3- or 4-digit VTAM message number associated with the alert message. This number corresponds to the *nnnn* part of the VTAM IST*nnn* or IST*nnnn* message ID.

MNAME=*name*

The minor resource name (MNAME) associated with the message. This must form a valid resource name and may be from 1 to 8 characters long.

PNAME=*name*

The group resource name (PNAME) associated with the message. This must form a valid resource name and may be from 1 to 8 characters long.

MNETNAME=*name*

The minor network name (MNETNAME) associated with the message. This must form a valid network name and may be from 1 to 8 characters long.

PNETNAME=*name*

The group network name (PNETNAME) associated with the message. This must form a valid network name and may be from 1 to 8 characters long.

ONETNAME=*name*

ODOMAIN=*name*

OSSCP=*name*

OLINK=*name*

These four parameter options let you override the originating network name, domain ID, SSCP name, or link name, if necessary. This may be useful when using &PPOALERT procedures to generate test messages.

LDOMAIN=*name*

Lets you override the last domain name.

ALARM={ YES | NO }

Specifies whether the terminal alarm sounds when a message is delivered.

COLOR=*color*

Specifies the screen display color for the message.

HLIGHT=*highlight*

Specifies whether the message is to be highlighted.

INTENS={ HIGH | NORMAL }

Specifies the message intensity required.

NRD={ NO | OPER }

Sets the non-roll delete attribute for the message.

SCAN={ YES | NO }

Specifies whether at signs (@) within text are to be interpreted as word highlight markers.

DATA | TEXT=*message-text*

The alert message text generated.

Examples: &PPOALERT

```
&PPOALERT LINK=CENTRAL TEXT=Successful Recovery of Area-3
```

Notes:

&PPOALERT simulates a VTAM message if the MSGID operand is specified, or if the message identifier within TEXT is a valid VTAM message ID.

The ALL, LOCAL, and REMOTE operands on the &PPOALERT verb represent the delivery options for the location at which the message is simulated. (This is different from the &PPOCONT and &PPODEL verbs, where these operands direct message flow from PPOPROC.) The LINK= or DOMAIN=operands can also be specified on the &PPOALERT statement. The delivery options are then used to override DEFMSG on the targeted domains. For example:

&PPOALERT LOCAL PPOPROC LINK=CENTRAL + TEXT=Node ASYD01 is now active

This statement causes the message to be delivered to link name CENTRAL, and at CENTRAL it assumes the delivery options of LOCAL and PPOPROC.

Message COLOR and HIGHLIGHT values you specify in the &PPOALERT verb override the PPOCOLOR and PPOHLITE SYSPARMS operands.

If LINK={ *linkname* | * } or DOMAIN={ *domainid* | * } is specified, the &ZFDBK system variable is set, following execution, as follows:

0

The message was enqueued successfully.

8

The specified link or domain was not enabled for PPO. This may mean that the link has not been started or PPO ISR has not been enabled for the link.

12

The specified link or domain is enabled for PPO but is not enabled for outbound unsolicited PPO traffic.

16

Storage shortage, message not sent.

20

ISR queue overflow on one or more links, &ZPPOSCNT has a count of the links on which a message was successfully sent.

28

ISR internal error.

In all other cases, &ZFDBK is set to zero.

If sends on multiple links are attempted because an asterisk has been specified on the LINK or DOMAIN operands, any error code returned will reflect the first error encountered, except for a queue overflow condition, which will always be indicated if it has occurred on any of the links.

Note: For information about how the ISR queue limit is calculated, see the description of the ISR QMAXK operand in the Online Help.

&PPOCONT

Resumes normal processing of a VTAM PPO message and optionally overrides delivery options or queues a copy of the message to a specific ISR link.

```
&PPOCONT [ NONE | LOCAL | REMOTE | ALL |
           LINK={ linkname | * } |
           DOMAIN={ domainid | * } ]
         [ ALARM={ YES | NO } ]
         [ COLOR=color ]
         [ HLIGHT=highlight ]
         [ INTENS={ HIGH | NORMAL } ]
```

Used within a PPOPROC procedure to return a message from the previous &PPOREAD to your product region for processing.

The message may either be returned for standard delivery, or delivery may be overridden by the &PPOCONT verb.

Operands:

NONE

This operand indicates that delivery will take place according to delivery options specified for the VTAM message in the DEFMSG table. All other delivery options will override those specified in DEFMSG.

LOCAL

The PPO message will be released for delivery to local PPO recipients only. The message is no longer available to PPOPROC.

REMOTE

The PPO message will be released for delivery to eligible ISR links enabled for PPO/ISR outbound traffic. LOCAL delivery is suppressed. The message is no longer available to PPOPROC.

ALL

This releases the PPO message for delivery to authorized PPO receivers, and to all eligible ISR-connected systems (unless any ISR delivery has already been performed). The message is no longer available to PPOPROC.

LINK={ *linkname* | * } | DOMAIN={ *domainid* | * }

If a specific link or domain is indicated, a copy of the message is queued to the specified link. Addressing to a local link name, or a remote domain ID may be used.

When an asterisk (*) is specified, a copy of the message is queued to all ISR links that are enabled for PPO/ISR outbound traffic.

The message is not released, but remains available to PPOPROC.

ALARM={ YES | NO }

Specifies whether the terminal alarm is sounded when the message is delivered.

COLOR=*color*

Specifies the screen display color for the message. Valid values are:

RED GREEN BLUE TURQUOISE YELLOW PINK WHITE

HLIGHT=*highlight*

Specifies whether the message is to be highlighted. Valid values are:

REVERSE BLINK USCORE

INTENS={ HIGH | NORMAL }

Specifies the message intensity required.

Examples: &PPOCONT

```
&PPOCONT LINK=CENTRAL
```

An &PPOCONT issued when no VTAM PPO message is available is ignored.

If the current message is a copy of an SPO (VTAM command response) message, or a copy of a VTAM command, the message or command is never delivered locally.

Notes:

An &PPOCONT need not be issued if another &PPOREAD is issued. If a message is being processed and another &PPOREAD is issued without an intervening &PPODEL or &PPOREPL, an implied &PPOCONT is performed and the message is returned for standard processing before the next &PPOREAD is carried out.

&PPOCONT is used to free a message for delivery even though the PPOPROC procedure continues processing before issuing another &PPOREAD.

When PPO messages are delivered to connected systems the Inter System Routing facility (ISR) is used to carry the PPO message traffic; this facility allows centralized collection of PPO message traffic.

If LINK=*linkname* or DOMAIN=*domainid* is specified, the &ZFDBK system variable is set, following execution, as follows:

0

Message queued satisfactorily.

4

Not sent, message originated or arrived from nominated link, or already sent to this link.

8

The specified link or domain was not enabled for PPO. This may mean that the link has not been started or PPO ISR has not been enabled for the link.

12

The specified link or domain is enabled for PPO but is not enabled for outbound unsolicited PPO traffic.

16

Message not sent, storage error.

20

Message not sent, ISR queue overflow.

24

Previous &PPODEL LINK/DOMAIN has blocked this link.

28

ISR internal error.

If &PPOCONT LINK=* or DOMAIN=* is used, the &ZFDBK system variable is set as follows:

0

Message queued to at least one ISR link.

4

Message not queued to any ISR link.

8

Previous &PPODEL LINK/DOMAIN=* has prevented any delivery.

In all other cases, &ZFDBK is set to zero (0) by &PPOCONT.

Message COLOR and HIGHLIGHT values you specify in the &PPOCONT verb override the PPOCOLOR and PPOHLITE SYSPARM variables.

&PPODEL

Deletes a VTAM PPO message, or blocks ISR delivery of a VTAM PPO message.

```
&PPODEL [ { ALL | DOMAIN={ domainid | * } |
            LINK= { linkname | * } |
            LOCAL | REMOTE } ]
```

Used within the PPOPROC procedure to delete a message previously delivered for processing by an &PPOREAD, or else used to block automatic ISR delivery for a message.

Depending on the option specified, the message may not be available for any further processing.

Operands:

ALL

(The default) means the PPO message is deleted and no delivery takes place. Copies queued to ISR links by previous &PPOCONT LINK= or &PPOCONT DOMAIN= statements are not affected.

The message is no longer available to PPOPROC.

DOMAIN={ *domainid* | * } LINK={ *linkname* | * }

A specific domain or link is indicated to block ISR delivery of the message to the specified ISR link. Addressing to a local link name, or a remote domain ID may be used.

When an asterisk (*) is specified, ISR delivery to all ISR links is blocked.

The message is not released, and remains available to PPOPROC.

LOCAL

The PPO message is released for delivery to eligible ISR links that are enabled for PPO/ISR outbound traffic. The message is not delivered to any local PPO receivers.

The message is no longer available to PPOPROC.

REMOTE

The PPO message is released for delivery to local PPO receivers only. No ISR delivery occurs (any copies sent previously by &PPOCONT LINK/DOMAIN=*name* have already been sent).

The message is no longer available to PPOPROC.

Notes:

An &PPODEL issued when no VTAM PPO message is available is ignored.

&PPODEL is normally used to eliminate many of the excess messages presented by VTAM, particularly some group messages.

If LINK=*linkname*, or DOMAIN=*domainid* is specified, the &ZFDBK system variable is set, following execution, as follows:

0

Message delivery has been blocked.

8

Specified link is not enabled for PPO.

24

A copy of the message has already been sent by &PPOCONT LINK/DOMAIN=*name*.

In all other cases, &ZFDBK is set to 0.

&PPOREAD

Requests the next VTAM PPO message be made available to PPOPROC.

```
&PPOREAD { SET |
           VARS=prefix* [ RANGE=(start,end) ] |
           VARS={ name | (name,name, ..., name) } |
           STRING=( name, name, ..., name ) |
           ARGS [ RANGE=(start,end) ] |
         [ WAIT={ YES | NO | nnnn.nn } ]
```

The PPOPROC system level procedure uses &PPOREAD to receive the next VTAM message.

If no VTAM PPO message is immediately available, processing of the procedure will be suspended at this point and will resume when the next VTAM PPO message arrives.

If DEFMSG has been used to limit the messages that are to be intercepted, only those messages will satisfy an &PPOREAD.

Multiple &PPOREAD statements is present within PPOPROC, thus making the processing of group messages easier. Delivery of all unsolicited VTAM messages to the one NCL process allows PPOPROC to correlate and react intelligently to VTAM's notification of network events.

On completion of &PPOREAD the system variable &ZVARCNT is set to the number of variables created or modified by the operation.

The profile of the message received by &PPOREAD is set in a suite of reserved system variables. The message profile (which includes attributes such as color, highlighting, and source information) provides a complete description of all the PPO message attributes in addition to the message text.

Operands:

SET

Specifies that no tokenization of the PPO message is to be performed, but that the &PPOREAD statement is to return only the message profile of the next PPO message.

If SET is not specified, instructions must be coded on the &PPOREAD statement specifying the tokenization requirements for the message text by using the other &PPOREAD operands.

VARS=

Specifies that the message is to be tokenized into the nominated variables before control is returned to the procedure. Each word of the command output line will be tokenized into the nominated variables from left to right. If insufficient variables are provided, some data will not be available to the procedure. Excess variables will be set to a null value. The formats of the operands that may be coded with VARS= are described below. The format of the operands associated with VARS= are:

prefix*

Variables will be generated automatically during the tokenization process, and that the variable names will be *prefix1* .. *prefix2*, and so on. The RANGE= operand may be specified to indicate a starting and ending suffix number. *prefix** cannot be used in conjunction with other variable names.

name

The name of a variable, excluding the ampersand (&).

***name*(*n*)**

As *name*, but *n* denotes the length of the data that is to be placed in the variable.

***(*n*)**

Denotes a skip operation, where n represents the number of units to be skipped during the tokenization process. On VARS= statements *n* denotes 'skip this number of words'. An asterisk (*) by itself is the same as *(1).

STRING=

Specifies that no tokenization is to be performed. The entire text of the command line is to be treated as a single string and returned to the procedure in the nominated variables. The formats of the operands associated with STRING are:

name

User-specified variables (excluding the leading &) into which the string text is to be placed. Text will be placed into each variable for the maximum length of a variable.

***name*(*n*)**

User-specified variables (excluding the leading &) into which the string text is to be placed. Text will be placed into each variable for specified length *n*.

***(*n*)**

Denotes a skip operation, where *n* represents the number of units to be skipped during the tokenization process. On STRING statements n denotes 'skip this number of characters'. An asterisk (*) by itself is the same as *(1).

ARGS

Denotes that the line of text retrieved will be tokenized and placed word by word into automatically generated variables of the form &1 through &*n*, depending on how many are required to hold the text. The RANGE= operand may be coded to designate a start number and optionally an end number, which delimits the number of variables that will be generated.

WAIT={ YES | NO | *nnnn.nn* }

YES signifies that if no message is immediately available, the PPOPROC procedure is suspended until a message arrives. NO indicates that if no message is available control is to be returned to the procedure immediately. *nnnn.nn* signifies the number of seconds, and 1/100th seconds the procedure is suspended for, awaiting a message to arrive, before control will be returned to the procedure (maximum is 9999.99 seconds). WAIT=0 has the same effect as WAIT=NO.

Examples:

```
&PPOREAD ARGS RANGE=(20,80)
```

Reads or waits for the next unsolicited VTAM message specifying that it is tokenized into variables &20 to a maximum of &80. &ZVARCNT will be set to indicate how many variables were created.

```
&PPOREAD VARS=(*(3),A(2),B(3),C,D,E,F)
```

Reads or waits for the next unsolicited VTAM message, and tokenizes the message into individual words. *(3) indicates that the first 3 words are ignored, 2 characters of the next word are placed in the variable &A, three characters of the next word are placed in the variable &B and the next 4 words are placed in variables &C, &D, &E and &F respectively.

```
&PPOREAD VARS=ABC* RANGE=(1,50)
```

Reads or waits for the next unsolicited VTAM message, tokenizes it and places the text word by word into a series of automatically generated variables of the form ABC1 ABC2 ... ABC50 and so on. As many variables as required are generated, to the limit specified by the RANGE operand.

```
&PPOREAD STRING=(A(3),B(2),*(5),C(3))
```

Reads the next unsolicited VTAM message as a single string of text. The first 3 characters are placed in &A, the next 2 characters are placed in &B, the next 5 characters are ignored and the next 3 characters are placed in &C.

Notes:

Following an &PPOREAD a useful technique is the use of an &GOTO statement, using the VTAM message number, to go to the routine that will process the message.

```
&CONTROL NOLABEL
.READ
   &PPOREAD VARS=(A,B,C,D,E,F,G,H)
     &GOTO .&A
          -* Unexpected messages will be caught here,
          -* and returned for normal processing.
     &WRITE MON=YES DATA=UNEXPECTED MESSAGE NO: &A
     &GOTO .READ
.ISTnnnI
.ISTnnnI
:
: process ...
```

While testing and developing a PPOPROC procedure, you may need to terminate the current version and invoke a new updated copy. This is achieved using the SYSPARMS PPOPROC=FLUSH command, which is specifically designed to force termination of PPOPROC, followed by a SYSPARMS PPOPROC=member command to reinstate PPOPROC.

The &RETCODE system variable is set to zero (0) by &PPOREAD, except when WAIT=NO or WAIT=*n* was coded and no message arrived. In this case, &RETCODE is set to 12.

When PPOPROC terminates for any reason, standard PPO processing is resumed, and no PPO messages will be lost.

Preloading of PPOPROC using the LOAD command offers no performance advantage as the procedure remains loaded for the duration of processing.

&PPOREPL

Resumes normal processing of a VTAM PPO message, after replacing the message text.

```
&PPOREPL [ ALARM={ YES | NO } ]
         [ COLOR=color ]
         [ HLIGHT=highlight ]
         [ INTENS={ HIGH | NORMAL } ]
         [ NRD={ NO | OPER } ]
         [ SCAN={ YES | NO } ]
         [ DATA | TEXT=replacement text ]
```

Used within a PPOPROC procedure to return the updated message text from the previous &PPOREAD to your product region for processing.

The message is returned for standard delivery as specified in the DEFMSG table. NPF processing is not affected by the updated text. Network Partitioning (NPF) processing is not affected by the updated text. The resource name used by NPF is the name extracted from the original text.

Operands:

ALARM={ YES | NO }

Specifies whether the terminal alarm sounds when the message is delivered.

COLOR=*color*

Specifies the screen display color for the message. Valid values are:

RED GREEN BLUE TURQUOISE YELLOW PINK WHITE

HLIGHT=*highlight*

Specifies whether the message is to be highlighted. Valid values are:

REVERSE BLINK USCORE

INTENS={ HIGH | NORMAL }

Specifies the message intensity required.

NRD={ NO | OPER }

Sets the non-roll delete attribute for the message.

SCAN={ YES | NO }

Specifies whether @ characters within text are to be interpreted as word highlight markers.

DATA | TEXT=*replacement text*

The alert message text replacement generated.

Examples: &PPOREPL

```
&PPOREPL TEXT=APPLICATION &7 HAS FAILED
```

Notes:

An &PPOREPL issued when no VTAM PPO message is available is ignored.

After issuing an &PPOREPL, the message is no longer available in its original form and no &PPOCONT need be issued to return the message for normal processing. &PPOREPL is ignored if the message is an SPO message delivered to the PPO interface. An &PPOREPL is followed immediately by an &PPOREAD to make the next PPO message available.

&PPOREPL lets you replace a VTAM message with replacement text of your own choice.

&PROMPT

Writes the specified text to the user's terminal and waits for input.

```
&PROMPT [ NR ]
        [ NOPRT ]
        [ WAIT=nn ]
        { VARS=prefix* [ RANGE=(start,end) ] |
          VARS={ name | (name,name, ..., name) } |
          STRING=(name, name, ..., name) |
          ARGS [  RANGE=(start,end) ] }
        [ INPUT={ CHAR | HEX | HEXEXP } ]
        [ AUTONL={ YES | NO } ]
        [ prompt ]
```

&PROMPT is used within an NCL procedure to converse with a user at an LU1 terminal. Having written the supplied text to the terminal, the procedure will be suspended until the user enters some data in reply, or until any supplied time limit expires. Data entered is made available in user variables, as detailed below.

On completion of &PROMPT the system variable &ZVARCNT is set to the number of variables created or modified by the operation.

Operands:

NR

Specifies that no carriage return/line feed function is to be performed following the text. The print head of the terminal will be left at the end of the text. If NR is not specified, the print head will be positioned at the beginning of the next line for subsequent entry of data.

NOPRT

Specifies that the Inhibit Print SCS character is to be sent with the prompt text. For terminals that support this character, the subsequent input will not be printed or displayed at the terminal. The next &WRITE or &PROMPT will automatically re-enable printing.

WAIT=*nnnn.nn*

Specifies that the procedure can wait for *nnnn.nn* seconds for the user to enter some data (maximum value is 9999.99 seconds). If, after *nnnn.nn* seconds, no data has been entered, control is returned to the procedure. The &INKEY system variable set to a null value. If data is entered within the time specified, or if no time is specified, &INKEY contains the word ENTER.

VARS=

Specifies that the input is to be tokenized into the nominated variables before control is returned to the procedure. Each word of the input will be tokenized into the nominated variables from left to right. If insufficient variables are provided, some data will not be available to the procedure. Excess variables will be set to a null value. The formats of the operands that may be coded with VARS are described below.

prefix*

Denotes that variables are generated automatically during the tokenization process, and that variable names will be '*prefix1*,…,*prefix2*' and so on. The RANGE operand may be specified to indicate a starting and ending suffix number. *prefix** cannot be used with other variable names.

name

The name of a variable, excluding the ampersand (&).

***name*(*n*)**

As name, but *n* denotes the length of the data to be placed in the variable.

***(*n*)**

Denotes a skip operation, where n represents the number of units to be skipped during the tokenization process. On VARS= statements, n denotes 'skip this number of words'. An asterisk (*) by itself is the same as *(1).

STRING=

Specifies that no tokenization is to be performed. The entire text of the command line is treated as a single string and returned to the procedure in the nominated variables. The formats of the operands associated with STRING are:

name

User-specified variables, excluding the leading &, into which the string text is put. Text is placed into each variable up to the maximum length of that variable.

***name*(*n*)**

User-specified variables, excluding the leading &, into which the string text is put. Text is placed into each variable for the length specified *n*.

***(*n*)**

Denotes a skip operation, where n represents the number of units to be skipped during the tokenization process. On STRING statements, *n* denotes 'skip this number of characters'. An asterisk (*) by itself is the same as *(1).

ARGS

Denotes that the input will be tokenized and placed word by word into automatically generated variables of the form &1 through &*n*, depending on how many are required to hold the text. The RANGE= operand may be coded to designate a start number and optionally an end number, which delimits the number of variables that will be generated.

INPUT={ CHAR | HEX | HEXEXP }

Specifies the format of the data returned in the variables created by the &PROMPT operation. Default is standard character data. If HEX is specified it means that the data in the variables is pure hexadecimal (and therefore not directly procurable by NCL). HEXEXP means that the data in the variables is hexadecimal data represented as expanded hexadecimal, so that a hexadecimal byte with a value of, for example, 0A will appear in a variable as two characters 0A. HEX and HEXEXP support provide data transparency across &PROMPT operations.

AUTONL={ YES | NO }

YES specifies that the next output will be preceded by a new line character, if necessary to ensure the output starts in position 1. Specify NO if you do not wish this extra character to be automatically added.

prompt

The text of the message to be written. Normal variable substitution is performed before sending the message. Text is in upper and lower case.

Examples: &PROMPT

```
&PROMPT ARGS Please enter your log on request
&PROMPT NR WAIT=900 STRING=(RESPONSE) Enter log on request ===>
```

Notes:

Any data entered by the user when an &PROMPT statement is not outstanding will be queued pending the next &PROMPT statement, but only up to the 32 KB.

The text of an &PROMPT message can contain non-printable or hexadecimal characters.

Messages written with &PROMPT will not be logged to the activity log. The use of the &WRITE statement should be considered if information is to be written to the log.

The use of the &WRITE LF=NO operand is considered to create 'strike-over masks' on those terminals that do not support the Print Inhibit SCS character.

If the device to which an &PROMPT message is sent responds with a multiple-element chain in reply, &PROMPT completes on the arrival of each chain element. Multiple &PROMPT statements must be issued to receive the entire reply from the terminal. The &ZLU1CHN system variable (see page 910) indicates the position within the current chain of each message received.

Note: See also the sample EASINET procedure $EASINET in the distribution library.

&QEXIT

Terminates the current procedure and all higher levels, and optionally ends the current processing window.

```
&QEXIT [ NOMSG | PMENU ]
```

&QEXIT provides a means of ending the current processing window from within an NCL procedure. If issued, the current procedure is terminated, all higher levels of nested procedures are ended without further processing, and the window under which the procedure was executing is ended. If the user has only one processing window, &QEXIT terminates the window and logs the user off.

Operands:

NOMSG

If the termination of the window results in a user log off, then this operand determines if the standard disconnect message

N20005 SESSION TERMINATED AT *hh.mm.ss* ON *day dd-mon-year* FOR USERID=*userid*

will be issued indicating that the session has terminated. If this operand is specified then the session is terminated without issuing the termination message.

PMENU

If you code the PMENU operand, &QEXIT terminates all processing in the current window but returns the window to the primary menu position rather than terminating the window environment completely.

Examples: &QEXIT

```
&IF .&INKEY = .PF04 &THEN &QEXIT PMENU
&IF .&COMMAND = .LOGOFF &THEN &QEXIT
```

&REMSTR

Returns the data following the first occurrence of a specified character in a supplied string.

```
&REMSTR (c) text
```

&REMSTR is a built-in function and must be used to the right of an assignment statement.

&REMSTR will scan the specified data looking for the nominated selection character and returns the data following it into the target variable.

&REMSTR is used in conjunction with &SELSTR to split and manipulate strings of data.

&SELSTR assigns the data up to the nominated search character into the nominated variable. &REMSTR assigns the data following the nominated search character into the nominated variable.

Operands:

(*c*)

The selection character. The scan will be terminated at the first occurrence of the character '*c*'. If this character is not found the target variable will be set to null.

The data assigned to the target variable will not include the selection character.

If the selection character is a blank then a space should be specified. Where the selection character is a blank that is one of many successive blanks, only the first blank will be dropped. To eliminate multiple leading blanks in a variable, use either the &LBLSTR built-in function or the &PARSE verb.

text

The string of data to be split at the specified character.

Examples: &REMSTR

```
&MMSS = &REMSTR (.) &TIME  -* If &TIME is currently
                           -* 09.23.50
                           -* then &MMSS will be set
                           -* to 23.50
```

Notes:

&REMSTR is used with data returned from full-screen panels where it is necessary to internally manipulate the data for further processing.

Having split a string using &REMSTR, the &NBLSTR, &TBLSTR and &LBLSTR functions is used to strip leading and/or trailing blanks as necessary. It may be desirable to use &PARSE, where complex parsing is required.

If &CONTROL DBCS or DBCSN or DBCSP is in effect, &REMSTR is sensitive to the presence of DBCS data (see page 1204).

More information:

&PARSE (see page 526)
&SELSTR (see page 601)

&RETCODE

Returns the current system return code or sets a new return code value.

```
&RETCODE [ value ]
```

&RETCODE may be referenced as a system variable to return the current return code value or as a verb to set a new value.

A nested procedure can set a return code on the &END statement. On return to the higher level, &RETCODE contains the return code value. In addition, many NCL functions can set &RETCODE as an indication of the success or otherwise of the function.

&RETCODE is in the range 0 to 99 and is used to indicate the completion of a function performed by a nested procedure.

When used as a verb, &RETCODE sets the value of the &RETCODE variable.

Operands:

value

A new value in the range 0 to 99 to be placed in &RETCODE.

Examples: &RETCODE

```
&GOSUB .GETREQ
&IF &RETCODE NE 0 &THEN +
  &DO
    &SYSMSG = &STR NO REQUESTS IN QUEUE
    &RETCODE 4
    &RETURN &SYSMSG
  &DOEND

&CONTROL FINDRC
EXEC &REQUEST
&IF &RETCODE EQ 100 &THEN +
  &WRITE DATA=Requested Procedure &REQUEST not found
```

A procedure may set a value in the range 0 to 99 for &RETCODE. A value of 100 is set by the system if the &CONTROL FINDRC option is set. This option allows a procedure to determine the success of a request for a nested procedure. If the requested procedure does not exist and &CONTROL FINDRC is set processing continues but &RETCODE is set to a value of 100. If &CONTROL FINDRC is not set, the requesting procedure will terminate.

If no user exit is installed, &SECCALL EXIT sets the &RETCODE to 100.

When a process is initiated, &RETCODE will have a default value of 0.

An alternative to using the &END statement to pass a return code is to use the &RETURN statement, which can return variables to a higher nesting level, or &CONTROL SHRVARS, which allows sets of variables to be shared between procedures.

The &RETSUB statement may be coded with a value to be set as &RETCODE. Alternatively, &RETCODE is set within the body of the subroutine.

More information:

&END (see page 314)
&RETURN (see page 576)
&RETSUB (see page 575)
&CONTROL (see page 266)

&RETSUB

Returns from a subroutine within a procedure.

```
&RETSUB [ returncode ]
```

Common processing routines within an NCL procedure may be placed in subroutines. Control is transferred to a subroutine using the &GOSUB statement. The subroutine returns control to the statement following the &GOSUB by issuing an &RETSUB statement. There is a one to one correlation between each &GOSUB and its associated &RETSUB, allowing nested subroutine calls.

Operands:

returncode

A return code in the range 0 to 99 may be specified. This is available in the variable &RETCODE on return to the calling statement. A return code outside these ranges will cause termination of the procedure.

Examples: &RETSUB

```
.GETSUB  -* subroutine to get a request from a user

   -*
   -* Subroutine processing
   -*
 &IF &RETCODE = 4 &THEN +
     &RETSUB
   &RETSUB 0
```

Notes:

If &RETSUB is issued without a preceding &GOSUB, the statement is ignored.

If &RETSUB is specified without any other operands, any existing value of &RETCODE is retained. Thus the following are equivalent:

```
&RETSUB 4
```

```
&RETCODE 4
&RETSUB
```

More information:

&GOSUB (see page 354)

&RETURN

Returns to a higher nesting level passing the nominated variables.

```
&RETURN [ &var1 &var2 .... &varn ]
```

To return to a higher NCL nesting level, optionally passing the nominated variables.

Operands:

&*var*1 &*var*2 .... &*varn*

names of user variables that are to be returned to the higher nesting level. The variables must be specified including the leading ampersand (&). Resolution of complex variables (for example, &&1) is not supported on this statement. Any number of user variables is nominated.

Following an &RETURN statement in a nested procedure, processing will resume in the higher nesting level (the invoking procedure). The variables specified on the &RETURN statement will then be available to this higher level. If a variable of the same name existed in the higher level before the &RETURN statement was processed, it will be updated to reflect the value in the lower nesting level when the &RETURN statement was processed. If no variable of that name existed in the higher level, then a new variable will be created containing the data from the lower level. If the variable had a null value in the lower level when the &RETURN statement was issued a null value will also be set in the higher level when processing resumes.

Examples: &RETURN

```
& IF &RETCODE EQ 4 &THEN +
    &DO
     &SYSMSG = &STR DETAILS NOT ON FILE
     &RETCODE 8
     &RETURN &SYSMSG
    &DOEND
&RETCODE 0
&RETURN &DEVICEID &DEVICELOCN &DEVICEADDR
```

Notes:

If no variables are nominated, the &RETURN statement acts as an &END.

An &RETURN statement processed in a non-nested NCL procedure will act as an &END statement. If &CONTROL ENDMSG is in effect, messages indicating the return variables and their values will be generated. This allows processing of subroutine procedures to be verified by direct invocation.

The &RETURN statement makes it possible to develop modular procedures that perform common functions. Such procedures is invoked passing data using the EXEC statement. Having completed processing the procedure can return control using the &RETURN statement to return required data.

An &RETURN statement does not modify or change the names of the variables being returned. They must be referenced in the higher level procedure using the names specified on the &RETURN statement.

An alternative to the use of variables on &RETURN, is &CONTROL SHRVARS, which allows selected groups of user variables to be shared between nested levels.

If &CONTROL SHRVARS is in effect when a procedure is invoked, any variables specified on an &RETURN statement are returned and will supplement any variables returned as a result of SHRVARS processing.

The &RETCODE value returned to the calling procedure, will be the value that is current at the time &RETURN is executed.

More information:

&END (see page 314)
&CONTROL (see page 266)

&RSCCHECK

Returns a value indicating a user's access to resources within a specified resource group.

```
&RSCCHECK resource-group resource [ resource .... resource ]
```

&RSCCHECK is a built-in function and must be used to the right of an assignment statement.

The resource validation facility provides a means of controlling access to installation-defined resource groups from an NCL procedure.

If the specified resource is found within the user's scope, a value of YES is returned in the variable specified to the left of the assignment statement.

Multiple resources within a single resource group may be tested in a single statement. When multiple resources are tested, the user must be authorized for all the resources specified. If the user is not authorized for one or more of the resources, a NO value is returned.

Operands:

resource-group

The name for the resource group containing the resources for testing. The resource-group is a mandatory operand (resource groups are defined in the Resource List member specified in the user's UAMS record).

resource .... resource

The name for the resource to be tested. This is qualified by a network name for network resources, or by a resource qualifier if this is present in the resource group member definition control statement.

If the resource name can contain a period (.), and no qualifier is needed, then use a leading period in the value to indicate that there is no qualifier.

Examples: &RSCCHECK

```
&AUTH = &RSCCHECK $NMMSG NETA.NODE1
&IF &AUTH = NO &THEN +
    &ENDAFTER &WRITE DATA=NOT AUTHORIZED
```

```
&AUTH = &RSCCHECK USERGRP resourcea resourceb resourcec
```

Notes:

The maximum length for a resource is an optional 8-character qualifier, followed by a period (.), followed by a 64-character resource name.

If no RESOURCE LIST member is specified in the user ID record, then &RSCCHECK returns YES.

&RSCCHECK $NMMSG *resource* replaces &NFPMCHK *resource*.

&RSCCHECK $NMCMD *resource* replaces &NFPVCHK *resource*.

&RSCCHECK $NMCMDD *resource* replaces &NFPDCHK *resource*.

More information:

&NPFxCHK (see page 516)

&SECCALL

Communicates with the Userid Access Security Subsystem (UAMS) or your installation security exit.

```
&SECCALL ADD USERID=userid
             PWD=password
             FIELDS={ (nnnn,...,nnnn) | * }
           [ TYPE={ USER | GROUP } ]
           [ PREFIX=prefix ]

&SECCALL CHANGE USERID=userid
              [ PWD=password ]
              [ NEWPWD=newpassword ]
              [ FIELDS={ (nnnn,...,nnnn) | * } |
                DETAILS={YES | NO } ]
              [ PREFIX=prefix ]

&SECCALL CHECK USERID=userid
               PWD=password

&SECCALL DELETE USERID=userid

&SECCALL EXIT DATA={ xxx ...  xxx } |
              VARS={ xxx | (xxx,...,xxx) } |
              prefix* [ RANGE=( start,end ) ] |
              ARGS [ RANGE=( start,end ) ]

&SECCALL GET USERID=userid
            FIELDS={ (nnnn,...,nnnn) | * }
          [ OPT={ KEQ | KGT | KLT } ]
          [ PREFIX=prefix ]

&SECCALL QUERY [ PREFIX=prefix ]
               [ FIELDS={ (nnnn,...,nnnn) | * } ]

&SECCALL UPDATE USERID=userid
                FIELDS={ (nnnn,...,nnnn) |  * }
              [ PREFIX=prefix ]
```

The &SECCALL verb provides NCL with a method of communication with the security subsystem or security exit. Security is provided for the use of functions that potentially update the security definition of a user, that is, ADD, UPDATE, and DELETE. The CHANGE function is restricted to changing the issuer's user password unless the user is authorized for UAMS. The QUERY function returns the attributes that the user is using in their currently active region.

Examples: &SECCALL

```
&SECCALL ADD USERID=userid PWD=password FIELDS=*

&SEC0010 = NAME
&SEC0012 = LOCATION
&SEC0013 = 123-4567
&SECCALL CHANGE USERID=&USERID PWD=password +
               NEWPWD=newpwd DETAILS=YES

&SECCALL CHECK USERID=&USERID PWD=password
&SECCALL DELETE USERID=userid

&SECCALL GET USERID=userid FIELDS=(0010,0012,0013) OPT=KEQ

&SEC0010 = NEW-NAME
&SEC0012 = NEW-LOCATION
&SEC0013 = 123-4567
&SECCALL UPDATE USERID=userid FIELDS=(0010,0012,0013)

&A = PARM1
&B = PARM2
&C = PARM3
&SECCALL EXIT VARS=(A,B,C)
```

Notes:

&SECCALL is particularly useful from EASINET procedures, where it may be used to verify user ID privileges, validity, and passwords before passing the user to a selected application.

&SECCALL is designed to shield the issuing NCL procedure from any knowledge of the type of security subsystem in operation, and provides the same function whether UAMS is in use or an installation-provided security exit is active. If a security exit is provided, &SECCALL causes a call to that exit.

Note: For more information about parameters passed to the exit and information expected from the exit, see the *Security Guide*.

&SECCALL CHECK function can validate access by a user from the IP host to the local port. This verification is controlled by the system parameter IPCHECK=REGISTER | NONE, which is set only during system initialization. If a user is successfully validated by &SECCALL CHECK, with IPCHECK set to REGISTER, then the system associates their user ID with the IP connection.

IPCHECK=REGISTER results in user IDs being recorded for IP connections by product region signon and &SECCALL CHECK processing.

Structured Fields:

All user ID security attributes use structured fields to exchange data between NCL and the security subsystem or security exit.

All user variables referring to structured fields that are exchanged between NCL and the security subsystem are in the form &*prefixnnn* where *prefix* is defined by the PREFIX operand and *nnn* refers to a structured field that is defined for your product region. (Structured fields referring to components and applications that are not licensed, or that have been excluded, are not available from the security subsystem.)

For example, structured field number 0010 represents the USERID NAME. On completion of &SECCALL GET USERID=*userid* FIELDS=0010, a variable exists called &SEC0010, containing the user ID of the user for which information was requested.

If a particular structured field contains more than one subfield, one variable is returned for each subfield. In this case the variable representing the first subfield is named in the format described previously, and the remaining subfields are returned in variable names in the format:

&SEC*xxxx*B
&SEC*xxxx*C
...
&SEC*xxxx*Z

For example, structured field number 0016 represents the terminals to which a user is restricted and includes three subfields. On completing &SECCALL GET USERID=*userid* FIELDS=0016, three variables exist, in the format:

```
&SEC0016
&SEC0016B
&SEC0016C
```

Optional features can also generate structured fields to represent feature-dependent user ID attributes or privileges.

Note: For more information, see the *Security Guide*.

&SECCALL ADD

Requests the nominated user ID be added to the UAMS database or to the external security exit. UAMS authority is required to use this function.

```
&SECCALL ADD USERID=userid
             PWD=password
             FIELDS={ (nnnn,...,nnnn) | * }
           [ TYPE={ USER | GROUP } ]
           [ PREFIX=prefix ]
```

Operands:

USERID=*userid*

The user ID of the user definition to be added. This user ID must be 1 to 8 characters long.

PWD=*password*

The initial password for this user ID. This password is used the first time the user logs on to your product region. It is then expired and the user is requested to enter a new password.

This field is required if a user definition is being added to a local UAMS data set. It is not required if a security exit is installed. However, if a password is supplied, it is available to the security exit. It is not required if a group definition is being added.

The password must be 1 to 8 characters long, or a minimum length as specified by the SYSPARMS PWMIN operand and a maximum length as specified by the SYSPARMS PWMAX operand. (This default is overridden by using the SYSPARMS PWMAX and PWMIN operands.)

FIELDS={ (*nnnn,...,nnnn*) | *}

Provides a list of nominated structured field values for the user ID being defined. A list of structured field values is supplied; all fields not supplied are set to defaults. To set all structured fields, you can use an asterisk (*).

Note: For more information about structured fields, see the *Security Guide*.

The user variables that identify the structured fields in the field list must be in the form *&prefixnnnn*, where *prefix* is defined by the PREFIX operand and *nnnn* is a defined structured field for this system.

TYPE={ USER | GROUP }

Defines the type of definition to be added:

TYPE=USER (the default) indicates that a user definition is to be added and represents an individual user ID definition.

TYPE=GROUP identifies the definition being added as being a group definition. Group definitions are used to group the security definitions for a number of users. For example, when a user's security definition is retrieved at logon time and the user is defined with a group definition, the security attributes for the group definition are used.

PREFIX=*prefix*

Defines a 1- to 7-character prefix of the variables referred to by the FIELDS operand. The default is SEC.

Return Codes:

The result of the ADD function sets &RETCODE as follows:

0

Request successful. The user definition has been added to the UAMS database, or to the external security exit.

4

Request unsuccessful. The user definition was not added to the UAMS database, or the external security exit rejected the add. &SYSMSG is set to contain an error message indicating cause of failure.

More information:

&SECCALL (see page 580)

&SECCALL CHANGE

Allows a user's password and/or user details to be changed. This function is used to change the user's password for the user executing this function, or to force change another user's password (in this case the password is also expired). The user's new password and/or user details must be specified.

```
&SECCALL CHANGE USERID=userid
              [ PWD=password ]
              [ NEWPWD=newpassword ]
              [ FIELDS={ (nnnn,...,nnnn) | * } |
                DETAILS={ NO | YES } ]
              [ PREFIX=prefix ]
```

Operands:

USERID=*userid*

The user ID of the target user for which the password is to be changed. The user ID must be 1 to 8 characters long and defined to the security subsystem. If the user ID is other than that of the issuing user ID then the issuing user must be authorized for UAMS.

PWD=*password*

The current password of the issuing user ID. The current password must be supplied if the user is changing their own password. This operand is required if the issuing user is not authorized for UAMS. If the user is authorized for UAMS and the current password is not provided then the new password is expired and the user must change their password the next time they log on to your product region.

NEWPWD=*newpassword*

The new password to be used by the user the next time they log on to your product region. The new password must be 1 to 8 characters long or a minimum length as specified by the SYSPARMS PWMIN operand and a maximum length as specified by the SYSPARMS PWMAX operand.

FIELDS={ (*nnnn*,...,*nnnn*) | * }

The FIELDS operand provides a list of nominated structured field values for the user ID being defined. A list of structured field values is supplied. To set all structured fields, you can use an asterisk (*). The user variables that identify the structured fields in the field list must be in the form &*prefixnnnn*, where *prefix* is defined by the PREFIX operand and *nnnn* is a defined structured field for this system.

Note: For more information about structured fields, see the *Security Guide*.

Valid structured fields for &SECCALL CHANGE are as follows:

0011

User name

0012

User address

0013

User telephone phone number

0014

User language code

001D

User email address

0030

User time zone name

0520

Event notification services attribute 1

0521

Event notification services attribute 2

0522

Event notification services attribute 3

0523

Event notification services attribute 4

Note: The FIELDS operand cannot be used in conjunction with the DETAILS operand.

DETAILS={ NO | YES }

Specifies whether user details, name, location, and phone number are to be changed.

DETAILS=NO specifies that user details are not to be changed.

DETAILS=YES indicates that the user details are to be changed. The user details are identified as the following structured fields:

- 0011 (user name)
- 0012 (user location)
- 0013 (user phone number)
- 0014 (user language code)
- 001D (user email address)
- 0030 (user time zone name)

The user details are referred to from NCL in the form &*prefix*0011, &*prefix*0012, and &*prefix*0013, where *prefix* is defined by the PREFIX operand.

The DETAILS operand cannot be used in conjunction with the FIELDS operand.

PREFIX=*prefix*

Defines a 1- to 7-character prefix of the variables referred to by the DETAILS operand. The default is SEC.

Return Codes:

The result of the CHANGE function sets &RETCODE as follows:

0

Request successful. The user's password has been successfully updated.

4

The user's password and/or user details update was unsuccessful, function unsupported.

8

The user's password and/or user details were not updated. Function supported but an error occurred. &SYSMSG is set to contain an error message indicating cause of failure.

The &ZFDBK system variable is set as follows after the fields or user details CHANGE function:

0

User details update was successful.

4

User details update was unsuccessful.

More information:

&SECCALL (see page 580)

&SECCALL CHECK

&SECCALL CHECK Provides the ability to determine whether the nominated user ID and password combination would be allowed to log on to the system in which the NCL procedure is executing.

```
&SECCALL CHECK USERID=userid
               PWD=password
```

Operands:

USERID=*userid*

The user ID of the target user for which the check is to be performed. The user ID must be 1 to 8 characters long and defined to the security subsystem. The call is valid only for the issuing user ID or from an EASINET NCL procedure.

PWD=*password*

The current password of the user ID.

Return Codes:

The result of the CHECK function is indicated by the setting of &RETCODE as follows:

0

The password is correct and logon to this system would be successful.

4

The password is correct but logon would fail for other reasons. &SYSMSG is set to contain an error message indicating the cause of the failure. The conditions under which your product region sets this return code are:

- User ID already logged on and the user is not authorized for multiple logons.
- The user ID is suspended.
- Logon not allowed from this terminal.
- Maximum users logged on.
- System is shutting down-logons have been stopped.

If a security exit is installed, the exit may refuse logons for other reasons. An appropriate error message is set in &SYSMSG.

8

Password is correct but it has expired. Logon would succeed but the user is required to change their password.

12

Password is correct but this is a new user ID definition. Logon would succeed but the user is required to change their password.

16

Password is invalid. Logon fails

20

User ID is unknown. Logon fails.

24

Request failed or the function is not available.

28

Password is valid but the user ID is not defined as a user on UAMS. (This applies only if a partial exit is in place.)

The return codes from the &SECCALL CHECK function are supported as documented by native UAMS processing. The &ZFDBK system variable reflects the setting of the return code as set by the security exit.

Note: For more information about these return code settings, see the *Security Guide*.

Notes:

If a security exit is in use, the CHECK call is handled by the exit. The exit may choose to support the same return codes for the same results, or it may not. Check the return codes supported by your security exit before using the CHECK option of &SECCALL.

The CHECK call completing with return code 28 indicates that the security exit has verified the user ID/password combination as valid, but the user ID is not actually defined to your product region as a valid user. This is common in cases where EASINET is used as the network security gateway and user ID/password checking is performed by your product region before the user is allowed to access any application in the network. In these circumstances user ID/password validation is performed for users who may not have access to your product region but do have access to other network applications.

If a security exit is in place, then &ZFDBK reflects the value of the return code set by the security exit. This is useful as supplemental information to &RETCODE. For example, if a user who is not authorized for multiple signons attempts to unlock his terminal, an &SECCALL CHECK is executed on his behalf. As the user is already signed on, &SECCALL CHECK returns an &RETCODE of 4. If the user's password expires while the terminal is locked, &RETCODE is still 4. It is not possible to discern from &RETCODE that the password has expired. However, in the first case, &ZFDBK is set to 0, and with the expired password condition, it is set to 4. The $NMLOCK procedure is then able to determine that the password has expired and react accordingly.

The call is valid only for the issuing user ID or from an EASINET NCL procedure.

&SECCALL DELETE

Provides the ability to delete a nominated user definition from the UAMS database or the external security system. UAMS authority is required to use this function.

```
&SECCALL DELETE USERID=userid
```

Operands:

USERID=*userid*

The user ID of the target user that is to be deleted. The user ID must be 1 to 8 characters long and defined to the security subsystem.

Return Codes:

The result of the DELETE function is indicated by the setting of &RETCODE as follows:

0

Result successful. The user ID has been deleted.

4

Result unsuccessful. The user definition was not deleted from the UAMS database or the external security exit rejected the exit. &SYSMSG is set to contain an error message indicating the cause of failure.

&SECCALL EXIT

Provides a direct interaction between an NCL procedure and the installation's full security exit or partial security exit.

```
&SECCALL EXIT DATA=xxx ...  xxx |
              VARS={ xxx | (xxx, ...,xxx) } |
              prefix* [ RANGE=(start,end) ] |
              ARGS [ RANGE=(start,end) ]
```

Operands:

DATA=*xxx* ... *xxx*

Any user variable that contains data to be passed to the security exit. Any number of variables is passed to the exit within the limitations of the maximum NCL statement length. The data passed to the exit is segmented into words on blanks.

VARS=

Specifies the names of the source variables to be passed to the security exit. Each variable will be passed as a separate word to the security exit. The formats of the operands that is coded with VARS= are:

VARS=*xxx* | (*xxx*, ..., *xxx*)

The name of a variable, excluding the ampersand (&) prefix. A variable list is supplied by enclosing in brackets multiple names separated by commas.

VARS=*prefix [RANGE=(*start*, *end*)]**

Supplies the leading characters terminated by an asterisk that denote a numeric range of variables. If the RANGE= operand is specified or allowed to default, then an ascending numeric range is generated by concatenating the supplied prefix with a numeric suffix that is sequentially incremented within the supplied start and end values. The start and end values must be in the range 0 to 65535.

ARGS [RANGE=(*start*, *end*)]

Supplies variables in the form &1 to &*n*. The RANGE= operand, as described previously, is coded to designate a start number and optionally an end number, which delimits the number of variables passed.

Return Codes:

On return from this function, &RETCODE is set to whatever return code was set (in Register 15) on return from the security exit. This must be in the range 0 to 99.

On return from the exit, returned data is constructed into variables &1, &2 ... &*n*, regardless of the names of the variables used to pass data to the exit. This function returns as many variables as there were operands passed.

Notes:

&SECCALL EXIT allows direct communication between EASINET NCL procedures and a security exit. This function allows a procedure to pass the contents of nominated variables to the security exit for processing and the security exit can return information exchanged between the procedure in defined variables. The content and processing of the information exchanged between the procedure and the exit is determined absolutely by the installation. Your product region has no knowledge of, or impact on, that information.

Using &SECCALL EXIT is valid only if a full or partial security exit is installed.

The NCL procedure must specify sufficient variables to hold all expected information from the security exit. Null variables are passed to the security exit when using &SECCALL EXIT DATA=. If the exit deletes all information from one of the variables, that variable is returned as a null variable. The procedure should be written to cater for this circumstance.

&SECCALL GET

Provides the ability to retrieve the nominated user's security attributes and privileges.

```
&SECCALL GET USERID=userid
             FIELDS={ (nnnn,...,nnnn) | * }
           [ OPT={ KEQ | KGT | KLT } ]
           [ PREFIX=prefix ]
```

Operands:

USERID=*userid*

The user ID of the target user which is to be retrieved. The user ID must be 1 to 8 characters long and defined to the security subsystem.

FIELDS={ (*nnnn*,...,*nnnn*) | * }

Provides a list of nominated structured field values for which the security attributes for the user ID are to be retrieved.

Note: For more information about structured fields, see the *Security Guide*.

A list of structured field values is supplied or an asterisk (*) is used to specify that all security attributes are to be returned. The variables are set as indicated by the required structured fields and are returned to the NCL procedure in the form &*prefixnnnn*, where *prefix* is defined by the PREFIX operand and *nnnn* is a defined structured field for this system.

OPT={ KEQ | KGT | KLT }

Indicates which record, in relation to the user ID specified in the USERID operand, is to be retrieved.

OPT=KEQ indicates that you wish to retrieve the user definition with an exact match on the specified user ID.

OPT=KGT indicates that you wish to retrieve the user definition with the lowest key value greater than the specified user ID. A key of blanks retrieves the first user ID record on UAMS.

OPT=KLT indicates that you wish to retrieve the user definition with the highest key value less than the specified user ID.

If a full security is installed, then the options KGT and KLT are presented to the security exit. It is up to the security exit to decide whether to support the call and to return the correct user ID information.

PREFIX=*prefix*

Defines a 1- to 7-character prefix of the variables referred to by the FIELDS operand. The default is SEC.

Return Codes:

The result of the GET function is indicated by the setting of &RETCODE as follows:

0

Request successful. The user ID attributes are available in the user variables generically named &SEC*nnnn*.

4

Request unsuccessful. The user ID information was not available. &SYSMSG is set to contain an error message indicating cause of failure.

More information:

&SECCALL (see page 580)

&SECCALL QUERY

Provides the ability to return the security attributes that the user is using in their current region.

```
&SECCALL QUERY [ PREFIX=prefix ]
               [ FIELDS={ (nnnn,...,nnnn) | * } ]
```

Operands:

PREFIX=*prefix*

Defines a 1- to 7-character prefix of the variables referred to by the FIELDS operand. The default is SEC.

FIELDS={ (*nnnn*,...,*nnnn*) | * }

Provides a list of nominated structured field values for which the security attributes for the user ID are to be returned.

Note: For more information about structured fields, see the *Security Guide*.

A list of structured field values is supplied, or an asterisk (*) is used to specify that all security attributes are to be returned. The variables are set as indicated by the required structured fields and are returned to the NCL procedure in the form &*prefixnnnn*, where *prefix* is defined by the PREFIX operand and *nnnn* is a defined structured field for this system. (For more information about the FIELDS operand, see Structured Fields in this chapter.)

If the FIELDS operand is omitted, then all security attributes are returned.

Return Codes:

The result of the QUERY function sets &RETCODE as follows:

0

Request successful. The user ID attributes are available in the user variables generically named &*prefixnnnn*.

4

The request was unsuccessful. The &SYSMSG system variable is set to an error message indicating the cause of failure.

Notes:

The query function is issued only by a user logged on to your product region. Default values for all structured fields are returned if the function is issued from EASINET.

There are no calls made to either the native UAMS database or to the external security exit, as the information is obtained from the user's current region.

If a request is made for a structured field that is part of an unlicensed or excluded feature, then the relevant variable is null.

The following structured fields are valid for &SECCALL QUERY:

0010

Current user ID

0014

User language code

0017

User time-out control (Y/N)

0019

Multiple signon capacity (Y/N)

0020

OCS access privilege (Y/N)

0021

Broadcast Services access privilege (Y/N)

0022

Network Services access privilege (Y/N)

0023

System Support privilege (Y/N)

0025

CA SOLVE:FTS access privilege (Y/N)

0026

NEWS access privilege (Y/N)

0027

MAI-FS access privilege (Y/N)

0028

User Services procedure name

0029

User's NCL procedure library

002A

UAMS access privilege (Y/N)

002B

Operations Management privilege (Y/N)

002D

NCS access privilege (Y/N)

002E

User's SPLIT/SWAP privilege (Y/N)

002F

Library Services path name

0030

User's time zone name

0050

OCS command authority level

0051

OCS Monitor status (Y/N)

0055

PPO message receipt option (Y/N)

0057, B

NPF message restriction option (Y/N)

0059

OCS MSG message receipt (Y/N)

005A

OCS unsolicited message receipt (Y/N)

0081

Information Services access (Y/N)

0100

CA SOLVE:FTS definition privilege (P/S/N)

0101

CA SOLVE:FTS private request privilege (Y/N)

0102

CA SOLVE:FTS system request privilege (Y/N)

0103

CA SOLVE:FTS private control privilege (Y/N)

0104

CA SOLVE:FTS system control privilege (Y/N)

0105

CA SOLVE:FTS private function mask (12 characters)

0106

CA SOLVE:FTS system function mask (12 characters)

0150

NEWS statistics reset privilege (Y/N)

0151

NTS access privilege (Y/N)

0180, B

AOM message delivery and routing codes

0181, D

AOM MVS SYSCMD console authority

0182

AOM MSG level (20 characters)

0183, E

AOM VM SYSCMD authority (Y/N)

0185

AOM VOS3/JSS4 SYSCMD authority (0 to 15)

0200

MAI-FS privilege class (A/B/C/D)

0201

MAI-FS model user ID (8 characters)

0202

MAI-FS A and E command capability (Y/N)

0510

Panel command access authority (Y/N)

0511

System services access (Y/N)

0530

TCP/IP Services access privilege (0-2)

0580

CA SOLVE:NetMail access (Y/N)

0601

MODS access (Y/N)

&SECCALL UPDATE

Provides the ability to update the nominated user's security attributes and privileges. UAMS authority is required for this function.

```
&SECCALL UPDATE USERID=userid
                  FIELDS={ (nnnn,...,nnnn) | * }
                [ PREFIX=prefix ]
```

Operands:

USERID=*userid*

The user ID of the target user which is to be updated. The user ID must be 1 to 8 characters long and defined to the security subsystem.

FIELDS={ (*nnnn,...,nnnn*) | * }

Provides a list of nominated structured field values for which the security attributes for the user ID are to be updated.

Note: For more information about structured fields, see the *Security Guide*.

A list of structured field values is supplied, or an asterisk (*) is used to specify that all security attributes are to be updated. The user variables expected by the UPDATE function must be in the form &*prefixnnnn*, where *prefix* is defined by the PREFIX operand and *nnnn* is a defined structured field for this system and nominated in the field list (or with an asterisk).

The following fields are not valid for &SECCALL UPDATE:

- 0010 User ID
- 0018 Last updated date/time
- 001B User/group definition type

PREFIX=*prefix*

Defines a 1- to 7-character prefix of the variables referred to by the FIELDS operand. The default is SEC.

Return Codes:

The result of the UPDATE function is indicated by the setting of &RETCODE as follows:

0

Request successful. The user ID information has been updated.

4

Request unsuccessful. The user ID information was not updated. The &SYSMSG system variable is set to an error message indicating cause of failure.

More information:

&SECCALL (see page 580)

&SELSTR

Returns the data preceding a specified character in a supplied string.

```
&SELSTR (c) text
```

&SELSTR is a built-in function and must be used to the right of an assignment statement.

To scan the specified data looking for the nominated selection character and return the data up to the specified selection character.

&SELSTR is used to split a variable at a nominated point.

&SELSTR is used in conjunction with &REMSTR to split and manipulate strings of data.

Operands:

(*c*)

The selection character. The scan will be terminated at the first occurrence of the character *c*. If this character is not found the entire string will be assigned to the target variable. If the selection character is the first character found, a null value will be assigned to the target variable.

If the selection character is a blank then a space should be specified.

text

The data to be scanned for the occurrence of the character *c*.

Examples: &SELSTR

```
&A = &SELSTR ( ) &1      -* select data up to first blank

&HOUR = &SELSTR (.) &TIME
```

Notes:

&SELSTR is ideal for use with data returned from full-screen panels where it is necessary to internally manipulate the data for further processing.

&REMSTR acts as the converse to &SELSTR. &SELSTR assigns the data up to the specified character, while &REMSTR assigns the data following the specified character.

&PARSE may also be used for dissecting strings.

If &CONTROL DBCS or DBCSN or DBCSP is in effect, &SELSTR is sensitive to the presence of DBCS data (see page 1205).

More information:

&PARSE (see page 526)
&REMSTR (see page 572)
&SUBSTR (see page 650)

&SETBLNK

Returns a blank string.

```
&SETBLNK [ length ]
```

&SETBLNK is a built-in function and must be used to the right of an assignment statement.

The &SETBLNK function sets a variable to blanks and optionally sets it to a specific length. If the target variable already exists, then it will be set to blanks for the current length of the variable, if the length operand is omitted. If length is specified, then the length of the variable will be truncated or extended as necessary.

If the target variable (to the left of the assignment statement) does not exist (a null), then one will be created and blank filled to the specified length. If length is not specified, then the maximum length for a variable will be assumed.

Setting a length of zero will cause the variable to be deleted (set to null).

Operands:

length

The new length for the target variable. This may be from 0 to the maximum length of a variable.

Examples: &SETBLNK

```
&VAR = &SETBLNK 20

&ABC = &SETBLNK
```

Notes:

Use of &SETBLNK will destroy any data currently assigned to an existing variable.

&SETLENG

Sets the length of a variable.

```
&SETLENG length
```

&SETLENG is a built-in function and must be used to the right of an assignment statement.

The &SETLENG function forces a variable to a specific length.

If the specified length is less than the current length, the data within the variable will be truncated. If the specified length is greater than the current length of data in the variable, trailing blanks will be added.

If the variable does not exist (a null), then it is created and padded with blanks to the specified length.

Setting a length of zero causes the variable to be deleted (set to null).

Operands:

length

The new length for the target variable. This must be in the range 0 to the maximum length of the variable.

Examples: &SETLENG

```
&KEY = AAABBBCCC
&KEY = &SETLENG 5  -* &KEY is set to AAABB

&NAME = &SETLENG &NAMELEN
```

Notes:

The &SETLENG function is ideal for setting up keys of fixed length that are to be used in subsequent file processing and for setting fields to known lengths, perhaps to interface with other off-line systems.

Using &SETLENG can add trailing blanks to pad out existing data to the new length. If necessary, use the &TBLSTR function to remove these trailing blanks when no longer required.

If a series of variables are to be displayed in a tabular form (for example, in a report, or as message lines on a panel), the &OVERLAY statement is used instead.

If &CONTROL DBCS or DBCSN or DBCSP is in effect, &SETLENG is sensitive to the presence of DBCS data (see page 1207).

&SETVARS

Extracts named keywords and associated data from a data string.

```
&SETVARS [ PREFIX=ppp ]
         [ MODFLD={ NO | YES } ]
         [ ERROR={ ABORT | CONTINUE } ]
         [ DUPLICATE={ YES | NO } ]
         [ CONCAT={ YES | NO } ]
         [ KEYWORDS=( list-of-keywords ) ]
         [ PARMS | ARGS [ RANGE=( start,end ) ] |
           VARS=prefix* [ RANGE=( start,end ) ] |
           DATA=value ]
```

&SETVARS allows a procedure to analyze either a data string or a set of variables, to extract named keywords and their associated data. &SETVARS takes data in the form *keyword=data*, *keyword=data*,..., creates variables for each keyword, and assigns the corresponding data to it. The variable names created are (by default) the name of the keyword. However, a prefix may be specified for the names.

Operands:

PREFIX=*ppp*

An optional 1- to 8-character prefix for the generated variable names. If PREFIX is not specified, the variable created is named the same as the corresponding keyword.

If you specify a numeric prefix, all keywords must be all-numeric.

MODFLD={ NO | YES }

When set to YES, the MODIFIED attribute is reset for all existing variables, and the MODFLD attribute is set on generated output variables.

ERROR={ ABORT | CONTINUE }

Indicates what action is to be taken if an error is found during processing of input data or variables.

ABORT indicates that the procedure is to abort. CONTINUE causes &RETCODE to be set to 8, &SYSMSG to be set to an error message, and creation of variables to cease at the variable in error.

Any syntax errors always result in the procedure aborting, regardless of the setting of the ERROR operand.

DUPLICATE={ YES | NO})

Indicates whether duplicate keywords are allowed. The default, YES, indicates that duplicates are allowed.

CONCAT={ YES | NO }

When CONCAT=YES is specified, all the input data specified by the PARMS, ARGS, and VARS= operands is concatenated together (without blanks), and treated as if the entire string had been specified on the DATA= operand.

CONCAT=YES cannot be specified when DATA= is specified.

KEYWORDS=(*list-of-keywords*)

This is an optional list of valid keywords for input. If specified, all keywords must be found in the list. Any listed keywords (prefixed by the optional prefix) have the associated variable deleted before processing.

Each keyword must be a valid part of a variable name and, if the specified prefix is numeric, then the keywords must all be numeric.

PARMS

This is the default operand meaning that &1, &2, &3, from 1 to &PARMCNT, are used (that is, the parameters supplied to the procedure).

This is similar to ARGS RANGE=(1,&PARMCNT), except that, if there are no parameters, then there are no errors.

ARGS [RANGE=(*start*,*end*)]

Indicates that numeric variables in the range are to be used as input.

VARS=*prefix [RANGE=(*start*,*end*)]**

Indicates that nominated prefixed variables are to be used as input.

Note: The previous three options mean that a set of variables is used as input. The range is processed as follows:

- Null variables are ignored.
- Each non-null variable must be in the format: *keyword=value*, where the keyword and the equal sign are required and the value (which may be null) is assigned to the prefix keyword variable, without modification.
- Duplicate and keywords list checks are done as requested.

DATA=*value*

This optional operand can contain input data in the following formats:

kwd=*value*

kwd='*value*'

kwd="*value*"

kwd=

or any combination of the above.

If quotes are used then they must be validly paired, and are stripped. Any double quote is singled up in a target variable, and blanks (real ones) is placed in a variable. Otherwise data is delimited on blanks (real or embedded).

Example 1:

If your procedure is executed as follows:

```
EXEC MYPROC PARM1='VALUE1 VALUE2' PARM2=OPTION
```

the parameter string could be analyzed using &SETVARS as follows:

```
&SETVARS PREFIX=AA DATA=&ALLPARMS
```

This results in the variable &AAPARM1 containing VALUE1 VALUE2, and &AAPARM2 containing OPTION.

Example 2:

If your procedure is executed as follows:

```
EXEC MYPROC PARM1=VALUE PARM2=OPTION
```

the parameter string is analyzed using &SETVARS as follows:

```
&SETVARS PREFIX=AA
```

This results in the variable &AAPARM1 containing VALUE and &AAPARM2 containing OPTION.

Notes:

The DATA= option allows you to easily pass data to a procedure and parse it into variables, including embedded blanks.

The ERROR=CONTINUE option allows you to keep control even if the data is incorrect.

&SMFWRITE

(z/OS systems only) Writes a record to the SMF data set.

```
&SMFWRITE    [ TEST ] [ RECID=nnn ]
             hexadecimal-string
```

NCL processes executing in z/OS regions could need to record SMF information for later analysis and off line processing. You can write a procedure that creates an SMF record, holding the information that you wish to record, then use &SMFWRITE to write the record to the SMF data set.

Operands:

TEST

Specifies that all formatting of the SMF record is to be performed but the record will not be written to SMF. Instead the data provided is dumped to the activity log with an SMFWRITE eye catcher.

RECID=*nnn*

Specifies the (decimal) SMF record type to be generated. Range is 0 to 255. Default is as set by the SYSPARMS SMFID operand.

If RECID is not specified, the following occurs:

- The default as set by the SMF Customizer Parameter Group (/PARMS) is used.
- The SMF Record Identifier field is set to the desired default value. To Display this value, use the /SYSPARM shortcut to display the SMFID sysparm.

hexadecimal-string

The data to be written as an SMF record. This is provided in expanded hexadecimal form, that is, if the record is to contain the characters AB, the hexadecimal string will express them as C1C2.

Example1:

The following &SMFWRITE call uses the default SMF Record Identifier value and generates the SMF data set record containing the string set in the &MYDATA variable.

```
&MYDATA = &STR THIS DATA WILL BE WRITTEN TO SMF
&SMFREC = &HEXEXP &MYDATA
&SMFWRITE &SMFREC
```

Example 2:

The following code extract only sends the output to the product region's activity logs as TEST output.

```
&MYDATA = &STR THIS TEST SMF DATA IS SENT TO LOG ONLY
&SMFREC = &HEXEXP &MYDATA
&SMFWRITE TEST RECID=132 &SMFREC
```

This produces the following output in the activity log:

```
SMFWRITE +0000 00380000 3E840003 79630107 024FC3C1  .... .d.. `... .|CA
SMFWRITE +0010 F3F1E3C8 C9E240E3 C5E2E340 E2D4C640  31TH IS T EST SMF
SMFWRITE +0020 C4C1E3C1 40C9E240 E2C5D5E3 40E3D640  DATA IS SENT TO
SMFWRITE +0030 D3D6C740 D6D5D3E8                    LOG ONLY
```

Notes:

- The header of the SMF record is created by the &SMFWRITE verb.
- *hexadecimal-string* is compressed by the &SMFWRITE verb. The use of the &HEXEXP built-in function assists in the formatting of hexadecimal data strings prepared for &SMFWRITE.
- Enter **/SYSPARM** to access the SMFTRACE system parameter.
- The SMFTRACE system parameter (/SYSPARM at the command prompt), when set to YES, can be used to dynamically request the writing of the SMF record to the activity log. It is done regardless of the TEST operand specification. When the TEST operand is specified on the &SMFWRITE verb, the SMF record is written to the activity log only, regardless of the SMFTRACE system parameter setting. Normally, only background users can write data to SMF using &SMFWRITE verb. The SMF record is written to the activity log regardless of the authorization. Message N25B02 is logged if authorization fails.
- The use of &SMFWRITE is authorized by the NCL file ID access authorization exit. For more information about using &SMFWRITE, see the *Security Guide*. By default, &SMFWRITE is always authorized if executed by a system level procedure and always denied if executed by a process executing on behalf a real user. You must implement an NCL File ID Access Authorization exit to authorize use of &SMFWRITE by ordinary users.
- &SMFWRITE operates only if your product region is running as an authorized task.

Return Codes:

On completion, &RETCODE is set to one of the following values:

0

Operation was successful. &ZFDBK = SMF return code.

4

SMF inactive. &ZFDBK = SMF return code.

8

SMF operation failed. &ZFDBK = SMF return code.

12

Invalid system for request or not APF authorized.

16

User is not authorized for request.

More information:

&HEXEXP (see page 362)

&SNAMS CANCEL

The &SNAMS CANCEL verb cancels an outstanding MDS request. Effectively, an MDS error message is generated and sent to the responder, indicating that the requester no longer requires a reply.

All MDS replies already received by SNAMS on behalf of the canceled request are discarded. The senders of these messages are not informed of this situation.

This verb has the following format:

```
&SNAMS CANCEL ID=requestid
```

Operands:

ID=*requestid*

Specifies the identifier of the outstanding request to cancel. Its value was returned in the &ZSNAMID variable when the outstanding MDS request was issued using the &SNAMS SEND verb.

Return Codes:

&RETCODE	&ZFDBK	Meaning
0	0	Request canceled, no MDS replies
	1	Request canceled, MDS replies discarded
4	0	Not an outstanding request, no MDS replies
	1	Not an outstanding request, MDS replies discarded

For &RETCODE 0, an MDS error is sent to the responder.

&SNAMS DEREGISTER

The &SNAMS DEREGISTER verb deregisters a registered NCL application from SNAMS.

A registered NCL application uses this option to deregister itself from SNAMS. Consequently, it is no longer a valid destination for incoming MDS-MUs. Messages already received by the MDS Router on behalf of this application are discarded; the senders are not informed of this situation.

Deregistration also flushes active transactions for which the deregistered application is the responder. Partner applications are informed of this deregistration through an MDS error (sense code X'08A90003' for unit-of-work canceled).

If the application has registered interest in focal point updates with MS-CAPS (that is, it is an entry point application) or if it is registered as a focal point application, then deregistration signifies the end of such roles. Active entry point nodes are informed of the deregistration through an MDS error (sense code X'08A80003' for unknown application name).

This verb has the following format:

```
&SNAMS DEREGISTER
     [ NAME=applicationname | ID=registrationid ]
```

Operands:

NAME=*applicationname*

Specifies the name of the application to deregister. The operand is not required but, if supplied, the ID= operand cannot be specified. If both NAME= and ID= operands are omitted, all applications registered by the requesting NCL process are deregistered.

Limit: 1 through 32 characters in length

ID=*registrationid*

Specifies the non-zero registration identifier returned in the &ZSNAMID system variable for the registered application. This operand is not required but, if supplied, the NAME= operand cannot be specified. If both NAME= and ID= operands are omitted, all applications registered by the requesting NCL process are deregistered.

Return Codes:

&RETCODE	&ZFDBK	Meaning
0	0	Applications deregistered successfully
4	0	Application not registered by this process

For &RETCODE 8, the &SYSMSG variable contains an error message.

Notes:

Deregistration can trigger the following EDS system configuration notification events:

- $$SNAMS.APPL.DEREGISTER

  This is always generated for application deregistration. The resources for this event are, respectively, the application routing name and the local node name.

- $$SNAMS.FP.INACTIVE.LOCAL

  This is generated if it was registered as a focal point application. The resources are respectively, the MS category and the focal point address.

- $$SNAMS.EP.INACTIVE

  This is generated for each entry point within the sphere of control of the local node for the MS category registered by the application. The resources are respectively the MS category and the entry point node name.

Processes that are profiled for the receipt of the events have the message N00102 queued to their response queues and accessible using the &INTREAD statement. The resources for this message are formatted as follows:

Application routing name, or MS category

Is a hexadecimal quoted string if it is an SNA-architected value, that is, a 4-byte value containing non-displayable characters (for example, '23F0F3F1'x). Otherwise, it is formatted as a text string (for example, USERAPPL).

Node name

Contains the network identifier and the NAU name, separated by a period (for example, NETID.NAUNAME).

Focal point address

Contains the network identifier, NAU name, and focal point application routing name, each separated by a period (for example, NETID.NAUNAME.'23F0F3F1'x).

&SNAMS RECEIVE

The &SNAMS RECEIVE verb receives an MDS-MU that is a reply or error message.

An NCL application uses this option to receive an MDS request (if explicitly registered), an MDS reply (if a requester) following a previous SEND operation, or an MDS error message.

This verb has the following format:

```
&SNAMS RECEIVE MU=mdo
               [ ID=requestID ]
               [ WAIT=nnn ]
```

Operands:

MU=*mdo*

Specifies the target MDO for the MDS-MU received. The value is a stem name or a compound MDO name.

ID=*requestid*

(Optional) Specifies the message identifier.

When omitted, SNAMS returns the first MDS-MU received on behalf of this NCL process.

If supplied, it is the request identifier for an outstanding &SNAMS SEND request, or the registration identifier of an application registration. Both of these identifiers are returned in the &ZSNAMID system variable after their respective &SNAMS requests.

WAIT=*nnn*

Specifies the time, in seconds (for example, 15) or seconds and hundredths (for example, 1.25), for which the procedure is prepared to wait for the arrival of an MDS-MU. If one is not available before this interval expires, the receive request is canceled and an unsuccessful return code results (that is, &RETCODE is set to 12).

Return Codes:

&RETCODE	&ZFDBK	Meaning
0	0	MDS-MU received successfully
4	0	Request cannot be satisfied - no outstanding request
8	0	Data is invalid for the specified MDO (mapped)
12	0	Request timed out - no data

For &RETCODE 0, the &ZSNAMID variable contains the message identifier for the MDS-MU received. This value is the request identifier for an outstanding &SNAMS SEND request; otherwise, the registration identifier of the receiving application.

For &RETCODE 8, the &SYSMSG variable contains an error message.

Notes:

This verb affects outstanding &SNAMS RECEIVE_NOTIFY requests as follows:

- A request for a specific message cancels outstanding RECEIVE_NOTIFY requests for nonspecific messages, and requests for that particular message.
- A request for a nonspecific message cancels all outstanding RECEIVE_NOTIFY requests.

&SNAMS RECEIVE_NOTIFY

The &SNAMS RECEIVE_NOTIFY verb notifies the procedure when data is available to be received.

This option is an asynchronous request which enables the procedure to continue execution and is notified of data arrival by a notification event being queued to the internal environment of the procedure. This request returns no data. When the notify event is received the &SNAMS RECEIVE request is used to access any data that arrived.

This verb has the following format:

```
&SNAMS RECEIVE_NOTIFY [ ID=requestid ]
```

Operands:

ID=*requestid*

(Optional) Specifies the message identifier.

When omitted, SNAMS issues a notification for the first MDS-MU received on behalf of this NCL process.

If supplied, it is the request identifier for an outstanding &SNAMS SEND request or the registration identifier of an application registration. Both of these identifiers are returned in the &ZSNAMID system variable after their respective &SNAMS requests.

Return Codes:

&RETCODE	&ZFDBK	Meaning
0	0	Request successful
4	0	Request cannot be satisfied - no outstanding request

Notes:

This verb affects outstanding &SNAMS RECEIVE_NOTIFY requests in one of the following ways:

- A request for a specific message cancels outstanding RECEIVE_NOTIFY requests for nonspecific messages
- A request for a nonspecific message cancels all outstanding RECEIVE_NOTIFY requests for specific messages

Thus at any instance, either multiple RECEIVE_NOTIFY requests for specific messages, or a single nonspecific RECEIVE_NOTIFY request is outstanding. When the request is satisfied the following message is placed on the request queue of the internal environment for the NCL procedure:

```
N00101 NOTIFY: SNAMS EVENT: RECEIVE RESOURCE: &ZSNAMID
```

where the value &ZSNAMID is the message identifier, that is, the request identifier for an outstanding &SNAMS SEND request, or the registration identifier of the destination application.

&SNAMS REGISTER

The &SNAMS REGISTER verb registers an application with SNAMS.

An NCL application uses this option to register itself with SNAMS. Once registered, it is deemed to be a valid destination for MDS-MUs.

The application can optionally register a category for which it intends to be the local focal point. This registration indicates to MS_CAPS that the local node can negotiate for the role of a focal point for that category in an MS Capabilities exchange.

Alternatively, the application can register interest in a category to receive focal point updates from MS_CAPS. This option however, is unnecessary for applications that use such notification messages solely to obtain addressing details, to send MDS-MUs to the active focal point. The &SNAMS SEND verb provides an option for sending MDS-MUs to the focal point of a specified category, eliminating the need for applications to track focal point activation/deactivation.

This verb has the following format:

```
&SNAMS  REGISTER NAME=applicationname
              [ APPLID=routingname ]
              [ CATEGORY=fpcategory [ TYPE={ FP | FPN } ] ]
```

Operands:

NAME=*applicationname*

(Mandatory) The name of the application being registered and must be 1 through 32 characters long. If the APPLID= operand is not supplied, the first eight characters of this name are used as the routing name. Otherwise it is purely a descriptive name known only to the local system.

APPLID=*rountingname*

This operand, if supplied, is the name recognized by SNAMS as a potential target application for incoming MDS-MUs. If omitted, the first 8 bytes of the NAME= operand is used as the routing name. This operand is designed for SNA-architected applications with nondescriptive names (for example, '23F0F3F1'X). The value must be four bytes specified as a hexadecimal quoted string.

CATEGORY=*fpcategory* [TYPE={ FP | FPN }]

The CATEGORY operand indicates the focal point category for the TYPE= operand. The operand is not required, but if supplied, it must be one through eight characters long and specified as a hexadecimal quoted string.

TYPE={ FP | FPN }

Indicates the role of the registered application with respect to the category specified for the CATEGORY= operand. This operand itself is optional. If supplied, the CATEGORY= operand must be specified.

TYPE=FP registers the application as the local focal point for the specified category. This registration authorizes MS-CAPS to accept negotiations with remote nodes aimed at establishing the local node as the focal point for that category, and with this application acting as the focal point application.

TYPE=FPN indicates to MS_CAPS that this application wishes to be notified of all focal point updates for the specified category.

Default: FP

Return Codes:

&RETCODE	&ZFDBK	Meaning
0	0	Registration successful
4	0	Duplicate registration for this process
8	0	Application registered by another process
	1	Supplied details conflict with an existing registration

For &RETCODE 0 and 4, the &ZSNAMID variable contains a non-zero application registration identifier.

For &RETCODE 8, the &SYSMSG variable contains a system error message.

Notes:

New registrations trigger the following EDS system configuration notification events:

- $$SNAMS.APPL.REGISTER

  This is generated when the application is registered for the first time. The resources for this event are, respectively, the application routing name and the local node name.

- $$SNAMS.FP.ACTIVE.LOCAL

  This is generated if the registration is for a focal point application. The resources for this event are respectively, the MS category and the focal point address.

Processes that are profiled for the receipt of the events have the message N00102 queued to their response queues and accessible using the &INTREAD statement. The resources for this message are formatted as follows:

Application routing name, or MS category

Is a hexadecimal quoted string if it is an SNA-architected value, that is, a 4-byte value containing non-displayable characters (for example, '23F0F3F1'x). Otherwise, it is formatted as a text string (for example, USERAPPL).

Node name

Contains the network identifier and the NAU name, separated by a period (for example, NETID.NAUNAME).

Focal point address

Contains the network identifier, NAU name, and focal point application routing name, each separated by a period (for example, NETID.NAUNAME.'23F0F3F1'x).

&SNAMS SEND

The &SNAMS SEND verb sends a formatted MDS-MU request, reply, or error to a nominated target application or focal point. In addition to the application data GDS variable, the following MDS-MU Routing Information GDS variable (X'1311') contents must be present:

- Origin application name (if it is not registered)
- Destination application name (if not sending to a focal point)
- MDS-MU type (Request or Reply or Error)
- MDS-MU flags (First Message/Last Message)
- Origin and destination network identifier and NAU name, if omitted, default to values of the local node
- Origin application name, if omitted, defaults to that of the last registered application

If the MDS-MU is destined for a focal point, destination values in the MDS-MU header are set to those destinations of the active focal point for the supplied category.

If the MDS_MU is flagged as the FIRST message, the Agent Unit of Work Correlator GDS variable (X'1549') is not required (SNAMS generates one for it). Otherwise, Agent Unit of Work Correlator is mandatory (that is, for MDS replies).

This verb has the following format:

```
&SNAMS SEND MU=mdo [ CATEGORY=fpcategory ]
```

Operands:

MU=*mdo*

Specifies the name of the MDO that contains a formatted MDS-MU to send. The value is a stem name or a compound MDO name.

CATEGORY=*fpcategory*

(Optional) If specified, must be one through eight characters long-it is a hexadecimal quoted string. The MDS-MU is sent to the active focal point application for this category, with respect to the original application.

The active focal point is determined as follows:

- A local focal point exists on the same system:

  If the origin application is the local focal point application itself, ASM concludes that the MDS-MU is sourced from a nested focal point and targeted at the nesting, or remote focal point. The assigned focal point is selected as its destination. Otherwise, ASM sends the MDS-MU to the local focal point application.

- No local focal point exists:

  The MDS-MU is sent to the assigned focal point, if one is active.

Return Codes:

&RETCODE	&ZFDBK	Meaning
0	0	Send request accepted
4	0	Focal point is unavailable
8	0	MDS-MU header format exception
	1	MDS routing exception
	2	Request rejected by internal application
	3	Invalid origin node

For &RETCODE 0, if it is an MDS-MU request that expects one or more MDS-MU replies (that is, it is not flagged as the LAST message), the &ZSNAMID variable contains a non-zero request identifier, which is used on subsequent &SNAMS RECEIVE verbs for correlating replies.

For &RETCODE 8, the &SYSMSG variable contains an error message. A sense code is also returned for &ZFDBK 0 and 1.

Notes:

Although SNAMS can accept the SEND request, it does not guarantee delivery of the MDS-MU to a remote destination. If delivery failed for one reason or another, an MDS error message is returned.

MDS replies, if any, are delivered to the initiator of the request and not the registered receiver of the destination application. Thus, the origin application of MDS requests need not be registered with SNAMS. MDS errors however, are delivered to the registered receiver if it does not correlate to an outstanding request. Thus, if the origin application is not registered, errors are lost.

&SOCKET ACCEPT

Services incoming connection requests.

```
&SOCKET ACCEPT ID=socket_id [TYPE={SYNC | ASYNC}]
```

ACCEPT is used by a server application to accept incoming connection requests from a client and create a new socket.

Operands:

ID=*socklet_id*

Specifies the identity of the socket created by &SOCKET REGISTER.

TYPE={SYNC | ASYNC}

Indicates whether this is a synchronous (SYNC) or asynchronous (ASYNC) socket request. If TYPE=ASYNC is specified, the verb returns control immediately, and a notification message is queued to the dependent environment when ACCEPT completes:

```
N00101 NOTIFY: TCP/IP EVENT: ACCEPT RESOURCE: RC=rc RSN=rsn ERR=errno
VERRIN=vendor_info ID=new_socket_id
```

Note: ID= is omitted if the ACCEPT fails.

Examples: &SOCKET ACCEPT

```
&SOCKET ACCEPT ID=&SOCKETID TYPE=SYNC
```

Return Codes:

0

Accept successful

8

Accept failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

When you are writing a server application, use &SOCKET REGISTER to start listening for incoming connection requests.

You would normally start another NCL process to handle the new connection and use &SOCKET TRANSFER_REQUEST to pass the connection to that process.

The &ZSOCID system variable contains the socket number to be used for communications with the client.

The port number and IP address of the remote system are available by referencing the &ZSOCPRT and &ZSOCHADR system variables.

Use TYPE=ASYNC so that your NCL process does not wait for connection requests in &SOCKET ACCEPT.

The &SOCKET ACCEPT verb is executed only by the server in client/server applications.

More information:

&SOCKET REGISTER (see page 638)
&SOCKET TRANSFER_REQUEST (see page 647)
&SOCKET TRANSFER_ACCEPT (see page 646)

&SOCKET CLOSE

Closes the socket specified by the socket ID.

```
&SOCKET CLOSE ID=socket_id
```

CLOSE indicates that the NCL process has finished using the socket.

Operands:

ID=*socklet_id*

(Mandatory) Specifies the identifier of the socket to be closed.

Examples:

```
&SOCKET CLOSE ID=&NSOCKID
```

Return Codes:

0

Close successful

8

Close failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

Use &SOCKET CLOSE on sockets created by &SOCKET OPEN, CONNECT, and ACCEPT.

More information:

&SOCKET OPEN (see page 630)
&SOCKET CONNECT (see page 624)
&SOCKET ACCEPT (see page 621)

&SOCKET CONNECT

Connects a process to a server.

```
&SOCKET CONNECT
      { ADDRESS=ip_address | HOSTNAME=host_name }
        PORT=port_id
      [ WAIT={ time | 30 } ]
      [ TYPE={ SYNC | ASYNC }]
```

CONNECT attempts to establish a connection to the server, at the host specified by IP address or host name on the specified port number.

Operands:

ADDRESS=*ip_address*

Specifies the IP address of the host.

HOSTNAME=*host_name*

Specifies the name of the host to which you are connecting.

PORT=*port_id*

(Mandatory) Specifies the TCP port number that the server is waiting on for client connections. TCP port numbers range from 0 to 65535.

WAIT={ *time* | 30 }

Specifies the time (in seconds)to wait for the host to respond. The default value is 30 seconds.

TYPE={ SYNC | ASYNC }

Indicates whether this is a synchronous (SYNC) or asynchronous (ASYNC) socket request.

If TYPE=ASYNC is specified, WAIT cannot be specified.

If TYPE=ASYNC is specified, the verb returns control immediately, and a notification message is queued to the dependent environment when CONNECT completes:

```
N00101 NOTIFY: TCP/IP EVENT: CONNECT RESOURCE: RC=rc RSN=rsn ERR=errno
VERRIN=vendor_info ID=new_socket_id **addr/name**
```

where **addr/name** is set to the destination specified on the CONNECT request (that is, either ADDRESS=*ip_address* or HOSTNAME=*host_name*)

Note: ID= is omitted if the CONNECT fails.

Examples: &SOCKET CONNECT

```
&SOCKET CONNECT ADDRESS=172.24.91.45 PORT=&PRT
```

```
&SOCKET CONNECT HOSTNAME=TESTMVS1 PORT=&PRT
```

Return Codes:

0

Connect successful

8

Connect failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

The verb is executed only by the client in client/server applications using TCP sockets.

The &ZSOCID system variable contains the socket number to be used for communications with the server.

&SOCKET GETHOSTBYADDR

Obtains name information for a specified host.

```
&SOCKETGETHOSTBYADDR ADDRESS=ip_address
                     [ MDO=mdo_name ]
                     [ WAIT=time ]
                     [ TYPE={SYNC | ASYNC} ]
```

GETHOSTBYADDR is used to obtain the *real* name of the host at an IP address specified by the ADDRESS operand.

Operands:

ADDRESS=*ip_address*

(Mandatory) Specifies the IP address of the host about which information is required. No alias name information is returned.

MDO=*mdo_name*

Specifies the name of the Mapped Data Object (MDO) into which information about the host is to be formatted. The MDO is mapped by the $NMTCPHE map.

WAIT=*time*

Specifies the time (in seconds)to wait for the host to respond. The default value is 30 seconds.

TYPE={SYNC | ASYNC}

Indicates whether this is a synchronous (SYNC) or asynchronous (ASYNC) socket request.

If TYPE=ASYNC is specified, WAIT cannot be specified.

If TYPE=ASYNC is specified, the verb returns control immediately, and a notification message is queued to the dependent environment when GETHOSTBYADDR completes:

```
N00101 NOTIFY: TCP/IP EVENT: GETHOSTBYADDR RESOURCE: RC=rc RSN=rsn ERR=errno
VERRIN=vendor_info ADDRESS=ip_address
```

The return MDO is available in $INT.USERMDO after &INTREAD receives the message.

Examples: &SOCKET GETHOSTBYADDR

```
&SOCKET GETHOSTBYADDR ADDRESS=172.24.91.45
&WRITE NAME=&ZSOCHNM IPADDRESS=&ZSOCHADR
```

Return Codes:

0

Get successful

8

Get failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

The information returned is set into the following NCL variables:

&ZSOCHNM contains the host name.

- &ZSOCFHNM contains the full name of the host.
- &ZSOCHADR contains the IP address of the host.

More information:

&SOCKET GETHOSTBYNAME (see page 627)

&SOCKET GETHOSTBYNAME

Obtains name and address details for a specified host.

```
&SOCKET GETHOSTBYNAME HOSTNAME=host_name
                      [ MDO=mdo_name ]
                      [ WAIT=time ]
                      [ TYPE={SYNC | ASYNC} ]
```

GETHOSTBYNAME is used to obtain the IP address and, if an alias name is used in the HOSTNAME parameter, the full real name of a specific host.

Operands:

HOSTNAME=*host_name*

(Mandatory) Specifies the name of the host about which information is required. If an alias name is used both the full real name and full alias name of the host are returned. If a real name is specified no alias name information is returned.

MDO=*mdo_name*

Specifies the name of the Mapped Data Object (MDO) into which information about the host is to be formatted. The MDO is mapped by the $NMTCPHE map.

WAIT=*time*

Specifies the time (in seconds) to wait for the host to respond. The default value is 30 seconds.

TYPE={SYNC | ASYNC}

Indicates whether this is a synchronous (SYNC) or asynchronous (ASYNC) socket request.

If TYPE=ASYNC is specified, WAIT cannot be specified.

If TYPE=ASYNC is specified, the verb returns control immediately, and a notification message is queued to the dependent environment when GETHOSTBYNAME completes:

```
N00101 NOTIFY: TCP/IP EVENT: GETHOSTBYNAME RESOURCE: RC=rc RSN=rsn ERR=errno
VERRIN=vendor_info HOSTNAME=host_name
```

The return MDO is available in $INT.USERMDO after &INTREAD receives the message.

Examples: &SOCKET GETHOSTBYNAME

```
&SOCKET GETHOSTBYNAME HOSTNAME=TESTMVS1
&WRITE NAME=&ZSOCHNM IPADDRESS=&ZSOCHADR
```

Return Codes:

0

Get successful

8

Get failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

The information returned is set into the following NCL variables:

- &ZSOCHNM contains the host name.
- &ZSOCFHNM contains the full name of the host.
- &ZSOCHADR contains the IP address of the host.

More information:

&SOCKET GETHOSTBYADDR (see page 625)

&SOCKET OPEN

Creates a UDP socket.

```
&SOCKET OPEN [ PORT=port_id ]
```

OPEN is used to create a socket with a specified UDP port.

Operands:

PORT=*port_id*

Specifies the port number to be associated with the new socket. Port numbers have a range of 0 to 65535 with the default being 0, in which case the system allocates its own port number.

Examples: &SOCKET OPEN

```
&SOCKET OPEN PORT=&PRT
```

Return Codes:

0

Open successful

8

Open failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

This verb is executed only by applications that want to use UDP sockets.

You need to do an &SOCKET OPEN before using &SOCKET SEND_TO or &SOCKET RECEIVE_FROM.

The information returned is set into the following NCL variables:

- &ZSOCID contains the socket number to be used for communications.
- &ZSOCPRT contains the UDP port number associated with the socket.

Some TCP/IP interfaces, for example Fujitsu TISP, have special requirements regarding pre-definitions of ports.

More information:

&SOCKET SEND_TO (see page 642)
&SOCKET RECEIVE_FROM (see page 636)
&SOCKET CLOSE (see page 623)

&SOCKET PING

Tests connectivity with a specified host.

```
&SOCKET PING
      { ADDRESS=ip_address | HOSTNAME=host_name }
      [ PACKETSIZE=nn ]
      [ COUNT=nn ]
      [ MDO=mdo_name ]
      [ GETNAME={ YES | NO } ]
      [ WAIT=time ]
      [ TYPE={ SYNC | ASYNC }]
```

PING is used to send an echo request to a host specified either by the ADDRESS or HOSTNAME parameter, and waits a specified period of time for a response from the host.

Operands:

ADDRESS=*ip_address*

Specifies the IP address of the host.

HOSTNAME=*host_name*

Specifies the name of the host.

PACKETSIZE=*nn*

Specifies the length of the packets to send to the host. This value must be in the range of 16 to 2048 bytes. The default value is 64.

COUNT=*nn*

Specifies the number of times to execute the ping. This value must be in the range of 1 to 999. The default value is 1.

MDO=*mdo_name*

Specifies the name of the Mapped Data Object (MDO) to receive the statistical information collected by this verb. The MDO is mapped by the $NMTCPPG map.

GETNAME={ YES | NO }

If the default value YES is used the name of the destination host is resolved.

WAIT=*time*

Specifies the time (in seconds) to wait for the host to respond. The default value is 30 seconds.

TYPE={SYNC | ASYNC}

Indicates whether this is a synchronous (SYNC) or asynchronous (ASYNC) socket request.

If TYPE=ASYNC is specified, WAIT cannot be specified.

If TYPE=ASYNC is specified, the verb returns control immediately, and a notification message is queued to the dependent environment when each PING completes:

```
N00101 NOTIFY: TCP/IP EVENT: PING RESOURCE: RC=rc RSN=rsn ERR=errno
VERRIN=vendor_info RESULT=**result_type**  **addr/name**
```

where:

****result_type**** is PARTIAL (intermediate result) or FINAL (final result)

****addr/name**** is set to the destination specified on the PING request (that is, either ADDRESS=*ip_address* or HOSTNAME=*host_name*)

The return MDO is available in $INT.USERMDO after &INTREAD receives the message.

Examples: &SOCKET PING

```
&SOCKET PING HOSTNAME=TESTMVS1 MDO=MDO1 COUNT=&COUNT WAIT=30
```

Return Codes:

0

Ping successful

4

Ping timed out

8

Ping failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

The information returned is set into the following NCL variables:

- &ZSOCHNM contains the host name.
- &ZSOCFHNM contains the full name of the host.
- &ZSOCHADR contains the IP address of the host.

Some interfaces may not support &SOCKET PING. In this case, &ZFDBK is set to 36 and &ZSOCERRN to 7 (EUNSUPP). The &ZTCPSUPP function is used to test if &SOCKET PING is supported.

&SOCKET RECEIVE

Receives data from a TCP socket specified by the socket ID.

```
&SOCKET RECEIVE

        ID=socket_id
      { MDO=mdo_name |
        VARS={ name | (name,name,... name) [SEGMENT] } |
        { ARGS | VARS=prefix* } [ RANGE=(start,end) ] [SEGMENT] }
      [ LENGTH=0..4 ]
      [ WAIT=time ]
      [ TYPE={ SYNC | ASYNC } ]
```

RECEIVE is used to receive data on a nominated TCP socket. The socket must have been connected to a foreign host by using &SOCKET CONNECT, or &SOCKET ACCEPT.

Operands:

ID=*socklet_id*

(Mandatory) Specifies the identifier of the socket to be used for communication.

MDO=*mdo_name*

Specifies the name of the Mapped Data Object (MDO) into which data will be received.

VARS={ *name* | (*name*,*name*,... *name*) }

Specifies the names of the variables to be the target of the RECEIVE operation. If insufficient variables are provided, some data will not be available to the procedure. Excess variables are set to a null value.

SEGMENT

Specifies that there is no delimiter character but that the parsed string will be placed into the receiving variables in segments that correspond to the length of the individual variables. The length defaults to the maximum variable length unless overridden by length specifications in a variable list.

ARGS | VARS=*prefix**

Specifies that the receive operation modifies or creates variables in a numeric range (&1 through &*n* for ARGS, *prefix1* through *prefixn* for VARS) depending on how many are needed to satisfy the operation.

RANGE=(*start*,*end*)

The RANGE operand is coded to designate a start number and an end number to delimit the number of variables generated. The start and end values must be in the range 1 to 32767 and the end value must be equal to or greater than the start value.

LENGTH=0..4

Specifies the length of the received data prefix that contains the data length. The LENGTH operand is useful when incoming messages are prefixed by the length of the rest of the message. The LENGTH value tells the &SOCKET verb the length of the prefix, not the length of the data.

The default value is 4. If a value other than 0 is used, the receive will complete only when all of the data specified by the data prefix length is received by the verb (see Notes).

WAIT=*time*

Specifies the period of time (in seconds) to wait for the receive to be completed. The default value is 0, meaning to wait until there is a successful receive.

TYPE={SYNC | ASYNC}

Indicates whether this is a synchronous (SYNC) or asynchronous (ASYNC) socket request.

If TYPE=ASYNC is specified, WAIT cannot be specified.

If TYPE=ASYNC is specified, the verb returns control immediately.

If data is immediately available, return code 0 is set, and VARS or MDO contain the data.

If no data is immediately available, return code 12 is set, and a notification message is queued to the dependent environment when data is available:

```
N00101 NOTIFY: TCP/IP EVENT: RECEIVE RESOURCE: RC=rc RSN=rsn ERR=errno
VERRIN=vendor_info ID=socket_id
```

After receiving this message, the procedure performs another RECEIVE with the same parameters to obtain the data.

Examples: &SOCKET RECEIVE

```
&SOCKET RECEIVE LENGTH=2 ID=&NSOCKID VARS=R* WAIT=300
```

Return Codes:

0

Receive successful

4

Receive timed out

8

Receive failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

12

Wait for notification of data availability

When null data is received, the connection has been terminated.

Notes:

TCP protocol makes packets of the data in discriminately-it does not retain the boundaries between sends. This means that an application may do five sends and the application on the remote end may do 10 receives to get all of the data. There is no correlation between the number and size of sends at one end and the number and size of receives at the other end.

If you are communicating with another NCL process using &SOCKET, the LENGTH value should match on both the SEND and RECEIVE verbs at both ends of the connection.

The LENGTH value is in network byte order (that is, the most significant byte first). If you are communicating with a machine that uses a different byte order to the network (for example, Intel X86), the machine you are communicating with needs to ensure that it sends and receives lengths in the correct format.

If the LENGTH value specified in the RECEIVE parameters does not match the incoming data, unexpected results might occur.

The &SOCKET RECEIVE verb is executed only by the client/server applications that are using TCP sockets.

More information:

&SOCKET SEND (see page 639)
&DECODE (see page 294)

&SOCKET RECEIVE_FROM

Receives data from a UDP socket specified by the socket ID.

```
&SOCKET RECEIVE_FROM
        ID=socket_id
      { MDO=mdo_name |
        VARS={ name | (name,name,... name) [SEGMENT] } |
        { ARGS | VARS=prefix* } [ RANGE=(start,end) ] [SEGMENT] }
      [ WAIT=time ]
      [ TYPE={ SYNC | ASYNC } ]
```

RECEIVE_FROM is used to receive data from a UDP socket.

Operands:

ID=*socklet_id*

(Mandatory) Specifies the identifier of the socket to be used for communication.

MDO=*mdo_name*

Specifies the name of the Mapped Data Object (MDO) into which information about the host is to be formatted.

VARS={ *name* | (*name*,*name*,... *name*) }

Specifies the names of the variables to be the target of the receive operation. If insufficient variables are provided, some data will not be available to the procedure. Excess variables are set to a null value.

SEGMENT

Specifies that there is no delimiter character but that the parsed string will be placed into the receiving variables in segments that correspond to the length of the individual variables. The length defaults to the maximum variable length unless overridden by length specifications in a variable list.

ARGS | VARS=*prefix**

Specifies that the receive operation modifies or creates variables in a numeric range (&1 through &*n* for ARGS, *prefix1* through *prefixn* for VARS) depending on how many are needed to satisfy the operation.

RANGE=(*start*,*end*)

The RANGE operand is coded to designate a start number and an end number to delimit the number of variables generated. The start and end values must be in the range 1 to 32767, and the end value must be equal to or greater than the start value.

WAIT=*time*

Specifies the period of time (in seconds) to wait for the receive to be completed. The default value is 0, meaning to wait until there is a successful receive.

TYPE={SYNC | ASYNC}

Indicates whether this is a synchronous (SYNC) or asynchronous (ASYNC) socket request. If TYPE=ASYNC is specified, WAIT cannot be specified. If TYPE=ASYNC is specified, the verb returns control immediately. If data is immediately available, return code 0 is set, and vars/MDO contain the data. If no data is immediately available, return code 12 is set, and a notification message is queued to the dependent environment when data is available:

```
N00101 NOTIFY: TCP/IP EVENT: RECEIVE_FROM RESOURCE: RC=rc RSN=rsn ERR=errno
VERRIN=vendor_info ID=socket_id
```

After receiving this message, the procedure performs another RECEIVE_FROM with the same parameters to obtain the data.

Examples: &SOCKET RECEIVE_FROM

```
&SOCKET RECEIVE_FROM ID=&NSOCKID VARS=R* WAIT=300
```

Return Codes:

0

Receive_from successful

4

Receive_from timed out

8

Receive_from failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

12

Wait for notification of data availability

Notes:

The &SOCKET RECEIVE_FROM verb is executed only by applications that are using UDP sockets.

The information returned is set into the following NCL variables:

- &ZSOCHNM contains the name of the data's source host.
- &ZSOCFHNM contains the full name of the data's source host.
- &ZSOCHADR contains the IP address of the data's source host.
- &ZSOCPRT contains the port number associated with the source socket on the remote host.

More information:

&SOCKET OPEN (see page 630)
&SOCKET SEND_TO (see page 642)
&DECODE (see page 294)

&SOCKET REGISTER

Registers a socket.

```
&SOCKET REGISTER [ PORT=port_id ] [ CONVLIM=nn ]
```

REGISTER is used by a server application to obtain a socket, bind it to a specified TCP port, and issue a listen on the socket, specifying the queue depth given by the CONVLIM parameter.

Operands:

PORT=*port_id*

Specifies the TCP port number to be used by the socket. Port numbers range from 1 to 65535, with the default being 0, in which case the system allocates its own port number. (See Notes below.)

CONVLIM=*nn*

Specifies the limit for waiting conversations, this being the maximum number of clients that is waiting for their connection request to be ACCEPTED at any one time. The default value is 100.

Examples: &SOCKET REGISTER

```
&SOCKET REGISTER CONVLIM=10 PORT=&PRT
```

Return Codes:

0

Register successful

8

Register failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

The &SOCKET REGISTER is executed only by the server in client/server applications.

Usually a server application will have an NCL process handling a socket created by &SOCKET REGISTER and using &SOCKET ACCEPT to accept inbound connection requests. This process also starts independent processes to handle each connection. The &SOCKET TRANSFER_REQUEST and TRANSFER_ACCEPT verbs are used to transfer the connection to the new process.

The information returned is set into the NCL variable, &ZSOCID. This variable will contain the socket number to be used when referring to the registered socket.

Some TCP/IP interfaces, for example Fujitsu TISP, have special requirements regarding pre-definitions of ports.

More information:

&SOCKET ACCEPT (see page 621)

&SOCKET SEND

Sends data to a TCP socket.

```
&SOCKET SEND
        ID=socket_id
      { MDO=mdo_name | VARS=prefix* [ RANGE=(start,end) ] |
        VARS={ name | name,name,... name) } |
        ARGS [ RANGE=(start,end) ] | DATA=data }
      [ LENGTH=0..4 ]
```

SEND is used to send data to a TCP socket, identified by the ID parameter, that has been connected by &SOCKET CONNECT or &SOCKET ACCEPT.

Operands:

ID=*socklet_id*

(Mandatory) Specifies the identifier of the socket that is to be used for communication.

MDO=*mdo_name*

Indicates that the data to be sent is contained in the specified MDO.

VARS=*prefix [RANGE=(*start*,*end*)]**

Supplies leading characters terminated by an asterisk to denote a numeric range of variables, which contain the data to be sent.

VARS={ *name* | (*name*,*name*,... *name*) }

Specifies the names of the variables containing the data to be sent.

ARGS [RANGE=(*start*,*end*)]

The RANGE operand is coded to designate a start number and an end number to delimit the number of variables sent. The start and end values must be in the range 0 to 32767 and the end value must be equal to or greater than the start value.

DATA=*data*

Specifies the data to be sent.

LENGTH=0..4

Specifies the length of the data prefix that contains the data length. The default value is 4 which means that there is a 4 byte prefix added to the data sent containing the length of the data itself. Specifying LENGTH=0 means that no prefix is added to the message. Setting the data prefix length allows a RECEIVE operation to get all of (and only) the data sent by the current execution of the SEND verb (see Notes).

Examples: &SOCKET SEND

```
&SOCKET SEND LENGTH=2 ID=&NSOCKID VARS=V1
```

Return Codes:

0

Send successful

8

Send failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

TCP protocol makes packets of the data indiscriminately-it does not retain the boundaries between sends. This means that an application may do 5 sends and the application on the remote end may do 10 receives to get all of the data. There is no correlation between the number and size of sends at one end with the number and size of receives at the other end.

If you are communicating with another NCL process using &SOCKET, the LENGTH value should match on both the SEND and RECEIVE verbs at both ends of the connection.

The LENGTH value is in network byte order (that is, the most significant byte first). If you are communicating with a machine that uses a different byte order to the network (for example, Intel X86), the machine you are communicating with must send and receive lengths in the correct format.

If the LENGTH value specified in the SEND parameters does not match the incoming data, unexpected results might occur.

The &SOCKET SEND verb is executed only by the client/server applications using TCP sockets.

More information:

&SOCKET RECEIVE (see page 633)
&ENCODE (see page 309)

&SOCKET SEND_TO

Sends data to a UDP socket.

```
&SOCKET SEND_TO ID=socket_id
              { MDO=mdo_name | VARS=prefix* [ RANGE=(start,end) ] |
                VARS={ name | name,name,... name) } |
                ARGS [ RANGE=(start,end) ] | DATA=data }
              { ADDRESS=ip_address | HOSTNAME=host_name }
                PORT=port_id
```

SEND_TO is used to send the data on a UDP socket, identified by the ID parameter, to the specified host and port number.

Operands:

ID=*socket_id*

(Mandatory) Specifies the identifier of the socket that is to be used for communication.

MDO=*mdo_name*

Indicates that the data to be sent is formatted in the specified MDO.

VARS=*prefix [RANGE=(*start*,*end*)]**

Supplies leading characters terminated by an asterisk to denote a numeric range of variables, which contain the data to be sent.

VARS={ *name* | (*name*,*name*,... *name*) }

Specifies the names of the variables containing the data to be sent.

ARGS [RANGE=(*start*,*end*)]

The RANGE operand is coded to designate a start number and an end number to delimit the number of variables sent. The start and end values must be in the range 0 to 32767 and the end value must be equal to or greater than the start value.

DATA=*data*

Specifies the data to be sent.

ADDRESS=*ip_address*

Specifies the IP address of the destination host.

HOSTNAME=*host_name*

Specifies the name of the destination host.

PORT=*port_id*

Specifies the UDP port number of the destination. Port numbers range from 0 to 65535.

Examples: & SOCKET SEND_TO

```
&SOCKET SEND_TO ID=&NSOCKID MDO=MDO1 PORT=&PRT +
                ADDRESS=172.24.91.45
```

Return Codes:

0

Send_to successful.

8

Send_to failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

UDP protocol does not guarantee delivery of the data, however when an application sends data out, it arrives at the remote end as a single unit. For example, if an application does five sends, the application at the remote end will do five receives. Also, the size of each send matches the size of each receive.

The &SOCKET SEND_TO verb is executed only by applications using UDP sockets.

The maximum size datagram that is sent is determined by the vendor interface. Attempting to send a larger datagram results in ZFDBK being set to 3, for example, &ZSOCERRN to 29 (EMSGSIZE).

More information:

&SOCKET OPEN (see page 630)
&SOCKET RECEIVE_FROM (see page 636)
&ENCODE (see page 309)

&SOCKET TRACEROUTE

Obtains a list of routers along the route to the host.

```
&SOCKET TRACEROUTE
      { ADDRESS=ip_address | HOSTNAME=host_name }
      [ PACKETSIZE={ nn | 64 } ]
      [ COUNT={ nn | 3 } ]
      [ HOPS={ nn | 10 } ]
      [ FROMHOP={ nn | 1 } ]
      [ WAIT={ time | 3 } ]
      [ TYPE={ SYNC | ASYNC } ]
      [ GETNAME={ YES | NO } ]
        MDO=mdo_name
```

TRACEROUTE is used in the diagnosis of connectivity and performance related problems on a TCP/IP network. It is used to find breaks in the route from a host to a remote host, or to verify that a network path to a remote host exists, displaying times for hops along the path.

Operands:

ADDRESS=*ip_address*

Specifies the IP address of the destination host.

HOSTNAME=*host_name*

Specifies the name of the destination host.

PACKETSIZE={ *nn* | 64 }

Specifies the length of the packets to send to the host. This value must be in the range of 40 to 2048 bytes. The default value is 64.

COUNT={ *nn* | 3 }

Specifies the number of times to execute the trace. This value must be in the range of 1 to 10. The default value is 3.

HOPS={ *nn* | 10 }

Specifies the maximum number of devices to locate on the route to the host. This value must be in the range of 1 to 256. The default value is 10.

FROMHOP={ *nn* | 1 }

Specifies the number of hops to the first device for which data is to be returned. This value must be in the range of 1 to 256. The default value is 1.

WAIT={ *time* | 3 }

Specifies the time in seconds to wait for a response from any one host on the route to the destination host. The default value is 3 seconds.

TYPE={ SYNC | ASYNC }

Indicates whether this is a synchronous (SYNC) or asynchronous (ASYNC) socket request.

If TYPE=ASYNC is specified, WAIT cannot be specified.

If TYPE=ASYNC is specified, the verb returns control immediately, and a notification message is queued to the dependent environment when each hop of the TRACEROUTE completes:

```
N00101 NOTIFY: TCP/IP EVENT: TRACEROUTE RESOURCE: RC=rc RSN=rsn ERR=errno
VERRIN=vendor_info RESULT=**result_type** **addr/name**
```

where:

- **_result_type_** is PARTIAL (intermediate result) or FINAL (final result).
- **_addr/name_** is set to the destination specified on the TRACEROUTE request (that is, either ADDRESS=_ip_address_ or HOSTNAME=_host_name_).

The return MDO is available in $INT.USERMDO after &INTREAD receives the message.

GETNAME={ YES | NO }

If the default value YES is used, the names of the hosts on the route to the destination are resolved.

MDO=*mdo_name*

Specifies the name of the MDO that contains the response from the traceroute. This MDO is mapped by the $NMTCPTC map.

Examples: & SOCKET TRACEROUTE

```
& SOCKET TRACEROUTE HOSTNAME=TESTMVS1 MDO=MDO1 HOPS=15 WAIT=5
```

Return Codes:

0

Traceroute successful.

4

HOPS parameter is too low.

8

Traceroute failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notes:

The information returned is set into the following NCL variables:

- &ZSOCHNM contains the host name.
- &ZSOCFHNM contains the full name of the host.
- &ZSOCHADR contains the IP address of the host.

Some interfaces may not support TRACEROUTE. In this case, &ZFDBK is set to 37 and &ZSOCERRN to 7 (EUNSUPP). The &ZTCPSUPP function is used to test whether TRACEROUTE is supported.

&SOCKET TRANSFER_ACCEPT

Accepts a socket ID from a donor NCL process.

```
&SOCKET TRANSFER_ACCEPT ID=socket_id
```

TRANSFER_ACCEPT is used to accept a socket from a donor NCL process. The TRANSFER_ACCEPT should be performed upon receipt of a notification message (see below) from the donor NCL process, indicating that it has been targeted to receive a socket.

Operands:

ID=*socket_id*

(Mandatory) Specifies the identifier of the socket to be used for communication.

Examples: &SOCKET TRANSFER_ACCEPT

```
&INTREAD ARGS TYPE=ANY
&IF &1 EQ N00101 &THEN +
  &DO
     &NSOC = &REMSTR (=) &7
     &SOCKET TRANSFER_ACCEPT ID=&NSOC
  &DOEND
```

Return Codes:

0

Transfer successful.

8

Transfer failed; see &ZFDBK for reason code (see page 1220).

Notes:

An &SOCKET TRANSFER_REQUEST issued by another process causes a notification message to be queued to the request queue of the target process.

The information returned is set into the NCL variables, &ZSOCID. This variable will contain the socket number to be used for communications.

Notification Message Format:

&SOCKET TRANSFER_ACCEPT should be issued upon receipt of the following message, queued to the internal environment of this process, indicating that it has been targeted for a transfer request:

```
N00101 NOTIFY: TCP/IP EVENT: TRANSFER RESOURCE: ID=socket_id +
    NCLID=ncl_id
```

More information:

&SOCKET TRANSFER_REQUEST (see page 647)

&SOCKET TRANSFER_REQUEST

Transfers the socket ID from the current NCL process to another NCL process identified by NCL ID.

```
&SOCKET TRANSFER_REQUEST ID=socket_id NCLID=ncl_id
```

To transfer ownership of a socket from one process to another.

Operands:

ID=*socklet_id*

(Mandatory) Specifies the identifier of the socket to be transferred.

NCLID=*ncl_id*

(Mandatory) Nominates the NCL process to which ownership of the socket is being transferred. The notification message (see below) will be queued to the internal environment of the process to indicate that it has been targeted for a transfer request.

Examples: &SOCKET TRANSFER_REQUEST

```
&SOCKET TRANSFER_REQUEST ID=&NSOC NCLID=&NID
```

Return Codes:

0

Transfer successful

8

Transfer failed; see &ZFDBK for reason code, &ZSOCERRN and &ZSOCVERR for further error information (see page 1219).

Notification Message Format:

&SOCKET TRANSFER_REQUEST issues the following message and queues it to the request queue of an NCL process that it has targeted for a transfer request:

```
N00101 NOTIFY: TCP/IP EVENT: TRANSFER RESOURCE: ID=socket_id NCLID=ncl_id
```

This request is completed after the target NCL process executes the &SOCKET TRANSFER_ACCEPT verb.

&APPC START NOTIFY=YES is a useful means to start a process and find out the NCL ID (this does not require an APPC conversation).

There should be no asynchronous requests outstanding when &SOCKET TRANSFER_REQUEST is issued.

More information:

&SOCKET TRANSFER_ACCEPT (see page 646)

&STR

Returns a string that is the sum of the supplied text.

```
&STR text text ... text
```

To return a string made up of nominated text.

&STR is a built-in function and must be used to the right of an assignment statement.

The assigned data commences at the first non-blank following the &STR keyword and includes any following data up to and including the last non-blank character.

&ASISTR differs from &STR in that &STR does not retain blanks that follow the &STR keyword.

Operands:

text text ... text

One or more words or variables to be assigned into the variable to the left of the assignment character (=).

Examples: &STR

```
&SYSMSG = &STR X11103 INVALID DATA ENTERED
&1 = &STR &1 MYTEXT &2
```

Notes:

&STR is ideal for constructing error messages to be displayed using full-screen panels.

The total size of the constructed variable or constant cannot exceed the maximum size for a variable, that is, 256 characters. (If it exceeds the maximum, it will be truncated to 256 characters.)

If &CONTROL DBCS or DBCSN or DBCSP is in effect, &STR is sensitive to the presence of DBCS data (see page 1209).

More information:

&ASISTR (see page 205)

&SUBSTR

Returns a string that is a section of a nominated variable or constant.

```
&SUBSTR data i [ j ]
```

&SUBSTR provides a means of isolating part of a string or variable. &SUBSTR is a built-in function and must be used to the right of an assignment statement.

Operands:

data

A variable or constant from which the extraction of data is to be performed.

i

The position within the data at which the extraction is to start. The value of *i* must be greater than 0. The first character in a variable is counted as 1. If *i* exceeds the length of the variable, then a null value is extracted.

j

The length of data to be extracted. If *j* is omitted or exceeds the length remaining in the variable, the remaining length is used. *j* must be 0 or greater. If 0 is specified, then a null value is extracted.

Examples: &SUBSTR

```
&A = ABCDEF
&B = &SUBSTR &A 3 2  -* &B is set to CD
&C = &SUBSTR &A 4    -* &C is set to DEF
```

Used in calculating a number as a percentage.

```
&PC = (&CURR * 10000) / &INIT
&A = &LENGTH &PC
&A = &A - 1
&B = &SUBSTR &PC &A
&A = &LENGTH &PC
&A = &A - 2
&PC = &SUBSTR &PC 1 &A
&PC = &CONCAT &PC .  &B %
&WRITE Percentage is &PC
```

Notes:

The process of variable substitution eliminates null variables from a statement. Therefore care should be taken to ensure any expected variables are supplied on the &SUBSTR statement or invalid results can occur.

If &CONTROL DBCS or DBCSN or DBCSP is in effect, &SUBSTR is sensitive to the presence of DBCS data (see page 1210).

&TBLSTR

Returns a string with trailing blanks deleted.

```
&TBLSTR string
```

&TBLSTR is a built-in function and must be used to the right of an assignment statement.

It is possible for user variables to contain trailing blanks. These blanks can have been added by the use of other built-in functions such as &SUBSTR and &SETLENG.

&TBLSTR will remove any trailing blanks from the data and returns this value.

If the data consists entirely of blanks then a null value is returned.

Operands:

string

Data or a variable containing data from which the trailing blanks are to be removed.

Examples: &TBLSTR

```
&1 = &TBLSTR &INPUT
&A = &TBLSTR &A
```

Notes:

To remove both leading and trailing blanks the &NBLSTR function is used. To remove only leading blanks the &LBLSTR function is used.

More information:

&ASISTR (see page 205)
&NBLSTR (see page 442)
&LBLSTR (see page 386)

&TRANS

Performs character translation within a string.

```
&TRANS { C'x1 y1....  xn yn | X'xx1 yy1.... xxn yyn |
         UPPER | LOWER | PRINT |
         NLUPPER [ =lc ] | NLLOWER [ =lc ] }
       string
```

&TRANS is a built-in function that translates occurrences of nominated characters, which is specified either in character form or as a hexadecimal value, to an alternative character or hexadecimal value. The translation occurs when the source string is assigned to the target variable, which is specified to the left of the &TRANS function. The source string is unchanged. Translation occurs after substitution of any variables in the source string. The source string is therefore assumed to start one blank position following the end of the translation control argument. Any additional blanks will be treated as part of the source string and will be translated if applicable, as shown in the second example.

Operands:

***C'x*^{1} y^{1} ... x^{n} y^{n}**

Indicates that all occurrences of the character x^m in the source string are to be replaced by the character y^m that is paired with x^m in the translate control string.

***X'xx*^{1} yy^{1} ... xx^{n} yy^{n}**

Indicates that all occurrences of the character whose hexadecimal value is xx^m in the source string are to be replaced by the character whose hexadecimal value is yy^m that is paired with xx^m in the translate control string.

UPPER

Translates any lowercase characters to uppercase.

LOWER

Translates any uppercase characters to lowercase.

PRINT

Translates non-printable characters to blanks.

NLUPPER [=*lc*]

Uses the language code of the user to decide the character set that is used for the translation. A predefined language code can optionally be specified, for example, NLUPPER=GR.

When the language code (either that of the user or the supplied value) does not match one of the supported language codes (see page 1187), the language code of the system is used to perform the translation. When the language code of the system is not one of the supported values, the value UK is used.

NLLOWER [=*lc*]

Uses the language code of the user to decide the character set that is used for the translation. A predefined language code can optionally be specified, for example, NLLOWER=GR.

When the language code (either that of the user or the supplied value) does not match one of the supported values, the language code of the system is used to perform the translation. When the language code of the system is not one of the supported values, the value UK is used.

string

Data which is to be translated.

Examples: &TRANS

```
&A = ABC
&A1 = ABD
&B = &TRANS C'A1B2C3' &A &A1     -* (single blank after
                                 -* translate argument)
                                 -* Result &B = 123 12D

&A = ABC
&A1 = ABD
&B = &TRANS C'A1B2C3 %'  &A &A1 -* (two blanks after translate
                                 -* argument
                                 -* Result &B = %123%12D

&A = ABC
&B = &TRANS X'C1C2C3C4' &A       -* Result &B = BBD

&A = john smith
&B = &TRANS UPPER &A             -* Result &B = JOHN SMITH
```

Notes:

Two adjacent single quotes must be used if a quote forms part of the translate control string.

This function is particularly useful for screening data that is to be displayed on a panel using the preparse facility.

&TYPECHK

Returns one of a supplied list of data types based on supplied variables.

```
&TYPECHK (type1, type2 ....) &var1 [ &var2 .... &varn ]
```

Allows you to test one or more variables against a list of type attributes.

&TYPECHK is a built-in function and must be used to the right of an assignment statement. The statement returns a value after assignment that represents the first matching type attribute that was found or else a null value.

The first supplied value is tested against each of the type values specified in the list until a match is found. If the variable's type attribute does not match any in the list, then a null value is returned. If a match is found, then that type is returned.

If more than one source value is supplied, then &TYPECHK produces one of the following results:

- If all values match the type of the first value, then that type is returned.
- If any of the values have a different type attribute from the first value, then a null value is returned.

Operands:

type

One or more type values against which the source values are to be tested. A single type value is coded on its own, or multiple values is coded enclosed in parentheses. Valid values and value meanings for type are:

- ALPHA, an alphabetic character value
- ALPHANUM, an alphanumeric character value
- ALPHANUMNAT, an alphanumeric or national character value
- DATE1, a date in the format of &DATE1 system variable
- DATE2, a date in the format of &DATE2 system variable
- DATE3, a date in the format of &DATE3 system variable
- DATE4, a date in the format of &DATE4 system variable
- DATE5, a date in the format of &DATE5 system variable
- DATE6, a date in the format of &DATE6 system variable
- DATE7, a date in the format of &DATE7 system variable
- DATE8, a date in the format of &DATE8 system variable &TYPECHK
- DATE9, a date in the format of &DATE9 system variable
- DATE10, a date in the format of &DATE10 system variable

- DATE11, a date in the format of &DATE11 system variable
- DATE12, a date in the format of &DATE12 system variable
- DATE13, a date in the format of &DATE13 system variable
- DATE14, a date in the format of &DATE14 system variable
- DATE16, a date in the format of &DATE16 system variable
- DATE17, a date in the format of &DATE17 system variable
- DOMAIN, a valid NCL domain ID
- DSN, a valid data set name (with or without member name in z/OS)
- HEX, a hexadecimal value
- IPADDR, a valid IP address, in the format A.B.C.D, where each of A, B, C, and D have a valid range of 0 to 255
- MIXED, a string containing valid DBCS data
- MSGLVL, a valid AOM message level value
- NAME, a valid PDS member name (that is, starts with an alphabetic or national character and all other characters are alphanumeric or national characters)
- NAME12, a variable, up to 12 characters, conforming to member name rules
- NAME256, a variable, up to 256 characters, conforming to member name rules
- NULL, a null value
- NUM, a numeric value
- REAL, a real number
- ROUTCDE, a valid AOM message route code
- SIGNNUM, a signed number in the range -2,147,483,647 to +2,147,483,647
- TIME1, a time value with the format of the &ZTIME1 system variable
- TIME2, a time value with the format of the &ZTIME2 system variable
- TIME3, a time value with the format of the &ZTIME3 system variable
- Y, Y or YES
- N, N or NO

&var

This operand is a system or user variable. Multiple variables is coded.

Examples: &TYPECHK

```
&1 = ABC
&A = &TYPECHK (NUM,ALPHA) &1
```

Result: &A = ALPHA

```
&1 = +111
&A = &TYPECHK (NUM,SIGNNUM) &1
```

Result: &A = SIGNNUM

```
&1 = 1234567890
&A = &TYPECHK (SIGNNUM,NUM) &1
```

Result: &A = SIGNNUM

```
&1 = 12345678901
&A = &TYPECHK (SIGNNUM,NUM) &1
```

Result: &A = NUM (Because the number exceeds the maximum for NCL arithmetic).

```
&1 = 123
&2 = ABC
&A = &TYPECHK (NUM,HEX) &1 &2
```

Result: &A is set to NULL, because &1 and NUM match but NUM does not match &2.

Notes:

By using &TYPECHK to test variables before using them in arithmetic functions, you can avoid the procedure being terminated with an invalid arithmetic function error condition due to invalid input.

The NUM type applies to variables containing a number with no leading sign. This number can, however, be too large for arithmetic operations. To determine whether you can use a numeric variable in arithmetic, check its type attribute for SIGNNUM or REAL. SIGNNUM is returned if the number is a positive or negative integer that is used in arithmetic.

ALPHA means that the variable contains alphabetic characters only, and no national characters. ALPHANUM means that the variable can contain alpha and numeric characters, but not necessarily a mixture.

When working with full-screen procedures, Panel Services provides facilities that automatically perform validation of data entered by the operator.

Note: For more information, see the *Network Control Language Programming Guide*.

If &TYPECHK is used to test a list of variables which include NULL values, then NCL variable substitution logic does not let it recognize the existence of those null variables.

&VARTABLE

&VARTABLE statements add, maintain, monitor, or delete tables of variables (vartables), and vartable entries.

```
&VARTABLE     ADD
              ID=tablename
              KEY=fieldname
              [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
              [ ADJUST=n | COUNTER=n ]
              [ FIELDS=fieldlist { VARS=(var1, var2, ..., varn) |
                        VARS=prefix* [ RANGE=( start, end ) ] |
                        ARGS [ RANGE=( start, end ) ] |
                        MDO=mdoname } ]
```

```
&VARTABLE    ALLOC
             ID=tablename
             [ SCOPE={ PROCESS | REGION |
                       SYSTEM | AOM } ]
             [ AGE={ NO | NEW | ALL | UPDATE | GET } ]
             [ DATA={ 1 | n | MAPPED } ]
             [ DELOLD={ YES | NO } ]
             [ KEYFMT={ CHAR | UCHAR | NUM } ]
             [ KEYLEN=keylen ]
             [ LIMIT={ 0 | n } ]
             [ USERCORR={ NO | YES } ]

&VARTABLE    DELETE
             ID=tablename
             [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
             [ KEY=fieldname ]
             [ FIELDS=fieldlist VARS=varlist ]

&VARTABLE    FREE
             ID=tablename
             [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]

&VARTABLE    GET
             ID=tablename
             KEY=fieldname
             [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
             [ AGE={ YES | NO } ]
             [ DELETE={ YES | NO } ]
             [ FIELDS=fieldlist { VARS=(var1, var2, ..., varn) |
                       VARS=prefix* [ RANGE=( start, end ) ] |
                       ARGS [ RANGE=( start, end ) ] |
                       MDO=mdoname } ]
             [ OPT={ KEQ | KGE | KLE | KGT | KLT | FIRST |
                       LAST | GEN | IGEN | OLDEST | NEWEST } ]

&VARTABLE    PUT | UPDATE
             ID=tablename
             KEY=fieldname
             [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
             [ ADJUST=n | COUNTER=n ]
             [ FIELDS=fieldlist { VARS=(var1, var2, ..., varn)
                       VARS=prefix* [ RANGE=( start, end ) ] |
                       ARGS [ RANGE=( start, end ) ] |
                       MDO=mdoname } ]

&VARTABLE    QUERY
             ID=tablename
             [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
             [ FIELDS=fieldlist VARS=varlist ]
```

```
&VARTABLE    RESET
             ID=tablename
             [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
             [ OLDEST=n | NEWEST=n ]
```

&VARTABLE ADD

Allows an NCL procedure to add a vartable entry to an existing vartable.

```
&VARTABLE ADD ID=tablename KEY=fieldname
           [ SCOPE={ PROCESS | REGION | SYSTEM | AOM} ]
           [ ADJUST=n | COUNTER=n ]
           [ FIELDS=fieldlist { VARS=(var1, var2, ...  varn) |
                                VARS=prefix* [ RANGE=( start, end ) ] |
                                ARGS [ RANGE=( start, end ) ] | MDO=mdoname } ]
```

Operands:

ID=*tablename*

(Mandatory) Indicates the name of the table to which you want to add a variable entry. The table must have been previously allocated, although not necessarily by this procedure (particularly if SCOPE=REGION is specified).

KEY=*fieldname*

(Mandatory) Indicates the name of the NCL variable that contains the value of the key to assign to this table entry. Do not code an ampersand (&) unless the name of the variable containing the key is stored in the variable specified on the KEY= operand. For example:

- KEY=KEYFIELD means extract the contents of NCL variable &KEYFIELD, and use those contents as the key value.
- KEY=&KEYFLDNM means extract the contents of NCL variable &KEYFLDNM, and use those contents as the name of an NCL variable that contains the key value. If &KEYFLDNM contained KF, the contents of variable KF would be used as the key value.

If the table was allocated with KEYFMT=CHAR, the nominated key value is padded with blanks if shorter than the declared key length of this table, or an error response is set and the entry not added if the value is longer. KEYFMT=UCHAR performs an uppercase translation before using the supplied key value.

If the table was allocated with KEYFMT=NUM, the key value must be a valid, signed number, that is, it must satisfy an &TYPECHK of SIGNNUM.

SCOPE= { PROCESS | REGION | SYSTEM | AOM }

An optional parameter, indicating the scope of the table. Allowed values are:

PROCESS

(Default) Indicates the table is visible only to the NCL process that allocates it. This includes any nested or higher-level executed procedures.

REGION

Indicates the table is visible to all NCL procedures executing in the current region. This includes any procedures executing in another window, if you have more than one window open.

SYSTEM

Indicates the table is visible to all NCL procedures executing in the same system.

AOM

Indicates that this verb refers to a mirrored vartable. The entry is also added to the mirrored copy if AOM is started.

Note: SCOPE=AOM is available only if your region includes Automation Services products.

ADJUST=*n* | COUNTER=*n*

These parameters let you set or adjust the counter field of the new entry. Only one is coded, and they are mutually exclusive with the use of COUNTER or ADJUST as field list values (see later). If none of these parameters are specified, the counter field is initialized to 0. *n* must be an integer, optionally signed, that is valid for NCL arithmetic.

COUNTER=*n* sets the counter field of the new entry to the value of *n*.

ADJUST=*n* adds the value of *n* to the counter field of the new entry (*n* can be negative).

Because the counter field of a new entry is initialized to 0, ADJUST=*n* on an ADD operation is the same as COUNTER=*n*.

FIELDS=*fieldlist* { VARS=(*var*1, *var*2, ... *varn*) |

VARS=*prefix [RANGE=(*start*, *end*)] |**

ARGS [RANGE=(*start*, *end*)] |

MDO=*mdoname* }]

These parameters let you specify the information you want to store in the new table entry, and the NCL variables that the information is to be extracted from. If specified, the FIELDS and VARS operands must have the same number of entries in their lists. If VARS=, ARGS or MDO= is specified and the FIELDS= operand is not specified, the equivalent of FIELDS=DATA* is assumed.

FIELDS=*fieldlist* nominates the information you want to store. *fieldlist* is a list of names in one of the following formats:

name

(*name*)

(*name*,*name*,...)

where each name is one of the following (you cannot duplicate any of these names in the list):

DATA*n*

Indicates that you are operating on a data value for the *n*th data field in this entry. *n* must be from 1 to the value specified on the DATA= parameter when the table was allocated. You can have several DATA*n* entries, as long as each has a unique number *n*.

DATA*

Indicates that you are operating on the data for all the data fields in this entry, from 1 to the value specified on the DATA= parameter when the table was allocated. The accompanying variable name in the VARS= list must be in the format *prefix**, and the suffixes generated to access the variables are 1 to the number of allocated data fields.

MDO

Indicates that the entire entry is to be updated from an MDO. An MDO name must be located in the corresponding position in the VARS list.

.COUNTER

Indicates that you are supplying an initial value for the counter field in this entry. The field is mutually exclusive with .ADJUST, and COUNTER= or ADJUST=.

.ADJUST

Indicates that you are supplying an adjustment value for the counter field in this entry. The field is mutually exclusive with .COUNTER, and COUNTER= or ADJUST=.

.USERCORR

Indicates that you are supplying a user correlator check value. Because a new entry cannot have a value to check against, the supplied value is ignored (but must be numeric).

AOM tables can have the following additional field names specified:

.AOMID

Indicates the AOM ID value. The associated variable must be null, or contain a value from 1 to 12 characters. The value is folded to uppercase.

.AOMATTR

Is the AOM attribute string.

VARS=(*var*1,*var*2,...,*varn*) nominates the NCL variables that contain the information for each entry in the *fieldlist*. A one-to-one correspondence exists from each entry in the variables list to the same entry in *fieldlist*. Thus the first entry in variables list nominates the variable containing the data for the first entry in *fieldlist*. The variables list must be in one of the following formats:

varname

(*varname*)

(*varname*,*varname*,...)

where *varname* is a valid NCL variable name, without the ampersand (&), unless you want to refer to the variable containing the data indirectly (see KEY=). If DATA* or D* was specified in the FIELDS list, the matching variable name must be specified as *prefix**.

VARS | ARGS | MDO defines the source data structures for the ADD operation against the vartable.

The FIELDS keyword is optional, and if not specified, defaults to FIELDS=DATA*.

If the FIELDS operand is specified, then the VARS operand must be specified, and must be a list of variables that are the target of the ADD operation. The list must contain one of the following:

- The names of known VARTABLE fields (such as COUNTER and ADJUST).
- An operand of the form DATA*n*. *n* is any integer within the range 1 through 999 for a DATA=MAPPED vartable. Otherwise n is in the range1 to the data limit (as determined by the DATA operand on the &VARTABLE ALLOC statement).
- The operand DATA* (meaning all data fields).
- The operand MDO (meaning the entire data object in the vartable entry).

Each entry in the VARS list must parallel an entry in the FIELDS list and must be one of the following:

- An NCL token name.

- A generic name (for example, ABC*) if it parallels the DATA* entry in the FIELDS list.
- An MDO name (for example, ABC.) if it parallels the MDO operand (or DATA* operand) in the FIELDS list.

When the FIELDS operand is omitted (or specified as FIELDS=DATA*), the source variables are specified by the usual NCL syntax (that is, as ARGS [RANGE=], VARS=*varslist*, *VARS=prefix* [RANGE=], or MDO=*mdoname*).

If an MDO is nominated as the source data structure it is placed intact into the vartable as the vartable entry. Mapping Services maintains the mapping for a subsequent GET operation.

Examples: &VARTABLE

```
&K = KEY001
&D = DATA001
&VARTABLE ADD ID=MYTABLE KEY=K FIELDS=DATA1 VARS=D
```

This example adds an entry to the private (SCOPE=PROCESS) vartable named MYTABLE. The entry has a key value of KEY001 and a data1 content of DATA001.

Return Codes:

System variable &ZFDBK is set after an &VARTABLE ADD statement to indicate the result of the operation:

0

The entry was added successfully.

1

The entry was added successfully. The table was at the limit specified by the &VARTABLE ALLOC, and DELOLD=YES was also specified on the ALLOC. The oldest entry was deleted to make room for this entry.

4

An entry with the nominated key value exists.

12

The supplied key value was longer than the table key length.

16

No table of this name exists in this scope.

24

The table is already at the limit specified by &VARTABLE ALLOC. The entry could not be added.

28

Variable specified for .AOMID is longer the 12 characters.

32

SCOPE=AOM table has been disabled due to a storage error.

100

Variable specified for .AOMATTR is longer than 30 characters.

101 to 130

Variable specified for .AOMATTR has an invalid value at the position indicated by 130 subtracting 100 from the &ZFDBK code.

&ZFDBK values 28, 32, 100, and 101 through 130 are possible only with AOM tables. &ZFDBK value 1 cannot occur with AOM.

Syntax errors in a &VARTABLE ADD statement terminate the NCL procedure. Always specify SCOPE=REGION or SCOPE=SYSTEM to refer to a table of that scope.

&VARTABLE ALLOC

Allows an NCL procedure to allocate a new vartable. Once allocated, other &VARTABLE statements can refer to the table. The table is defined without entries.

```
&VARTABLE ALLOC ID=tablename
              [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
              [ AGE={ NO | NEW | ALL | UPDATE | GET } ]
              [ DATA={ 1 | n | MAPPED } ]
              [ DELOLD={ YES | NO } ]
              [ KEYFMT={ CHAR | UCHAR | NUM } ]
              [ KEYLEN=keylen ]
              [ LIMIT={ 0 | n } ]
              [ USERCORR={ NO | YES } ]
```

Operands:

ID=*tablename*

(Mandatory) Names the table which other &VARTABLE statements can then refer to. *tablename* must be a 1- to 12-character name; the first character alphabetic or national, the rest alphanumeric or national.

SCOPE= { PROCESS | REGION | SYSTEM | AOM }

An optional parameter, indicating the scope of the table. Allowed values are:

PROCESS

(Default) The table is visible only to the NCL process that allocates it. This includes any nested or higher-level EXECuted procedures.

REGION

Indicates the table is visible to all NCL procedures executing in the current region. This includes any EXECuted, STARTed, or &INTCMDed procedures in this region, and any procedures executing in another OCS window, if you have more than one window open.

SYSTEM

Indicates that the table is visible to all NCL procedures executing in the same system.

AOM

Indicates this statement allocates a mirrored vartable. If AOM is not started the actual mirrored copy is not allocated. When AOM is started the mirrored copy is built.

Note: SCOPE=AOM is available only if your region includes Automation Services products.

The amount of storage needed for the mirrored copy is determined by the LIMIT parameter. This parameter is required if SCOPE=AOM is specified. If the maximum amount of storage allowed for mirrored vartables is exceeded, as set by the SYSPARMS AOMMIRST command, the ALLOC command is rejected with an &ZFDBK of 24. This also happens when the maximum allowable storage is set to zero, to disable the use of mirrored vartables.

KEYLEN=*klen*

Specifies the length of the keys in this table. This value is required if KEYFMT=CHAR is specified or assumed.

Range: 1 through 256

AGE={NO | NEW | ALL | UPDATE | GET}

Specifies whether to age entries when certain operations are performed on them. For all specifications, an added entry is always marked as the newest. Aging allows a table to be used as a cache to keep frequently referenced entries in the table and to allow automatic deletion of old entries.

NO or NEW

Indicates that only entries added to the table become the newest entries. All other references leave an entry in relative age order.

ALL

Indicates that any reference to a table entry makes that entry the newest. This reference includes GET, PUT, ADD, or UPDATE.

UPDATE

Indicates that an entry updated by PUT or UPDATE is also made the newest entry.

GET

Indicates that an entry retrieved by GET is also made the newest entry.

Default: NO

DATA={ 1 | *n* | MAPPED }

Indicates how many data fields is stored in each table entry. A number from 1 to 16 is specified. If omitted, this field defaults to 1. Each data field can hold up to 256 characters of data.

The number specified determines the maximum number of fields, which correspond to NCL variables, that is placed in a single vartable entry.

DATA=MAPPED

Specifies that each entry in the vartable is either an MDO, or can contain an unrestricted number of NCL variables.

DELOLD={ YES | NO }

Indicates whether &VARTABLE ADD or &VARTABLE PUT (when adding) can delete the oldest entry automatically when the table is full (for tables allocated with a LIMIT that is not zero; this option has no meaning if LIMIT=0 is specified or defaulted):

- DELOLD=NO means that the ADD or PUT does not proceed; a &ZFDBK value of 24 is returned.
- DELOLD=YES means that the ADD or PUT will proceed. The oldest entry is deleted and an &ZFDBK value of 1 is returned to warn the user.

KEYFMT={CHAR | UCHAR | NUM}

Specifies whether the table has a numeric or character format key:

CHAR

Indicates that the key is a character string. The table is ordered for sequential retrieval based on the character value of the key and blank padded if necessary. KEYLEN is required for this value.

UCHAR

Is the same as KEYFMT=CHAR except that lowercase characters are translated to uppercase. KEYLEN is required for this value.

NUM

Indicates that the key is a signed number. The table is ordered based on the numeric value of the key (largest negative through 0 to largest positive). Key values must always be a valid, optionally signed number, from -2147483648 to 2147483647. KEYLEN cannot be specified for this value.

Default: CHAR

KEYLEN=*keylen*

Must be provided if KEYFMT=CHAR or UCHAR is specified or defaulted, to indicate the key length of the entries in this table. *keylen* must be a number from 1 to 256. Key values supplied when adding entries to this table are padded with blanks to the nominated length, if shorter. If key values are longer, an error response is returned.

LIMIT={ 0 | *n* }

Indicates whether the table is to have a limit on the number of entries.

LIMIT=0

(Default) Indicates that the table can have any number of entries.

Limit=*n*

n is from 1 to 1,000,000 indicating that no more than *n* entries is added to the table. If an &VARTABLE ADD or &VARTABLE PUT operation exceeds this limit and DELOLD=NO is specified or defaulted on the ALLOC, the addition is not performed, and an &ZFDBK value of 24 is returned. If DELOLD=YES is specified, the addition is performed, and the oldest entry automatically deleted to make room.

USERCORR= { NO | YES }

An optional parameter, allowing control over the use of the USERCORR field in table entries when performing &VARTABLE UPDATE or &VARTABLE PUT operations.

USERCORR=NO (the default) means that use of the user correlator is optional.

USERCORR=YES means that use of the user correlator is required when updating table entries.

Examples: &VARTABLE ALLOC

```
&VARTABLE ALLOC ID=MYTABLE KEYLEN=20
```

This example allocates a private table called MYTABLE with a key length of 20.

```
&VARTABLE ALLOC ID=STABLE SCOPE=SYSTEM KEYLEN=30 LIMIT=50 USERCORR=YES
```

This example allocates a table called STABLE, which is visible to all NCL procedures running in this system. The key length is 30, and update operations require use of the user correlator. A limit of 50 entries is placed on the table.

Return Codes:

System variable &ZFDBK is set after an &VARTABLE ALLOC statement to indicate the result of the operation:

0

Indicates that the table was allocated successfully.

16

Indicates that a table of this name exists in this scope. A table of the same name is allocated in each of the three scopes. For example, an NCL procedure could allocate a table called TAB1 with a scope of PROCESS, another with a scope of REGION, and another with a scope of SYSTEM. All statements that want to refer to the REGION or SYSTEM level table must specify the SCOPE= parameter.

20

Indicates that 16 tables are already allocated.

24

Indicates that the allocation causes the total storage necessary for mirroring to exceed AOMMAXIS.

&ZFDBK values 20 and 24 are only possible with AOM tables.

Notes:

Syntax errors in a &VARTABLE ALLOC statement will terminate the NCL procedure.

AOM requires KEYLEN=16 and a nonzero limit value on an &VARTABLE ALLOC.

&VARTABLE DELETE

Allows an NCL procedure to delete an entry from an existing vartable. The delete can be optionally synchronized with any concurrent updating.

```
&VARTABLE DELETE ID=tablename
                 [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
                 [ KEY=fieldname ]
                 [ FIELDS=fieldlist VARS=varlist ]
```

Operands:

ID=*tablename*

(Mandatory) Indicates the name of the table you wish to delete a vartable entry from. The table must have been previously allocated, although not necessarily by this procedure (particularly if SCOPE=REGION is specified).

SCOPE= { PROCESS | REGION | SYSTEM | AOM }

An optional parameter, indicating the scope of the table you wish to delete. Allowed values are:

PROCESS

(Default) Indicates that the table is a private table allocated by this NCL process.

REGION

Indicates that the table has been allocated a scope of REGION.

SYSTEM

Indicates that the table has been allocated by all NCL procedures executing in the same system.

AOM

Indicates that this statement refers to a mirrored vartable. The entry is deleted from the mirrored copy if AOM is started.

Note: SCOPE=AOM is available only if your region includes Automation Services products.

KEY=*fieldname*

(Mandatory) Indicates the name of the NCL variable that contains the value of the key of the table entry you wish to delete. Do not code an ampersand(&) unless the name of the variable containing the key is stored in the variable specified on the KEY= operand. For example:

- KEY=KEYFIELD means extract the contents of NCL variable &KEYFIELD, and use those contents as the key value.
- KEY=&KEYFLDNM means extract the contents of NCL variable &KEYFLDNM, and use those contents as the name of an NCL variable that contains the key value. If &KEYFLDNM contains KF, the contents of variable KF are used as the key value.

If the table was allocated with KEYFMT=CHAR, the nominated key value is padded with blanks, if shorter than the declared key length of this table. If the value is longer, an error response is set and the entry not added. KEYFMT=UCHAR performs on upper case translation before using the supplied key value.

If the table was allocated with KEYFMT=NUM, the key value must be a valid, signed number, that is, it must satisfy an &TYPECHK of SIGNNUM.

FIELDS=*fieldlist* VARS=*varlist*

These parameters let you specify a user correlator value for validation against the current table contents. You must specify both these parameters, or omit both. If specified, the two parameters must have one entry in each list.

FIELDS=*fieldlist* nominates the information you want to delete. *fieldlist* is a list of names in one of the following formats:

name

(*name*)

(*name*,*name*,...)

where each name is an unduplicated user correlator value (.USERCORR) from a previous &VARTABLE GET supplied to check synchronization (the value corresponds to that for the variable in the VARS list).

VARS=*varlist* nominates the NCL variables that contain the information for each entry in *fieldlist*. There is a one-to-one correspondence between each entry in *varlist* and the same entry in *fieldlist*. Thus the first entry in *varlist* nominates the variable containing the data for the first entry in *fieldlist*. *varlist* must be in one of the following formats:

varname

(*varname*)

(*varname*,*varname*,...)

where *varname* is a valid NCL variable name, without the ampersand (&), unless you wish to indirectly refer to the variable containing the data (see discussion under KEY= above).

Examples: &VARTABLE DELETE

```
&K = KEY001 &VARTABLE DELETE ID=MYTABLE KEY=K
```

This example deletes the entry with key value KEY001, in the private vartable (SCOPE=PROCESS), called MYTABLE. If there is no entry with that key, &ZFDBK is set to 4.

Return Codes:

System variable &ZFDBK is set after an &VARTABLE DELETE statement to indicate the result of the operation:

0

The entry was deleted successfully.

4

No entry with the nominated key value exists.

8

The value of the supplied user correlator does not match the value in the table entry.

12

The supplied key value was longer than the table key length.

16

No table of this name exists in this scope.

32

SCOPE=AOM table has been disabled due to a storage error.

&ZFDBK value 32 is only possible with AOM tables.

Notes:

Syntax errors in a &VARTABLE DELETE statement will cause the NCL procedure to terminate.

You must always specify SCOPE=REGION to refer to a table of that scope.

&VARTABLE FREE

Allows an NCL procedure to delete an entire vartable, including entries and the vartable definition, to free storage.

```
&VARTABLE FREE ID=tablename
              [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
```

Operands:

ID=*tablename*

(Mandatory) Indicates the name of the table you wish to free. *tablename* must be the name of an existing vartable, within the specified scope.

SCOPE= { PROCESS | REGION | SYSTEM | AOM }

An optional parameter, indicating the scope of the table you wish to free. Allowed values are:

PROCESS

Indicates the table is visible only to the NCL process that allocates it. This includes any nested or higher-level EXECuted procedures.

REGION

Indicates the table is visible to all NCL procedures executing in the current region. This includes any procedures executing in another window, if you have more than one window open.

SYSTEM

Indicates that the table is visible to all NCL procedures executing in the same system.

AOM

Indicates the statement refers to a mirrored vartable. The mirrored copy of the vartable is also freed if AOM is started.

Note: SCOPE=AOM is available only if your region includes Automation Services products.

Examples: &VARTABLE FREE

```
&VARTABLE FREE ID=MYTABLE
```

This example frees the private table called MYTABLE.

```
&VARTABLE FREE ID=RTABLE SCOPE=REGION
```

This example frees a table in the current region called RTABLE, if it exists.

Return Codes:

System variable &ZFDBK is set after an &VARTABLE FREE statement to indicate the result of the operation:

0

The table was freed successfully.

16

No table with this name exists within this scope.

20

SCOPE=AOM table was in use at the exact time the deletion of the mirrored copy was attempted. The table remains allocated.

&ZFDBK value 20 is possible with AOM tables only. If this happens, the NCL procedure should delay for approximately one second and try again.

Note:

Syntax errors in a &VARTABLE FREE statement will cause the NCL procedure to terminate.

&VARTABLE GET

Allows an NCL procedure to retrieve an entry from an existing vartable. The exact key of the record is not required.

```
&VARTABLE GET ID=tablename
              KEY=fieldname
            [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
            [ AGE={ YES | NO } ]
            [ DELETE={ YES | NO } ]
            [ FIELDS=fieldlist { VARS=(var1, var2, ... varn) |
              VARS=prefix* [ RANGE=( start, end ) ] |
              ARGS [ RANGE=( start, end ) ] |
              MDO=mdoname } ]
            [ OPT={ KEQ | KGE | KLE | KGT | KLT | GEN |
                    IGEN | FIRST | LAST | OLDEST | NEWEST } ]
```

Operands:

ID=*tablename*

(Mandatory) Indicates the name of the table you want to retrieve the entry from. The table must have been previously allocated, although not necessarily by this procedure (particularly if SCOPE=REGION is specified).

SCOPE= { PROCESS | REGION | SYSTEM | AOM }

An optional parameter, indicating the scope of the table. Allowed values are:

PROCESS

Indicates the table is a private table, visible only to the NCL process that allocated it.

REGION

Indicates the table is visible to all NCL procedures executing in the current region.

SYSTEM

Indicates that the table is visible to all NCL procedures executing in the same system.

AOM

Indicates that this statement refers to a mirrored vartable. Entries added or updated from the screening table can also be retrieved.

Note: SCOPE=AOM is available only if your region includes Automation Services products.

AGE={ YES | NO }

(Optional) Indicates whether to make the retrieved entry the newest in the table (AGE=YES). The default depends on the value for the AGE parameter specified on the &VARTABLE ALLOC statement for this table. If the table has been allocated with aging on GET, using AGE=NO on an &VARTABLE GET lets you access entries for maintenance without aging entries.

DELETE={ YES | NO }

Indicates whether to delete the entry retrieved. DELETE=NO indicates the entry is not deleted. DELETE=YES indicates the retrieved entry is deleted from the table.

KEY=*fieldname*

(Must not be specified if OPT=FIRST, LAST, OLDEST, or NEWEST is specified) Indicates the name of the NCL variable that contains the value of the search key. Do not code an ampersand (&) unless the name of the variable containing the key is stored in the variable specified on the KEY= operand. For example:

- KEY=KEYFIELD means extract the contents of NCL variable &KEYFIELD, and use those contents as the key value.
- KEY=&KEYFLDNM means extract the contents of NCL variable &KEYFLDNM, and use those contents as the name of an NCL variable that contains the key value. If &KEYFLDNM contained KF, the contents of variable KF is used as the key value.

If the table was allocated with KEYFMT=CHAR, the nominated key value is padded with blanks, if shorter than the declared key length of this table. If the value is longer, an error response is set. KEYFMT=UCHAR performs an uppercase translation before using the supplied key value.

If the table was allocated with KEYFMT=NUM, the key value must be a valid, signed number. That is, it must satisfy an &TYPECHK of SIGNNUM.

FIELDS=*fieldlist* VARS=*varlist*

Defines the source variables from the vartable to retrieve by the GET operation. If specified, the FIELDS and VARS operands must have the same number of entries in their lists. If VARS=, ARGS or MDO= is specified and the FIELDS= operand is not specified, the equivalent of FIELDS=DATA* is assumed.

FIELDS=*fieldlist* nominates the information you want to retrieve. *fieldlist* is a list of names in one of the following formats:

name

(*name*)

(*name*,*name*,...)

Each name is one of the following (you cannot duplicate any of these names in the list):

DATA*n*

Indicates that you are operating on a data value for the *n*th data field in this entry. *n* must be from 1 to the value specified on the DATA= parameter when the table was allocated. You can have several DATA*n* entries, as long as each has a unique number *n*.

DATA*

Indicates that you are operating on the data for all the data fields in this entry, from 1 to the value specified on the DATA= parameter when the table was allocated. The accompanying variable name in the VARS= list must be in the format *prefix**, and the suffixes generated to access the variables are 1 to the number of allocated data fields.

MDO

Indicates that you want to place the entire entry into an MDO. An MDO name must be located in the corresponding position in the VARS list.

Note: When MDO access is used, individual data fields cannot be requested.

MAP

Indicates that you want to retrieve the map name for the corresponding entry. A single NCL variable must be located in the corresponding position of the VARS list. For entries containing NCL variables, the map name returned is always $NCL, regardless of whether the vartable was allocated as mapped or not. For entries containing MDOs, the map name returned is that associated with the MDO when the entry was last updated.

.KEY

Indicates that you want to retrieve the actual key value of this entry. This value is different from the supplied search key (KEY=) if you are not using OPT=KEQ. If the same variable as the search key is used in the field list, its value is updated after the search key value is extracted.

.COUNTER

Indicates that you want to retrieve the current value of the counter field for this entry.

.USERCORR

Indicates that you want to retrieve the user correlator value for this entry. This value is used in a later UPDATE, PUT, or DELETE operation to help ensure that no other updates have taken place.

AOM tables can have the following additional field names specified:

.AOMID

Indicates the AOM ID value. The associated variable is set to the stored AOM ID value. If the stored ID is blank, the value is null.

.AOMATTR

Is the AOM attribute string. A full 30-character string is always returned.

.AOMCOUNT

Is the AOM count for this entry. If the COUNT option is specified, a LOOKUP statement that matches an entry increments this counter. If GAOM is stopped, a zero value is always returned.

VARS=(*var1*,*var2*,...,*varn*) nominates the target structures for the GET operation against the vartable. Each entry in *varlist* corresponds to the same entry in *fieldlist*. Thus the first entry in *varlist* nominates the variable containing the data for the first entry in *fieldlist*. *varlist* must be in one of the following formats:

```
varname
```

```
(varname)
```

```
(varname,varname,...)
```

varname is a valid NCL variable name, without the ampersand (&), unless you want to refer to the variable containing the data indirectly (see the example for the KEY operand). If DATA* or D* was specified in the FIELDS list, the matching variable name must be specified as *prefix**.

VARS | ARGS | MDO defines the target data structures for the GET operation against the vartable.

The FIELDS keyword is optional, and if not specified, defaults to FIELDS=DATA*.

If the FIELDS operand is specified, then the VARS operand must be specified, and must be a list of variables to contain the source fields. The list must contain one of the following:

- The names of known VARTABLE fields (such as COUNTER and ADJUST).
- An operand of the form DATA*n*. *n* is any integer within the range 1 through 999 for a DATA=MAPPED vartable. Otherwise n is in the range1 to the data limit (as determined by the DATA operand on the &VARTABLE ALLOC statement).
- The operand DATA* (meaning all data fields).
- The operand MDO (meaning the entire data object in the VARTABLE entry).

Each entry in the VARS list must parallel an entry in the FIELDS list and must be one of the following:

- An NCL token name
- A generic name (for example, ABC*) if it parallels the DATA* entry in the FIELDS list
- An MDO name (for example, RECORD.) if it parallels the MDO operand (or DATA* operand) in the FIELDS list

When the FIELDS operand is omitted (or specified as FIELDS=DATA*), the target variables are specified by the usual NCL syntax (that is, ARGS [RANGE=], VARS=*varslist*, VARS=*prefix* [RANGE=], or MDO=*mdoname*).

When VARS or ARGS are the target of a GET operation, but the vartable entry was created as an MDO (and not mapped by $NCL), the target variables are segmented from the MDO contents.

If an MDO is nominated as the target data structure, all data from the vartable entry is placed in the single MDO. If no map name is supplied on the GET operation, the map name specified for the MDO on the PUT operation is supplied as the default. If the entry was created from NCL variables and not an MDO, the $NCL map maps the resulting MDO.

OPT={KEQ | KGE | KLE | KGT | KLT | GEN | IGEN | FIRST | LAST | OLDEST | NEWEST}

Indicates the relationship between the supplied search key and the matching table entry (if one is found).

KEQ

(Default) Indicates that you want to retrieve the table entry with an exact match on the supplied search key.

KGE

Indicates that you want to retrieve the table entry with the lowest key value greater than or equal to the supplied search key.

KLE

Indicates that you want to retrieve the table entry with the highest key value less than or equal to the supplied search key.

KGT

Indicates that you want to retrieve the table entry with the lowest key value greater than the supplied search key.

KLT

Indicates that you want to retrieve the table entry with the highest key value less than the supplied search key.

GEN

Indicates that you want to retrieve the table entry with the lowest key value generically equal to the search key value for its non-blank length, but possibly with other characters after it.

IGEN

Indicates that you want to retrieve the table entry with the longest non-blank key value that matches the search argument.

FIRST

Indicates that you want to retrieve the table entry with the lowest key. If this option is specified, you cannot specify the KEY operand.

LAST

Indicates that you want to retrieve the table entry with the highest key. If this option is specified, you cannot specify the KEY operand.

OLDEST

Indicates that you want to retrieve the oldest table entry, that is, the entry that was first added, updated, or retrieved (depending on the ALLOC AGE= option). If this option is specified, you cannot specify the KEY operand.

NEWEST

Indicates you want to retrieve the newest table entry, that is, the entry that was last added, updated, or retrieved (depending on the ALLOC AGE= option). If this option is specified, you cannot specify the KEY operand.

If no entry is found that matches the passed key, &ZFDBK is set to 4, and none of the nominated variables are updated.

Examples: &VARTABLE GET

```
&K = KEY001
&VARTABLE GET ID=MYTABLE KEY=K FIELDS=DATA1 VARS=D
&WRITE DATA FOR KEY &K IS &D
```

This example retrieves the entry from the private (SCOPE=PROCESS) vartable named MYTABLE. The field, &D, contains the returned user data in that entry.

```
&VARTABLE GET ID=GTABLE SCOPE=SYSTEM OPT=FIRST +
    FIELDS=(KEY,DATA1) VARS=(K,D)
&DOWHILE &ZFDBK = 0
    .
    ...  process entry, key in &K, data in &D
    .
    &VARTABLE GET ID=GTABLE SCOPE=SYSTEM OPT=KGT KEY=K +
                  FIELDS=(KEY,DATA1) VARS=(K,D)
&DOEND
```

This example illustrates a technique to read sequentially through a table. The first GET, using OPT=FIRST, retrieves the entry with the lowest key, and the key value is returned in &K. The second GET, using OPT=KGT, retrieves the entry with the next higher key, and sets that key value into &K.

Use OPT=LAST and OPT=KLT for backward retrieval.

```
&VARTABLE GET ID=FRED SCOPE=GLOBAL OPT=KGT KEY=#OS$STARTKEY +
              FIELDS=(KEY,MDO) VARS=(#OS$NEWKEY,#OS$MDO.)
```

This example reads a vartable record that is mapped into an MDO, and obtains the key of the vartable record at the same time. Note the period after the MDO name in the VARS list.

Return Codes:

System variable, &ZFDBK, is set after an &VARTABLE GET statement to indicate the result of the operation:

0

The entry was retrieved successfully. Any nominated variables are updated.

4

No entry with the requested key value exists.

12

The supplied key value was longer than the table key length.

16

No table of this name exists in this scope.

32

SCOPE=AOM table has been disabled due to a storage error. &ZFDBK value 32 is possible only with AOM tables.

Notes:

Syntax errors in a &VARTABLE GET statement cause the NCL procedure to terminate.

Always specify SCOPE=REGION to refer to a table of that scope.

&VARTABLE PUT or UPDATE

The &VARTABLE PUT statement allows an NCL procedure to add or update an entry within an existing vartable. The entry is added if there is no entry with a matching key, or updated if an entry with a matching key already exists.

The &VARTABLE UPDATE statement allows an NCL procedure to update an entry within an existing vartable.

```
&VARTABLE { PUT | UPDATE }
            ID=tablename
            KEY=fieldname
          [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
          [ ADJUST=n | COUNTER=n ]
          [ FIELDS=fieldlist
           { VARS=(var1, var2, ...  varn) |
             VARS=prefix* [ RANGE=( start, end ) ] |
             ARGS [ RANGE=( start, end ) ] |
             MDO=mdoname } ]
```

Operands:

ID=*tablename*

(Mandatory) Indicates the name of the table you wish to retrieve the entry from. The table must have been previously allocated, although not necessarily by this procedure (particularly if SCOPE=REGION is specified).

KEY=*fieldname*

(Mandatory) Indicates the name of the NCL variable that contains the value of the search key. Do not code an ampersand (&) unless the name of the variable containing the key is stored in the variable specified on the KEY= operand. For example:

- KEY=KEYFIELD means extract the contents of NCL variable &KEYFIELD, and use those contents as the key value.
- KEY=&KEYFLDNM means extract the contents of NCL variable &KEYFLDNM, and use those contents as the name of an NCL variable that contains the key value. If &KEYFLDNM contained KF, the contents of variable KF is used as the key value.

If the table was allocated with KEYFMT=CHAR, the nominated key value is padded with blanks, if shorter than the declared key length of this table. An error response will be set, and the entry not added if the value is longer. KEYFMT=UCHAR performs an upper case translation before using the supplied key value.

If the table was allocated with KEYFMT=NUM, the key value must be a valid, signed number. That is, it must satisfy an &TYPECHK of SIGNNUM.

SCOPE= { PROCESS | REGION | SYSTEM | AOM }

An optional parameter, indicating the scope of the table. Allowed values are:

PROCESS

Indicates the table is a private table, visible only to the NCL process that allocated it.

REGION

Indicates the table is visible to all NCL procedures executing in the current region.

SYSTEM

Indicates that the table is visible to all NCL procedures executing in the same system.

AOM

Indicates that this statement refers to a mirrored vartable. The entry is added to or updated in the mirrored copy if AOM is started.

Note: SCOPE=AOM is available only if your region includes Automation Services products.

ADJUST=*n* | COUNTER=*n*

These parameters let you set or adjust the counter field for a new or existing entry. Only one is coded, and they are mutually exclusive with the use of COUNTER, .COUNTER, ADJUST, and .ADJUST field list values (described below). If none of these parameters is specified, the counter field is initialized to 0 when adding, or left as is when updating. *n* is any valid, signed number that will fit into a full word.

COUNTER=*n* will cause the counter field of the new or updated entry to be set to the value of *n*.

ADJUST=*n* adds *n* to the counter field value of the new or updated entry (*n* can be negative). If the table for updating does not contain a matching key entry, the old counter value is taken as 0.

FIELDS=*fieldlist* { VARS=(*var*1, *var*2, ... *varn*) |

VARS=*prefix [RANGE=(*start*, *end*)] |**

ARGS [RANGE=(*start*, *end*)] |

MDO=*mdoname* }

These optional parameters let you specify the information you want to store in the new table entry, or to replace information in an existing table entry, specifying the NCL variables the information is to be extracted from. If specified, the FIELDS and VARS operands must have the same number of entries within their lists. If VARS=, ARGS or MDO= is specified and the FIELDS= operand is not specified, the equivalent of FIELDS=DATA* is assumed.

FIELDS=*fieldlist* nominates the information you want to store. *fieldlist* is a list of names in one of the following formats:

name

(*name*)

(*name*,*name*,...)

where each name is one of the following (you cannot duplicate any of these in the list):

- DATA*n* indicates you want to update the nth user data field in this entry. Several DATA*n* entries is supplied in the list, as long as each has a unique value for *n*. *n* must be from 1 to the number of allocated data fields (that is, to the number specified for the DATA= operand).
- DATA* indicates you want to add or update all the allocated data fields, extracting the values from NCL variables for the names *prefix1* to *prefixn*. The associated VARS= list entry must be in the format *prefixn*.
- MDO indicates that the entire entry is to be updated from an MDO. An MDO name must be located in the corresponding position in the VARS list.

  Note: When an entry contains an MDO, individual fields cannot be accessed.
- .COUNTER indicates you are supplying an initial or new value for the counter field in this entry. Mutually exclusive with .ADJUST, COUNTER=, or ADJUST=.
- .ADJUST indicates you are supplying an adjustment amount for the counter field in this entry. Mutually exclusive with .COUNTER, COUNTER=, or ADJUST=.
- .USERCORR indicates you are supplying a user correlator check value. If updating an existing entry, the supplied value for the user correlator must match the present value in the existing entry. If not, the update is not performed, and &ZFDBK is set to 8. If inserting a new entry, the value supplied for the user correlator is ignored (but must be numeric).

AOM tables can have the following additional field names specified:

- .AOMID indicates the AOM ID value. The associated variable must be null, or contain a value from 1 to 12 characters. The value will be folded to uppercase.
- .AOMATTR is the AOM attribute string

VARS=*varlist* nominates the NCL variables that contain the information for each entry in *fieldlist*. There is a one-to-one correspondence from each entry in *varlist* to the same entry in *fieldlist*. Thus the first entry in *varlist* nominates the variable containing the data for the first entry in *fieldlist*, and so on. *varlist* must be in one of the following formats:

varname

(*varname*)

(*varname*,*varname*,...)

where *varname* is a valid NCL variable name, without the ampersand (&), unless you wish to indirectly refer to the variable containing the data (see discussion under KEY=).

If DATA* or D* is specified in the FIELDS= list, the associated VARS= list entry must be in the format *prefix**.

VARS | ARGS | MDO defines the source data structures for the ADD operation against the vartable.

The FIELDS keyword is optional, and if not specified, defaults to FIELDS=DATA*.

If the FIELDS operand is specified, then the VARS operand must be specified, and must be a list of variables that are the target of the ADD operation. The list must contain one of the following:

- The names of known VARTABLE fields (such as COUNTER and ADJUST).
- An operand of the form DATA*n*. *n* is any integer within the range 1 to 999 for a DATA=MAPPED vartable. Otherwise n is in the range1 to the data limit (as determined by the DATA operand on the &VARTABLE ALLOC statement).
- The operand DATA* (meaning all data fields).
- The operand MDO (meaning the entire data object in the VARTABLE entry).

Each entry in the VARS list must parallel an entry in the FIELDS list and must be one of the following:

- An NCL token name
- A generic name (for example ABC*) if it parallels the DATA* entry in the FIELDS list
- An MDO name (for example ABC.) if it parallels the MDO operand (or DATA* operand) in the FIELDS list

When the FIELDS operand is omitted (or specified as FIELDS=DATA*) the source variables is specified by the usual NCL syntax (that is, as ARGS [RANGE=], VARS=*varslist*, *VARS=prefix* [RANGE=], or MDO=*mdoname*).

If an MDO is nominated as the source data structure it is placed intact into the vartable as the vartable entry. Mapping Services will maintain the mapping for a subsequent ADD operation.

Examples: &VARTABLE PUT

```
&K = KEY001
&D = DATA001
&VARTABLE PUT ID=MYTABLE KEY=K FIELDS=DATA VARS=D
```

This example adds or updates an entry in the private (SCOPE=PROCESS) vartable called MYTABLE. The entry has a key value of KEY001 and a data content of DATA001.

```
.LOOP &MSGREAD ARGS
&VARTABLE PUT ID=IDTABLE KEY=1 ADJUST=1 +
      FIELDS=DATA VARS=ZMTEXT
&GOTO .LOOP
```

This example builds a table containing all uniquely identified messages received in a MSGPROC, the identifier being the first word. The data for each entry is the last complete message text with that identifier, and the counter field contains the count of messages with that identifier received.

This illustrates the ease with which event counting is performed. The NCL procedure need not be concerned with whether a particular event (message, and so on) has already been seen, as the PUT logic handles this.

Changing the scope in this example to REGION allows another NCL procedure in the NCL region to sequentially read the table and display information to an operator.

Examples: &VARTABLE UPDATE

```
&K = KEY001
&D1 = DATA001A
&D2 = DATA001B
&D3 = DATA001C
&VARTABLE UPDATE ID=MYTABLE KEY=K FIELDS=DATA* +
      VARS=D* ADJUST=1
```

This example updates an entry in the private (SCOPE=PROCESS) vartable called MYTABLE. The entry has a key value of KEY001, and the data fields (three assumed) have a data content of DATA001A, DATA001B, and DATA001C. The counter has 1 added to it.

```
.LOOP &MSGREAD ARGS
&VARTABLE UPDATE ID=IDTABLE KEY=1 ADJUST=1 +
       FIELDS=DATA VARS=&ZMTEXT
&GOTO .LOOP
```

This example updates a table containing all previously nominated uniquely identified messages received in a MSGPROC, the identifier being the first word. The data for each entry is the last complete message text with that identifier, and the counter field contains the count of messages received with that identifier.

Compare this example with the example for &VARTABLE PUT. Only message identifiers previously entered into in the table are counted. If an entry is not found &ZFDBK is set to 4, and the update is not performed.

Changing this example by giving the table a scope of PROCESS, another NCL procedure in the NCL region could sequentially read the table and write the counts, and so on, to a screen.

Return Codes:

System variable &ZFDBK is set after an &VARTABLE PUT statement to indicate the result of the operation:

0

The entry was added or updated successfully.

1

The entry was added successfully. The table was at the LIMIT specified on the &VARTABLE ALLOC, and the oldest entry was deleted to make room for this entry.

8

An entry with the nominated key value already exists, and the supplied user correlator value did not match the user correlator value in that entry.

12

The supplied key value was longer than the table key length.

16

No table of this name exists in this scope.

20

This table was allocated with USERCORR=YES specified, and no user correlator was supplied on the &VARTABLE PUT statement.

28

This table was allocated with USERCORR=YES specified, and no user correlator was supplied on the &VARTABLE PUT statement.

32

SCOPE=AOM table has been disabled due to a storage error.

100

Variable specified for .AOMATTR is longer than 30 characters.

101 to 130

Variable specified .AOMATTR has an invalid value at the position indicated by 130 subtracting 100 from the &ZFDBK code.

&ZFDBK values 28, 32, 100, and 101 to 130 are possible with AOM tables only. &ZFDBK value 1 cannot occur with AOM.

System variable &ZFDBK is set after an &VARTABLE UPDATE statement to indicate the result of the operation:

0

The entry was updated successfully.

4

No entry with the supplied key value exists.

8

An entry with the nominated key value exists, but the supplied user correlator value did not match the user correlator value in that entry.

12

The supplied key value was longer than the table key length.

16

No table of this name exists in this scope.

20

This table was allocated with USERCORR=YES specified, and no user correlator was supplied on the &VARTABLE UPDATE statement.

28

Variable specified for .AOMID is longer the 12 characters.

32

SCOPE=AOM table has been disabled due to a storage error.

100

Variable specified for .AOMATTR is longer than 30 characters.

101 to 130

Variable specified .AOMATTR has an invalid value at the position indicated by 130 subtracting 100 from the &ZFDBK code.

&ZFDBK values 28, 32, 100, and 101 to 130 are possible with AOM tables only.

Notes:

Syntax errors in &VARTABLE PUT and VARTABLE UPDATE statements cause an NCL procedure to terminate.

You must always specify SCOPE=REGION to refer to a table of that scope.

&VARTABLE QUERY

Allows an NCL procedure to inquire about the existence of a given vartable, and an option to retrieve information when the table is found.

```
&VARTABLE QUERY ID=tablename
              [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
              [ FIELDS=fieldlist VARS=varlist ]
```

Operands:

ID=*tablename*

(Mandatory) Indicates the name of the table about which you want to inquire. The table need not have been previously allocated, as the return codes in &ZFDBK indicate this.

SCOPE= { PROCESS | REGION | SYSTEM | AOM }

An optional parameter, indicating the scope of the table about which you want to inquire. Allowed values are:

PROCESS

Indicates the table is a private table, visible only to the NCL process that allocated it.

REGION

Indicates the table was allocated with a scope of REGION.

SYSTEM

Indicates that the table is visible to all NCL procedures executing in the same system.

AOM

Indicates that this statement refers to a mirrored VARTABLE. Extra information is returned for SCOPE=AOM vartables.

Note: SCOPE=AOM is available only if your region includes Automation Services products.

FIELDS=*fieldlist* VARS=*varlist*

These optional parameters let you specify the information you want to retrieve about a table, and the NCL variables where the information is located. Specify both these parameters, or omit both. If specified, the two parameters must have the same number of entries within their lists.

FIELDS=*fieldlist* nominates the information you want returned. *fieldlist* is a list of names in the format:

name

(*name*)

(*name*,*name*,...)

where each name is one of the following (you cannot duplicate any of these names in the list):

.AGE

Indicates that you want the AGE option specified on ALLOC returned.

.DATA

Indicates that you want the DATA value specified on ALLOC returned.

.DELOLD

Indicates that you want the DELOLD option specified on ALLOC returned.

.KEYLEN

Indicates that you want the key length of the table returned. If the table was allocated with KEYFMT=NUM, the associated variable is set to NUM.

.LIMIT

Indicates that you want the LIMIT value specified on ALLOC returned.

.TOTAL

Indicates that you want the current number of entries in the table returned.

.USERCORR

Indicates that you want the USERCORR option specified on ALLOC returned.

AOM tables can have the following additional field names specified:

.AOMTHIT

Indicates the total number of matches that have been counted by screening table LOOKUP statements with the TOTAL option specified.

.AOMTMISS

Indicates the total number of no-matches that have been counted by screening table LOOKUP statements with the TOTAL option specified.

.AOMTADD

Indicates the total number of ADDs that have been counted by screening table LOOKUP statements with the ADD and TOTAL option specified.

VARS=*varlist* nominates the NCL variables that receive the information for each entry in *fieldlist*. A one-to-one correspondence exists from each entry in *varlist* to the same entry in *fieldlist*. Thus the first entry in *varlist* nominates the variable that receives the data for the first entry in *fieldlist*, and so on. *varlist* must be in one of the following formats:

varname

(*varname*)

(*varname*,*varname*,...)

where *varname* is a valid NCL variable name, without the ampersand (&), unless you want to refer to the variable containing the data indirectly.

Examples: &VARTABLE QUERY

```
&VARTABLE QUERY ID=MYTABLE
```

This example queries the existence of a table named MYTABLE, in the current NCL process environment. &ZFDBK is set as a result.

```
&VARTABLE QUERY ID=RTABLE FIELDS=(.TOTAL,.KEYLEN) VARS=(T,K)
```

This example queries the existence of a table named RTABLE. If found, the key length and current number of entries are returned in NCL variables T, and K, respectively.

Return Codes:

System variable &ZFDBK is set after an &VARTABLE QUERY statement to indicate the result of the operation:

0

A table with the supplied name was found in the requested scope. Information variables are set if requested.

16

No table of this name exists in this scope.

Notes;

Syntax errors in a &VARTABLE QUERY statement terminates the NCL procedure.

Always specify SCOPE=REGION to refer to a table of that scope.

&VARTABLE RESET

Allows an NCL procedure to delete all entries from an existing vartable, while retaining the table definition. (The table is then empty as if it has been FREEd and reALLOCated.) As an option, only the oldest or newest n entries is deleted.

```
&VARTABLE RESET ID=tablename
              [ SCOPE={ PROCESS | REGION | SYSTEM | AOM } ]
              [ OLDEST=n | NEWEST=n ]
```

Operands:

ID=*tablename*

(Mandatory) Indicates the name of the table you wish to reset. This must be the name of an existing table within the requested scope.

SCOPE= { PROCESS | REGION | SYSTEM | AOM }

An optional parameter, indicating the scope of the table about which you want to inquire. Allowed values are:

PROCESS

Indicates the table is a private table, visible only to the NCL process that allocated it.

REGION

Indicates the table was allocated with a scope of REGION.

SYSTEM

Indicates that the table is visible to all NCL procedures executing in the same system.

AOM

Indicates that this statement refers to a mirrored VARTABLE. This specification precludes the use of the OLDEST and NEWEST operands.

Note: SCOPE=AOM is available only if your region includes Automation Services products.

OLDEST=*n* | NEWEST=*n*

Indicates that, rather than emptying the table completely, only the oldest or newest n entries are to be deleted. This option is useful for tables that are being used as caches.

Examples: &VARTABLE RESET

```
&VARTABLE RESET ID=MYTABLE
```

This example will reset the private table called MYTABLE.

```
&VARTABLE RESET ID=STABLE SCOPE=SYSTEM OLDEST=50
```

This example resets a table called STABLE, which is visible to all NCL procedures running in this product region. Only the oldest 50 entries are deleted.

Return Codes:

System variable &ZFDBK is set after an &VARTABLE RESET statement to indicate the result of the operation:

0

The table was reset successfully.

16

No table of this name exists in this scope.

20

SCOPE=AOM table in use at the exact time the deletion of the mirrored copy was attempted. The table remains allocated.

&ZFDBK value 20 is possible with AOM tables only. If this happens, the NCL procedure should delay for approximately one second and try again.

Note:

Syntax errors in the &VARTABLE RESET statement terminate the NCL procedure.

&WRITE

Writes the specified text.

```
&WRITE [ ALARM={ YES | NO } ]
       [ ALL={ YES | NO } ]
       [ COLOR=color | COLOUR=colour ]
       [ CR={ YES | NO } ]
       [ FF={ YES | NO } ]
       [ HLIGHT=hlight | HLITE=hlight ]
       [ INTENS={ HIGH | NORMAL } ]
       [ LF={ YES | NO } ]
       [ LOG={ YES | NO } ]
       [ { MDO=mdoname |
           ARGS [ RANGE=(start, end) ] |
           VARS={ var | ( var1, var2, ..., varn) } } ]
       [ TYPE={ REQUEST | RESPONSE } ]
       [ MON={ YES | NO } ]
       [ MSGCODE=xx ]
       [ NRD={ YES | NO | OPER } ]
       [ RC={ (n,n,...,n) | ALL | NONE } ]
       [ SCAN={ YES | NO } ]
       [ TERM={ YES | NO } ]
       [ USERID=userid | LUNAME=luname | NCLID=nclid | SERVER=servername ]

       [ AOM={ YES | NO } ]
       [ AOMAUTH=[ YES | NO ]
       [ AOMID=identifier ]
       [ AOMJOBID=jobid ]
       [ AOMJOBNM=jobname ]
       [ AOMMINOR={ YES | NO } ]
       [ AOMMSGID=msgid ]
       [ AOMMSGLV={ IN | msglevel } ]
       [ AOMRC= | ROUTCDE= | RC={ 2 | NONE | ALL | list } ]
       [ AOMSOS={ OS | VM } ]
       [ AOMTIME=hhmmss ]
       [ AOMTYPE={ WTO | WTOR | MSG } ]
       [ AOMUFLGS=nn | AOMUFLGn={ YES | NO } ]
       [ AOMUSERI=userid ]
       [ AOMUSERN=usernode ]

       [ DATA=message-text ]
```

The &WRITE verb lets an NCL procedure issue a message. By default, the message is sent to the environment in which the NCL procedure is executing.

Several operands allow the message to be sent to MONITOR class OCS users, all OCS users, and so on. Other operands allow setting of message attributes, such as color and alarm. Automation Services adds several additional operands, that allow assignment of AOM routing options, and AOM attributes.

Operands:

The following operands are available to all users for message attribute assignment and message routing. AOM-specific operands follow these.

ALARM={ YES | NO }

Specifies whether the message is to ring the terminal alarm when displayed on an OCS window.

ALL={ YES | NO }

specifies that the message is to be written to all OCS users and dependent environments profiled to receive general broadcast messages.

COLOR=*color* | COLOUR=*colour*

The color in which the message is to be displayed. This is ignored if the terminal does not support extended color data streams. Color must be one of the following:

RED GREEN BLUE TURQUOISE YELLOW PINK WHITE NONE

CR={ YES | NO }

Indicates, for LU1 type terminals, whether a carriage return function is to be performed after writing the specified text. CR=YES, the default, indicates that carriage return is required, CR=NO that it is not.

FF={ YES | NO }

Indicates, for LU1 type terminals, whether a form feed function is to be performed. The form feed is actioned before any supplied text is written to the device. If no text is supplied, a blank line will be written after the form feed is actioned, unless LF=NO is also specified.

HLIGHT=*hlight* | HLITE=*hlite*

The extended high-lighting in which the message is to be displayed. This is ignored if the terminal does not support extended high-lighting data streams. *hlight* must be one of the following:

REVERSE USCORE BLINK NONE

INTENS={ HIGH | NORMAL }

Specifies whether the message is to be displayed on the terminal in high or normal intensity.

LF={ YES | NO }

Indicates, for LU1 type terminals, whether a line feed function is to be performed after writing the specified text. LF=YES, the default if the LF operand is not specified, indicates that a line feed function is required, LF=NO that it is not. LF=NO is used to create a strike-over mask so that secure data is entered on the terminal. Two or three successive &WRITE statements with LF=NO, followed by an &PROMPT statement, will effectively obliterate the entered data. The LF operand is ignored if used to write a message to a terminal that is not an LU1.

LOG={ YES | NO }

LOG=YES specifies that the message is to be written to the activity log.

{ MDO=*mdoname* |

ARGS [RANGE=(*start*, *end*)] |

VARS={ *var* | (*var*1, *var*2, ...,*varn*) }

MDO=*mdoname* specifies a mapped data object (MDO) to be delivered to the target destination. The MDO will be embedded in the $MSG user MDO and the $MSG.MAPNAME element set to the stem MAP name.

Specifying VARS or ARGS results in a $NCL MDO being built and delivered in the $MSG MDO, containing the named variables or arguments.

Notes: The MDO, VARS, and ARGS operands are mutually exclusive.

RANGE=(*start*, *end*) is specified with ARGS to denote an argument range.

TYPE={ REQUEST | RESPONSE }

This operand allows messages to be written to the target's request queue or response queue. The default is the response queue. For TYPE=REQUEST, the target must be another NCL procedure.

MON={ YES | NO }

Specifies that the message is to be written to all OCS users and dependent environments profiled to receive Monitor class messages.

MSGCODE=*xx*

Supplies a two digit hex message delivery code. This value is used as an 8-bit mask matched against the profile of possible receiving environments. Its use implies ALL=YES as the default generic delivery option but is overridden by supplying other options such MON=YES, FTS=YES and so on. An OR comparison is used and delivery is initiated when at least one or more bits are matched. Thus the value is equivalent to an 8-bit routing code.

The message delivery code facility allows the installation to develop and control delivery of messages in a variety of classes designed to meet the needs of the installation.

NRD={ YES | NO | OPER }

Specifies whether the message is to be classified as non-roll delete, that is, whether it will remain on the operator's window until deleted rather that being rolled over by other messages as they arrive. If NRD=NO is coded, or allowed to default, the message will be a standard roll-delete message.

If NRD=YES is coded, the message is assigned a delete operator message identifier or DOM ID. The value of the DOM ID is returned on completion of the &WRITE in the &ZDOMID system variable. If the message is to be deleted at some future time, this DOM ID must be specified as an operand on a &NRDDEL statement. The NRD message will also be deleted when the NCL process that issued the &WRITE statement terminates.

If NRD=OPER is coded the message is classified as non-roll delete but no DOM ID is assigned to it and it will never be deleted by any means except NRD cursor deletion - even when the originating process terminates.

RC={ (*n*,*n*,...,*n*) | ALL | NONE }

Specifies the routing codes to be used to determine the delivery of the message. Use of this operand is supported only if products that include Automation Services are installed. Receivers must have a match on any of the nominated route codes (*n*) to qualify for receipt of the message. RC=NONE will deliver the message to all AOM receivers who have at least one routing code active.

SCAN={ YES | NO }

Indicates that the message can contain strings of characters, delimited by @s, meaning that the delimited string should be highlighted at the terminal. A scan of the message takes place and the @s removed, being replaced by the appropriate attribute settings to turn highlighting on and off at the terminal.

TERM={ YES | NO }

TERM=YES specifies that the message is to be written to the owning environment. In the case of an NCL process executing directly from OCS the message will appear on the user's OCS window. Line messages issued under a full-screen process, for example, User Services, will not be displayed until the full screen process terminates.

Messages issued from system procedure environments, such as LOGPROC, AOMPROC, PPOPROC and so on. will be treated as monitor class messages and will be generically delivered with an identifying character prefixing the text. For example, messages from LOGPROC are delivered to all monitor receivers with L as the prefix.

By default, messages issued from BSYS and BLOG background environments are directed to the log. Messages from BMON are treated as specifying MON=YES.

TERM=NO specifies that the message will not be delivered to the target response queue.

TERM=NO is mutually exclusive with the SERVER, USERID, LUNAME and NCLID operands.

USERID=*userid*

Specifies that the message is to be written to the specified user ID only. On completion, &RETCODE is set.

LUNAME=*luname*

Indicates the node name of the terminal to which the message is to be sent. The terminal is either an LU1 terminal or a terminal with an OCS window(including the CONSOLE device).

NCLID=*nclid*

Specifies that the message is to be delivered to a dependent queue of the nominated process.

SERVER=*servername*

This operand allows the messages to be directed at a registered server procedure.

DATA=*message-text*

The text of the message to be displayed at the terminal, written to an LU1 device or written to the activity log. Normal variable substitution will be performed before sending the message. Text is in upper and lower case. If no text is supplied, a blank line will be displayed or written (unless LF=NO is specified for LU1). DATA can only be specified as the last keyword on the statement since the data string is regarded as being everything to the right of the DATA= keyword to the end of the statement.

These are the AOM-specific operands and are available only in regions that include Automation Services products. The preceding operands are available to all users.

Note: Where defaults are shown for AOM attributes, they apply only if any AOM attributes are specified. If no AOM attributes are specified, no AOM attributes are carried with the message.

AOM={ YES | NO }

Indicates whether the message is to be routed to all authorized AOM message receivers or not.

AOM=NO (the default) means no AOM specific delivery will take place. Other &WRITE operands such as MON= can cause delivery to more than one user. In this case, other AOM operands is coded, to give the message AOM attributes without causing automatic AOM delivery.

AOM=YES will cause the message to be delivered to authorized AOM receivers.

If no AOM attributes are specified, all AOM receivers will receive the message irrespective of profiled routing codes and AOM message levels. If any other AOM attributes are specified, the message will be delivered to AOM receivers based on user profile routing codes and message levels (that is at least one message level in common, and at least one route code in common, except that BC message level bypasses routing codes, and if the message has ROUTCDE=NONE, then that matches any user profile ROUTCDE except NONE).

A prefix of A is attached to all AOM messages produced by &WRITE, except those sourced by NCL procedures in the AOMP environment that also have AOM attributes specified.

AOMAUTH={ YES | NO }

Allows setting of the AOM authorized issuer attribute. This attributes indicates whether the original issuer of the message (WTO) was MVS-authorized. This attribute is available after &MSGREAD or &INTREAD in the &ZMAOMAU system variable.

AOMID=*identifier*

Assigns the AOM ID attribute. identifier must be null, or from 1 to 12 characters in length. This value is available after &MSGREAD or &INTREAD in the &ZMAOMID system variable.

AOMJOBID=*jobid*

Assigns the AOM JOBID attribute. *jobid* must consist of the job type, either J, S, or T, and a 1 to 5 digit number. This value is available after &MSGREAD or &INTREAD in the &ZMAOMJI system variable.

This attribute implies a source operating system of OS. Thus, it is invalid with any VM attributes or AOMSOS=VM.

AOMOJBNM=*jobname*

Assigns the AOM JOBNAME attribute. *jobname* must be null, or from 1 to 8 characters. This value is available after &MSGREAD or &INTREAD in the &ZMAOMJN system variable.

This attribute implies a source operating system of OS. Thus, it is invalid with any VM attributes or AOMSOS=VM.

AOMMINOR={ YES | NO }

Indicates whether this message is a minor line. Minor lines are not subject to AOM prefixing. Normally, this is used for multi-line displays, where only the first line needs AOM prefixes. The value of this attribute is available after &MSGREAD or &INTREAD in the &ZMAOMMIN system variable.

This attribute implies a source operating system of OS. Thus, it is invalid with any VM attributes or AOMSOS=VM.

AOMMSGID=*msgid*

Assigns the AOM message ID attribute. msgid must be null, or from 1 to 12 characters. This value is available after &MSGREAD or &INTREAD in the &ZMAOMMID system variable.

AOMMSGLV={ IN | *msglevel* }

Assigns the AOM message level attribute. The default, INformational, only applies if any AOM attributes are specified.

If AOM=YES is also specified, the AOM message level is used to determine which authorized AOM receivers will receive a copy of the message.

AOMRC= | ROUTCDE= | RC={ 2 | NONE | ALL | *list* }

Allows specification of a list of routing codes. Routing code values are from 1 to 128. The default, 2, only applies if any AOM attributes are specified.

If AOM=YES is also specified, these routing codes will be used to determine which AOM receivers will receive a copy of the message. Receivers with at least one routing code in common, or, if ROUTCDE=NONE is specified, receivers with at least one routing code, receive the message (subject to MSGLEVEL screening).

A single value is specified as AOMRC=*n*. Multiple values, and ranges is specified as:

- AOMRC=(4,13,27)
- AOMRC=(1-5,16,40-55)

ROUTCDE and RC are alternative spellings of this parameter.

The value of this operand is available in the &ZMAOMRC system variable after &INTREAD or &MSGREAD.

AOMSOS={ OS | VM }

Allows explicit setting of the AOM source operating system attribute. The default is the operating system that this product region is executing under, unless an AOM operand that implies a specific operating system is specified.

Attributes that imply a specific source operating system are marked. Use of these operands must be consistent with the value specified for AOMSOS if coded.

The value of this operand is available after &MSGREAD or &INTREAD in the &ZMAOMSOS system variable.

AOMTIME=*hhmmss*

Allows a specific AOM message time to be set. The default, if this parameter is omitted, is the current time. *hhmmss* must be 6 digits, in the range 000001 to 240000.

AOMTYPE={ WTO | WTOR | MSG }

Allows setting of a specific AOM message type. The default value for this operand depends on the operating system, or specified AOMSOS value. If z/OS, then AOMTYPE=WTO is assumed. If z/VM, AOMTYPE=MSG is assumed.

The value of this attribute is available after &MSGREAD or &INTREAD in the &ZMAOMTYP system variable.

Specification of this attribute can imply a specific source operating system. This can cause an error if incompatible operating system related attributes are specified.

AOMUFLGS=*nn* | AOMUFLG*n*={ YES | NO }

Allows setting the 8 AOM user flag attributes. All 8 is set as a hexadecimal value, using the AOMUFLAGS=*nn* operand, or the flags is set individually, using AOMUFLGn=YES/NO.

The values of these flags are available after &INTREAD or &MSGREAD in the &ZMAOMUFn system variables.

AOMUSERI=*userid*

Assigns the AOM USERID attribute. *userid* must be null, or from 1 to 8 characters. This value is available after &MSGREAD or &INTREAD in the &ZMAOMUI system variable.

This attribute implies a source operating system of VM. Thus, it is invalid with any VM attributes or AOMSOS=OS.

AOMUSERN=*usernode*

Assigns the AOM USERNODE attribute. *usernode* must be null, or from 1 to 8 characters. This value is available after &MSGREAD or &INTREAD in the &ZMAOMUN system variable.

This attribute implies a source operating system of VM. Thus, it is invalid with any VM attributes or AOMSOS=OS.

DATA=*message-text*

The text of the message to issue. Maximum length of the message is 256 characters. This is a standard &WRITE operand.

Examples: &WRITE

```
&WRITE TEST MESSAGE

&WRITE ALARM=YES COLOR=RED HLITE=REVERSE DATA=ring the +
    alarm

&WRITE MON=YES LOG=YES DATA=Message to all Monitors and +
    to the log.

&WRITE LOG=YES TERM=NO DATA=message to the log only

&WRITE FF=YES LUNAME=PRINTER1 DATA=Network Report ----- +
    Page No: &PAGE

&WRITE FF=YES LF=NO

&WRITE SCAN=YES DATA=RELOAD@NCP23@NOW
```

AOM Examples:

```
&WRITE COLOR=RED AOM=YES DATA=AOM IS ALIVE

&WRITE USERID=U1234 AOMJOBNM=JOB1 DATA=job1 related msg
```

Note: When using &WRITE it is good practice to always include DATA= before the message text to differentiate between the message text and other operands. For example,

```
&WRITE DATA=RC=(&RETCODE)
```

rather than

```
&WRITE RC=(&RETCODE)
```

as RC would be interpreted as the ROUTCDE operand in the latter.

Notes:

Messages written using &WRITE ALARM=YES and &WRITE MON=YES are by default logged unless overridden by the LOG= operand.

Generic delivery options such as MON=YES, FTS=YES is used to broadcast to OCS operators and dependent environments. Dependent environments are by default set to a profile that disallows any generic message receipt. Use of &INTCMD PROFILE MONMSG=YES, and so on, allows the dependent environment to be used for unsolicited message receipt.

Messages generated to primary environments is processed by a MSGPROC before further processing or display. The MSGPROC associated with the primary environments of system procedures, for example, LOGPROC, is used to process messages before their generic delivery as monitor messages with an identifying prefix.

The use of ALL=YES either explicitly or implicitly via MSGCODE= without other generic delivery options is equivalent to the use of the MSG ALL command. The receipt of these messages is controlled by the PROFILE MSG=Y | N of the receiving environment.

If variables are used to provide the message text, the contents of the variables is hexadecimal data.

Alarm, high-intensity, NRD, and routing code options are ignored for LU1 type terminals.

&WRITE with AOM operands is similar to &AOMALERT ROUTE=MSG. However, &WRITE allows sending the message to a specific user, by using the LUNAME, USERID, and NCLID operands. &AOMALERT does not allow this.

If NRD=YES is specified, the system variable &ZDOMID contains the assigned message DOMID, in the format domain/n. This value is used in a subsequent &NRDDEL verb to delete the message.

Note: See also the PROFILE command used to control an environment's message receipt options in the Online Help.

Return Codes:

When one of the target operands of the &WRITE verb is specified (that is, LUNAME, NCLID, SERVER, or USERID), &RETCODE is set as follows:

0

The message was delivered to the specified destination.

4

The target was a closed OCS window.

8

The LUNAME specified was not available.

12

The USERID or LUNAME specified was not in OCS mode.

16

The target was not found in the System Services domain.

24

The destination procedure was at queue limit, or the storage limit was reached.

28

The target for a TYPE=REQ request was not an NCL procedure.

32

The contents of the source MDO exceeds the maximum allowable size.

&WTO

This verb issues an MVS or VM WTO.

The verb lets an NCL procedure issue an MVS WTO. Optional parameters allow specification of routing and descriptor codes, special MCS flags, and system console ID.

In z/VM, the &WTO verb writes a message to the GCS console.

This verb has the following format:

```
&WTO [ CONSOLE={ nn | name } ]
     [ DESC={ NONE | list } ]
     [ MCSFLAG=( [ RESP ] [ ,REPLY ] [ ,BRDCST ] [ ,HRDCPY ] ) ]
     [ ROUTCDE={ NONE | list } ]
     [ LINETYPE={ NO | C | L | D | DE | E } ]
       DATA=message text
```

Operands:

CONSOLE={ *nn* | *name* }

Specifies the ID of the system console to which to queue the message. The operand uses the REG0 MCSFLAG to queue the WTO conditionally to the nominated console. If this parameter is used with the ROUTCDE parameter, then the message is queued to consoles based on routing code and the specified console ID.

On z/OS, you can use CONSOLE=*name* (*name* being a valid console name).

DESC={ NONE | list }

Allows specification of a list of MVS descriptor codes. The list is a single code, from 1 to 16, or a list of codes and ranges, as shown:

DESC=5 DESC=(1,7,8-12)

If no descriptor codes are specified, MVS typically supplies a default of 7.

Some MVS descriptor codes (for example, 1, 2, or 11) cause the message to become Non-Roll-Delete (NRD). The message remains on MVS (and system) consoles until deleted by an MVS DOM (which the &DOM verb produces). Take care when using these descriptor codes, as overuse could lead to system console buffer shortages.

MCSFLAG=([RESP] [,REPLY] [,BRDCST] [,HRDCPY])

Allows specification of MCSFLAG values. If only a single value is required, MCSFLAG=*flag* is used. If more than one is required, the list must be enclosed in brackets and separated by commas, for example, MCSFLAG=(*flag*,*flag*...).

The allowable MCSFLAG options are:

RESP

Indicates that this WTO is a command response.

REPLY

Indicates that this WTO is an echo of a reply to a WTOR.

BRDCST

Broadcasts this WTO to all consoles, irrespective of routing codes.

HRDCPY

Indicates that this WTO is to be a hardcopy-only WTO; that is, to be logged to SYSLOG, but not sent to any console.

The only other MCSFLAG values that &WTO uses are the ROUTCDE/DESC present flag, and the REG0 flag, set if CONSOLE= is specified.

ROUTCDE={ NONE | *list* }

Allows specification of a list of MVS routing codes. ROUTCDE values are from 1 to 128.

A single value is specified as ROUTCDE=n. Multiple values and ranges are specified as follows:

```
ROUTCDE=(4,13,27)
ROUTCDE=(1-5,16,40-55)
```

LINETYPE={ NO| C | L | D | DE | E }

Allows specification of a multiline WTO.

The operand has the following values:

- NO generates a single-line WTO.
- C (Control) starts the multiline WTO.
- L (Label) specifies a label line.
- D (Data) specifies the body of the multiline WTO.
- DE specifies a data line that also ends the multiline WTO.
- E ends the multiline WTO when no more data exists.

Values other than NO allow you to build up a multiline WTO, one line at a time. The multiline WTO is complete when a data-end (DE) or end (E) line is sent. &ZDOMID contains the same value after each line of a multiline WTO. You can write a single-line WTO during generation of a multiline WTO.

All other &WTO operands except DATA are ignored when specified on the second and subsequent lines of a multiline WTO.

The DATA operand is not mandatory for LINETYPE=E.

&RETCODE is set to the WTO macro return code after an &WTO when LINETYPE is other than NO.

Default: NO

DATA=*message text*

Specifies the text of the message to WTO. Maximum length of the message is 126 characters.

Examples: & WTO

```
&WTO DATA=AOM IS HERE!!!!
&WTO CONSOLE=1 DESC=1 DATA=HELP!!!!
&WTO ROUTCDE=(1,2,11) +
        DATA=Note: Configuration file updated.
```

Notes:

When using &WTO, consider these recommendations:

- Always provide a message identifier at the start of the message. Use this identifier to establish some connection with the issuer of the &WTO.
- Avoid using descriptor codes 1, 2, or 11. These descriptor codes cause the messages to become Non-Roll-Delete (NRD) and can lead to excessive numbers of NRD messages being displayed.
- Excessive use of &WTO can lead to console buffer shortages.

Following &WTO, the system variable &ZDOMID contains the MVS-assigned message identification or DOMID, as an eight-hexadecimal digit value. This value is used in a subsequent &DOM verb to delete the message. This is important if descriptor codes that make the message NRD are used (typically 1, 2, and 11).

The &WTO verb generates an MVS WTO. Thus, the message is seen and processed like any other WTO message.

While z/VM supports the WTO macro, only the text parameter is used.

The &RETCODE variable is set to the MVS WTO return code when a multiline WTO is issued.

Note: For more information about MCSFLAG values, see the IBM publication *MVS Programming Authorized Assembler Services Reference*.

More information:

&WTOR (see page 710)
&DOM (see page 303)

&WTOR

Issues an MVS WTOR and waits for a reply.

```
&WTOR[ CONSOLE={ nn | name } ]
     [ RLEN={ 119 | nn } ]
     [ MCSFLAG=([ RESP ][ ,REPLY ][ ,BRDCST ][ ,HRDCPY ] )]
     [ ROUTCDE={ NONE | list } ]
     [ WAIT={ YES | nn.nn } ]
     { [ VARS=prefix* [ RANGE=(start,end) ] ] |
       [ VARS={ name | (var-name-list ) } ] |
       [ STRING ] | STRING=(var-name-list)
       [ ARGS [ RANGE=(start,end) ] ] }
      DATA=message text
```

The &WTOR verb lets an NCL procedure issue an MVS WTOR. Optional parameters allow specification of routing and descriptor codes, special MCS flags, and system console ID. By default, the procedure waits indefinitely for a reply. However, a maximum wait time is specified.

The &WTOR verb is not supported on z/VM.

Operands:

CONSOLE={ *nn* | *name* }

Indicates a system console ID that the WTOR is to be queued to. It uses the REG0 MCSFLAG to conditionally queue the WTOR to the nominated console. If this parameter is used with the ROUTCDE parameter, the WTOR will be queued to consoles based on routing code as well as the specified console ID.

RLEN={ 119 | *nn* }

Indicates the maximum reply length permitted. The default, and maximum (MVS-imposed) is 119 characters. Otherwise, specify a value of 1 to 119.

MCSFLAG=([RESP] [,REPLY] [,BRDCST] [,HRDCPY])

Allows specification of some MCSFLAG values. If only a single value is required, *MCSFLAG=flag* is used. If more than one is required, MCSFLAG=(*flag*,*flag*...) is used.

Note: Some MCSFLAG values are not meaningful for &WTOR for example, HRDCPY.

The allowable MCSFLAG options are:

RESP

Indicates that this WTOR is a command response.

REPLY

Indicates that this WTOR is an echo of a reply to a WTOR.

BRDCST

Indicates that this WTOR is to be broadcast to all consoles, irrespective of routing codes, and so on.

HRDCPY

Indicates that this WTOR is to be a hardcopy-only WTOR that is, is to be logged to SYSLOG, but not sent to any console.

The only other MCSFLAG values that &WTOR uses are the ROUTCDE/DESC present flag, and the REG0 flag, set if CONSOLE= is specified.

ROUTCDE={ NONE | *list* }

Allows specification of a list of MVS routing codes.

A single value is specified as ROUTCDE=*n*. Multiple values, and ranges is specified as:

```
ROUTCDE=(4,13,27)ROUTCDE=(1-5,16,40-55)
```

Values: 1 to 128

WAIT={ YES | *nn.nn* }

Specifies how long the NCL procedure is to wait for a reply to the WTOR.

WAIT=YES, (the default) will cause the procedure to be suspended indefinitely, pending a reply.

WAIT=*nn.nn* will cause the procedure to wait for up to *nn* seconds for a reply. If no reply is provided by this time, the outstanding WTOR is canceled (DOM issued) and &RETCODE will be set to 4 and the procedure resumes execution. The valid range is 0.01 to 9999.99.

If a reply is provided, &RETCODE is set to 0.

WAIT=0 or WAIT=NO is not supported on &WTOR, as it would be senseless to issue and immediately cancel the WTOR.

Regardless of the WAIT value used, if the procedure is flushed for any reason, the WTOR is canceled.

VARS=

Specifies that the reply is to be tokenized into the nominated variables before control is returned to the procedure. Each word of the reply will be tokenized into the nominated variables from left to right. If insufficient variables are provided, some data will be lost. Excess variables will be set to a null value. The format of the operands that is coded with VARS= are described below.

prefix*

Denotes that variables will be generated automatically during the tokenization process, and that the variable names will be prefix1 .. prefix2 and so on. The RANGE= operand is specified to indicate a starting and ending suffix number. Prefix* cannot be used in conjunction with other variable names.

name

The name of a variable, excluding the ampersand (&).

name(n)

As above, but n denotes the length of the data that is to be placed in the variable.

***(*n*)**

Denotes a skip operation, where n represents the number of units to be skipped during the tokenization process. On VARS= statements n denotes 'skip this number of words'. An asterisk (*) by itself is the same as *(1).

STRING

Specifies that no tokenization is to be performed. The entire text of the reply is to be treated as a single string and returned to the procedure in &1. No variables is nominated if STRING is specified.

STRING=(*var-name-list*)

Specifies that the reply is to be segmented into substrings and placed in the nominated receiving variables. For example STRING=(A(5),B(10)) indicates that the first 5 characters are placed in &A and the next 10 characters in &B.

ARGS

Denotes that the message received will be tokenized and placed word by word into automatically generated variables of the from &1 through &n, depending on how many are required to hold the message. The RANGE= operand is coded to designate a start number and optionally, an end number, which delimits the number of variables that will be generated.

DATA=*message text*

The text of the message to WTOR. Maximum length of the message is 122 characters.

Examples: &WTOR

```
&WTOR ARGS DATA=Reply with configuration parameters
&WTOR WAIT=10 CONSOLE=1 ARGS DATA=Reply with config. parameters
&IF &RETCODE=4 &THEN &WTO CONSOLE=1 +
   DATA=No parameters received within time limit
```

Notes:

When using &WTOR, consider these recommendations:

- Always provide a message identifier at the start of the message. This identifier should establish some connection with the issuer of the &WTOR.
- WTOR messages are always treated as Non-Roll-Delete. For this reason, keep to a minimum the number of WTOR messages outstanding at any one time.

No DOMID is returned after &WTOR, as the NCL procedure is always suspended until it is replied to, or until the WAIT interval is exceeded, in which case the WTOR is automatically deleted.

Careless use of &WTOR can tie up many system console buffers. WTOR messages are always Non-Roll-Delete (NRD).

The &WTOR verb generates an MVS WTOR. Thus, the message is seen and processed like any other WTOR message.

More information:

&WTO (see page 706)

&ZAMCHECK

Indicates whether support is enabled in your product region for a specified access method.

```
&ZAMCHECK accessmethod
```

&ZAMCHECK is a built-in function and must be used to the right of an assignment statement.

&ZAMCHECK is used to verify support for VTAM and XNF access methods.

The term access method refers to a communication program used to communicate with terminals, other product regions, or another application.

Operands:

accessmethod

The value of this field is VTAM or XNF. The value returned is YES or NO. If a name other than VTAM or XNF is specified, the result is NO.

Examples:

```
&A = &ZAMCHECK VTAM
&IF &A = YES &THEN &DO
   -SYSPARMS PPOACBNM=NMPPO1
   -PPO START
   &DOEND
```

More information:

Summary Table (see page 29)

&ZFEATURE

Returns an indication of the availability of one or more features.

```
&ZFEATURE feature1 [ feature2 .... featuren ]
```

&ZFEATURE returns a value of YES or NO to indicate whether your product region is configured with the single nominated feature or all the nominated features. This allows NCL procedures to include feature-dependent code that can or cannot be activated, depending on the presence of the feature.

Operands:

***feature*1 [*feature*2.... *featuren*]**

The names of one or more features. If the region is configured with all the nominated features, the built-in function returns a value of YES in the target variable. If any of the nominated features is not present in the system, &ZFEATURE returns a value of NO in the target variable.

Examples: &ZFEATURE

```
&A = &ZFEATURE TCPIP
&GOSUB .&A
.YES -* TCPIP feature is present, so use its facilities
 :
.NO -* TCPIP feature is not present.
```

Note: &ZFEATURE is used to test for the features as listed in the description of the PROD JCL parameter in the *Reference Guide*.

More information:

Summary Table (see page 29)

&ZNCLKWD

Returns a value indicating whether a given string is an NCL keyword.

```
&ZNCLKWD string
```

Provides a means of testing whether a given string is an NCL keyword.

&ZNCLKWD is a built-in function and must be used to the right of an assignment statement.

The specified string is tested and one of the following values is assigned to the variable to the left of the assignment statement.

YES

The string is an NCL keyword.

NO

The string is not an NCL keyword.

Operands:

string

The string to be tested.

Examples:

```
&A = &ZNCLKWD &INPUT
&IF &A EQ YES &THEN +
   &GOTO .INVALID
```

More information:

Summary Table (see page 29)

&ZOSCHK

Indicates whether support is enabled in your product region for a specified operating system (or family of operating systems) or capability.

```
&ZOSCHK { ANY | ALL } name [ name ... ]
```

&ZOSCHK is a built-in function and must be used to the right of an assignment statement.

The result is Boolean (0 or 1):

- 0 means that the test failed.
- 1 means that the test succeeded.

Operands:

{ ANY | ALL }

This operand is mandatory. Specifying ANY means that, if any of the following tests is true, then the result is true (1 is returned).

Specifying ALL means that all of the following tests must be true for the result to be true.

***name* [*name* ...]**

Each name that you specify here indicates an operating system (or family of operating systems) or capability that you want to test for.

The value of this field is any of the following:

GENOS

Any OS system (for example, z/OS or MSP)

GENVM

Any VM system

GENVOS

Any VOS3 system

IBM

Any IBM system

IBM_OS

Any IBM OS system

IBM_VM

Any IBM VM system

NONIBM

Any non-IBM system

FUJITSU

Any Fujitsu system

FUJITSU_OS

Any Fujitsu OS system

FUJITSU_FSP

Any Fujitsu FSP system

HITACHI

Any Hitachi system

HITACHI_OS

Any Hitachi OS system

OS390>

z/OS and OS/390

OS/390>

z/OS and OS/390

ZOS

z/OS only

Z/OS

z/OS only

ZOS>

z/OS or later Z/OS> z/OS or later

MSP

MSP only

MSP>

MSP or later (up to and beyond MSP/EX)

MSPAE

MSP/AE only

MSP/AE

MSP/AE only

MSPAE>

MSP/AE or later (up to and beyond MSP/EX)

MSP/AE>

MSP/AE or later (up to and beyond MSP/EX)

MSPEX

MSP/EX only

MSP/EX

MSP/EX only

MSPEX>

MSP/EX or later

MSP/EX>

MSP/EX or later

VOS3

Hitachi VOS3 only

VM

Any VM

GCS>

GCS or later (up to and beyond VM/ESA)

VMESA>

VM/ESA or later X

VM/ESA>

VM/ESA or later X

EXTMCS

System supports EXTMCS consoles

31BIT

System supports 31-bit addressing

XMS

System supports basic cross-memory services

ESAXMS

System supports ESA-style cross-memory services

SMF77

System supports SMF record 119 (x'77') (z/OS 1.2 and later)

IPV6

System supports IPv6 (z/OS 1.4 and later)

Notes:

The result is boolean (0 or 1):

- 0 means that the test failed.
- 1 means that the test succeeded. The remaining operands (at least one; as many as desired) identify a specific operating system or capability to test for. If an unrecognized name is used, then the test is treated as false.

Standard NCL function semantics apply. This function is used only on the right of an assignment statement.

This function returns 0 or 1, not YES or NO.

Examples: &ZOSCHK

```
&RESULT = &ZOSCHK ANY GENOS MSP
&RESULT = &ZOSCHK ALL IBM VM
```

More information:

Summary Table (see page 29)

&ZPSKIP

Sets a new string as the active panel skip data. Optionally, also restarts processing from the primary menu.

```
&ZPSKIP [ string | =string ]
```

Your product region supports the ability to perform menu jumps or panel skips as a means of abbreviating panel navigation. Panel skips is used to move rapidly from one panel display to another without viewing each individual panel which would normally be displayed. The panel skip data is a variable length string wherein data for each panel is delineated by the use of a period (.). This data is normally used to automatically satisfy &PANEL statements. The &ZPSKIP verb is used to set a new panel skip string. In addition, the use of a leading equal sign (=), is used to restart processing from the primary menu.

Operands:

=

An optional automatic trigger to flush current processing and re-invoke the Primary Menu. The associated panel skip string supplied will then be used to satisfy further panel processing.

string

The data to be stored as the current panel skip string. Periods are used to delineate the data that is applicable to each panel. In addition, semicolons is used as field separators to supply data to multiple fields on a panel. Neither separator is used as part of the data, or is returned to the panel or &ZPSKIP system variable.

A null value is used to reset any panel skip data.

Examples: &ZPSKIP

```
&ZPSKIP =U.4  -* jump to user services and then option 4
&ZPSKIP         -* stop any further panel skipping
&ZPSKIP &REMCMD -* set new string for skipping forward
&ZPSKIP =M.TSO  -* jump to MAI session TSO
```

An NCL menu procedure should always support forward panel skipping. The &ZPSKIP verb should be used to set forward skipping. For example:

```
&SELECT = &SELSTR (.) &CMD -* extract first option
&REMOPT = &REMSTR (.) &CMD -* extract remainder of data
&ZPSKIP &REMOPT  -* set forward skipping
```

Panel skipping is terminated by a panel display that signifies an error. If the alarm is sounded or #ERRFLD highlighting is used, then panel skip data is not used to satisfy the panel and the panel skip string is set to a null value.

Note: For more information about the &ZPSKIP and &ZPSKPSTR system variables, see the *Network Control Language Programming Guide*.

More information:

Summary Table (see page 29)

&ZQUOTE/&ZQUOTE2

Places quotes around a string and places it into a variable.

```
&ZQUOTE text ...  text
&ZQUOTE2 text ...  text
```

&ZQUOTE provides a means of quoting a string using single or double quotes. &ZQUOTE2 also quotes data, but uses only single quotes. For the rules used, see Notes.

&ZQUOTE and &ZQUOTE2 are built-in functions and must be used to the right of an assignment statement.

Operands:

text

The text to be quoted

Return Codes:

&ZFDBK is set to indicate the success or failure of the quoting procedure:

0

Quote successful

4

The data cannot be quoted. This occurs if the string for quoting is too long to fit within an NCL variable. The input is assigned to the output variable without change.

Examples:

```
&A=&ZQUOTE ABC DEF  -* results in &A set to “ABC DEF”
&B=&ZQUOTE say “Hello”  -* &B is set to 'say “Hello”'
&C=&ZQUOTE say “G'day”  -* &C is set to 'say “G''day”'
&D=&ZQUOTE2 say “G'day”  -* &D is set to 'say “G”day”'
```

Notes:

&ZQUOTE and &ZQUOTE2 is used to quote a string which is later analyzed by &SETVARS.

&ZQUOTE quotes the text according to the following rules:

- If the data contains no double quotes (”), a double quote is placed at each end of the text.
- If the data contains no single quotes, a single quote is placed at each end of the text. Otherwise a single quote is placed at each end and every embedded single quote is replaced by two single quotes
- If the data is null, the nominated variable is set to null.

&ZQUOTE2 uses the following rules:

- A single quote is placed at each end of the text and every embedded single quote is replaced by two single quotes
- If the data is null, a null quoted string is returned (that is, '').

More information:

&SETVARS (see page 604)
&ZUNQUOTE (see page 731)

&ZSHRINK

Removes leading and trailing spaces and reduces multiple spaces within a string to one space.

```
&ZSHRINK text ... text
```

&ZSHRINK is a built-in function and must be used to the right of an assignment statement.

&ZSHRINK provides a means of removing spaces from the beginning or end of a string, or removing multiple spaces from within a string.

Operands:

text

The text to be shrunk.

Notes:

The supplied text is returned with spaces removed according to the following rules:

- Leading and trailing spaces are removed.
- Two or more adjacent spaces are compressed into one space.
- If the last two characters are a space followed by a period (.), the space is removed and the period placed following the last non-blank character.

Examples:

```
&A = &ZSHRINK THIS  IS A TEST .
sets &A to  THIS IS A TEST.
```

&ZSOCINFO

Obtains information about a specific socket owned by the process.

```
&ZSOCINFO socket_id infotype
```

&ZSOCINFO returns information according to the socket as identified by socket_id and the value of *infotype* as explained below.

Operands:

EXIST

Returns YES if the NCL process owns a socket with the specified socket ID or NO if the NCL process does not own a socket with the specified socket ID.

TYPE

Returns a character string representing the type of socket: TCP, TCPLISTEN, or UDP.

PORT

Returns the port number assigned to the socket by the local host.

ADDR

Returns the IP address assigned to the socket by the local host.

PEERPORT

Returns the port number of the peer host on a TCP connection.

For a UDP socket this is the port number last referenced by a SEND_TO or RECEIVE_FROM verb.

PEERADDR

Returns the IP address of the peer host on a TCP connection.

For a UDP socket this is the IP address last referenced by a SEND_TO or RECEIVE_FROM verb.

PEERNAME

Returns the name of the peer host on a TCP connection (where HOSTNAME was used to establish the connection).

RETCODE

Returns the return code from the last operation on this socket.

FDBK

Returns the reason code from the last operation on this socket.

VERRIN

Returns the vendor error information from the last operation on this socket.

ERRNO

Returns the error number value from the last operation on this socket.

BYTESIN

Returns the number of bytes of data received by this socket.

BYTESOUT

Returns the number of bytes of data sent by this socket.

Examples:

```
&STYPE = &ZSOCINFO &SOCK TYPE
&WRITE DATA=SOCKET &SOCK IS &STYPE
```

&ZSUBST

Returns a string with substituted data.

```
&ZSUBST subchar data
```

&ZSUBST provides a means of substituting data in a string or variable. Normally NCL substitution uses the ampersand (&) character to indicate the start of a variable name. You can use the &ZSUBST built-in function to substitute a string of data using a different character.

&ZSUBST is a built-in function and must be used to the right of an assignment statement.

Operands:

subchar

This is the substitution character. It is:

c

Any single character

C 'c'

Any single quoted character (including a blank)-double quotes is used

X 'xx'

A hexadecimal pair representing the wanted character

data

The data to substitute. The data is in a variable. It is the data after normal substitution that is then passed through a single substitution pass using the nominated character instead of the ampersand.

Examples: &ZSUBST

```
&A = PARIS
&TEST = @A
&RESULT = &ZSUBST @ &TEST -* &RESULT will contain PARIS
```

&ZSYSPARM

Obtains the value of a system parameter (SYSPARM).

```
&ZSYSPARM operand
```

NCL procedures can use the &ZSYSPARM built-in function to directly access system parameters which have been set using the SYSPARMS command. The value of the SYSPARMS operand specified on the built-in function is returned in the identical format to the output for the same operand in a SHOW SYSPARMS display. (For information about the SHOW SYSPARMS command, see the Online Help.)

Operands:

operand

The SYSPARMS operand for which the setting information is required.

Return Codes:

&ZSYSPARM sets the &ZFDBK system variable as follows: 0 SYSPARMS operand was valid

4

SYSPARMS operand not applicable to current operating system

8

SYSPARMS operand name is unknown to the system

Examples: &ZSYSPARM

```
&IPPORT = &ZSYSPARM IPPORT
```

Note: The operand name specified on the &ZSYSPARM built-in function must be an exact match for a known operand.

&ZTCPERDS

Returns the short message for a TCP/IP error code.

```
&ZTCPERDS n
```

Operands:

n

A valid error code number. If the number is not a recognized error code, the message, EUNKNOWN - UNKNOWN ERRNO is returned.

Examples: &ZTCPERDS

```
&B = &ZTCPERDS 40
```

sets the value of &B to the following:

```
N3AD01 - 40 - ENETUNREACH - DESTINATION NETWORK UNREACHABLE
```

If the number is not a recognized error number, &B is set to:

```
N3AD01 -xxx - EUNKNOWN - UNKNOWN ERRNO
```

where xxx is the error number supplied.

&ZTCPERNM

Returns the logical name of a TCP/IP error code.

```
&ZTCPERNM n
```

Operands:

n

A valid error code number. If the number is not a recognized error code, the message, EUNKNOWN is returned.

Examples: &ZTCPERNM

```
&A = &ZTCPERNM 40
```

sets the value of &A to ENETUNREACH. If the number is not a recognized error code, &A is set to EUNKNOWN.

&ZTCPINFO

Obtains information about the local host or TCP/IP vendor stack.

```
&ZTCPINFO infotype
```

&ZTCPINFO returns data according to the value of infotype as explained below.

Operands:

TYPE

Returns the vendor stack type as specified on the TCPIP START command TYPE=.

STATUS

Returns ACTIVE, STARTING, STOPPING, QUIESCING, or INACTIVE.

HOSTADDR

Returns the local host IP address.

HOSTNAME

Returns the local host alias name.

HOSTFULLNAME

Returns the local host full name.

INTERFACE

Returns descriptive interface information returned by the vendor interface.

CONNECTION

Returns connection information from the vendor interface.

SOLVE_DNR

Returns SOLVE Domain Name Resolver (DNR) information.

Examples: &ZTCPINFO

```
&INTF = &ZTCPINFO INTERFACE &WRITE DATA=TCPIP INTERFACE IS &INTF
```

Note: For more information, see the DNR and SHOW DNR command descriptions in the Online Help.

&ZTCPSUPP

Determines if a function is supported by the current TCP/IP vendor stack.

```
&ZTCPSUPP function
```

&ZTCPSUPP returns NO if the function is unknown or not supported, or if the TCPIP START command has never been issued. It returns YES if the function is supported.

Operands:

PING

Returns YES if &SOCKET PING is supported.

TRACEROUTE

Returns YES if &SOCKET TRACEROUTE is supported.

GETHOSTBYADDR

Returns YES if &SOCKET GETHOSTBYADDR is supported.

GETHOSTBYNAME

Returns YES if &SOCKET GETHOSTBYNAME is supported.

SOLVE_DNR

Returns YES if the SOLVE Domain Name Resolver (DNR) is supported.

Examples: &ZTCPSUPP

```
&CANPING = &ZTCPSUPP PING
&WRITE DATA=PING? &CANPING!
```

Note: For more information, see the DNR and SHOW DNR command descriptions in the Online Help.

More information:

&SOCKET PING (see page 631)
&SOCKET TRACEROUTE (see page 644)
&SOCKET GETHOSTBYADDR (see page 625)
&SOCKET GETHOSTBYNAME (see page 627)

&ZUNQUOTE

Removes one level of quotes from a string.

```
&ZUNQUOTE text ... text
```

&ZUNQUOTE is used to reverse the quoting procedures performed by &ZQUOTE. &ZUNQUOTE is a built-in function and must be used to the right of an assignment statement. If the input data is null, the nominated variable will be set to null.

Return Codes:

The system variable &ZFDBK is set to indicate the success or failure of the unquoting procedure:

0

Unquote was successful

4

Data error occurred. For example, the string is too long for one NCL variable. The input is copied to the output variable, without change.

Examples: &ZUNQUOTE

```
& A = &STR 'say “G''day”'
&B = &ZUNQUOTE &A           -* &B will set to say “G'day”
&C = &ZUNQUOTE &B           -* &C will be the same as &B since
-* there are no more levels of quoting
&D = &ZUNQUOTE 'say “G'day”'-* Will result in &ZFDBK=4
                            -* and &D set to 'say “G'day”'
```

Note: For a definition of a quoted string, see description of &ZQUOTE .

More information:

&ZQUOTE/&ZQUOTE2 (see page 722)

Chapter 3: System Variables

About System Variables

The following table is a list of the system variables with a brief description of their function.

The Feature/Component column indicates whether a specific feature or component must be included in the initialization parameters at region startup before you can use the variable.

Note: For more information about system initialization parameters, see the *Reference Guide*.

Note: If any of the following products are configured in the region, the internal Automation Services (AS) component is enabled: FT, NETSPY, OPSCICS, OPSOS, SNA, SNAAUTO, or TCPIP.

Name	Description	Feature/ Component
&ALLPARMS	A user variable that supplies a single string for all parameters specified when an NCL procedure is invoked	
&AOMACCT1–4	Four system variables that return, for some MVS-sourced messages, the first four accounting fields from the JOB statement	AS
&AOMALARM	Returns the alarm attribute for the current message	AS
&AOMASID	Returns the address space ID (ASID) that issued the current message	AS
&AOMATEXT	Returns the text of the current line of a message that has been delivered to AOMPROC	AS
&AOMAUTH	Indicates whether the issuer of a WTO/WTOR is authorized	AS
&AOMAUTO	Returns the value of the automation flag	AS
&AOMAUTOT	Returns the value of the automation token	AS
&AOMBC	Indicates whether the current message is a broadcast message	AS
&AOMCHAR1	Returns the screen character that indicates the status of operator console format messages	AS
&AOMCOLOR	Indicates the color attribute of the current message	AS
&AOMCONNM	Returns the Extended MCS console name	AS

Name	Description	Feature/ Component
&AOMDESC	Returns the descriptor codes assigned to the current message, in list format	AS
&AOMDHEX	Returns the descriptor codes assigned to the current message, in hexadecimal	AS
&AOMDMASK	Returns the descriptor codes assigned to the current message, in &MASKCHK format	AS
&AOMDOM	Indicates whether the current message is a Delete Operator Message notification (DOM-Notify)	AS
&AOMDOMID	Returns the Delete Operator Message (DOM) ID of the current message	AS
&AOMEVCLS	Returns the EVENT class value	AS
&AOMHLITE	Returns the highlight attribute for the current message	AS
&AOMID	Returns an ID assigned by the screening table to the current message or event	AS
&AOMIJOBN	Returns the job name of the address space that issued the WTO, WTOR, or EVENT	AS
&AOMINTEN	Returns the intensity attribute for the current message	AS
&AOMJOBCL	Returns the job class of the job that issued the WTO or WTOR	AS
&AOMJOBID	Returns the JES job number that issued the current message	AS
&AOMJOBNM	Returns the job name of the active address space that issued the current message	AS
&AOMJSTCB	Returns the hexadecimal address of the job step TCB that either issued the current WTO or WTOR, or owns the TCB that issued the message	AS
&AOMLDID	Returns the domain ID of the last handler of this message, event, or DOM-Notify	AS
&AOMLROUT	Returns the local routing option for the message or event as set by the screening table ROUTE or LCLROUTE operands	AS
&AOMLRSLT	Returns the eight LOOKUP results from screening, in &MASKCHK format	AS
&AOMLRSL1–8	Eight system variables that return the results of up to eight LOOKUP statements	AS
&AOMLTCTL	Indicates whether the current line of the current message is a control line	AS
&AOMLTDAT	Indicates whether the current line of the current message is a data line	AS
&AOMLTEND	Indicates whether the current message is an end line	AS

Name	Description	Feature/ Component
&AOMLTLAB	Indicates whether the current line of the current message is a label line	AS
&AOMMAJOR	Indicates whether the current line of the current message is a major line	AS
&AOMMHEX	Returns the MCS flags assigned to the current WTO or WTOR	AS
&AOMMINOR	Indicates whether the current line of the current message is a minor line	AS
&AOMMMASK	Returns the MCS flags assigned to the current message in &MASKCHK format	AS
&AOMMONIT	Indicates whether to deliver the current message to monitor class message receivers	AS
&AOMMPFSP	Indicates whether the Message Processing Facility (MPF) initially suppresses the current message	AS
&AOMMSGCD	Returns the message code assigned to this message	AS
&AOMMSGID	Returns the extracted message ID of the current message	AS
&AOMMSGLV	Returns the highest message level of the current message	AS
&AOMMVCON	Returns the ID of the system console to which the current message was routed	AS
&AOMMVSDL	Indicates whether the screening table has deleted the current message	AS
&AOMNMCON	Returns the console ID to which the current message was routed	AS
&AOMNMDOM	Returns the assigned DOMID associated with a DOM-notify message	AS
&AOMNMIN	Returns the number of minor lines in a multiline WTO	AS
&AOMNRD	Indicates whether to display the current message as a non-roll delete message on OCS consoles	AS
&AOMODID	Returns the domain ID of the system where the message originated, as set by the NMDID JCL parameter	AS
&AOMRCLAS	Returns the ISR remote classes, as set by the screening table, in MASKCHK format	AS
&AOMRCLS1–8	Eight system variables which provide the individual values of the eight AOM ISR remote classes for this message or event	AS
&AOMREISS	Returns the value YES if the current message was reissued on a JES3 GLOBAL processor, otherwise its value is NO	AS
&AOMRHEX	Returns the routing codes assigned to the current message, in hexadecimal	AS
&AOMRKEY	Returns the retrieval key attribute	AS

Name	Description	Feature/ Component
&AOMRMASK	Returns the routing codes assigned to the current message, in &MASKCHK format	AS
&AOMROUTC	Returns the routing codes assigned to the current message	AS
&AOMROUTE	Returns the routing option for the current message, as set by the screening table	AS
&AOMRROUT	Returns the remote routing option for the current message, as set by the screening table	AS
&AOMRWTOR	Indicates whether the current message is a Replied-to-WTOR	AS
&AOMSALRT	Indicates whether the &AOMALERT verb sources the current message	AS
&AOMSDATA	Returns the saved data from a successful LOOKUP statement	AS
&AOMSINGL	Indicates whether the current message is a single-line message	AS
&AOMSOLIC	Indicates whether the current message is a solicited message	AS
&AOMSOLTP	Returns the solicit type of the current message	AS
&AOMSOS	Identifies the type of operating system that sourced this message	AS
&AOMSUBTP	Returns the subtype of the current line of the current message	AS
&AOMTEXT	Returns the major text of the current message	AS
&AOMTIME	Returns the timestamp of the current message	AS
&AOMTYPE	Identifies the current message as a WTO, WTOR, DOM, or EVENT	AS
&AOMUFLGS	Returns the eight user flags in &MASKCHK format	AS
&AOMUFLG1–8	Eight system variables which are user-defined flags, set by the screening table	AS
&AOMVMMCL	Returns the VM IUCV message class of a VM-sourced message	AS
&AOMVMSRC	Returns the AOM/VM message source	AS
&AOMVMUID	Returns the VM user ID that a message originated from	AS
&AOMVMUND	Returns the VM RSCS node that a message originated from	AS
&AOMWRID	Returns the WTOR reply ID of the current message	AS
&AOMWRLEN	Returns the length of the text that is passed in reply to a WTOR	AS
&AOMWTO	Indicates whether the current message is a write-to-operator (WTO)	AS
&AOMWTOR	Indicates whether the current message is a WTOR	AS
&BROLINE*n*	A series of system variables that return the current broadcast lines	

Name	Description	Feature/ Component
&CURSCOL &CURSROW	System variables that return the cursor location	
&DATE*n*	A series of system variables that return the current system date in different formats	
&DAY	Returns the current day of the week	
&FILEID	Returns the name of the file currently being processed	
&FILEKEY	Indicates the current position of an NCL process within a UDB	
&FILERC	Indicates the success or otherwise of a file processing function	
&FILERCNT	Provides a count of the number of records deleted by &FILE DEL processing	
&FSM	Indicates if the issuing procedure has access to a real window	
&INKEY	Returns a value representing the key last used to enter data	
&LOOPCTL	Returns the current setting of the automatic loop control counter	
&LUCOLS	Indicates the number of columns currently allocated to this processing window	
&LUEXTCO	Indicates whether the terminal supports extended color	
&LUEXTHI	Indicates whether the terminal supports extended highlighting	
&LUNAME	Returns the name of the terminal at which the NCL procedure is executing	
&LUROWS	Returns the number of rows currently allocated to this process window	
&MAI#SESS	Returns the number of currently defined sessions (equivalent to &MAINSESS)	SNAACCESS
&MAIAE	Indicates the availability of the A and E primary commands	SNAACCESS
&MAIAPPL	Returns the name of the application acting as the PLU on the MAI session	SNAACCESS
&MAICOLS	Returns the number of columns in the current screen of an MAI session	SNAACCESS
&MAICROWS	Returns the number of rows in the current screen of an MAI session	SNAACCESS
&MAIDISC	Indicates whether MAI honors a terminal disconnect request	SNAACCESS
&MAIFRLU	Returns the direction of the last data stream	SNAACCESS
&MAIIKEY	Indicates the value of the key used to enter data	SNAACCESS
&MAILOCK	Indicates whether MAI honors a terminal lock request	SNAACCESS
&MAILU	Returns the name of the VTAM APPL being used as the secondary LU	SNAACCESS

Name	Description	Feature/ Component
&MAIMNFMT	Returns the current menu format as long or short	SNAACCESS
&MAINSESS	Returns the number of currently defined sessions (equivalent to &MAI#SESS)	SNAACCESS
&MAIOCMD	Returns the outbound data stream sent by the PLU	SNAACCESS
&MAIREQ	Returns the MAI logon request	SNAACCESS
&MAISCANL	Returns the scan limit for session commands	SNAACCESS
&MAISID	Returns the session ID of the session of whose behalf the script is running	SNAACCESS
&MAISKIPP	Returns the systemwide value for the session command prefix character	SNAACCESS
&MAISKPK1	Returns the session command function key 1	SNAACCESS
&MAISKPK2	Returns the session command function key 2	SNAACCESS
&MAISMODE	Returns the mode in which the script procedure is running	SNAACCESS
&MAITITLE	Returns the title that is displayed at the top of the MAI-FS main menu	SNAACCESS
&MAIUNLCK	Indicates whether the data stream just received unlocks the keyboard	SNAACCESS
&MAIWNDOW	Indicates the visibility of the MAI-FS session	SNAACCESS
&NDBERRI	Returns additional information about an NDB warning or error condition	
&NDBRC	Indicates the success or otherwise of an &NDBxxx NCL statement	
&NDBRID	Returns the record ID of the current or new record	
&NDBSQPOS	Returns the relative position in an &NDBSCAN-built sequence	
&NEWSAUTH	Indicates whether a user is authorized for NEWS functions	SNA
&NEWSRSET	Indicates whether the user is authorized for NEWS reset (delete) functions	SNA
&NMID	Returns the ID of this system	
&OCSID &OCSIDO	Indicates the OCS ID name for the current window	
&PANELID	Indicates the name of the current panel	
&PARMCNT	Returns the count of the number of variables entered when a procedure was invoked	
&RETCODE	Returns the current system return code	
&ROUTECODE	Returns the routing codes assigned to the current message, in &MASKCHK format	AS

Name	Description	Feature/ Component
&SYSID	Returns the current operating system identification	
&TIME	Returns the current time	
&USERAUTH	Returns the command authority of the user who initiated the procedure	
&USERID	Returns the user ID of the user currently executing the procedure	
&USERPW	Returns the PASSWORD of the user	
&VSAMFDBK	Returns the VSAM return code from a file processing operation	
&ZACBNAME	Returns the primary VTAM ACB name in use by the system	
&ZAMTYPE	Returns the name of the access method used to connect the terminal on which the NCL procedure is executing	
&ZAPPCACC	Returns the number of active APPC conversations for the NCL process	
&ZAPPCCSI	Returns the client/server indicator for the APPC conversation	
&ZAPPCELM	Returns the message from an Error Log GDS variable received after an error, or deallocate abend	
&ZAPPCELP	Returns product set information from an Error Log GDS variable received after an error, or deallocate abend	
&ZAPPCID	Returns the conversation ID which identifies an APPC conversation (a unique integer)	
&ZAPPCIDA	Returns the APPC conversation ID for the transaction that started the NCL process	
&ZAPPCLNK	Returns the link name for an APPC conversation	
&ZAPPCMOD	Returns the mode name for an APPC conversation	
&ZAPPCPCC	Returns the number of pending APPC conversations for the NCL process	
&ZAPPCQLN	Returns the network qualified local LU name	
&ZAPPCQRN	Returns the network qualified remote LU name	
&ZAPPCRM	Returns the current receive map name	
&ZAPPCRTS	Indicates whether a request to send has been received	
&ZAPPCSCM	Returns the Server Connection Mode indicator	
&ZAPPCSM	Returns the current send map name	
&ZAPPCSND	Returns the APPC SEND protocol indicator	
&ZAPPCSTA	Returns the current state of an APPC conversation	

Name	Description	Feature/ Component
&ZAPPCSYN	Returns a character string, equivalent to that of the SYNC_LEVEL parameter of the LU6.2 MC_GET_ATTRIBUTES verb	
&ZAPPCTRN	Returns the locally known transaction identifier (up to 32 characters) for an APPC conversation	
&ZAPPCTYP	Returns a character string providing the APPC conversation type	
&ZAPPCVRB	Returns the last APPC verb that was issued	
&ZAPPCWR	Returns a character string, equivalent to that of the LU6.2 WHAT_RECEIVED parameter	
&ZAPPCWRI	Returns a character string, equivalent to that of the LU6.2 WHAT_RECEIVED parameter	
&ZBLANK1	Returns a single blank character	
&ZBROID	Returns the broadcast identifier associated with the NCL process	
&ZBROTYPE	Indicates the type of broadcast associated with the issuing procedure	
&ZCOLS	Indicates the number of columns associated with the physical terminal	
&ZCONSOLE	Returns the system console number associated with a console user ID	
&ZCURSFLD &ZCURSPOS	Returns the name of the field where the cursor is positioned and the offset within that field	
&ZDBCS	Indicates whether a terminal supports double byte character set data streams (DBCS)	
&ZDOMID	Returns the deletion identifier for a non-roll delete message	
&ZFDBK	Returns completion information following execution of selected NCL statements	
&ZDSNQLCL	Returns the value of the local data set qualifier	
&ZDSNQSHR	Returns the value of the shared data set qualifier	
&ZGDATE*n*	A set of system variables that return the date, in different formats, based on GMT	
&ZGDAY	Returns the day of the week, based on GMT	
&ZGOPS	Indicates the generic type of operating system	
&ZGTIME*n*	A set of system variables that return the time, based on GMT	
&ZGTIMEZ*n*	A set of system variables that indicate the difference in time between local (operating system) time and GMT	

Name	Description	Feature/ Component
&ZINTYPE	(Message profile variable) Specifies whether a request message or a response message satisfies an &INTREAD operation	
&ZIREQCNT	Returns the count of messages queued to dependent request queue of an NCL process	
&ZIRSPCNT	Returns the count of messages queued to dependent response queue of an NCL process	
&ZJOBNAME	Returns the job name	
&ZJOBNUM	Returns the JES2/3 job number for the last job submitted by NCL (OS/VS only)	
&ZJRNLACT	Returns the ddname of the active journal data set	
&ZJRNLALT	Returns the ddname of the alternate (or inactive) journal data set	
&ZLCLIPA	Returns the IP address of the local host for a TN3270 session	
&ZLCLIPP	Returns the IP port of the TN3270 server for a TN3270 session	
&ZLOGMODE	Returns the name of the VTAM logmode table entry used when the current terminal was connected	
&ZLUNETID	Returns the network ID of the currently connected terminal	
&ZLUTYPE	Indicates the type of device or environment	
&ZLU1CHN	Indicates the segment position of a message received from an LU1 device	
&ZMAIACT# or &ZMAIACTN	Returns the number of active sessions associated with the current window	SNAACCESS
&ZMALARM	Indicates whether the message causes the terminal alarm to sound	
&ZMALLMSG	Indicates whether an MSG ALL command generates the message	
&ZMAOMAU	Indicates whether the original WTO or WTOR issuer was authorized	AS
&ZMAOMBC	Indicates whether the current message has the AOM broadcast attribute	AS
&ZMAOMDTA	Indicates whether the current message contains AOM data	AS
&ZMAOMID	Returns the AOM ID value	AS
&ZMAOMJI	Returns the job ID of AOM MVS-sourced messages	AS
&ZMAOMJN	Returns the job name of AOM MVS-sourced messages	AS
&ZMAOMMID	Returns the AOM message ID	AS
&ZMAOMMIN	Indicates whether this is an AOM minor line	AS

Name	Description	Feature/ Component
&ZMAOMMLC	Indicates whether the current message is an MLWTO control line	AS
&ZMAOMMLD	Indicates whether the current message is an MLWTO data line	AS
&ZMAOMMLE	Indicates whether the current message is an MLWTO end line	AS
&ZMAOMMLL	Indicates whether the current message is an MLWTO label line	AS
&ZMAOMMLT	Indicates the type of MLWTO of the current message	AS
&ZMAOMMLV	Returns the highest AOM message level of the current message	AS
&ZMAOMMSG	Indicates whether the current message was marked for propagation to eligible AOM receivers	AS
&ZMAOMRC	Returns the AOM routing codes assigned to the current message	AS
&ZMAOMRCM	Returns the routing codes assigned to the current message, in &MASKCHK format	AS
&ZMAOMRCX	Returns the AOM routing codes assigned to the current message, in hexadecimal characters	AS
&ZMAOMSOS	Returns the operating system type from which the current AOM message came	AS
&ZMAOMSYN	Returns the originating system name for the current message	AS
&ZMAOMTM	Returns the AOM timestamp of the current message	AS
&ZMAOMTYP	Returns the AOM type of a message	AS
&ZMAOMUFM	Returns the eight AOM user flags in &MASKCHK format	AS
&ZMAOMUF1–8	Eight system variables which return the AOM user-defined flags, set in the screening table	AS
&ZMAOMUI	Returns the originating user ID of an AOM message from a VM system	AS
&ZMAOMUN	Returns the VM RSCS node name that an AOM/VM message came from	AS
&ZMAPNAME	(Message profile variable) Returns the map name for the embedded user MDO in the current $MSG MDO if present	
&ZMCOLOR &ZMCOLOUR	Returns the color attribute of the message	
&ZMDOCOMP	Returns the last name segment of the fully qualified name for the MDO component involved in the last operation	
&ZMDOFDBK	Returns the feedback code after any verb references an MDO	
&ZMDOID	Returns the identifier of the MDO involved in the last operation	

Name	Description	Feature/ Component
&ZMDOM	Indicates whether the message is a delete operator message instruction	
&ZMDOMAP	Returns the map name for &ZMDOID	
&ZMDOMID	Returns the delete operator message identifier (DOMID) of the message read, provided the message has the non-roll delete message attribute (as determined by the setting of the &ZMNRD terminal)	
&ZMDONAME	Returns the fully qualified name of the MDO component involved in the last operation	
&ZMDORC	Returns the return code after any verb references an MDO (used with &ZMDOFDBK)	
&ZMDOTAG	Returns the MDO tag value of the component involved in the last operation	
&ZMDOTYPE	Returns the ASN.1 type of &ZMDOCOMP	
&ZMEVONID	Returns the NCL ID of the procedure which issued the &EVENT which caused the message on the RESP queue	
&ZMEVPROF	Returns the EDS profile name which resulted in delivery of an event notification.	
&ZMEVRCDE	Returns the route code of an incoming event message	
&ZMEVTIME	Returns the time that an event originated, in the format HH.MM.SS.THT	
&ZMEVUSER	Returns the user ID of a user who issued the &EVENT verb which caused the message on the RESP queue	
&ZMHLIGHT &ZMHLITE	Returns the display highlighting attribute of the message. Values are NONE, USCORE, REVERSE, or BLINK	
&ZMINTENS	Returns the display intensity attribute of the message. Values are HIGH, or LOW, or null if no message is processed	
&ZMLNODE	Returns the terminal name of the user to whom the log message is to be attributed	
&ZMLOGCMD	Returns whether a log message is an echo to the log of a command (available to &LOGREAD only)	
&ZMLSRCID	Returns the message prefix of the last handler for the message just received	
&ZMLSRCTP	Returns the type of the last handler for the message just received	
&ZMLTIME	Returns the time stamp of a log message (available to &LOGREAD only) (format HH.MM.SS.THT)	

Name	Description	Feature/ Component
&ZMLUSER	Returns the user ID the log message came from (available to &LOGREAD only)	
&ZMMONMSG	Indicates whether the message received is a monitor class message	
&ZMMSG	Indicates whether the message received is a standard message	
&ZMMSGCD	Indicates the hexadecimal message code attribute for this message	
&ZMNMDIDL	Returns the domain ID for the previous system to handle this message	
&ZMNMDIDO	The domain ID for the system where this message originated	
&ZMNRD	Indicates whether the message carries the non-roll delete attribute	
&ZMNRDRET	Indicates whether the message is received as a result of the user issuing an NRDRET command	
&ZMODFLD	Returns the name of the next modified field on a panel	
&ZMOSRCID	Returns the message prefix for the originator of the message just received	
&ZMOSRCTP	Returns the type for the originator of the message just received	
&ZMPPODTA	Indicates whether any PPO message profile information is available regarding this message	AS
&ZMPPOMSG	Indicates whether the message originated from PPO	AS
&ZPPOSCNT	A counter of remote domains to which a PPO message was delivered.	AS
&ZMPPOSEV	If &ZMPPODTA=YES, then this variable includes gives the severity level of the PPO message	AS
&ZMPPOTM	If &ZMPPODTA=YES, this variable gives the time when the message was created	AS
&ZMPPOVNO	If &ZMPPODTA=YES, this variable returns the VTAM message number for the PPO message	AS
&ZMPREFXD	Indicates whether the message text has been prefixed with identifier values	
&ZMPTEXT	Returns the message text, prefixed according to the current profile settings	
&ZMREQID	Returns either user ID or NCL ID, if &ZINTYPE=REQ	
&ZMREQSRC	Returns the source of the INTQ command if &ZINTYPE=REQ	
&ZMSLEVEL	Returns the version of System Services	
&ZMSOLIC	Indicates whether the message was solicited or unsolicited	

Name	Description	Feature/ Component
&ZMSOURCE	Returns the verb that last set the values for the message profile variables	
&ZMTEXT	Returns the text of the message received	
&ZMTYPE	Returns the type of message received	
&ZNCLID	Returns the unique identifier of the NCL process	
&ZNCLNEST	Returns the EXEC nesting level of the current procedure within the method level	
&ZNCLTYPE	Returns the type of the current procedure	
&ZNETID	Returns the value of the VTAM network identifier	
&ZNETNAME	Returns the network name of the primary ACB	
&ZNMDID	Returns the value of the domain identifier	
&ZNMSUP	Returns the value of the system user prefix	
&ZOCS	Indicates whether the NCL process is associated with an OCS window	
&ZOPS	Returns the type of operating system	
&ZOPSVERS	Returns the version of the operating system	
&ZOUSERID	Returns the originating user ID for an NCL process	
&ZPERRORC	Returns the value of the standard panel field attribute COLOR for error fields and messages	
&ZPERRORH	Returns the value of the standard panel field attribute HLITE for error fields	
&ZPINPHIC	Returns the value of the standard panel field attribute COLOR for mandatory input data fields and command fields	
&ZPINPLOC	Returns the value of the standard panel field attribute COLOR for optional input data fields	
&ZPINPUTH	Returns the value of the standard panel field attribute HLITE for data input fields	
&ZPINPUTP	Returns the value of the standard panel field attribute PAD for data input fields	
&ZPLABELC	Returns the value of the standard panel field attribute COLOR for field labels and comments	
&ZPMTEXT1	Returns the text of the Primary Menu broadcast	
&ZPOUTHIC	Returns the value of the standard panel field attribute COLOR for output data fields that are always present	

Name	Description	Feature/ Component
&ZPOUTLOC	Returns the value of the standard panel field attribute COLOR for output data fields that are not always present	
&ZPPFKEYC	Returns the value of the standard panel field attribute COLOR for the output fields on the left and right of the panel title and the function key area	
&ZPPI	Indicates whether PPI is available	
&ZPPINAME	Returns the defined receiver-ID of the current NCL process if it has one	
&ZPRINAME	Returns the name of the primary ACB or XNF UCE	
&ZPRODNAM	Returns the product name	
&ZPSERVIC	Returns the value of the first four bytes of the PSERVIC field of the BIND for the current terminal	
&ZPSKIP	Returns the next available segment of panel skip data	
&ZPSKPSTR	Returns the current panel skip string in its entirety	
&ZPSUBTLC	Returns the value of the standard panel field attribute COLOR for subtitles, headings and trailers	
&ZPTITLEC	Returns the value of the standard panel field attribute COLOR for the panel title	
&ZPTITLEP	Returns the value of the standard panel field attribute PAD for the panel title	
&ZPWSTATE	Returns the state of a user's password	
&ZREMIPA	Returns the IP address of the remote host for a TN3270 session	
&ZREMIPP	Returns the remote host IP port for a TN3270 session	
&ZROWS	Returns the number of rows available to the physical terminal	
&ZSCOPE	Returns the scope of the server name if the current NCL process is registered as a server	
&ZSECEXIT	Returns the type of security exit installed	
&ZSERVER	Returns the server name if the current NCL process is registered as a server	
&ZSNAMID	Returns an integer when using the &SNAMS verbs	SNA
&ZSOCCID	Returns the socket ID used by the interface	
&ZSOCERRN	Returns the error number value associated with the last referenced socket	
&ZSOCFHNM	Returns the full host name of the host referenced by some requests	

Name	Description	Feature/ Component
&ZSOCHADR	Returns the IP address of the host referenced by some requests	
&ZSOCHNM	Returns the host name of the host referenced by some requests	
&ZSOCID	Returns the socket ID of the last referenced socket	
&ZSOCPRT	Returns the port number of the last referenced socket	
&ZSOCTYPE	Returns the type of the last referenced socket	
&ZSOCVERR	Returns vendor error information from the last referenced socket	
&ZSSCPNAM	Returns the value of the VTAM SSCP name	
&ZSYSNAME	Returns the SYSNAME value	
&ZTCP	Returns the status of the socket interface	
&ZTCPHSTA	Returns the value of the IP address of the local host	
&ZTCPHSTF	Returns the value of the full name of the local host	
&ZTCPHSTN	Returns the value of the short name of the local host	
&ZTIMEn	Returns system variables for different formats of the current time	
&ZTSOUSER	Indicates if the user has connected through the TSO or TSS interface	
&ZUCENAME	Returns the UCE name that the product region is using to communicate with XNF	
&ZUDATE*n*	A set of system variables that return the user's date, in different formats, time zone adjusted	
&ZUDAY	Returns the user's day of the week, time zone adjusted	
&ZUSERLC	Returns the language code for this user	
&ZUSERSLC	Returns the system recognized language code for a user	
&ZUSRMODE	Returns a value indicating special conditions of this signed on user	
&ZUTIME*n*	A set of system variables that return the user's time, time zone adjusted	
&ZUTIMEZ*n*	A set of system variables that indicate the difference in time between local (operating system) time and the user's time zone	
&ZUTIMEZN	Returns the user's time zone name	
&ZVARCNT	Returns the number of variables created or modified by the last NCL verb that used generic processing	
&ZVTAMLVL	Returns the VTAM release and version number, if available	
&ZVTAMPU	Returns the host PU name of VTAM	
&ZVTAMSA	Returns the subarea number of VTAM	

Name	Description	Feature/ Component
&ZWINDOW	Returns the identifier of the current window	
&ZWINDOW#	Returns the number of active windows	
&ZWSTATE	Returns a value indicating the state of the current window	
&0	Returns the name of the procedure currently being executed	
&00	Returns the name of the base procedure of the NCL process	
&000	Returns the system global variable prefix	

&ALLPARMS

A user variable that supplies a single string for all parameters specified when an NCL procedure is invoked.

&ALLPARMS is a user variable that is a single string containing all parameters entered when the procedure was invoked.

&ALLPARMS is created automatically when the process is invoked if any data was supplied on the initiating command.

Examples: &ALLPARMS

If a procedure is invoked with the following command:

```
EXEC ROUTPROC text of command to be sent to remote location
```

then &ALLPARMS is set to ”text of command to be sent to remote location”.

The variable can then be used in commands such as:

```
ROUTE NM2 &ALLPARMS
```

Notes:

&ALLPARMS is set only when a procedure is invoked with parameters. If the procedure is invoked without parameters, &ALLPARMS is set to null. Subsequent functions such as &PAUSE which allow the operator to enter more optional variables do not cause &ALLPARMS to be reset.

The data contained within &ALLPARMS is referenced on an individual variable basis using &1, &2, and so on. These variables, however, remain available only until reset by another function or assignment that specifically references that variable. The maximum length of data that is set in &ALLPARMS is 256 characters.

Because &ALLPARMS is a user variable, it is deleted by the procedure if necessary.

&AOMACCT1-4

Four system variables containing, for some MVS-sourced messages, the first four accounting fields from the JOB statement.

The system variables &AOMACCT1-4 contain the first four accounting fields from the JOB statement of a job issuing a WTO or WTOR message. Each variable has a maximum length of 10 characters.

Accounting information is not always available. If the message is sourced by JES2, or the accounting control blocks could not be located, these system variables are null. They are also null for DOM-notify or VM-sourced messages.

Examples: &AOMACCT1

```
&AOMREAD STRING &MSG
 .
 .
 .
&IF .&AOMACCT1 = .PROD &THEN ...  process prod job
                       &ELSE ...  process non-prod job.
 .
 .
 .
```

Note: If accounting information is to be used, it should be saved in a table when a message that carries it is detected.

More information:

&AOMJOBID (see page 770)

&AOMALARM

A system variable containing the alarm attribute for the current message.

The &AOMALARM system variable is set to either NO or YES corresponding to the value coded on the [ALARM=NO | YES] keyword operand of the GLOBAL or MSGGROUP statements in the AOM screening table, or as altered by a SET statement.

If the ALARM= operand is not coded in the screening table then the default value assigned to this system variable is NO.

Example: &AOMALARM

```
&AOMREAD STRING &MSG
 .
 .
 .
&IF &AOMALARM = NO &THEN &WRITE ALARM=YES DATA=&MSG +
                  &ELSE &WRITE DATA=&MSG
 .
```

The &ZMALARM system variable contains the ALARM attribute for AOM messages delivered beyond AOMPROC.

Note: For more information, see the GLOBAL, MSGGROUP, and SET statements in the AOM screening table.

More information:

&AOMREAD (see page 65)
&ZMALARM (see page 912)

&AOMASID

A system variable containing the MVS address space ID (ASID) that issued the current message, in hexadecimal format.

&AOMASID contains a four-digit hexadecimal number which is the address space ID of the job that generated the message.

Sometimes messages are generated by a job, such as JES2, on behalf of another job. In such an instance the value in &AOMASID is the ID of the actual address space that issued the message, such as JES2, and not the ID of the address space that the message actually refers to.

Examples: &AOMMSGID

```
.LOOP
&AOMREAD SET &TESTJOB = &SUBSTR &AOMJOBNM 1 4
&IF .&AOMMSGID = .$HASP373 &IF &TESTJOB = TEST &GOTO .CANJOB
 .
 .
 .
&GOTO .LOOP
-*Pass ASID to Worker NCL proc with request to CANCEL job.
.CANJOB
-INTQ ID=xxx CANCEL &AOMASID
&GOTO .LOOP
```

Note: This variable is null if the current message is VM-sourced or a DOM-Notify.

More information:

&AOMJOBID (see page 770)
&AOMJOBNM (see page 771)

&AOMATEXT

A system variable containing the text of the current line of a message that has been delivered to AOMPROC.

When the &AOMREAD verb is issued to get the next message for processing by AOMPROC, the message is tokenized by using the VARS or ARGS operand.

Where access to the complete text of the original message is required, &AOMATEXT is used.

Examples: &AOMATEXT

```
&AOMREAD ARGS
  .
  .
  .
&IF &AOMID EQ WRITEOCS &THEN +
    &WRITE MON=YES MSGCODE=EF DATA=&AOMATEXT
  .
  .
  .
```

Notes:

&AOMATEXT equals &AOMTEXT for single line WTOs and WTORs, and VM messages. For multi-line WTOs, &AOMATEXT is always set to the current or minor line.

The three system variables &AOMMAJOR,&AOMMINOR, and &AOMSINGL is used to determine if the current line is a multi-line WTO and whether the current line is a major or minor line.

In the case of reading a minor line of a multi-line WTO, &AOMATEXT contains the actual text of the minor line.

Following is a table of the possible settings for all these variables:

&AOMMAJOR	&AOMMINOR	&AOMSINGL	&AOMTEXT
NO	NO	YES	CURRENT LINE
YES	NO	YES	CURRENT LINE
NO	YES	NO	FIRST/MAJOR LINE

In all the above cases, &AOMATEXT has the text of the current line.

The text returned includes any MVS screen characters.

More information:

&AOMCHAR1 (see page 756)
&AOMTEXT (see page 810)
&AOMMAJOR (see page 780)
&AOMMINOR (see page 782)
&AOMSINGL (see page 805)

&AOMAUTH

A system variable that indicates if the issuer of a WTO/WTOR is authorized.

Any program, either authorized or unauthorized, can issue a WTO or WTOR. It might be necessary to determine if the issuer of a WTO/WTOR is authorized before proceeding with any processing associated with that message.

&AOMAUTH is set to YES if the program issuing the WTO/WTOR is authorized; otherwise it is set to NO.

By examining &AOMAUTH it is possible to effectively ignore a message that was issued from an unauthorized program (for example, a tape mount message), which contains the same text as a critical message.

Examples: &AOMAUTH

```
&AOMREAD SET
 .
 .
 .
&IF &AOMAUTH = NO &THEN &AOMDEL
```

Notes:

In MSGPROC, the system variable &ZMAOMAU contains the same value as &AOMAUTH.

This system variable is null for VM messages and DOM-notifications.

&AOMAUTO

A system variable that returns the value of the automation flag.

MPF lists can flag messages as eligible for automation. This system variable contains the value YES or NO, reflecting this automation option.

Example: &AOMAUTO

```
&AOMREAD SET
  .
  .
  .
&IF &AOMAUTO = YES &THEN ... automate...
```

Notes:

This variable is useful when converting from other automation products that use the MPF list facilities.

This system variable is null for VM messages and DOM-notifies.

More information:

&AOMAUTOT (see page 755)

&AOMAUTOT

A system variable that contains the value of the automation token.

MPF lists can flag messages as eligible for automation. An optional 1- to 8-character automation token can also be assigned. This system variable contains the automation token value.

Example: &AOMAUTO

```
&AOMREAD SET
 .
 .
 .
&IF &AOMAUTO = YES &THEN &GOTO .&AOMAUTOT
```

Notes:

This variable is useful when converting from other automation products that use the MPF list facilities.

This system variable is null for VM messages and DOM-notifies.

More information:

&AOMAUTO (see page 754)

&AOMBC

A system variable that indicates whether or not the current message is a broadcast message.

This variable is set to YES if the current message is a console broadcast message; otherwise it is set to NO.

Example: &AOMBC

```
& IF &AOMBC = YES &THEN &AOMCONT COLOR=WHITE
```

Note: Console broadcast is also indicated in the message level attribute.

More information:

&AOMMSGLV (see page 788)

&AOMCHAR1

A system variable containing the screen character that indicates the status of operator console format messages.

When an MVS message is to be sent to an operator console, it contains a screen character that indicates the status of the message. This screen character immediately precedes the message text.

Valid values for the screen character are as follows:

***asteriskl* (*)**

The message was issued by an authorized system program with a descriptor code of 1, 2, or 11, and requires specific operator action.

***at sign* (@)**

The message was issued by a problem program with a descriptor code of 1, 2, or 11, and requires specific operator action.

blankl

The message does not require any specific operator action.

Example: &AOMCHAR

```
&IF .&AOMCHAR1 = . &THEN &AOMDEL
```

Notes:

When a message is parsed by &AOMREAD the first character of the message is not tokenized into an NCL variable. The only method of obtaining the screen character is by referring to this system variable or by sub-stringing the first character of the &AOMTEXT or &AOMATEXT system variables.

For VM-sourced messages, this system variable is null.

More information:

&AOMREAD (see page 65)
&AOMTEXT (see page 810)
&AOMATEXT (see page 751)

&AOMCOLOR

A system variable that indicates the color attribute of the current message.

After an &AOMREAD verb is issued to get the next available message for processing by AOMPROC, &AOMCOLOR is set to indicate the color attribute for that message.

The &AOMCOLOR system variable is set to one of the following values corresponding to the value coded on the [COLOR=value] keyword operand of the GLOBAL or MSGGROUP statements in the AOM screening table, or as altered by a SET statement.

If the COLOR operand is coded on both the GLOBAL and MSGGROUP statements, then the value on the MSGGROUP statement takes precedence for a particular message.

Valid values for message color are:

RED, BLUE, GREEN, WHITE, PINK, YELLOW, TURQUOISE, NONE.

Example: &AOMREAD

```
&AOMREAD SET
  .
  .
  .
&IF .&AOMID EQ .PRODIMS &THEN +
   &WRITE MSGCODE=04 COLOR=&AOMCOLOR DATA=&ATEXT
```

Notes:

In MSGPROC, the system variable &ZMCOLOR contains the same value as &AOMCOLOR.

If the screening table does not set a color attribute, then &AOMCOLOR is set to NONE.

More information:

&AOMINTEN (see page 768)
&AOMHLITE (see page 764)

&AOMCONNM

Returns the Extended MCS console name.

Provides the Extended MCS console name that a WTO or WTOR is directed to or that a command was issued from.

Example: &AOMCONNM

```
& IF &AOMCONNM = MASTER &THEN +
   &GOSUB .MSGMAST
```

More information:

&AOMMVCON (see page 789)

&AOMDESC

A system variable that contains the descriptor code(s) assigned to the current message, in list format.

The descriptor code(s) that is assigned to a message are in the range from 1 to 16. This system variable contains a list-format representation of the descriptor codes. For example, if descriptor codes 1 and 7 are set, &AOMDESC has the value (1,7).

Example: &AOMDESC

```
&AOMREAD ARGS
  .
  .
  .
&WTO DESC=&AOMDESC DATA=I saw msg: &AOMATEXT
  .
  .
  .
```

Note: This system variable is particularly useful for the DESC operand of the &AOMALERT and &WTO verbs.

More information:

&WTO (see page 706)
&AOMALERT (see page 35)
&AOMDMASK (see page 760)
&AOMDHEX (see page 759)

&AOMDHEX

A system variable that contains the descriptor code(s) assigned to the current message, in hexadecimal.

The descriptor code(s) that is assigned to a message are in the range from 1 to 16. This is a two-byte (four-character) system variable containing the hexadecimal representation of the descriptor code(s).

For example, a WTO macro specifying DESC=(3,7,9) yields a hexadecimal value of X'2280' in &AOMDHEX.

Example: &AOMDHEX

```
& AOMREAD ARGS
  .
  .
  .
&CALL USERPROG &AOMDHEX
  .
  .
  .
```

Note: This system variable is particularly useful for passing the descriptor code(s) to a called program.

More information:

&AOMDMASK (see page 760)
&AOMCONNM (see page 758)

&AOMDMASK

A system variable that contains the descriptor code(s) assigned to the current message, in &MASKCHK format.

The descriptor code(s) that is assigned to a message are in the range from 1 to 16. This is a sixteen-byte system variable containing sixteen Y/N values representing the descriptor code(s).

For example, a WTO macro with DESC=(3,7,9) would yield a sixteen-byte value of NNYNNNYNYNNNNNNN.

Example: &AOMDMASK

```
& NRD = &MASKCHK **Y***Y*Y******* &AOMDMASK
```

Note: &AOMDMASK provides easy access to message descriptor code(s) via NCL.

More information:

&AOMDHEX (see page 759)
&AOMDESC (see page 758)

&AOMDOM

A system variable that indicates whether or not the current message is a Delete Operator Message notification (DOM-Notify).

If an AOMPROC issues &AOMCONT, REPL, or DEL with the DOM-NOTIFY=YES option for a WTO, WTOR, or MVS-sourced EVENT, then AOM enqueues a DOM-Notify message to the AOMPROC if either of the following happens:

- A corresponding MVS DOM (in any format) is received.
- AOM is stopped.

When this message is read by &AOMREAD, &AOMDOM is set to YES.

The &AOMDOMID system variable contains the domain ID of the original message. If the message was sent to AOM receivers, as an NRD message, then the domain ID assigned by Automation Services is also available in &AOMNMDOM.

This system variable is accessed in an AOMPROC only, and is available after an &AOMREAD statement is issued.

Example: &AOMDOM

```
& AOMREAD SET
 .
 .
 .
&IF &AOMDOM = YES &THEN ... process dom notify
```

Notes:

An AOMPROC can indicate an interest in a DOM by issuing &AOMCONT/REPL/DEL with DOM-NOTIFY=YES. AOM handles the various MVS DOM types and automatically queues the notification to the correct AOMPROC.

A value of DOM in system variable &AOMTYPE, after an &AOMREAD DOM=YES verb is executed, also indicates that the current message is a DOM-Notify.

More information:

&AOMDOMID (see page 762)
&AOMNMDOM (see page 792)
&AOMREAD (see page 65)

&AOMDOMID

A system variable that contains the MVS-assigned Delete Operator Message (DOM) ID of the current message.

Every WTO/WTOR message has an associated DOMID. This ID is normally used to associate a non-roll delete WTO/WTOR message with a subsequent DOM. A DOM is generated to delete a particular non-roll delete WTO/WTOR message that is no longer required by the system.

For example, when an address space terminates it can generate a DOM message to delete outstanding non-roll-delete messages that were associated with the address space.

This DOMID is supplied in the &AOMDOMID system variable following an &AOMREAD that returns a WTO, WTOR, or MVS-sourced EVENT. &AOMALERT-generated WTO or EVENT messages are also assigned a DOMID. It is also supplied when a DOM-Notify message is received by AOMPROC.

The DOMID is formatted as 8 hexadecimal digits. The first 2 are a system ID and the last 6 are the message number.

Examples: &AOMDOMID

```
-* read messages and DOM-notifies.
.LOOP
&AOMREAD SET
-* if NRD001 save domid
&IF .&AOMMSGID = NRD001 &THEN &DO  -* want notify
   &SAVEID = &AOMDOMID             -* save domid
   &AOMCONT DOM-NOTIFY=YES         -* indicate notify
                                   -* wanted & GOTO .LOOP
&DOEND
&IF &AOMTYPE = DOM &THEN &DO       -* got dom notify
   &if .SAVEID = .&AOMDOMID &THEN &DO
   .
   . -* process notify
   .
   &GOTO .LOOP
&DOEND
```

Notes:

ID contained in this system variable is generated by MVS, and as such is useful only for message correlation within an AOMPROC, or for use as the ID for an &DOM verb. Automation Services generates its own internal DOMID for messages sent to OCS consoles.

The DOMID is used to correlate a message with the arrival of the associated DOM-NOTIFY.

Another use is to issue an MVS DOM using the &DOM verb for NRD messages that MVS does not itself delete.

More information:

&AOMDOM (see page 761)
&DOM (see page 303)
&AOMREAD (see page 65)

&AOMEVCLS

Returns the EVENT class value.

When an EVENT statement is executed by a screening table, or an &AOMALERT TYPE=EVENT verb is executed, the generated event is assigned a class. This class is a 1- to 12-character value. &AOMEVCLS returns this event class, or a null value if no event class is specified for this event.

The meaning of event classes is user defined.

Example: &AOMDOMID

```
-* process events
.LOOP
&AOMREAD SET
-* if NRD001 save domid
&IF &AOMTYPE = EVENT &THEN &DO -* want notify
   &GOTO .&AOMEVCLS            -* use class as label.
&DOEND
```

Notes:

AOM places no meaning on an event class. It is entirely user defined.

An event class is specified along with an ID value (as set in &AOMID). This is useful for major and minor subdivision of events.

More information:

&AOMTYPE (see page 812)
&AOMID (see page 765)

&AOMHLITE

Returns the highlight attribute for the current message.

The &AOMHLITE system variable is set to one of the following values taken from the value coded on the [HLIGHT=value] keyword operand of the GLOBAL or MSGGROUP statements in the AOM screening table, or as altered by a SET statement.

Valid values for message highlight are:

```
NONE
USCORE
BLINK
REVERSE
```

Example: &AOMHLITE

```
-* process events
.LOOP
&AOMREAD SET
 .
 .
 .
&IF &AOMHLITE = NONE &THEN &GOTO .SETHLITE
 .
 .
 .
&GOTO .LOOP -* Set messages to REVERSE highlighting.
.SETHLITE
&AOMCONT HLIGHT=REVERSE
&GOTO .LOOP
```

More information:

&AOMINTEN (see page 768)
&AOMCOLOR (see page 757)
&AOMALARM (see page 750)

&AOMID

Returns an ID that has been assigned to the current message or event.

Enables AOM messages to be classified into groups for processing.

Example: .&AOMID

```
&CONTROL NOLABEL
 .
 .
 .
&AOMREAD SET
&GOTO .&AOMID   -* Use &AOMID as a branch destination
 .
 .
 .
.JESMSG         -* $HASP messages
 .
 .
 .
.ACFMSGS        -* ACF2 messages
 .
 .
 .
```

Notes:

The GLOBAL, MSGGROUP, and SET statements in the AOM screening table allow the specification of an ID, from 1 to 12 characters in length. If a message passes screening at a particular level then the ID specified at that level, or a default value, is assigned.

If no screening table is active, this variable has a value of NOTABLE. AOMPROC can contain logic to detect whether the screening table is not loaded by checking for ID=NOTABLE.

If a screening table is active but no ID has been specified on a GLOBAL or MSGGROUP statement, a message passing the relevant screening statements is assigned a default ID of AOMGLOBAL or AOMMSGGROUP respectively.

Events have a default ID of AOMEVENT. Messages and events sourced by the &AOMALERT NCL verb have a default ID of AOMALERT.

This ID attribute is also propagated to AOM receivers, and is inspected using the &ZMAOMID system variable.

Normally a group of messages are all assigned the same ID, for example, all $HASP messages might be assigned an ID of JESMSG. This means &AOMID is used in AOMPROC for processing a particular group of messages.

The EVENT statement in the screening table also allows an ID value to be assigned.

Note: For more information, see the ID operand on the GLOBAL, MSGGROUP, SET, and EVENT screening table statements.

&AOMIJOBN

A system variable containing the MVS job name of the address space that issued the WTO, WTOR, or EVENT.

This system variable contains the MVS job name of the address space that actually issued the WTO or WTOR.

This cannot be the same as the value in &AOMJOBNM, which contains the job name that the WTO can refer to (if JES has provided it).

Example: &AOMIJOBN

```
& CONTROL NOLABEL
 .
 .
 .
&AOMREAD SET
&IF .&AOMIJOBN = .JES2 &THEN GOTO .JES2SRCD
```

Note: &AOMIJOBN will be null if the current message or event is sourced from VM.

More information:

&AOMJOBNM (see page 771)
&AOMJOBID (see page 770)

&AOMINTEN

A system variable containing the intensity attribute for the current message.

The &AOMINTEN system variable is set to one of the following values corresponding to the value coded on the INTENS=value keyword operand of the GLOBAL or MSGGROUP statements in the AOM screening table, or as altered by a SET statement.

Values for message intensity are:

```
NORMAL

HIGH
```

Example: &AOMINTEN

```
.LOOP
&AOMREAD SET
 .
 .
 .
&IF &AOMINTEN = HIGH &THEN &GOTO .SETINTEN
 .
 .
 .
&GOTO .LOOP
-*Turn HIGH intensity off.
.SETINTEN
&AOMCONT INTENS=NORMAL
&GOTO .LOOP
```

More information:

&AOMALARM (see page 750)
&AOMCOLOR (see page 757)
&AOMHLITE (see page 764)

&AOMJOBCL

A system variable that contains the job class of the job that issued the WTO or WTOR.

This variable contains the job class that the issuing JOB had specified in the JOB statement in the JCL, if possible.

If JES issued the message, or JOB accounting control blocks are not accessible, this variable is null.

Example: &AOMJOBCL

```
.LOOP
&AOMREAD SET
&IF .&AOMJOBCL = .P &THEN &GOTO .PRODJOB
```

Note: Because the job class information is not available on all messages, the value should be saved in storage, such as a VARTABLE keyed by, for example, JOBID.

More information:

&AOMACCT1-4 (see page 749)

&AOMJOBID

A system variable that contains the JES job number that issued the current message.

This variable contains the job number associated with the current message. The first character is a letter describing the type of job followed by a five-digit number.

```
J00005 = JOB 5
T00120 = TSU 120
S03453 = STC 3453
```

Example: &AOMJOBID

```
.LOOP
&AOMREAD SET
&JTYP = &SUBSTR &AOMJOBNM 1 4
&IF &JTYP = TEST &THEN &GOTO .CANJOB
 .
 .
 .
&GOTO .LOOP
 .
 .
 .
-*Issue JES2 Cancel via SYSCMD.
.CANJOB
-SYSCMD $C&AOMJOBID
GOTO .LOOP
```

Note: When the job number related to a message is unavailable to AOM, &AOMJOBID is null. This occurs for messages generated by the MASTER address space, or by any subsystem not started by JES.

More information:

&AOMJOBNM (see page 771)

&AOMJOBNM

A system variable that contains the job name of the active address space that issued the current message.

AOM provides the job name of the job that issued the WTO or WTOR in this system variable. If an overriding job name is supplied by JES, then it is used. This means that messages originating from JES can have the job name of the target job, rather than the JES job name.

Example: &AOMJOBNM

```
.LOOP &AOMREAD SET
&JOBTYPE = &SUBSTR &AOMJOBNM 1 4
&IF &JOBTYPE = TEST &THEN &GOTO .TESTJOB
 .
 .
 .
&GOTO .LOOP
 .
 .
 .
.TESTJOB
   &AOMCONT COLOR=PINK HLIGHT=REVERSE
   &GOTO .LOOP
```

Notes:

Care should be taken when processing $HASP messages ,as this variable can contain the value JES2 rather than the name of the job that the message applies to. &AOMJOBID is a more reliable way of identifying which job the message relates to.

The name of the address space that issued a message might also be useful in constructing another, perhaps more meaningful, message.

The &AOMIJOBN system variable *always* contains the job name of the issuing address space.

More information:

&AOMJOBID (see page 770)
&AOMIJOBN (see page 767)

&AOMJSTCB

A system variable that contains the hexadecimal address of the job step TCB that either issued the current WTO or WTOR, or owns the TCB that issued the message.

This variable contains the hexadecimal address of the TCB that actually issued the WTO/WTOR. Thus, it is possible to differentiate between separate WTO/WTORs that have been issued from the same address space.

This variable is useful to &CALLed programs.

Examples: &AOMJSTCB

```
&CALL STATPROG &AOMATEXT &AOMJSTCB
```

Notes:

The ID of the address space that issued the current message is available in hexadecimal format in the system variable &AOMASID.

It might be necessary to go one step further and determine which TCB, within a particular address space, issued the WTO/WTOR.

Non-Roll-Delete messages is deleted by a JSTCB level DOM. AOM manages this automatically.

More information:

&AOMASID (see page 751)

&AOMLDID

A system variable that contains the Automation Services domain ID of the last handler of this message, event, or DOM-Notify.

Since AOM traffic can arrive across an ISR link, a way is needed to identify the system that originated this message, and the system that passed the message to this system.

&AOMLDID contains the 4-character domain ID, as set by the NMDID JCL parameter, of either the system that sent this message to this system, if it originated across an ISR link, or the domain ID of this system, if sourced locally.

This allows an AOMPROC, when handling automation centrally, to identify the sender (but not originator) of this message.

Example: &AOMLDID

```
&IF .&AOMLDID NE .&ZNMDID &THEN &GOTO .REMOTE
```

Notes:

For messages sourced locally, the value is equal to the value in the &ZNMDID system variable.

For messages that originated from a directly linked system, the value is that system's &ZNMDID value.

For messages that originated at least two systems away, the value is that of the system that the message was last handled by.

A MSGPROC can use the &ZMDIDL system variable to access this value if the message is delivered to AOM receivers.

All messages, events, and so on, including DOM-Notify messages, carry this attribute.

More information:

&AOMODID (see page 795)

&AOMLROUT

A system variable that contains the local routing option for the message or event as set by the screening table ROUTE or LCLROUTE operands.

A routing option is used to control delivery of messages processed by AOM. Any messages or events that arrive at an AOMPROC have this attribute, and the value is interrogated by the &AOMLROUT system variable.

&AOMLROUT contains the routing value set for the local system. &AOMRROUT contains the value set for ISR delivery.

Example: &AOMLROUT

```
&IF .&AOMLROUT = .PROCONLY &THEN &AOMDEL
```

Note: Because &AOMALERT can queue a message to a specific AOMPROC, it is possible to see ROUTE values other than PROC, PROCONLY, or BOTH.

More information:

&AOMROUTE (see page 800)
&AOMRROUT (see page 801)

&AOMLRSLT

A system variable that contains the eight LOOKUP results from screening, in &MASKCHK format.

The screening table LOOKUP statement allows setting of eight result values. Each is set to YES or NO, indicating success or failure of a LOOKUP.

For a WTO, WTOR, or MSG, &AOMLRSLT will contain the eight lookup results, formatted as a string of 8 Y or N characters. For example, if a LOOKUP statement sets result 5 true, and another sets result 8 true, then &AOMLRSLT is formatted as:

```
NNNNYNNY
```

This format is useful with the &MASKCHK built-in function.

Example: &AOMLRSLT

```
&CHECK = &MASKCHK ***Y***N &AOMLRSLT
```

Note: Individual LOOKUP results is checked by using the &AOMLRSL*n* system variables.

More information:

&AOMLRSL1-8 (see page 775)

&AOMLRSL1-8

Eight system variables that provide the results of up to eight LOOKUP statements.

The screening table LOOKUP statement allows setting of eight result values. Each is set to YES or NO, indicating the success or failure of a LOOKUP.

The system variables &AOMLRSL1 to &AOMLRSL8 each contains NO or YES, indicating whether a particular LOOKUP succeeded or failed (or was not done).

Example: &AOMLRS

```
& IF .&AOMLRSL3 = .YES &THEN &GOTO L3WORKED
```

Note: If several LOOKUP results need to be checked, see the &AOMLRSLT system variable.

More information:

&AOMLRSLT (see page 774)

&AOMLTCTL

A system variable that indicates whether or not the current line of the current message is a control line.

A control line can occur only as the first line of a multi-line WTO message, and it normally contains the message title.

A control line is optional, but if it occurs it must be no more than 34 characters in length.

&AOMLTCTL is set to YES if the message has been generated as a control line; otherwise, it is set to NO.

The line type of any line of a multi-line WTO is examined using the &AOMMINLT built-in function.

Example: &AOMLTCTL

```
-* Ensure Control line stands out.
&IF &AOMLTCTL = YES &THEN &AOMCONT HLIGHT=REVERSE
 .
 .
 .
```

Notes:

&AOMLTCTL is used to isolate control lines. These can then be enhanced to provide a more meaningful message title.

If not reading minor lines (&AOMREAD MINOR=NO), this system variable is of little use. Use the &AOMMINLT built-in function instead.

The value of &AOMMAJOR is set to YES if this variable contains the value YES.

Note: For more information, see the description of the WTO macro in the appropriate system reference manual.

More information:

&AOMLTDAT (see page 777)
&AOMLTEND (see page 778)
&AOMLTLAB (see page 779)
&AOMMAJOR (see page 780)
&AOMMINLT (see page 64)

&AOMLTDAT

A system variable that indicates whether or not the current line of the current message is a data line.

&AOMLTDAT is set to YES for all data lines following a control line of a multi-line message. For control, label, end-only, and single line messages, &AOMLTDAT is set to NO.

The &AOMMINLT built-in function allows access to the line type of any line of a multi-line WTO.

Example: &AOMLTDAT

```
.LOOP
&AOMREAD SET MINOR=YES
&IF &AOMLTDAT = YES &THEN &GOTO .MULTILINE
 .
 .
&GOTO .LOOP
 .
 .
 .   -* Delete Minor or Data lines from delivery.
.MULTILINE
&AOMDEL
&GOTO .LOOP
```

Notes:

The value of &AOMMINOR is normally set to YES if &AOMLTDAT contains a value of YES.

If not reading minor lines (&AOMREAD MINOR=NO), this system variable is of little use. Use the &AOMMINLT built-in function instead.

A data line can also be an end line.

Note: For more information, see the description of the WTO macro in the appropriate system reference manual.

More information:

&AOMLTCTL (see page 776)
&AOMLTEND (see page 778)
&AOMLTLAB (see page 779)
&AOMMINOR (see page 782)
&AOMMINLT (see page 64)

&AOMLTEND

A system variable that indicates whether or not the current message is an end line.

&AOMLTEND is set to YES if the current line of a multi-line message is an end line else &AOMLTEND is set to NO.

Example: &AOMLTEND

```
...

.LOOP &AOMREAD SET MINOR=YES
   .
   .
   .
&IF &AOMMINOR = YES AND &MULTILINEMSG = YES &THEN +
   &GOTO .MULTILINE
   .
   .
   .
.MULTILINE
 -* Reset the multi-line message flag
 -* when last line is detected.
   &IF &AOMLTEND = YES &THEN &MULTILINEMSG = NO
   .
   .
   .
   &GOTO .LOOP
```

Notes:

An end line is the last line of a multi-line WTO. Not all multi-line WTOs have an end line, as this is an option of the WTO macro. Sometimes the end line is also a data line.

&AOMLTEND is used, in conjunction with &AOMLTCTL and &AOMLTDAT, to manipulate and enhance multi-line messages.

The value of &AOMMINOR is set to YES if this variable contains a value of YES.

If not reading minor lines (&AOMREAD MINOR=NO), this system variable is of little use. Use the &AOMMINLT built-in function instead.

Note: For more information, see the description of the WTO macro in the appropriate system reference manual.

More information:

&AOMLTCTL (see page 776)
&AOMLTDAT (see page 777)
&AOMLTLAB (see page 779)
&AOMMINLT (see page 64)

&AOMLTLAB

A system variable that indicates whether or not the current line of the current message is a label line.

A label line can occur as the first line of a multi-line WTO message if there is no control line, or must immediately follow the control line or another label line. It normally contains message heading information.

&AOMLTLAB is set to YES if the current line is a label line or NO if it is not a label line.

Example: &AOMLTLAB

```
.LOOP
&AOMREAD SET MINOR=YES
&IF &AOMLTLAB = YES &THEN &GOTO .LABLINE
 .
 .
 .
```

Notes:

Multi-line messages can contain a control line which is usually the message title line. Label lines are often used as headings for the data lines.

A label line is optional, but if it occurs must be no more than 70 characters in length.

If not reading minor lines (&AOMREAD MINOR=NO), this system variable is of little use. Use the &AOMMINLT built-in function instead.

Note: For more information, see the description of the WTO macro in the appropriate system reference manual.

More information:

&AOMLTCTL (see page 776)
&AOMLTDAT (see page 777)
&AOMLTEND (see page 778)
&AOMMINLT (see page 64)

&AOMMAJOR

A system variable that indicates whether or not the current line of the current message is a major line.

This variable is set to YES if the current line is the first line of a multi-line WTO message.

If the current message is a single line WTO, as indicated by a value of YES in &AOMSINGL, then &AOMMAJOR is set to NO.

Example: &AOMMAJOR

```
.LOOP

&AOMREAD SET  MINOR=&MULTI

&IF &AOMMAJOR = YES &GOTO .MULTILINE
 .
 .
 .
&MULTI=NO
&GOTO .LOOP
 .
 .
 .
.MULTILINE
&MULTI = YES
 .
 .
 .
&GOTO .LOOP
```

Notes:

The &AOMMINLN and &AOMMINLT built-in functions is used while holding the major line to access all the other lines of a multi-line WTO. If &AOMREAD MINOR=NO is issued after reading a major line, the minor lines are not presented individually.

This variable is used, in conjunction with &AOMMINOR and &AOMSINGL, to manipulate message flow for multi-line WTOs.

This system variable indicates the start of a multi-line message. Based on other criteria, you might then decide to enter a loop where you read individual lines (using &AOMREAD MINOR=YES).

More information:

&AOMMINOR (see page 782)
&AOMSINGL (see page 805)
&AOMATEXT (see page 751)
&AOMTEXT (see page 810)

&AOMMHEX

A system variable that contains the MCS flag(s) assigned to the current WTO or WTOR.

This variable is a four character expansion of a two-byte field showing the MCS flag(s) settings of a message.

This variable is passed to a user program for interrogation using the &CALL verb.

Example: &AOMMEX

```
&CALL MYPROG &AOMMHEX
 .
 .
 .
```

Note: For an explanation on MCS flag settings, see the IBM *Supervisor Services and Macro Instructions or System Macros and Facilities* manuals.

More information:

&AOMMMASK (see page 783)

&AOMMINOR

A system variable that indicates whether or not the current line of the current message is a minor line.

This variable is set to YES if the current line is the second or subsequent line of a multi-line WTO. It is set to NO if a single line WTO or WTOR is received or if the current line is a major line for a multi-line WTO.

Example: &AOMMINOR

```
.LOOP
&AOMREAD SET MINOR=YES
&IF &AOMMINOR = YES &THEN &GOTO .DELMINOR
   .
   .
   . -* If minor line received delete from delivery
   . -* to minimize message flow.
   .
.DELMINOR
&AOMDEL
&GOTO .LOOP
```

Notes:

This variable is used, in conjunction with &AOMMAJOR and &AOMSINGL, to perform complex manipulation of multi-line messages.

If using &AOMREAD MINOR=NO, no minor lines are ever presented as the current line. In this case, the &AOMMINLN and &AOMMINLT built-ins are used to access the minor lines.

More information:

&AOMMAJOR (see page 780)
&AOMSINGL (see page 805)

&AOMMMASK

A system variable that contains the MCS flags assigned to the current message in &MASKCHK format.

This is a sixteen-character variable containing the MCS flag settings for a message in &MASKCHK format.

For example, if MCSFLAG=1, then &AOMMMASK contains YNNNNNNNNNNNNNN

Example: &MASKCHK

```
& MCSFLAGON = &MASKCHK YNNNNNNNNNNNNNNN &AOMMMASK
```

Note: The MCS flag(s) settings are also available in expanded hexadecimal format in &AOMMHEX.

More information:

&AOMMHEX (see page 781)

&AOMMONIT

A system variable that indicates whether or not the current message is also to be delivered to monitor class receivers.

&AOMMONIT is set to YES if the message is to be delivered to Monitor class users, otherwise NO, if delivery is to AOM receivers only.

AOM messages are normally only delivered to users who are profiled to receive AOM messages. There is instances when it is desirable to deliver messages to all MON class OCS users. The decision to also send an AOM message to MON class OCS users is initially made in the screening table by specifying MONITOR=YES at either the GLOBAL or MSGGROUP level.

This system variable is used to check that the screening table specification for the current message is still valid before the message is delivered to OCS consoles.

Example &AOMMONIT:

```
.LOOP
&AOMREAD SET &MSGPRF = &SUBSTR &AOMMSGID 1 3
&GOTO .&MSGPRF
   .
   .
   . -* Ensure VTAM messages are sent to Monitor class Users.
   .
.IST
&IF &AOMMONIT = NO &AOMCONT MONITOR=YES
&GOTO .LOOP
```

Note: Messages destined for monitor users would normally be identified within the screening table, but there might be specific conditions that require message analysis within AOMPROC before delivery can continue.

More information:

&AOMROUTE (see page 800)

&AOMMPFSP

A system variable that indicates whether or not the current message was initially suppressed by the Message Processing Facility (MPF).

This system variable contains the value YES if the current message was suppressed by MPF and in turn was processed by the screening table because GLOBAL MPFSUPP=YES was specified.

Before a message is processed by the screening table it may have been suppressed by MPF. Messages suppressed by MPF are not delivered to system consoles.

The AOM screening table does not normally process messages that have been suppressed by MPF. If MPFSUPP=YES is specified on the GLOBAL screening table statement, then MPF suppressed messages are processed by the screening table.

Example: &AOMMPFSP

```
-* Set MPF suppressed messages to reverse highlighting.
&IF &AOMMPFSP = YES &THEN &AOMCONT HLIGHT=REVERSE
```

Notes:

By using this variable MPF suppressed messages is singled out for special processing.

Note: For more information, see the MPFSUPP operand on the GLOBAL screening table statement.

&AOMMSGCD

A system variable that indicates the message code assigned to this message, as set by the screening table.

As well as route codes and message levels, further restrictions on message delivery is performed by setting specific message codes.

A users profile is set with a message code mask to restrict message delivery to that user.

For more information about user profiles, see the *Security Guide*.

Default setting of 00 is assigned if message code is not set by AOM screening table.

Example: &AOMMSGCD

```
.LOOP &AOMREAD SET
&IF &AOMMSGCD NE 00 &THEN &GOTO .MSGRESTRICT
  .
  .
  .
```

Note: Message code masks provide a method of selective message delivery.

More information:

&AOMROUTC (see page 799)

&AOMMSGID

A system variable containing the extracted message ID of the current message.

&AOMMSGID provides an easy mechanism for identifying messages.

Some examples of the possible contents of this variable are, $HASP150,IEC450I, or IST097I.

The screening table SET statement can alter the value of the MSGID, if, for example, messages that do not follow standard MVS or VM message naming rules are encountered.

Example: &AOMMSGID

```
.LOOP &AOMREAD SET
&GOTO .&AOMMSGID
 .
 .
 .
.$HASP150
 .
 .
 .
&GOTO .LOOP
```

Notes:

&AOMMSGID is always taken from &AOMTEXT. For a multi-line WTO the contents of &AOMMSGID are obtained from the first or major line.

In VM, &AOMMSGID is derived from the first word of the message text.

The value in &AOMMSGID is normalized. This means that for unauthorized messages, which are normally indicated by a plus sign (+) in position 1, the plus sign (+) is removed. &AOMAUTH is used to determine if the message is authorized.

The maximum length of this system variable is 12 characters.

More information:

&AOMTEXT (see page 810)

&AOMMSGLV

A system variable that contains the highest message level of the current message.

Message levels is used to limit the messages that is delivered to a specific AOM authorized environment.

The possible values in &AOMMSGLV, in order of decreasing severity, are: WTOR, R, I, CE, E, BC and IN.

Example: &AOMMSGLV

```
&IF &AOMMSGLV = I &THEN &AOMCONT COLOR=RED HLIGHT=REVERSE
```

Notes:

Message levels is selected/modified in the AOM screening table.

Authorized AOM receivers can profile their environment to receive one or more message levels.

More information:

&AOMROUTC (see page 799)

&AOMMVCON

A system variable that indicates the ID of the system console to which the current message was routed.

&AOMMVCON contains the system console ID of the console to which the current message is being routed. This is a console acquired by your product region for AOM.

The &AOMMVCON value can range from 0 to 255. It is null for VM-sourced messages.

Example: &AOMMVCON

```
.LOOP
&AOMREAD SET &IF &AOMMVCON GT 2 &GOTO .PERIPHCONS
  .
  .
  .
```

Notes:

Messages with a non-zero &AOMMVCON value are always regarded as solicited. The GLOBAL statement in the screening table must specify SOLICIT=YES to allow delivery of solicited messages to AOMPROC.

If using Extended MCS consoles, this variable can contain zero, but the message could still be directed at a console. This is because the target console cannot have a 1-byte ID.

More information:

&AOMNMCON (see page 791)
&AOMCONNM (see page 758)

&AOMMVSDL

A system variable that indicates whether the screening table has deleted the current message.

&AOMMVSDL is set to YES if the message was deleted, else it is set to NO. Messages being processed by the AOM screening table travel along two separate paths. One path is for delivery to Automation Services, and the other is for delivery to system consoles.

This system variable is used to determine if the message has appeared on system consoles or if it was deleted by the screening table.

Example: &AOMMVSDL

```
&IF &AOMMVSDL = YES &THEN &MVSSTATS = &MVSSTATS + 1
  .
  .
  .
```

Notes:

As indicated by the example, &AOMMVSDL is used to keep statistics on the number of messages suppressed by the AOM screening table. For this figure to be accurate, any messages deleted from the path must flow through to Automation Services.

For messages sourced by AOM/VM, this system variable is null.

More information:

&AOMMVCON (see page 789)
&AOMNMCON (see page 791)

&AOMNMCON

A system variable that contains the Automation Services console ID to which the current message was routed.

&AOMNMCON contains the Automation Services console ID of the console to which the current message is being routed.

The possible values are 0 to 255. For VM-sourced messages, &AOMNMCON is null.

Example: &AOMNMCON

```
...
&IF &AOMNMCON GT 0 &THEN &GOTO .CONSTATS
 .
 .
 .
-* Maintain statistics on solicited traffic
-* from AOM consoles.
.CONSTATS &CONSTATS = &CONSTATS + 1
 .
 .
 .
```

Notes:

Any messages delivered to a specific Automation Services console are regarded as solicited. The GLOBAL statement in the screening table must specify SOLICIT=YES to allow delivery of solicited messages to AOMPROC.

Note: For more information, see the SHOW CONSOLES command in the *Reference Guide*.

More information:

&AOMMVCON (see page 789)

&AOMNMDOM

Returns the internal DOM identifier associated with a DOM-notify message.

If an AOMPROC executes an &AOMCONT, &AOMDEL or &AOMREPL verb with the DOM-NOTIFY=YES operand specified, a DOM-Notify message is enqueued to that AOMPROC, when an eventual MVS DOM is received. If the message had the NRD=YES attribute, this system variable contains the assigned DOMID that was assigned to the message.

&AOMNMDOM is set following the &AOMREAD that receives the DOM-Notify message when the MVS DOM is received.

Example: &AOMNMDOM

```
&AOMREAD SET
&IF &AOMDOM EQ YES &THEN &DO
 .  -*process &AOMDOMID and &AOMNMDOM...
 .
 .
 .
&DOEND
 .
 .
 .
```

Notes:

This system variable is null unless the current message being processed by an AOMPROC is a DOM-Notify. It is also null if the original message was not NRD=YES, since no DOMID was assigned.

The AOMPROC need not issue the DOM (using &NRDDEL); AOM does this automatically.

More information:

&AOMDOMID (see page 762)
&AOMCONT (see page 48)
&AOMREPL (see page 69)
&AOMDEL (see page 55)

&AOMNMIN

A system variable that contains the number of minor lines in a multi-line WTO.

&AOMNMIN contains the number of minor lines in a multi-line WTO message when any line of the message is current. It is used as the upper bound in a loop that reads all the minor lines (using &AOMREAD MINOR=YES), or accesses minor lines using the &AOMMINLN built-in function.

Example: &AOMNMIN

```
&AOMREAD SET MINOR=NO
&IF &AOMMAJOR = YES &THEN &DO
   &I = 1
   &DOWHILE &I LE &AOMNMIN
   &LINE&I = &AOMMINLN &I
   &I = &I + 1   &DOEND
&DOEND
 .
 .
 .
```

Note: This system variable is null when the current message is not part of a multi-line WTO.

More information:

&AOMREAD (see page 65)
&AOMMINLN (see page 63)
&AOMMINLT (see page 64)

&AOMNRD

A system variable that indicates whether or not the current message is to be displayed as a non-roll-delete message on OCS consoles.

&AOMNRD is set to YES or OPER for a non-roll-delete message, otherwise NO.

A non-roll-delete message is a message that remains on the OCS screen until some action is performed that enables the message to be deleted. For example, a tape mount message rolls off the screen when the tape is mounted.

Example: &AOMNRD

```
.LOOP
&AOMREAD SET
&IF .&AOMNRD = .YES &THEN &GOTO .NRDMSG
 .
 .
 .
-* Change all NRDs to roll-delete.
.NRDMSG
&AOMCONT NRD=NO
 .
 .
 .
 &GOTO .LOOP
```

Notes:

For every NRD=YES message, there is an associated Delete-Operator-Message ID (DOMID), which is a correlation number generated with the message. A DOM is issued against the NRD message to mark it as deletable when the required action has been performed or when the job or step terminates.

NRD=OPER messages do not have associated DOMIDs. When an OCS user deletes the message from the screen, it is not recallable; the system retains no memory of the message.

More information:

&AOMDOM (see page 761)
&AOMDOMID (see page 762)

&AOMODID

A system variable that contains the domain ID of the Automation Services system where the message originated, as set by the NMDID JCL parameter.

In an ISR-connected AOM environment, an AOMPROC can tell where a message came from by referencing this variable. The &AOMODID system variable contains the domain ID of the message originator system.

If the message was sourced by the local system, the value is the same as the &ZNMDID system variable. Otherwise, it contains the domain ID of the originator, regardless of how many ISR links it traveled across to arrive here.

Example: &AOMODID

```
.LOOP
&AOMREAD SET
&IF &AOMODID NE &ZNMDID &THEN &GOTO .REMOTE
  .
  .
  .
```

Notes:

This system variable is set for all messages read by &AOMREAD. This includes DOM-notify, event, and so on.

The ISR connected system that actually delivered the message is determined by the &AOMLDID system variable.

More information:

&AOMLDID (see page 773)

&AOMRCLAS

A system variable that contains the ISR remote classes, as set by the screening table, in MASKCHK format.

If an AOMPROC needs to know or analyze the remote classes that are assigned to a message, the &AOMRCLAS system variable provides a formatted list of the eight classes, each being indicated as a single character, Y or N.

For example, if RMTCLASS=(1,6,8) is applied to a message, then &AOMRCLAS is set to YNNNNYNY.

Example: &AOMRCLAS

```
.LOOP
&AOMREAD SET
&IF &AOMRCLAS = YYYYYYYY &THEN &AOMCONT RMTCLASS=5
```

Notes:

Remote classes determine which ISR links a message or event is automatically delivered to.

The eight individual classes can also be referenced using the &AOMRCLAS1 to 8 system variables.

More information:

&AOMRCLS1-8 (see page 797)

&AOMRCLS1-8

Eight system variables providing the individual values of the eight AOM ISR remote classes for this message or event.

If an AOMPROC needs to know or analyze the remote classes that are assigned to a message, the system &AOMRCLS1....&AOMRCLS8 variables provide access to the individual values. Each is set to YES or NO.

For example, if RMTCLASS=(1,6,8) is applied to a message, then &AOMRCLS1 is YES, &AOMRCLS2 is NO, and so on.

Example: &AOMRCLS4

```
.LOOP
&AOMREAD SET &IF &AOMRCLS4 = YES &THEN &AOMCONT RMTCLASS=7
```

Notes:

Remote classes determine which ISR links a message or event is automatically delivered to.

The eight classes can also be referenced in MASKCHK format using the &AOMRCLAS system variable.

More information:

&AOMRCLAS (see page 796)

&AOMREISS

System variable that has the value YES if the current message was reissued on a JES3 GLOBAL processor, or across a sysplex, or on a VOS3/JSS4 global processor. Otherwise its value is NO.

Used to test for JES3 reissued messages on a GLOBAL processor

Example: &AOMREISS

```
&IF &AOMREISS = YES &THEN +
   -EXEC GLOBAUTO
```

This system variable is useful only in a multi-CPU environment.

Note: For more information, see the REISSUED screen table criterion.

&AOMRHEX

A system variable that contains the routing code(s) assigned to the current message, in hexadecimal.

The routing codes(s) that is assigned to a message are in the range from 1 to 128. This is a sixteen-character system variable containing the hexadecimal representation of the routing code(s).

For example, &AOMRHEX contains E020000000000000 for ROUTCDE=(1,2,3,11), which is equal to binary 1110000000100000....

Example: &AOMRHEX

```
& CALL STATPROG &AOMRHEX
```

Note: &AOMRHEX is HEX packed before passing to a user program via &CALL by using the built-in function &HEXPACK.

More information:

&AOMRKEY (see page 799)
&AOMROUTC (see page 799)

&AOMRKEY

A system variable that returns the retrieval key attribute.

If AOMPROC wishes to use the retrieval key attribute of a message, this system variable contains the value. The value is 1 to 8 characters. Where not available, this system variable returns a null value.

Example: &AOMRKEY

```
&IF .&AOMRKEY NE .&THEN +
    &GOSUB .TRACK_RKEY
```

The value of the retrieval key is determined by the WTO issuer.

Note: For more information, see RKEY screening table criterion.

More information:

&AOMMSGID (see page 787)

&AOMRKEY

A system variable that contains the routing code(s) assigned to the current message, in &MASKCHK format.

The routing codes(s) that can be assigned to a message are in the range from 1 to 128. This is a 128-character system variable containing 128 Y/N values representing the routing code(s).

For example, ROUTCDE=(1,2,3,11) yields a 128-byte value of YYYNNNNNNNYNNNNN....N.

Example: &AOMRMASK

```
&ROUTECODE = &MASKCHK YYYNNNNNNNYNNNNN &AOMRMASK
```

Note: &AOMRMASK can be used to identify invalid route codes.

More information:

&AOMRHEX (see page 798)
&AOMROUTC (see page 799)

&AOMROUTC

A system variable that contains the routing code(s) assigned to the current message.

&AOMROUTC is set with the routing code(s) of the current message enclosed in parentheses, for example, (1,3,11).

Example: &AOMROUTC

```
.LOOP &AOMREAD SET
&GOTO .&AOMID
 .
 .
 .
.PREPMSG
&WRITE RC=&AOMROUTC NRD=OPER +
   DATA=PLEASE PREPARE PRINTER 1 +
    FOR SPECIAL PRINT - AWZ001
&AOMCONT
&GOTO .LOOP
```

Note: As is seen from the example, &AOMROUTC is formatted so it is inserted directly into an &WRITE, &WTO, &WTOR, or &AOMALERT statement.

More information:

&AOMRHEX (see page 798)
&AOMRKEY (see page 799)

&AOMROUTE

A system variable that contains the routing option for the current message, as set by the screening table.

A message processed in the AOM screening table can go down two paths. One path relates to system console delivery while the other relates to Automation Services delivery.

The delivery of a message to Automation Services is specified by the ROUTE operand in the screening table. See the description of the GLOBAL, SET, and MSGGROUP screening table statements.

&AOMROUTE contains the local delivery ROUTE option.

Example: &AOMROUTE

```
.LOOP
&AOMREAD SET
&GOTO .&AOMID
 .
 .
 .
.SPECMSG &IF &AOMROUTE = PROC &THEN &AOMCONT COLOR=RED
 .
 .
 .
&GOTO .LOOP
```

Notes:

If &AOMROUTE contains BOTH, then the message has already been delivered to relevant OCS screens.

This system variable always contains the same value as the &AOMLROUT system variable.

More information:

&AOMLROUT (see page 774)
&AOMRROUT (see page 801)

&AOMRROUT

A system variable that contains the remote routing option for the current message, as set by the screening table.

A message processed in the AOM screening table can go down two paths. One path relates to system console delivery, while the other relates to Automation Services delivery.

The delivery of a message to Automation Services is specified by the ROUTE operand in the screening table. See the description of the GLOBAL, SET, and MSGGROUP screening table statements in the relevant chapter. For ISR delivery, the routing option to be used at the other end of a link can also be set.

&AOMROUTE contains the remote delivery ROUTE option.

Example: &AOMRROUT

```
.LOOP
&AOMREAD SET
&GOTO .&AOMID
 .
 .
 .
.SPECMSG
&IF &AOMRROUT = NO &THEN &AOMCONT RMTROUTE=PROCONLY
 .
 .
 .
&GOTO .LOOP
```

Note: If &AOMRROUT contains NO and is not overridden, no automatic ISR delivery occurs.

More information:

&AOMLROUT (see page 774)
&AOMROUTE (see page 800)

&AOMRWTOR

A system variable that indicates whether or not the current message is a Replied-to-WTOR.

The AOM screening table can process a WTOR message by issuing the outstanding reply text.

If a WTOR has been replied to by the screening table and the message was allowed to continue through to AOMPROC, this variable is set to YES. In all other cases, it is set to NO.

Example: &AOMRWTOR

```
.LOOP
&AOMREAD SET &IF &AOMRWTOR = YES &THEN &GOTO .REPLIEDTO
 .
 .
 .
-* Set message with under score to denote replied-to-wtor
.REPLIEDTO
&AOMCONT HLIGHT=USCORE
&GOTO .LOOP
```

Note: If the WTOR statement in the screening table does not have the CONTINUE option coded, WTORs which have been replied to are not delivered to Automation Services and are not seen by AOMPROC.

&AOMSALRT

A system variable that indicates whether or not the current message was sourced by the &AOMALERT verb.

If an AOMPROC needs to know whether or not a particular message is sourced by the &AOMALERT verb, this system variable is used. Thus, AOMPROC can prevent critical actions being taken because of counterfeit messages.

&AOMSALRT contains YES if the current message was sourced by an &AOMALERT verb. In all other cases, it is set to NO.

Example: &AOMSALRT

```
.LOOP
&AOMREAD SET
&IF &AOMSALRT = YES &THEN &GOTO .LOOP -* ignore
 .
 .
 .
```

Note: There is no way to alter the setting of this system variable. It is always set to YES for messages sourced by &AOMALERT, and to NO for all messages originating from the operating system.

More information:

&AOMALERT (see page 35)

&AOMSDATA

A system variable that contains the saved data from a successful LOOKUP statement.

The AOM screening table allows the DATA1 field of a mirrored VARTABLE to be saved when a match is found by a LOOKUP statement. This system variable contains the saved value.

Example: &AOMSDATA

```
.LOOP
&AOMREAD SET
&IF &AOMID = QPROC &THEN &AOMCONT NCLID=&AOMSDATA
 .
 .
 .
```

Note: An AOMPROC can maintain any useful data it likes in the DATA1 field of a mirrored VARTABLE, for use in any way it sees fit. The example above keeps the NCL IDs of secondary AOMPROCs. The key could have been a MSGID.

More information:

&VARTABLE (see page 657)

&AOMSINGL

A system variable that indicates whether or not the current message is a single line message.

Set to YES for a single line WTO or WTOR, and set to NO for multi-line WTOs.

Example: &AOMSINGL

```
.LOOP
&AOMREAD SET
&IF &AOMSINGL = NO &THEN &GOTO .MULTILINE
 .
 .
 .
.MULTILINE-* Multi-line processing
 .
 .
 .
&GOTO .LOOP
```

Note: &AOMSINGL is used in conjunction with &AOMMAJOR and &AOMMINOR to manipulate or enhance multi-line WTOs.

More information:

&AOMATEXT (see page 751)
&AOMMAJOR (see page 780)
&AOMMINOR (see page 782)
&AOMTEXT (see page 810)

&AOMSOLIC

A system variable that indicates whether or not the current message is a solicited message.

This variable is set to NO for all unsolicited messages. Any messages that have been solicited via an operating system command cause &AOMSOLIC to be set to YES.

Example: &AOMSOLIC

```
&IF &AOMSOLIC = YES &THEN &AOMDEL
```

Notes:

The screening table must have SOLICIT=YES on the GLOBAL statement for solicited messages to be received by AOMPROC.

All authorized AOM users receive the messages if AOMPROC allows solicited messages to be delivered. The user that issued the command can receive the messages twice since the messages are routed to the AOM console assigned to that environment as well as to AOMPROC.

The &AOMSOLTP system variable indicates the type of issuer that solicited the message.

More information:

&AOMSOLTP (see page 807)

&AOMSOLTP

A system variable that contains the solicit type of the current message.

This variable is set to NO for all unsolicited messages. Any messages that have been solicited via an operating system command cause &AOMSOLIC to be set to one of the following values:

NM

Solicited by an Automation Services user

TABLE

Solicited in response to a screening table REPLY or ISSUE statement

OTHER

Solicited by some other user

Example: &AOMSOLTP

```
& IF &AOMSOLIC = YES AND &AOMSOLTP = OTHER &THEN +
    &GOTO .SOLLOG
```

Notes:

The screening table must have SOLICIT=YES on the GLOBAL statement for solicited messages to be received by AOMPROC.

If an AOMPROC allows solicited messages to be delivered, then all authorized AOM users receive the messages. The user that issued the command receives the messages twice, since the messages are routed to the AOM console assigned to that OCS screen as well as AOMPROC.

The &AOMSOLIC system variable also indicates whether or not a message is solicited.

More information:

&AOMSOLIC (see page 806)

&AOMSOS

A system variable that identifies the type of operating system that sourced this message.

This system variable contains a value identifying the operating system that sourced this message. The following values are possible:

OS

Sourced by z/OS, MSP, or VOS3

VM

Sourced by VM

Examples:

```
& IF &AOMSOS = VM &THEN &GOTO .VMPROC
```

Note: Because an AOMPROC can receive messages from other systems via an ISR link, the value in this system variable is important in a mixed operating system network.

More information:

&ZMAOMSOS (see page 924)

&AOMSUBT

A system variable containing the subtype of the current line of the current message.

&AOMSUBTP is a two-character variable containing the hexadecimal representation of the bit settings used by AOM to describe the current message. Possible values for &AOMSUBTP are:

80

On for all WTOs (z/OS) or messages (z/VM)

40

On for WTOR

20

ON with 80 if a single line message

10

ON with 80 if a major line

08

ON with 80 if a minor line

04

Force routed for Automation Services console 02 - nn or master console messages

02

Forced routing

01

Replied-to-WTOR

Notes:

&AOMSUBTP can contain one or a combination of the above settings, for example, A0 - WTO (80), single line message (20).

&AOMSUBTP is helpful for debugging purposes.

&AOMTEXT

A system variable containing the major text of the current message.

This variable contains the text of an incoming message. If the message is a multi-line WTO/WTOR, then the text is taken from the first or major line. The text in &AOMTEXT does not change for following minor lines. For VM-sourced messages, &AOMTEXT contains the message text.

&AOMATEXT contains the minor line text associated with the major line in &AOMTEXT. For the first or major line, &AOMTEXT has the same contents.

Example: &AOMTEXT

```
&AOMDEL
&WRITE RC=&AOMROUTC DATA=&AOMTEXT
```

Notes:

The three system variables &AOMMAJOR, &AOMMINOR, and &AOMSINGL is used to determine whether or not the current line is from a multi-line WTO and whether the current line is a major or minor line.

Following is a table of the possible settings for all these variables:

&AOMMAJOR	&AOMMINOR	&AOMSINGL	&AOMTEXT
NO	NO	YES	SINGLE LINE text
YES	NO	NO	CURRENT LINE text
NO	YES	NO	FIRST/MAJOR LINE text

&AOMSINGL is also set to YES for WTOR messages.

The text returned includes any screen character. For more information, see the &AOMCHAR1 description.

More information:

&AOMATEXT (see page 751)
&AOMMAJOR (see page 780)
&AOMMINOR (see page 782)
&AOMCHAR1 (see page 756)
&AOMSINGL (see page 805)

&AOMTIME

A system variable containing the timestamp of the current message.

&AOMTIME is set to the time that the current message was generated, in the form *hhmmss*.

Examples: &AOMTIM

```
.LOOP
&AOMREAD SET
&GOTO .&AOMID
 .
 .
 .
&GOTO .LOOP

.PRODJOB3
&IF &AOMTIME GT &TIME3 &THEN &WRITE AOM=YES NRD=OPER +
       DATA=WARNING JOB &AOMJOBNM IS RUNNING LATE.
 .
 .
 .
&GOTO .LOOP
```

Note: &AOMTIME is used to compare the time difference between delivery of a particular non-roll delete message and its associated DOM.

&AOMTYPE

A system variable identifying the current message as a WTO, WTOR, DOM, or EVENT.

This variable is set to either WTO, WTOR, EVENT, or DOM, indicating that the current message is a WTO (or VM MSG), WTOR, screening table, or &AOMALERT generated EVENT, or MVS DOM-NOTIFY message.

Example: &AOMTYPE

```
.LOOP
&AOMREAD SET
&GOTO .&AOMTYPE
 .
 .
 .

.WTO
&AOMCONT
&GOTO .LOOP
-* Give WTOR to 'worker' procedure for analysis and reply.
.WTOR
-INTQ ID=&WRK1 &AOMWRID &AOMWRLEN &AOMATEXT
&AOMCONT
&GOTO .LOOP
```

Note: This variable can have a value of DOM only if a previous MVS-sourced message was released from an AOMPROC by &AOMCONT/REPL/DEL DOM-NOTIFY=YES.

More information:

&AOMWTO (see page 820)
&AOMWTOR (see page 821)
&AOMDOM (see page 761)

&AOMUFLGS

A system variable containing the eight user flags in &MASKCHK format.

&AOMUFLGS contains a string of eight characters which match the settings of the eight user flags (&AOMUFLG1-8) with Y or N. For example:

```
YNNNYNNY
```

The built-in function &MASKCHK is used to test the settings.

Example: &AOMUFLGS

```
&USRFLAG3 = &MASKCHK **Y***** &AOMUFLGS
&IF &USRFLAG = EQ &THEN &FLAG3 = ON
```

Notes:

&AOMUFLG1-8 are user definable flags set in the screening table.

The user flag mask is available to AOM message receivers in the system variable &ZMAOMUFM.

More information:

&AOMUFLG1-8 (see page 814)

&AOMUFLG1-8

These are eight system variables which are user-defined flags, set by the screening table.

The default for the system variables &AOMUFLG1.....&AOMUFLG8 is NO. Each is set to YES by the screening table or reset via &AOMCONT or &AOMREPL.

Example: &AOMUFLG1

```
.LOOP
&AOMREAD SET
&IF &AOMUFLG1 = YES &THEN &GOTO .SPECPROC
 .
 .
 .
&GOTO .LOOP
.SPECPROC  -* Special message processing
 .
 .
 .
&AOMCONT ROUTCDE=15
&GOTO .LOOP
```

Note: The user flag mask is available to AOM message receivers in the system variable &ZMAOMUFM.

More information:

&AOMUFLGS (see page 813)

&AOMVMMCL

The VM IUCV message class of a VM-sourced message.

Messages sourced by AOM/VM contain the VM *MSG IUCV message class.

Classes 1 to 8 are IUCV message types; class 30 is a programmable operator facility message type. Valid values are:

1

Message sent using CP MESSAGE and CP MSGNOH

2

Message sent using CP WARNING

3

Asynchronous CP messages, CP responses to a CP command executed by the programmable operator facility virtual machine, and any other console I/O initiated by CP

4

Message sent using CP SMSG command

5

Any data directed to the virtual console by the virtual machine (WRTERM, LINEDIT, and others)

6

Error messages from CP (EMSG)

7

Information messages from CP (IMSG)

8

Single Console Image Facility (SCIF) message from CP

30

Message coming from Automation Services

Example: &AOMVMMCL

```
.LOOP
&AOMREAD SET
&IF &AOMVMMCL = 30 &THEN &GOTO .PROPMSG
 .
 .
 .
&GOTO .LOOP
.PROPMSG  -* Special PROP-sourced message processing.
 .
 .
&AOMCONT ROUTCDE=15
&GOTO .LOOP
```

Note: &AOMVMMCL normally has the value '30'.

More information:

&AOMVMSRC (see page 816)

&AOMVMSRC

The AOM/VM message source.

Messages generated by AOM/VM can come from either the PROP IUCV connection or the GCS machine that Automation Services is running on.

Messages that come from the PROP connection have a source of PROP.

Messages that come from the GCS connection have a source of GCS. These are always solicited command responses.

Example: &AOMVMSRC

```
.LOOP
&AOMREAD SET
&IF &AOMVMSRC = GCS &THEN &AOMDEL
 .
 .
 .
```

Note: Because the GCS IUCV connection is used only to obtain responses to SYSCMD DEST=GCS, these messages are not normally seen by an AOMPROC unless the screening table is processing solicited responses.

More information:

&AOMVMMCL (see page 815)

&AOMVMUID

The VM user ID that a message originated from.

Messages generated by AOM/VM contain the user ID that originated the message. This user ID is the virtual machine name that issued the CP MSG command. For messages that originate from CP, the user ID is CP.

Example: &AOMVMUID

```
.LOOP
&AOMREAD SET
&IF &AOMVMUID = USER1 &THEN +
   &SYSCMD DEST=GCS MSG USER1 WHATS WRONG?
 .
 .
 .
```

Note: The user ID system variable is useful for replying to the originating user when a problem is solved.

More information:

&AOMVMUND (see page 818)

&AOMVMUND

The VM RSCS node that a message originated from.

Messages generated by AOM/VM contain the name of the RSCS node that the message originated from. For CP-generated messages in the local system, this has a value of CP.

In a networked VM system, this field is useful for identifying the original source of a message.

Example: &AOMVMUND

```
.LOOP
&AOMREAD SET
&IF &AOMVMUND NE VM1 &THEN &GOTO .REMOTEVM
 .
 .
 .
```

Note: If VM RSCS networking is not used, this field is not useful.

More information:

&AOMVMUID (see page 817)

&AOMWRID

A system variable containing the WTOR reply ID of the current message.

&AOMWRID contains a number, normally from 00 to 99, which is the reply ID of the current message. It is 3 or 4 digits long.

This variable is used from AOMPROC to automate replies.

Example:

```
.LOOP
&AOMREAD SET
&GOTO .&AOMTYPE
 .
 .
 .
&GOTO .LOOP
-* Pass WTOR to 'worker' procedure for analysis and reply.

-* Reply Command is SYSCMD R &AOMWRID,GO

.WTOR
&AOMCONT NCLID=&WRK1
&GOTO .LOOP
```

Note: If the current message is not a WTOR, &AOMWRID is null.

More information:

&AOMTYPE (see page 812)
&AOMWRLEN (see page 820)

&AOMWRLEN

A system variable containing the length of the text that is passed in reply to a WTOR.

This system variable contains the maximum length of data that is used for a reply to a WTOR.

When automating replies from AOMPROC this variable is used to ensure the reply is not rejected because the text is too long.

Example: &AOMWRLEN

```
 .
 .
 .
&RLEN = &LENGTH &RTEXT
&IF &RLEN GT &AOMWRLEN &THEN &GOTO .ERROR
 .
 .
 .
```

Note: If the current message is not a WTOR, &AOMWRLEN is null.

More information:

&AOMWRID (see page 819)
&AOMTYPE (see page 812)

&AOMWTO

A system variable that indicates whether or not the current message is a Write to Operator (WTO).

For messages generated by a z/OS system, this system variable is set to YES if the current message was generated by the WTO macro; otherwise, it is set to NO. For messages sourced by AOM/VM, &AOMWTO is always YES.

Example: &AOMWTO

```
&IF &AOMWTO EQ YES &THEN &GOTO .WTOPROCESS
 .
 .
 .
.WTOPROCESS
&AOMCONT &GOTO .AOMREAD
```

More information:

&AOMDOM (see page 761)
&AOMNRD (see page 794)
&AOMRWTOR (see page 802)
&AOMTYPE (see page 812)
&AOMWTOR (see page 821)

&AOMWTOR

A system variable that indicates whether or not the current message is a Write to Operator with Reply (WTOR).

This system variable is set to YES if current message was generated by the WTOR macro; otherwise, it is set to NO.

Example: &AOMWTOR

```
&IF &AOMWTOR = YES &GOTO .REPLYWTOR
  .
  .
  .
```

More information:

&AOMDOM (see page 761)
&AOMNRD (see page 794)
&AOMRWTOR (see page 802)
&AOMTYPE (see page 812)
&AOMWTO (see page 820)

&BROLINEn

A series of system variables that contain the current broadcast lines.

The system supports a maximum of four broadcast lines for use by the EASINET feature. They are:

- &BROLINE1
- &BROLINE2
- &BROLINE3
- &BROLINE4

The text of these broadcast lines is set using Broadcast Services.

The &BROLINE1, &BROLINE2, &BROLINE3, and &BROLINE4 system variables is included in any full-screen panel. When this panel is displayed, the current broadcast text associated with that line (if any) is displayed in place of the &BROLINE*n* variable.

If the panel is displayed by the EASINET component, use of any one of these variables signifies that the panel is to receive updated broadcasts as they are dispatched from Broadcast Services.

Examples: &BROLINE*n*

```
&BROLINE1
&BROLINE2
```

Notes:

The &BROLINE*n* variables is positioned without restriction on the panel. The maximum text that is displayed for a broadcast line is 78 characters. Therefore, the &BROLINE*n* variables are normally aligned at the left hand side of the panel. If there is insufficient room for the text, it is truncated.

The &BROLINE*n* variables apply to both general and specific broadcasts.

Under EASINET, a broadcast causes immediate redisplay of any candidate panel that contains any of the &BROLINE variables, if the panels are displayed by EASINET. Users logged on to a region have these variables refreshed at the next panel display.

Note: For more information, see the #OPT statement, described in the chapter "Designing Interactive Panels (Panel Services)" in the *Network Control Language Programming Guide* and the $EASILOGON panel in the panels data set.

More information:

About Broadcast Services (see page 1037)

&CURSCOL and &CURSROW

System variables that provide the cursor location.

These two system variables is used to determine the cursor row and column coordinates as at the last operator input from a panel displayed using the &PANEL statement.

&CURSROW is set to the number of the row that contained the cursor. When operating in split screen mode, the row is relative to the current window, regardless of where it commences on the physical screen.

&CURSCOL is set to the number of the column that contained the cursor. When operating in split screen mode, the column is relative to the current window, regardless of where it commences on the physical screen.

Note: If the last entry was caused by the INWAIT timer expiring, the value returned in

&CURSCOL/&CURSROW is indeterminate.

Examples: &CURSCOL and &CURSROW

```
&IF &CURSROW > 3 AND &CURSCOL > 6 &THEN +
    &GOSUB .XRACTSEL

&IF .&SYSMSG NE .  AND &CURSROW EQ 3 &THEN +
    &GOSUB .MSGHELP
```

These variables are designed to be used in conjunction with the CURSOR operand of the #OPT statement (of Panel Services) to effect precise cursor positioning to locations other than input fields.

Note: For information about using these facilities, see the *Network Control Language Programming Guide*.

&DATEn

This is a set of system variables which provide different formats of the current system date.

&DATE1 to &DATE17 (excluding &DATE15) supply the current system date in a variety of formats, as listed below:

- &DATE1—date as YY.DDD
- &DATE2—date as DAY DD-MON-YYYY
- &DATE3—date as DD-MON-YYYY
- &DATE4—date as DD/MM/YY
- &DATE5—date as MM/DD/YY
- &DATE6—date as YY/MM/DD
- &DATE7—date as YYMMDD
- &DATE8—date as YYYYMMDD
- &DATE9—date as *nnnnnn*
- &DATE10—date as YYYYMMDDHHMMSSpHHMM
- &DATE11—date as YYYYMMDDHHMMSS.FFFFFFpHHMM
- &DATE12—date as DD/MM/YYYY
- &DATE13—date as YYYY/MM/DD
- &DATE14—date as MM/DD/YYYY
- &DATE16—date as YYYY.DDD
- &DATE17—date as YYYYDDD

where:

DAY

Is the day of the week as follows:

- MON Monday
- TUE Tuesday
- WED Wednesday
- THU Thursday
- FRI Friday
- SAT Saturday
- SUN Sunday

DD

The day of the month as a 2-digit number

DDD

The Julian day within the year as a 3-digit number

MM

The month of the year as a 2-digit number

MON

The month of the year as follows:

- JAN January
- FEB February
- MAR March
- APR April
- MAY May
- JUN June
- JUL July
- AUG August
- SEP September
- OCT October
- NOV November
- DEC December

nnnnnn

The number of days from 1 January 0001 with no leading zeros

YYYY

The current year as a 4-digit number

YY

The current year as a 2-digit number

p

Plus or minus relative to Greenwich Mean Time (GMT)

FFFFFF

The time accurate to 10^{-6}

HHMMSS

The current time

HHMM

The GMT offset

Examples: &DATE*n*

```
&IF &DATE1 GT 98.001 &THEN +
 &END

&WRITE DATA=TODAY'S DATE IS &DATE2

&TODAY = &SUBSTR &DATE3 1 6
&IF &TODAY EQ 25-DEC &THEN +
    &GOTO .XMASDAY
```

Notes:

The current day is provided in the system variable &DAY.

&DATE6, &DATE7, &DATE8, and &DATE9 are useful where records are to be stored in chronological order.

See the &DATECONV function for the rules that apply to choice of century when converting from a form of the date that expresses the year in two digits (YY) to the form of the year in four digits (YYYY).

Each access to &DATE11 causes the system to re-fetch and synchronize time with the operating system, to format the result to microsecond accuracy. Use of this time should be avoided, to reduce unnecessary overheads, if such accuracy is not required.

More information:

&ZGDAY (see page 896)

&DAY

Provides the current day of the week.

&DAY provides a system variable for the current day of the week in the form DDD, where DDD is set to one of the following values:

MON

Monday

TUE

Tuesday

WED

Wednesday

THU

Thursday

FRI

Friday

SAT

Saturday

SUN

Sunday

Example: &DAY

```
&IF &DAY EQ SUN &THEN -EXEC SUNDAY&ELSE +
    -EXEC EVERYDAY
```

Notes:

The current date, based on the operating system time, is provided in different formats by the system variables &DATE1 to &DATE17.

The current date, based on GMT, is provided in different formats by the system variables &ZGDATE1 to &ZGDATE17.

&FILEID

Contains the name of the file currently being processed.

This is the file most recently actioned by either &FILE OPEN, &FILE PUT, &FILE ADD, &FILE SET, &FILE GET or &FILE DEL

Example: &FILEID

```
&FILE OPEN ID=FILE1 FORMAT=DELIMITED
&WRITE DATA=OPENED FILE : &FILEID
```

Note: When a file is closed, the contents of &FILEID revert to the name of the file most recently processed before closing the current file. If no other files are currently open, &FILEID is set to null.

More information:

&FILE (see page 321)

&FILEKEY

Indicates an NCL process's current position within a UDB.

This variable is set to the value of the full key of the last record read from the UDB identified by the last &FILEID statement issued by an NCL process. It can therefore be used to refer to the explicit key of each record read from a UDB when a file is being read using partial keys.

&FILEKEY reflects the private position of a process within its currently active file (which is the last file referenced on an &FILE statement).

Example: &FILEKEY

```
&FILE OPEN ID=HELPDESK FORMAT=DELIMITED -* open our file
&FILE SET ID=HELPDESK KEY='&1' -* set required key
&FILE GET ID=HELPDESK OPT=KGT VARS=TXT -* read that record
&IF &FILERC = 0 &THEN +
    &WRITE DATA=READ FOR RECORD &FILEKEY
```

Note: If the &FILEKEY variable can contain non-printable characters, it is recommended that it not be used directly as a parameter when invoking other procedures. In this case, use &HEXEXP to produce a character representation of the key, and pass this value.

More information:

&FILE (see page 321)

&FILERC

Indicates the success or otherwise of a file processing function.

A return code is set after the execution of the &FILE ADD, &FILE DEL, &FILE GET, &FILE OPEN, and &FILE PUT file processing statements. This return code is placed in the &FILERC system variable, which can then be tested to determine whether the operation was successful. The meaning of the various return codes is as follows:

For &FILE ADD:

0

Record added successfully.

1

Record added; truncation has occurred.

4

Record already exists (not replaced).

8

Error occurred; &VSAMFDBK is set.

16

NCL or Mapping Services processing error; &SYSMSG is set.

For &FILE DEL:

0

Record deleted successfully.

4

Record not found.

8

Error occurred; &VSAMFDBK is set.

16

NCL or Mapping Services processing error; &SYSMSG is set.

For &FILE GET:

0

Record retrieved successfully.

4

Record not found or end of data.

8

Error occurred; &VSAMFDBK is set.

16

NCL or Mapping Services processing error; &SYSMSG is set.

For &FILE OPEN:

0

Procedure is restricted to read only access.

4

Procedure is restricted to read and update access without delete authority.

8

Procedure is not restricted. Read, update and delete are authorized. If no authorization exit (NCLEX01) is in effect, then this value is set if the file is available for processing.

12

No access is authorized.

16

Specified file ID is not available for processing.

For &FILE PUT:

0

Record added or replaced successfully.

1

Record added; truncation has occurred.

4

Reserved for future use.

8

Error occurred; &VSAMFDBK is set.

16

NCL or Mapping Services processing error; &SYSMSG is set.

Example: &FILERC

```
& FILE OPEN ID=HELPDESK FORMAT=DELIMITED

&IF &FILERC EQ 16 &THEN +
    &ENDAFTER &WRITE DATA=NOT AVAILABLE

&FILE SET ID=HELPDESK KEY='PROB005'

&FILE GET ID=HELPDESK OPT=KEQ VARS=TXT

&IF &FILERC NE 0 &THEN +
    &WRITE DATA=RECORD NOT FOUND
```

Notes:

Simplify the testing for &FILERC and branching to the appropriate processing routine by using direct branching. For more information, see the *Network Control Language Programming Guide*.

For example:

```
&FILE GET ID=HELPDESK OPT=KEQ VARS=TXT

&GOTO
.GET&FILERC

.GET0
&ENDAFTER &WRITE DATA=RECORD RETRIEVED SUCCESSFULLY

.GET4
&ENDAFTER &WRITE DATA=RECORD NOT FOUND

.GET8
&ENDAFTER &WRITE DATA=ERROR VSAM CODE=&VSAMFDBK
```

The &VSAMFDBK system variable is also set on completion of a file processing operation, and is used to determine the exact cause of a VSAM-related error.

The values set in &FILERC after an &FILE OPEN statement are determined by the NCL file ID authorization exit NCLEX01. NCLEX01 is invoked the first time each new &FILE OPEN statement is referenced in a procedure. The name of the invoked exit is determined by the SYSPARMS command NCLEX01 operand. If no exit is in effect, only values of 8 (to authorize full access) and 16 (to indicate that the specified file is not available for processing) are set.

When &FILERC returns 16, &SYSMSG contains a message explaining the error. A value of 16 is also set for Mapping Services processing errors. This should occur only when in mapped processing mode.

Note: For more information, see the examples in the distribution library.

More information:

&VSAMFDBK (see page 866)

&FILERCNT

Provides a count of the number of records deleted by &FILE DEL processing.

NCL File Processing allows records to be deleted from a User DataBase (UDB) using the &FILE DEL statement, including support for the deletion of groups of records with a single statement. The deletion of groups of records is termed 'generic' deletion.

Generic deletion is triggered by use of the OPT=KEQALL or OPT=KGEALL operands on the &FILE DEL statement.

The &FILERCNT system variable provides a count of the number of records deleted during a generic process.

Example: &FILERCNT

```
&WRITE DATA=&FILERCNT RECORDS DELETED.
```

&FILERCNT remains unchanged until the next &FILE DEL or &FILE CLOSE statement.

Notes:

If no active file exists, &FILERCNT is set to 0.

If a non-generic deletion of a single record is performed, &FILERCNT is set to 1.

&FSM

Indicates if the issuing procedure has access to a real window.

Indicates whether the executing NCL process is directly associated with a real window, and therefore can issue an &PANEL statement. Any NCL procedure executing in an NCL processing environment that is associated with a real terminal window finds that &FSM returns a value of YES.

Any NCL procedure executing in an NCL processing environment which does not have an associated real terminal window (for example, the system background environments or ROF sessions) finds that &FSM returns a value of NO.

Example: &FSM

```
&IF &FSM = YES &THEN +
 &DO
     &PANEL MSGDISPLAY
     &END
 &DOEND
&WRITE DATA=&MSG1
&WRITE DATA=&MSG2
 .
 .
 .
```

&INKEY

The &INKEY system variable contains a value representing the key last used to enter data.

&INKEY is a system variable that is used to determine the last method of input to a procedure from either a full-screen panel or an LU1 type device.

&INKEY is set as follows:

- ENTER—Enter or Return key pressed.
- PF01 to PF24—program function key 1 through 24 pressed.
- PA1 to PA3—program attention key 1 through 3 pressed.
- Null—panel INWAIT or &PROMPT WAIT time period expired, no input made.

Program function key values (PF*nn*) and attention keys(PA*nn*) do not apply to an LU1 type terminal.

Program attention keys (PA1 to PA3) are available only to a procedure running with the &CONTROL PAKEYS option.

Typically, NCL procedures test &INKEY to determine the next action to take.

By default, certain function keys are allocated for use by the system and so are intercepted before reaching the NCL procedure. An example is F4, which is typically a return-to-menu key.

The &CONTROL PFKSTD, PFKALL, and NOPFK options allow the procedure to control the level of function key interception performed by the system. For example, &CONTROL PFKSTD indicates that function keys F3 and F4 are passed to the NCL procedure, but F2 and F9 continue to perform screen split and swap functions.

The &CONTROL PFKMAP option is used if function keys 13 through 24 are assigned the same functions as function keys 1 through 12. If this option is in effect, the NCL procedure is only required to cater for function keys 1 through 12. For instance, if a user presses the F13 key, NCL places the value PF01 in &INKEY; F15 results in PF03; and so on.

Examples: &INKEY

```
&IF .&INKEY EQ .  &THEN +
  &GOTO .TIMEOUT

&IF &INKEY EQ PF01 &THEN +
  &PANEL HELP
```

Notes:

If the panel has been defined with the #OPT control statement specifying a time interval on the INWAIT operand, and this time period has elapsed, &INKEY is set to null. Thus, it is possible for the procedure to determine whether validation is bypassed, and so on. &INKEY is also null if the time interval in the WAIT operand of an &PROMPT statement expires. Under these circumstances, some care must be taken in subsequent &IF statements that reference &INKEY, as it can have a null value and result in a syntax error. We recommend that you balance an &IF statement by using of an additional character that avoids such syntax errors when &INKEY is null, for example:

```
&IF X&INKEY EQ XPF01 &THEN +
 &GOTO .HELP-DISPLAY
```

An alternative method of determining this is the use of &CONTROL PANELRC to supply return codes to the invoking procedure on return from an &PANEL or &PROMPT statement. In this case, a return code of 12 in &RETCODE indicates that the INWAIT or WAIT time period has expired.

For function key entry, &INKEY is always four characters. Numbers below 10 always have a leading zero, for example: PF04.

The &INKEY value remains set until the next &PANEL statement.

If the full-screen environment associated with the NCL process is terminated (for example, by issuing &PANELEND), &INKEY returns a null value.

Note: For more information, see the $EASINET NCL procedure in the distribution library and the &CONTROL statement.

&LOOPCTL

Returns the current setting of the automatic loop control counter.

&LOOPCTL is a system variable that is used to determine the value of the system loop control counter current for the executing process. The system default value for &LOOPCTL is 1000, and this value decrements by one for each time a loop is executed. On exit from a loop, the counter returns to the default value. If the counter reaches0, then the process is terminated, on the assumption that it is looping uncontrollably.

Example: &LOOPCTL

```
&IF &LOOPCTL LT 10 &THEN +
  &GOTO .GIVEUP
```

Notes:

The &LOOPCTL verb is used to reset the loop control counter. You might need to do this if you expect to loop validly for a significant number of iterations during standard processing.

If your procedure has varying processing to perform, resulting in possible very lengthy iterations, you can check the decrementing value of &LOOPCTL and take action to avoid abnormal termination of the procedure.

More information:

&LOOPCTL (see page 405)

&LUCOLS

Indicates the number of columns currently allocated to this processing window.

NCL procedures displaying output might want to determine the width of the processing window. In this way, procedures is developed that cater for differing screen sizes.

The &LUCOLS system variable is tested to determine the number of display columns available to a processing window.

The system supports split screen operation with a maximum of two operational windows. When using split screen operation, the number of display columns available to the procedure might be less than the physical width for the screen. The &LUCOLS system variable always reflects the number of display columns for that window.

Example: &LUCOLS

```
&IF &MINWIDTH GT &LUCOLS &THEN +
  &GOTO .NOGOOD
```

Note: The dimensions of the physical terminal (regardless of the window dimensions) is determined from the &ZCOLS and &ZROWS system variables.

More information:

&LUROWS (see page 842)
&ZCOLS (see page 885)

&LUEXTCO

Indicates if the terminal supports extended color.

&LUEXTCO is a system variable that is used to determine if the terminal from which the NCL procedure is executing supports extended color facilities. If the terminal does support extended color &LUEXTCO is set to YES, if not &LUEXTCO is set to NO.

If the NCL procedure is executing in an environment which is not associated with a terminal (for example, a background environment), &LUEXTCO is set to a question mark (?).

&LUEXTCO applies only to IBM terminals which support full seven-color facilities or Fujitsu seven-color or three-color terminals.

&LUEXTCO is used within an NCL procedure to determine the type of processing possible for a particular terminal.

Example: &LUEXTCO

```
&IF &LUEXTCO EQ YES &THEN +
  &GOTO .7COLOR
```

Notes:

Testing of device attributes from within an NCL procedure displaying full-screen panels is not normally required as Panel Services automatically suppresses the generation of color data streams if not applicable to the device to which the panel is being sent. It is necessary if an NCL process is to display different panels depending on the terminal characteristics.

&LUEXTCO is set only if the BIND parameters for the terminal correctly indicate that the terminal supports READ PARTITION QUERY. If set, the system interrogates the terminal at connection time and determines which extended facilities are supported. Alternatively, the TERMINAL command is used to temporarily indicate that the terminal supports color.

More information:

&LUEXTHI (see page 840)

&LUEXTHI

Indicates if the terminal supports extended highlighting.

&LUEXTHI is a system variable that is used to determine if the terminal from which the NCL procedure is executing supports extended highlighting. If the terminal does support extended highlighting &LUEXTHI is set to YES; if not, &LUEXTHI is set to NO.

If the NCL procedure is executing in an environment which is not associated with a terminal (for example, a background environment), &LUEXTHI is set to a question mark (?).

&LUEXTHI is used within an NCL procedure to determine the type of processing possible for a particular terminal.

Example: &LUEXTHI

```
&IF &LUEXTHI EQ YES &THEN +
  &GOTO .BLINK
```

Notes:

Testing of device attributes from within an NCL procedure displaying full-screen panels is not normally required as Panel Services automatically suppresses the generation of extended highlighting data streams if not applicable to the device to which the panel is being sent.

&LUEXTHI is set only if the BIND parameters for the terminal correctly indicate that the terminal supports READ PARTITION QUERY. If set, the system interrogates the terminal at connection time and determines which extended facilities are supported. Alternatively, the TERMINAL command is used to temporarily indicate that the terminal supports extended highlighting.

More information:

&LUEXTCO (see page 839)

&LUNAME

Provides the name of the terminal at which the NCL procedure is executing.

&LUNAME provides a system variable for the name of the terminal at which the user is currently logged on or, in the case of the EASINET procedure, it provides the name of the terminal where the procedure is executing.

Example: &LUEXTHI

```
&IF &LUNAME EQ TERM1 &THEN +
  &GOTO .OK
```

The value returned from &LUNAME is a 1 to 8 character value.

Certain system environments use virtual user IDs and terminals names. The following values are returned for system environments:

BG-MON

background monitor

BG-LOG

background logger

BG-SYS

background system environment

AOM-PROC

AOMPROC procedure

CNM-PROC

CNMPROC procedure

LOG-PROC

LOGPROC procedure

PPO-PROC

PPOPROC procedure

CONSOLE

logical console associated with the sysoper user ID

CONS#*nn*

system console number nn

REMOTE

ROF user

Note: Other optional features that generate internal system environments allocate additional pseudo terminal names, which appear on a SHOW SESS command display as xxx-PROC, where xxx is the first three letters of the associated system level NCL procedure name.

&LUROWS

Indicates the number of rows currently allocated to this process window.

NCL procedures displaying multi-page output might want to determine the number of lines in the processing window. In this way, procedures is developed that cater for differing screen sizes.

It is good practice to write procedures that cater for the largest screen size (for example: 62 lines for a 3290) and that automatically adjust if used on a smaller screen.

The &LUROWS system variable is tested to determine the number of display lines available to a processing window.

When using split screen operation, the number of display lines available to the procedure might be less than the physical number for the screen. The &LUROWS system variable always reflects the number of display lines (rows) for that window.

When using &LUROWS, the NCL procedure must cater for any fixed overhead associated with a particular panel. For example, a panel can have a title on the top, followed by a line of column headings, and the third line blank. Data then commences on line 4 of the panel. The procedure must therefore allow for these three lines before attempting to calculate the number of display lines available for data.

Examples: &LUROWS

```
&IF &CNT LT &LUROWS &THEN +
  &GOTO .NEXTLINE
```

Notes:

When subtracting fixed panel overhead from &LUROWS, the NCL procedure must allow for &LUROWS having a value as low as when the window is not visible.

The dimensions of the physical terminal (regardless of the window dimensions) is determined from the &ZCOLS and &ZROWS system variables.

More information:

&LUCOLS (see page 838)
&ZROWS (see page 963)

&MAI#SESS

Returns the number of currently defined sessions. This is equivalent to &MAINSESS.

&MAIAE

Indicates the availability of the A and E primary commands. Is YES or NO.

&MAIAPPL

Returns the name of the application acting as the PLU on the MAI session.

Returns the name of the VTAM application acting as the PLU on the MAI session (that is, the application logged on to).

Example: &MAIAPPL

```
&IF &MAIAPPL = TSO &THEN &GOTO .TSO

&APPL = &SUBSTR &MAIAPPL 1 3
&IF &APPL = TSO &THEN &GOTO .TSO
```

Note: The value of &MAIAPPL can change during the session if the application performs a VTAM CLSDST/PASS operation to pass the MAI session to another application. For instance, &MAIAPPL might contain TSO at session start, then change to TSO0003 during the logon process.

&MAICCOLS

Returns the number of columns in the current MAI session's screen.

Provides screen size information to the script procedure.

Example: &MAICCOLS

```
& BUFFSIZE = &MAICROWS * &MAICCOLS
```

Note: Screen size is changed by the application issuing ERASE WRITE or ERASE WRITE ALTERNATE.

&MAICROWS

Returns the number of rows in the current MAI session's screen. Provides screen size information to the script procedure.

Example: &MAICROWS

```
& BUFFSIZE = &MAICROWS * &MAICCOLS
```

More information:

&MAICCOLS (see page 843)

&MAIDISC

Indicates whether MAI will honor a terminal disconnect request. Is YES or NO.

&MAIFRLU

Returns the direction of the last data stream.

Returns the direction of the last data stream as one of the following values:

PLU

The data stream last received was sent by the PLU, that is, the application.

SLU

The data stream last received was sent by the SLU, that is, the terminal.

Example: &MAIFRLU

```
&IF &MAIFRLU = SLU &THEN &GOTO .TERMINAL
```

Notes:

An &MAIFRLU issued when there is no data outstanding returns a null value. Data is outstanding from the time an &MAIREAD is satisfied until an &MAICONT, &MAIDEL, or another &MAIREAD is issued.

&MAIFRLU is most often used after an &MAIREAD ANY has been satisfied, so that the procedure can determine which data stream has been received.

More information:

&MAICONT (see page 407)
&MAIREAD (see page 419)
&MAIDEL (see page 411)

&MAIINKEY

Indicates the value of the key used to enter data.

Returns the value of the key used to enter data when a data stream is received from the SLU (terminal), as follows:

ATTN

the ATTN key

CLEAR

the CLEAR key

CARD

operator ID card

ENTER

the Enter key

MAG

magnetic card

NONE

no AID generated

PA1 to PA3

Program Attention Key 1 to 3

PF1 to PF24

Program Function Key 1 to 24

REQ

test request

SEL

selector pen attention

SF

inbound structured field

TRIG

trigger action

Example: &MAIINKEY

```
&IF &MAIINKEY = PF3 &THEN &GOTO .END
```

Notes:

When used to test the key used to enter data, an &MAIINKEY returns a null value if there is no outstanding data. Data is outstanding from the time an &MAIREAD is satisfied until an &MAICONT, &MAIDEL or another &MAIREAD is issued.

&MAIINKEY is sensitive to the setting of the &CONTROL PFKMAP option.

If &CONTROL PFKMAP is in effect, F3 to F24 are mapped into F1 to F12. For example, receipt of PF13 results in &MAIINKEY being set to a value of PF1.

More information:

&MAICONT (see page 407)
&MAIREAD (see page 419)
&MAIDEL (see page 411)

&MAILOCK

Indicates whether or not MAI will honor a terminal lock request. Is YES or NO. This represents the setting of the MAITLOCK system parameter.

&MAILU

Returns the name of the VTAM APPL being used by MAI as the secondary Logical Unit on the session.

Example: &MAILU

```
&IF &MAILU = MFMST &THEN &GOTO .MASTER
```

Note: Whenever MAI starts a session, it opens a VTAM ACB whose name is either chosen from a series or is specified by the user or an MAI exit. &MAILU returns the name of the VTAM ACB used for this session.

&MAIMNFMNT

Returns the current menu format as long or short.

&MAINSESS

Returns the number of currently defined sessions. This is equivalent to &MAI#SESS.

&MAIOCMD

Returns the command code contained in the first byte of the outbound data stream sent by the PLU. The following command codes is returned:

WRITE

WRITE

EW

ERASE/WRITE

EWA

ERASE/WRITE ALTERNATE

RM

READ MODIFIED

RMA

READ MODIFIED ALL

RB

READ BUFFER

WSF

WRITE STRUCTURED FIELD

EAU

ERASE ALL UNPROTECTED TO ADDR

&MAIREQ

Returns the MAI logon request.

Returns the value of the Logon Request field of the MAI Logon Details panel used to start the session.

Example: &MAIREQ

```
&IF &MAIREQ = TSOA &THEN &GOTO .TSOA
```

&MAISCANDL

Returns the scan limit for session commands.

&MAISID

Returns the session ID of the session on whose behalf the script is running.

Returns the session ID of the session on whose behalf the script procedure is running. The session ID is nominated on the Logon Details panel or allowed to default to the MID operand value of the DEFLOGON command used to provide the logon path.

Example: &MAISID

```
&IF &MAISID = TSO1 &THEN &GOTO .TSOFIRST
```

&MAISKIPP

Returns the system-wide value for the session command prefix character.

&MAISKPK1

Returns the session command function key 1 (for example, F12).

&MAISKPK2

Returns the session command function key 2 (for example, F24).

&MAISMODE

Returns the mode in which the script procedure is running.

Returns a value indicating the mode in which the script procedure is running (that is, how it was started). One of the following values is returned:

START

Indicates that the procedure was started at session start time. Parameters coded in the SCRIPT NCL PROC field of the Logon Details panel are passed as &1, &2, &3, and so on.

END

Indicates that the procedure was started at forced session end time. The procedure is driven under the following conditions: the user ends the window from which the MAI-FS session was created, or logs off from the product region, without logging off the application. The user is canceled by an OCS operator; the user's reconnect time limit expires; the session with the user's terminal is lost and session reconnect is disabled by the installation. Parameters coded in the SCRIPT NCL PROC field of the Logon Details panel are passed as &1, &2, &3, and so on.

SKIP

Indicates that the procedure was started by a .S session skip command entered by the user on a screen displayed on behalf of this session or the SCRIPT command. Words in the same input field as the .S command or in the DATA operand of the SCRIPT command are passed as parameters &1, &2, &3, and so on, to the procedure.

Examples: &MAISMODE

```
&IF &MAISMODE = END &THEN &END

&IF &MAISMODE = SKIP &THEN &GOTO .PROCESS

&GOSUB .MODE_&MAISMODE
```

Note: The script procedure must be able to handle being driven under all three of the above conditions. If no processing is to be performed for any of the conditions, the procedure should end.

&MAITITLE

Returns the title that is displayed at the top of the MAI-FS main menu.

&MAIUNLCK

Indicates whether the data stream just received would unlock the keyboard if sent to the terminal.

When a script procedure receives a data stream from the PLU (application), it should not attempt to automatically reply until a data stream is received that would unlock the keyboard if sent to the terminal. Failure to wait for the unlock condition could result in the reply being discarded because the SNA session state does not allow data to be sent.

&MAIUNLCK returns one of the following values:

YES

Indicates that the data stream would unlock the keyboard.

NO

Indicates that the data stream would not unlock the keyboard.

Example: &MAIUNLCK

```
&IF &MAIUNLCK = NO &THEN &GOSUB .WAITUNLCK
```

Notes:

&MAIUNLCK is normally used to determine if a reply is returned to the application. MAI delivers to the procedure each data stream as it is received. The script procedure might decide(using &MAIFIND) that it is time to reply to the application. However, some applications send data to the screen in multiple I/O operations, only unlocking the keyboard on the last one. A good example of such an application is TSO.

During logon, TSO sends 'logon in progress' type messages, followed by any broadcast messages, and finally the READY prompt. Multiple I/O operations is performed to send these messages, and the script procedure sees each one as it occurs. Even the READY prompt cannot actually unlock the keyboard. It could be followed by another data stream that contains the data necessary to unlock the keyboard.

A script procedure which wanted to reply after the READY prompt would find the prompt using &MAIFIND, but might still not be able to reply unless the prompt also unlocked the keyboard. If &MAIUNLCK returned NO on the READY data stream, the procedure would have to wait for the next data stream to unlock the keyboard before replying.

Most applications are far easier to handle than this. Usually one I/O operation is used to send data to the screen, and that data stream also unlocks the keyboard.

An &MAIUNLCK issued when there is no data outstanding returns a null value. Data is outstanding from the time an &MAIREAD is satisfied until an &MAICONT, &MAIDEL, or another &MAIREAD is issued. &MAIUNLCK issued while processing an SLU data stream returns the value NO.

Note: For more information, see the examples in the BASE.INSTALL library.

More information:

&MAIFIND (see page 414)
&MAICONT (see page 407)
&MAIREAD (see page 419)
&MAIDEL (see page 411)

&MAIWNDOW

Indicates the MAI-FS session's visibility.

A value of FOREGROUND is returned when the application is currently displayed. BACKGROUND indicates that the session is not displayed.

Example: &MAIWNDOW

```
&IF &MAIWNDOW =
```

Note: A value of BACKGROUND will be returned wherever the MAI session is not displayed. This includes whenever an NCL &PANEL statement has taken over the window, for example from the script itself, session help or broadcasts.

&NDBERRI

Provides additional information about an NDB warning or error condition.

The &NDBERRI system variable contains up to 12 characters of additional information when certain NDB error conditions occur. For example, if an &NDBADD or &NDBUPD statement set &NDBRC to 104 (duplicate unique key value detected), &NDBERRI contains the name of the field in error.

Not all error responses provide a value in &NDBERRI.

Example: &NDBERRI

```
& NDBADD ...
&IF &NDBRC = 104 &DO
   &WRITE DATA=DUPLICATE KEY ON ADD, FIELD = &NDBERRI
   &GOTO ERROR_EXIT
&DOEND
```

Notes:

This example displays the name of the field with a duplicated key value, if the add fails with that response.

If no NDB statements have been issued by the executing NCL process, &NDBERRI returns a null value.

&NDBERRI is always cleared when the next &NDBxxx statement is executed. For this reason, the value should be saved in a user variable if other &NDBxxx statements must be executed before using the saved value.

For more information about the contents of &NDBERRI for each possible &NDBRC value, see the *Network Control Language Programming Guide*.

More information:

&NDBRC (see page 853)

&NDBRC

Indicates the success or otherwise of an &NDBxxx NCL statement.

The &NDBRC system variable is set by all &NDBxxx NCL statements, to a value that indicates whether the requested function was performed successfully (&NDBRC set to 0), or not (&NDBRC set to a non-zero value).

The NCL procedure can test the value of &NDBRC to control processing. For example, &NDBRC is set to 1 if an &NDBGET statement does not find a record with the matching key or RID.

&NDBRC values fall into three categories:

OK

&NDBRC = 0

W*arning*

&NDBRC = 1 to 29

E*rror*

&NDBRC > 29

Warning responses include such things as: Record not found, End-of-file, Scan exceeded a limit.

Error responses include such things as: Invalid field name or value, Unknown keyword in free-format text, Unknown format or sequence name.

Example: &NDBRC

```
&NDBADD ...
&IF &NDBRC NE 0 &GOTO ADD_ERROR
```

Notes:

This example shows how the result of an &NDBADD is determined by using &NDBRC.

If no NDB statements have been issued by the executing NCL process, &NDBRC is always null. Once an &NDBxxx statement has been issued, &NDBRC always has a numeric value.

&NDBRC is always reset when the next &NDBxxx statement is executed. For this reason, the value should be saved in a user variable if other &NDBxxx statements must be executed before using the saved value.

Error responses (&NDBRC > 29) are not returned to an NCL procedure unless &NDBCTL ERROR=CONTINUE is in effect. Instead, the procedure will be aborted. &NDBOPEN and &NDBCLOSE are exceptions to this rule. They are always treated as if &NDBCTL ERROR=CONTINUE is in effect, except for responses 34 (Already open, on &NDBOPEN), and 35 (Not open, on &NDBCLOSE).

Note: For more information about &NDBRC values, see the *Network Control Language Programming Guide.*

More information:

&NDBERRI (see page 852)

&NDBRID

Provides the record ID of the current or new record.

The &NDBRID system variable is set by some NDB NCL statements to indicate the Record ID of a record. It is set as follows:

&NDBADD

The RID of the new record.

&NDBCLOSE

Set to 0.

&NDBCTL

Set to 0.

&NDBDEF

Set to 0.

&NDBDEL

Set to 0.

&NDBGET

The RID of the current record or 0 if retrieving key field statistics.

&NDBINFO

Set to 0.

&NDBOPEN

Set to 0.

&NDBSCAN

Set to the RID of the first record returned by the scan, or zero if no records were selected. If the &NDBSCAN parameter RECLIMIT=1 was specified, &NDBRID will remain set to zero even if one record passes the scan.

&NDBSEQ

Set to 0.

&NDBUPD

The RID of the updated record.

If a non-zero response is returned in &NDBRC, most statements set &NDBRID to 0. &NDBSCAN is the only exception to this rule.

Example: &NDBRID

```
&NDBGET MYDB FIELD=SURNAME VALUE=SMITH
&NDBDEL MYDB RID=&NDBRID
```

This example deletes the first record (lowest RID) on database MYDB, with the field SURNAME = SMITH.

Notes:

If no NDB statements have been issued by the executing NCL process, &NDBRID is always null. Once any &NDB*xxx* statements have been issued, &NDBRID is always numeric.

&NDBRID is always cleared when the next &NDB*xxx* statement is executed. For this reason, the value should be saved in a user variable if other &NDB*xxx* statements must be executed before using the saved value.

Returns the relative position in an &NDBSCAN-built sequence on &NDBGET and &NDBSEQ statements.

More information:

&NDBADD (see page 445)
&NDBGET (see page 472)
&NDBSCAN (see page 492)
&NDBUPD (see page 513)

&NEWSAUTH

Indicates whether a user is authorized for NEWS functions.

An NCL procedure might want to determine whether the user who invoked it is authorized to use NEWS functions. &NEWSAUTH is set to NO if the user ID definition of the user does not include NEWS authorization and YES if it does.

Example: &NEWSAUTH

```
&IF &NEWSAUTH = YES &THEN +
  &GOTO .OK
&ELSE +
  &GOTO .REJECT
```

&NEWSRSET

Indicates whether the user is authorized for NEWS reset (delete) functions.

An NCL procedure might want to determine whether the user who invoked it is authorized to reset (or delete) information held on a database.

&NEWSRSET is set to NO if the user ID of the user does not include NEWS reset authorization, and YES if it does. The meaning of NEWS reset is determined by the NCL procedure itself, but would normally determine whether the user is allowed to delete records from a database.

Example: &NEWSRSE

```
&IF &NEWSRSET = YES &THEN +
  &GOTO .OK
&ELSE +
  &GOTO .REJECT
```

Note: For more information, see the examples in the distribution library.

&NMID

Returns the 1- to 12-character ID of this system.

&NMID returns the 1- to 12-character ID as set by the SYSPARMS command ID= operand. If this value has not been set by the SYSPARMS command, the default of the primary VTAM ACBNAME is returned.

Example: &NMID

```
&IF &NMID EQ NMPROD &THEN +
  &GOTO .OK
```

Notes:

When multiple systems are running in the same network, on the same or different machines, each should have a unique NMID. This is not enforced but is good operational practice.

The NMID for each system should be meaningful so as to allow a functional use in an installation's operational procedures.

Note: For more information, see the NMDID JCL parameter an d the domain ID (DID) as specified on the NMDID initialization parameter. The DID must be unique for every connecting system; otherwise the systems limit certain functions across INMC links.

&OCSID and &OCSIDO

Indicates the OCS ID name for the current window.

When operating multiple Operator Console Services (OCS) windows it might be necessary to distinguish between the windows for operational reasons. The OCSID command allows a distinguishing ID to be assigned to a window. This ID remains displayed in the bottom right hand corner of the window until exited. Thus, a different ID is set in each window.

The &OCSID system variable enables an NCL procedure to test the ID of the current OCS window. &OCSID will be set to the same 1- to 8-character ID used in the OCSID command. If no ID has been set, &OCSID returns a null value.

The &OCSIDO system variable enables an NCL procedure to test the ID of the other OCS window if it is operational.

&OCSID and &OCSIDO would normally be used jointly to determine automatically, within an NCL procedure, what ID should be set for a particular window.

Consider the case where two windows are required. They are to have different IDs of TEST and LIVE. An NCL procedure is executed as the initial command (INITCMD) on entry to OCS. As the entry and exit from OCS can occur in a random fashion, the NCL procedure must be able to determine which of the two windows TEST or LIVE is to be assigned. The example below shows how this is achieved.

Example: &OCSID and &OCSIDO

```
&IF .&OCSIDO NE .LIVE &THEN +
  -OCSID LIVE
&IF .&OCSIDO EQ .LIVE &THEN +
  -OCSID TEST
```

Notes:

In the above example the OCSID command has been prefixed with the suppression character (-) to eliminate the echo of the command to the terminal. This makes the setting of the window ID transparent to the operator.

&OCSID is used in NCL procedures, such as a MSGPROC, to control the path of execution.

&PANELID

Indicates the name of the current panel.

To assist in documenting systems and to aid in problem reporting, an installation can standardize on the inclusion of the name of the panel in a set position on the screen for all full screen panels.

The &PANELID system variable provides an alternative to the inclusion of the actual panel name within the body of the panel.

The &PANELID system variable is always set to the name of the current panel.

Example: &PANELID

```
&PANELID
```

Notes:

The panel designer should allow up to 12 characters for a maximum size panel name.

Use of the &PANELID system variable other than in a panel yields a null value.

For testing purposes, an alternative to the inclusion of the &PANELID variable is the use of the &CONTROL PANELID statement to force the display of panel names in the upper left hand corner of all panels. See the &CONTROL statement for more details.

&PARMCNT

Supplies the count of the number of argument variables created when a procedure was invoked.

&PARMCNT is the count of the number of argument variables created on the statement when a procedure is invoked.

When a procedure is invoked, arguments is passed to the procedure by following the name of the procedure with the data to be passed.

The arguments is parsed into words and assigned into the user variables &1, &2, to & n, and &PARMCNT is set to the count of the variables created.

Example: &PARMCNT

```
&IF &PARMCNT EQ 0 &THEN +
  &WRITE DATA=REQUIRED DATA OMITTED
```

User enters:

```
EXEC PROC5 NCP1 NCP4
```

Notes:

&PARMCNT is set to 2, representing the entry of the variable values NCP1 and NCP4 (assigned to &1 and &2).

&PARMCNT is set only when a procedure is invoked and remains set to that initial value for the duration of processing. Subsequent functions do not change the value.

&PARMCNT applies only to the current nesting level. Each new nesting level establishes its own unique &PARMCNT value.

The &ALLPARMS variable provides a single variable that is a consolidation of all variables supplied on entry to the procedure.

More information:

&ALLPARMS (see page 748)
&CONTROL (see page 266)

&RETCODE

Returns the current NCL process's return code or sets a new return code value.

NCL statements can set &RETCODE as an indication of the success or otherwise of the function. A procedure can set a value in the range 0 to 99 for &RETCODE.

&RETCODE is used to indicate the completion of a function performed by a nested procedure.

&RETCODE used as a statement, sets the value to that specified.

A value of 100 is set by the system if the &CONTROL FINDRC option is set. This option allows a procedure to determine the success of a request for a nested procedure. If the requested procedure does not exist and &CONTROL FINDRC is set, processing continues but &RETCODE is set to a value of 100. If &CONTROL FINDRC is not set, the requesting procedure terminates.

value

A new value, in the range 0 to 99, to be placed in &RETCODE.

Example: &RETCODE

```
&IF &RETCODE NE 0 &THEN +
    &WRITE DATA=FUNCTION FAILED.

&RETCODE 4

&CONTROL FINDRC
-EXEC &REQUEST
&IF &RETCODE EQ 100 &THEN +
    &WRITE DATA=Requested Procedure '&REQUEST' Not Found
```

Notes:

A nested procedure can set a return code on the &END statement. On return to the higher level, &RETCODE contains the return code value.

If &RETCODE has not been set, it has a default value of 0.

An alternative to using the &END statement to pass a return code is to use the &RETURN statement, which can return variables to a higher nesting level, or &CONTROL SHRVARS, which allows sets of variables to be shared between procedures.

More information:

&END (see page 314)
&RETURN (see page 576)
&CONTROL (see page 266)

&ROUTECODE

A system variable that contains the routing code(s) assigned to the current message, in &MASKCHK format.

The routing codes(s) that is assigned to a message are in the range from 1 to 128. This is a 128-character system variable containing 128 Y/N values representing the routing code(s).

For example, ROUTCODE=(1,2,3,11) yields a 128-byte value of YYYNNNNNNNNYNNNNN....N.

Example: &ROUTECODE

```
&ROUTECODE = &MASKCHK YYYNNNNNNNNYNNNNN &AOMRMASK
```

&AOMRMASK is used to identify invalid route codes.

More information:

&AOMRHEX (see page 798)
&AOMROUTC (see page 799)

&SYSID

Returns the current operating system identification.

An operating system has an ID associated with it. This ID is used by the JES sub-system to identify jobs and so on. &SYSID is used to test the current value of this ID.

Example: &SYSID

```
&WRITE DATA=SYSTEM ID=&SYSID
&IF &SYSID = S001 &THEN +
    &GOTO .OK
```

Notes:

In z/OS and MSP installations the value returned from &SYSID is a 1- to 4-character SMF ID as set by your installation. In z/VM environments, &SYSID returns the 8-character Virtual Machine identifier of the machine in which your product region is executing. In VSE/SP environments, &SYSID returns a single question mark.

&NMID is used to determine the identity of the region under which the procedure is executing.

&TIME

Returns the current time. A system maintained variable that returns the current time of day in the format HH.MM.SS.

Example: &TIME

```
&WRITE DATA=THE TIME IS &TIME, ON &DATE2
&HHMM = &SUBSTR &TIME 1 5
```

Note: The value returned from &TIME is always an 8 character value.

More information:

&ZTIMEn (see page 972)
&ZGTIMEn (see page 898)
&ZGTIMEZn (see page 899)

&USERAUTH

Returns the command authority of the user who initiated the procedure.

Returns a numeric value in the range 0 to 255, indicating the command authority level of the user who initiated the procedure.

Example: &USERAUTH

```
&WRITE DATA=YOUR USERID IS &USERID AND AUTHORITY IS &USERAUTH
&IF &USERAUTH NE 255 &THEN +
   &GOTO .REJECT
```

Notes:

If an NCL process is executing on behalf of another user, for example, as a result of a SUBMIT command, the originating user's authority is propagated.

If USER1 issues the command

```
-SUBMIT BSYS -START PROC1
```

where USER1 has authority 1, and the BSYS environment has authority 9, use of &USERAUTH in this execution of PROC1 returns 1, not 9.

&USERID

Returns the user ID of the user currently executing the procedure.

A system maintained variable that returns the user ID of the user currently executing the procedure.

NCL procedures execute in an NCL processing region associated with a user ID. The user ID is a real user or alternatively the procedure can execute in the NCL processing region of one of the special internal user IDs that support the various system level processing regions within the system.

The internal user IDs within a system are formed from the system's domain ID (as specified by the NMDID initialization parameter) and a suffix that represents the particular internal environment that they support. However, the EASINET system user is always returned as EASINET, irrespective of the domain ID.

Typical internal user IDs are those that support the various background environments such as the background logger, the background monitor and the background system environment.

Example: &USERID

```
&WRITE DATA=YOUR USERID IS &USERID AND AUTHORITY IS &USERAUTH
&IF &USERID EQ SYSPROG &THEN +
    &GOTO .ALLOW
```

Notes:

The SHOW USERS command displays a list of the active user IDs.

The value returned from &USERID is a 1- to 8-character value.

&USERPW

Returns the PASSWORD of the user.

A system maintained variable that returns the PASSWORD of the user. The password returned is that used to log on. This variable is valid only for use within the LOGON REQUEST when starting an MAI-FS (Multiple Application Interface-Fullscreen) session, or from within an MAI session script procedure. For security reasons, it returns a null value if used elsewhere.

Example: &USERPW

```
TSO &USERID/&USERPW
```

Notes:

The password is stored in encrypted form, and is decrypted only when referenced. In addition, buffers containing the password are erased immediately after use.

The SYSPARMS USERPW operand determines the availability of &USERPW as logon data. For more information, see the *Reference Guide*. You can prevent the use of this system variable in your product region.

If a user changes their password, &USERPW returns the new value.

&VSAMFDBK

Indicates the VSAM return code from a file processing operation.

A VSAM return code (extracted from the VSAM RPL) is set after the execution of the &FILE ADD, &FILE DEL, &FILE GET and &FILE PUT file processing statements. This return code is placed in the &VSAMFDBK system variable, which can then be tested to determine the exact cause of a VSAM-related error. For logical processing errors, the variable contains the two hexadecimal characters of the VSAM feedback, for example X'1C'. If a physical error occurs, the variable is set to contain the characters PY and a message is written to the activity log giving the actual error code. The meaning of the various return codes is found in your VSAM documentation.

Although &VSAMFDBK is always set, it carries an error-related code only if the &FILERC system variable has been set to a value of 8, indicating that an error has occurred.

Example: &VSAMFDBK

```
&FILE OPEN ID=HELPDESK FORMAT=DELIMITED
&FILE GET ID=HELPDESK KEY='PROB005' VARS=TXT
IF &FILERC NE 8 &THEN +
 &GOTO .OK
&WRITE DATA=RECORD READ ERROR - VSAMFDBK = &VSAMFDBK
```

Notes:

The &FILERC system variable is set on completion of a file processing operation and should be used to determine whether the operation was generally successful.

Testing of the &VSAMFDBK system variable would normally only be performed if a severe error were detected. However, use of the &FILE GET OPT=UPD facility, where exclusive use of a record is requested, could result in the request failing if the record was already being processed elsewhere. In such a case, the user might have to test &VSAMFDBK to determine why the request failed and then retry if the failure was because exclusive use was not possible.

More information:

&FILERC (see page 829)

&ZACBNAME

Returns the primary VTAM ACB name in use by the system.

The variable allows an NCL procedure to determine in which system it is executing (as represented by the VTAM APPL definition that it is using).

Example: &ZACBNAME

```
&GOTO .&ZACBNAME
 .
 .
 .
.ERROR
   &ENDAFTER &WRITE DATA=this procedure is restricted to +
   production systems .
 .
 .
.NMCPU1 .
   -* production region on CPU 1
.
 .
.NMCPU2
   -* production region on CPU 2
```

&ZAMTYPE

Returns the name of the access method which is used to connect the terminal on which the NCL procedure is executing.

Allows an NCL procedure to determine the name of the access method which is used to connect the terminal on which it is executing.

The term access method refers to a communication program used to communicate with terminals, other systems, or another application.

&ZAMTYPE can contain one of the following values:

INTERNAL

The procedure is executing in a background region, an ROF region or a disconnected region

VTAM

The terminal is connected to your product region via VTAM

SSI

The terminal is connected to your product region via SSI

XNF

The terminal is connected to your product region via XNF

Example: &ZAMTYPE

```
&IF &ZAMTYPE = VTAM &THEN +
   D &LUNAME -* Do VTAM display of LU
```

&ZAPPCACC

Returns the number of active APPC conversations for the NCL process.

An NCL process can have multiple APPC conversations active at any one time. &ZAPPCACC returns the active conversation count. It is particularly useful when the NCL process is acting as a server for APPC conversations with a connect mode of ACCEPT. In this case, &ZAPPCACC can represent the number of client conversations waiting to be processed by the server.

Example: &ZAPPCACC

```
&APPC SET_SERVER_MODE CONNECT=ACCEPT
&IF &ZAPPCACC NE 0 &THEN +
 &DO
 &APPC RECEIVE_AND_WAIT ID=CLIENTS
```

&ZAPPCCSI

Indicates whether an APPC conversation is a client or server conversation.

CLIENT conversations are those that have been initiated remotely.

SERVER conversations are those that have been initiated locally using an &APPC ALLOCATE, &APPC ATTACH, or &APPC CONNECT statement.

Example: &ZAPPCCSI

```
&IF &ZAPPCCSI EQ CLIENT &THEN +
   &DO
   &APPC RECEIVE VARS=DATA
   .
   .
   .
```

&ZAPPCELM

Contains the message from an Error Log GDS variable received after an error, or deallocate abend has been received.

The Error Log GDS variable is used to send implementation-specific error information. It is sent as a consequence of issuing either of the following verbs:

```
&APPC SEND_ERROR LOG=
```

```
&APPC DEALLOCATE TYPE=ABEND LOG=
```

It is logged by the receiving system and made available to the receiving NCL procedure in the system variable &ZAPPCELM.

Example: &ZAPPCELM

```
&IF &RETCODE EQ 8 &THEN +
   &DO
   &WRITE DATA=REMOTE PGM ERROR = &ZAPPCELM
   &WRITE DATA=RECEIVED FROM &ZAPPCELP
  .
  .
  .
```

More information:

&ZAPPCELP (see page 870)

&ZAPPCELP

Contains any product set information from an Error Log GDS variable received after an error or deallocate abend has been received.

This system variable identifies the software product which sent an Error Log GDS variable as a result of issuing a either of the following verbs:

```
&APPC SEND_ERROR LOG=
```

```
&APPC DEALLOCATE TYPE=ABEND LOG=
```

The Error Log GDS variable message is contained in &ZAPPCELM. The information that identifies the product that sent the message is contained in &ZAPPCELP.

Example: &ZAPPCELP

```
&IF &RETCODE EQ 8 &THEN +
   &DO
   &WRITE DATA=REMOTE PGM ERROR = &ZAPPCELM
   &WRITE DATA=RECEIVED FROM &ZAPPCELP
  .
  .
  .
```

More information:

&ZAPPCELM (see page 869)

&ZAPPCID

Returns an integer when using the APPC facility.

The integer returned is within the range of 1 to 2147483647, and is equivalent to the usage of the LU6.2 verb RESOURCE parameter. It is set following successful completion of an &APPC ALLOCATE request, or after a procedure is first invoked as the conversation partner following an allocation request at the remote end.

The conversation identifier is assigned on a system-wide basis to uniquely identify a particular conversation within the local system. Local allocation requests and attach requests (following remote allocation requests) are both assigned unique identifiers. This means that the local and remote end of a same-LU conversation (for example started using the LOCAL option on an &APPC ALLOCATE request) have different conversation identifiers.

Example: &ZAPPCID

```
&IF &ZAPPCID NE &CONVB &THEN +
  &APPC TEST ID=&CONVB
```

&ZAPPCIDA

Indicates the APPC conversation ID for the transaction that started the NCL process. It is set for conversations that were attached only.

When an NCL process is attached, and then allocates further conversations, this system variable is used to provide the conversation identifier of the attaching conversation.

Example: &ZAPPCIDA

```
&APPC SEND VARS=DATA ID=&ZAPPCIDA
```

&ZAPPCLNK

Returns a character string when using the APPC facility.

&ZAPPCLNK returns a character string, equivalent to that of the PARTNER_LU_NAME parameter of the LU6.2 MC_GET_ATTRIBUTES verb, and provides the link name by which the remote LU name is addressed in the form:

linkname

where *linkname* is the locally-known name for the remote session partner connection, up to 12 characters long.

Example: &ZAPPCLNK

```
&WRITE DATA=CONVERSATION &ZAPPCID STARTED ON LINK &ZAPPCLNK
```

&ZAPPCMOD

Returns a character string when using the APPC facility.

&ZAPPCMOD returns a character string, equivalent to the MODE_NAME parameter of the LU6.2 MC_GET_ATTRIBUTES verb, and provides the mode name for the session on which the conversation is mapped.

Example: &ZAPPCMOD

```
&WRITE DATA=CONVERSATION &ZAPPCID STARTED USING MODE &ZAPPCMOD
```

&ZAPPCPCC

Returns the number of pending APPC conversations for the NCL process.

Incoming conversations requests for an NCL process that is registered as a server can accumulate in a pending queue if the connect server mode is NOTIFY or PENDING. &ZAPPCPCC returns the pending conversation count.

Example: &ZAPPCACC

```
&APPC SET_SERVER_MODE CONNECT=NOTIFY
&DOWHILE &ZAPPCACC NE 0 +
   &INTREAD TYPE=REQ VARS=MSG* RANGE=(1,7) +
   &CONVID=&MSG
```

&ZAPPCQLN

Returns a character string when using the APPC facility.

&ZAPPCQLN returns a character string, equivalent to that of the OWN_FULLY_QUALIFIED_LU_NAME parameter of the LU6.2 MC_GET_ATTRIBUTES verb, and provides the network qualified local LU name in the form:

netid.luname

where *netid* is the name of the SNA NETWORK where the local LU luname is defined. If the *netid* is blank or unknown, then it is omitted so that only *luname* is returned.

Example: &ZAPPCQLN

```
&WRITE DATA=CONVERSATION &ZAPPCID STARTED BY LOCAL LU &ZAPPCQLN
```

&ZAPPCQRN

Returns a character string when using the APPC facility.

&ZAPPCQRN returns a character string, equivalent to that of the PARTNER_FULLY_QUALIFIED_LU_NAME parameter of the LU6.2MC_GET_ATTRIBUTES verb, and provides the network qualified remote LU name in the form:

netid.luname

where netid is the name of the SNA NETWORK where the remote LU luname is defined. If the netid is blank or unknown, then it is omitted so that only luname is returned.

Example: &ZAPPCQRN

```
&WRITE DATA=CONVERSATION &ZAPPCID STARTED FROM REMOTE NODE &ZAPPCQRN.
```

&ZAPPCRM

Returns the current receive map name.

If data mapping is supported by an LU6.2 connection, an APPC conversation can specify a map name with each issuance of &APPC SEND_DATA. This map name is used by the receiving system to interpret the contents of the data transmitted. The system variable &ZAPPCRM is set to the map name received by the last receive operation.

Example: &ZAPPCRM

```
&IF &ZAPPCRM NE PROBLEM.SITE.DATE.DATA &THEN +
  &GOTO .MAPERR
```

Note: When sending NCL tokens, a map name of $NCL is sent with the data by default, to indicate that the data comprises one or more NCL tokens.

&ZAPPCRTS

Returns a character string when using the APPC facility.

&ZAPPCRTS returns a character string, equivalent to that of the LU6.2 REQUEST_TO_SEND_RECEIVED parameter (that, is either YES or NO).

Example: &ZAPPCRTS

```
&IF &ZAPPCRTS EQ YES &THEN +
  &GOSUB .RECV
```

Note: &ZAPPCRTS is either YES or NO. It is set to NO when the conversation is started. If during conversation operation a REQUEST_TO_SEND is issued by the conversation partner , this system variable reflects a value of YES. The YES value persist until the local conversation enters receive state, hence allowing the conversation partner to send, at which time the value reverts to NO, regardless of whether the system variable was ever examined by the local procedure.

&ZAPPCSCM

Returns a character string specifying the server connect mode for the NCL process.

An NCL process is in the following server connect modes:

- PENDING
- ACCEPT
- NOTIFY
- REJECT

Example: &ZAPPCSCM

```
&IF &ZAPPCSCM EQ PENDING &THEN +
   &DO
   APPC SET_SERVER_MODE CONNECT=ACCEPT
```

&ZAPPCSM

Returns the current send map name.

If data mapping is supported by an LU6.2 connection, an APPC conversation can specify a map name with each issuance of &APPC SEND_DATA. This map name is used by the receiving system to interpret the contents of the data transmitted. The system variable &ZAPPCSM is set to the map name sent by the last SEND_DATA operation.

Example: &ZAPPCSM

```
&WRITE DATA=DATA SENT USING MAP NAME: &ZAPPCSM
```

Note: When sending NCL tokens, a map name of $NCL is sent with the data by default, to indicate that the data comprises one or more NCL tokens.

&ZAPPCSND

Contains a send indication that has not yet been received, but will be set on the next receive operation.

APPC allows the NCL procedure to receive data on one operation, and send the data on to the next, but only if the conversation was about to change direction. In this case the &ZAPPCSND would indicate YES, meaning that if a subsequent receive is issued, the what-received indicator, &ZAPPCWRI, would be set to SEND. Otherwise it is set to NO.

Example: &ZAPPCSND

```
&DOWHILE &ZAPPCSND EQ NO
 &APPC RECEIVE VARS=DATA
  .
  .
  .
&DOEND
&APPC SEND VARS=DATA
```

&ZAPPCSTA

Returns the current state of an APPC conversation.

See the individual &APPC request details for valid state transition sequences.

&ZAPPCSTA is used to determine the current state of an APPC conversation. The following states are possible:

- CONFIRM
- CONFIRM_SEND
- CONFIRM_DEALLOCATE
- DEALLOCATE
- DEFER
- DEFER_DEALLOCATE
- DEFER_RECEIVE
- DEFER_SEND
- RECEIVE
- RESET
- SEND

Examples: &ZAPPCSTA

```
&IF &ZAPPCSTA EQ DEALLOCATE &THEN +
  &APPC DEALLOCATE TYPE=LOCAL
```

Note: Only certain APPC verbs is issued from a given state. See the individual &APPC request details for valid state transition sequences.

&ZAPPCSYN

Returns a character string, equivalent to that of the SYNC_LEVEL parameter of the LU6.2 MC_GET_ATTRIBUTES verb.

&ZAPPCSYN returns a character string, equivalent to that of the SYNC_LEVEL parameter of the LU6.2 MC_GET_ATTRIBUTES verb, and contains one of the following values:

- NONE
- CONFIRM

This system variable indicates the SYNC_LEVEL with which the conversation was started. A value of NONE means that the &APPC CONFIRM and CONFIRMED requests, the PREPARE_TO_RECEIVE TYPE=CONFIRM request, and the DEALLOCATE TYPE=CONFIRM request cannot be used.

Example: &ZAPPCSYN

```
&IF &ZAPPCSYN EQ CONFIRM &THEN +
  &GOSUB .CONF
```

&ZAPPCTRN

Returns the locally known transaction identifier (up to 12 characters) for an APPC conversation.

&ZAPPCTRN provides the unique transaction identifier as defined in the local system's TCT (Transaction Control Table) and used during the allocation and attach process.

For local allocation requests, it corresponds to the value specified for the TRANSID operand in the &APPC ALLOCATE verb.

Example: &ZAPPCTRN

```
&WRITE DATA=CONVERSATION &ZAPPCID STARTED FOR TRANSACTION &ZAPPCTRN
```

&ZAPPCTYP

Returns a character string providing the APPC conversation type.

&ZAPPCTYP returns a character string providing the conversation typed as one of the following values:

- MAPPED
- BASIC

This indicates whether the logical conversation boundary is mapped or basic. However, NCL operation always proceeds as though the conversation is mapped, regardless of its actual type.

Example: &ZAPPCTYP

```
&WRITE DATA=CONVERSATION &ZAPPCID TYPE = &ZAPPCTYP
```

&ZAPPCWR

Returns a character string, equivalent to that of the LU6.2 WHAT_RECEIVED parameter.

&ZAPPCWR returns a character string, equivalent to that of the LU6.2 WHAT_RECEIVED parameter, containing one of the following indicators:

ZERO

Nothing received

DATA_COMPLETE

Data received complete

DATA_TRUNCATED

Data was truncated

DATA_INCOMPLETE

Data was incomplete

FMH_DATA_COMPLETE

Data complete

FMH_DATA_TRUNCATED

FMH data truncated

FMH_DATA_INCOMPLETE

FMH data incomplete

SEND

send state

CONFIRM

confirm state

CONFIRM_SEND

confirm_send state

CONFIRM_DEALLOCATE

confirm_deallocate state

DEALLOCATE

deallocate state

This system variable is set following an &APPC RECEIVE_AND_WAIT or &APPC RECEIVE_IMMEDIATE request and indicates the type of data returned by the request.

A value of DATA_COMPLETE indicates that data was returned in the variables supplied.

A value of DATA_TRUNCATED or DATA_INCOMPLETE indicates that data was returned but an error has occurred that caused the data to be truncated.

A value of SEND indicates that no data was returned, but that the remote conversation partner has entered receive state, so the local conversation has entered send state.

A value of CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE indicates that no data was returned, but that the local conversation should issue an &APPC CONFIRMED request, following which receive, send, or deallocate state is entered respectively.

A value of DEALLOCATE indicates that no data was returned, but that the remote conversation partner issued a DEALLOCATE TYPE=FLUSH, and the local conversation should issue an &APPC DEALLOCATE TYPE=LOCAL to free all conversation resources.

Example: &ZAPPCWR

```
&IF &ZAPPCWR = CONFIRM &THEN +
    &GOSUB .CONF
```

&ZAPPCWRI

Returns a character string, equivalent to that of the LU6.2 WHAT_RECEIVED parameter.

&ZAPPCWRI returns a character string, equivalent to that of the LU6.2 WHAT_RECEIVED parameter, containing a short what received indicator as one of the following:

- ZERO nothing received
- DATC data received complete
- DATT data was truncated
- DATI data was incomplete
- FMHC FMH data complete
- FMHT FMH data truncated
- FMHI FMH data incomplete
- SEND send state
- CONF confirm state
- COSE confirm_send state
- CODE confirm_deallocate state
- DEAL deallocate state

This system variable is set following an &APPC RECEIVE_AND_WAIT or &APPC RECEIVE_IMMEDIATE request and indicates the type of data returned by the request. Descriptions are as for the &ZAPPCWR system variable above.

Example: &ZAPPCWRI

```
& IF &RETCODE = 0 &THEN +
  &GOTO .&ZAPPCWRI
```

&ZAPPCVRB

Returns the last APPC verb that was issued.

&ZAPBLANK1

Returns a single blank character.

&ZBROID

Returns the broadcast identifier associated with the NCL process.

To identify the broadcast request that resulted in the execution of this procedure.

Examples:

See $NMBRO distributed broadcast handler.

Notes:

This variable returns a value only when used in a procedure associated with a user broadcast.

The NSBRO command supports an ID operand that specifies a 1- to 8-character broadcast identifier.

This value is returned by &ZBROID when the broadcast procedure is executed on behalf of a user.

If no ID operand is used, a system value is generated.

Note: For more information, see the NSBRO command in the Online Help and the chapter "Broadcasts" in the *Reference Guide*.

&ZBROTYPE

Indicates the type of broadcast associated with the issuing procedure.

When a user receives a broadcast message an NCL procedure is invoked to perform associated processing. This procedure ($NMBRO) must be able to determine the type of broadcast to be processed and process accordingly. The &ZBROTYPE system variable is primed with the type of broadcast. One of the following values is set:

GENERAL

A general broadcast is being processed. A general broadcast is sent to all terminals and is not specifically destined for this user only.

SPECIFIC

A specific terminal broadcast is being processed. A specific broadcast is one that has been sent to a selective group of terminal using the wildcard selection techniques provided by the system.

USER

A broadcast destined for this user is being processed. A user broadcast is one that has been sent to one or more users using the wildcard selection techniques provided by the system.

NONE

No broadcast has been set.

Example: &ZBROTYPE

```
& GOTO .&ZBROTYPE
 .
 .
 .
.SPECIFIC
 .   .-* Specific broadcast processing
 .
.GENERAL
.
   .-* General broadcast processing
.
.USER
 .
   .-* User broadcast processing
.
```

Note: For more information, see the sample $NMBRO procedure in the distribution library.

&ZCOLS

Indicates the number of columns associated with the physical terminal.

NCL procedures might want to determine the number of columns associated with the physical terminal. &ZCOLS is the maximum number of columns available at the terminal regardless of the current processing window size.

Example: &ZCOLS

```
&IF &ZCOLS GT 80 &THEN +
  &GOTO .MODEL5
```

Note: When processing with an LU1 type device, a value of 80is returned. For VDU (screen) type devices, the actual number of columns is returned. If tested from a procedure running under a background environment, a value of 80 is returned.

More information:

&LUCOLS (see page 838)
&ZROWS (see page 963)
&LUROWS (see page 842)

&ZCONSOLE

Returns the system console number associated with a console user ID.

When a product command is received from a real console, a user ID environment is created and logically logged on to represent the console and the privileges associated with it. An NCL process executing in the processing region of this user ID can reference the &ZCONSOLE system variable to determine the real console number associated with the user ID. This variable returns a null value for any user ID that does not represent a console environment.

Example: &ZCONSOLE

```
&IF .&ZCONSOLE EQ . &THEN +
    &GOTO .NOCONSOLE
```

Note: For more information about console management, see the *Reference Guide*.

&ZCURSFLD

Indicates the field location and the offset within that field where the cursor is positioned.

These two system variables is used to determine the field and offset within the field where the cursor is positioned as at the last operator input from a panel displayed using the &PANEL statement.

&ZCURSFLD is set to the field location of the cursor. &ZCURSFLD is useful for providing context sensitive help. It contains the name of the field the cursor is in (TYPE=INPUT and TYPE=OUTVAR fields only—output fields have no name).

&ZCURSPOS is set to the offset within the field where the cursor is positioned.

Example: &ZCURSFLD

```
.HELP
&IF .&ZCURSFLD EQ .COMMAND &THEN +
   &GOTO .CMDHELP
   .
   .-* General Help
   .
.CMDHELP
  .
  .-* Help for the command field
  .
-* Split the line at the cursor position
&IF .&ZCURSPOS NE . &THEN +
   &PARSE SEGMENT VARS=(PREFIX(&ZCURSPOS),SUFFIX) +
      DATA=&
&ZCURSFLD
-* &PREFIX contains the data up to the cursor position
-* &SUFFIX contains the data after the cursor position
```

Note: &ZCURSFLD and &ZCURSPOS are set only when &CONTROL FLDCTL is set on. Otherwise, both are null.

&ZDBCS

Indicates whether a terminal supports double byte character set data streams (DBCS).

If the system is operating with double byte character set (DBCS), support enabled (SYSPARMS DBCS=YES), display terminals capable of operating in this mode are supported by the system, and can enter and display data represented by DBCS data streams. An example of DBCS implementation is the KANJI language used in Japan. If operating with DBCS enabled, the current terminal, that is, the terminal in whose environment the NCL procedure is executing, might or might not be capable of DBCS operation. Also, if the terminal is capable of DBCS operation, the implementation differs between IBM and Fujitsu terminals.

The &ZDBCS system variable therefore allows a procedure to determine whether the current terminal can support DBCS operation and, if so, whether it supports the IBM or the Fujitsu implementation of DBCS.

One of the following values will be returned:

DBCS1

The terminal supports the IBM implementation of DBCS

DBCS2

The terminal supports the Fujitsu implementation of DBCS

DBCS3

The terminal supports the Hitachi implementation of DBCS

NO

The terminal does not support DBCS

Example: &ZDBCS

```
&GOTO .&ZDBCS

 .

 .

 .

.NO

 .

 .

-* terminal does not support DBCS .

.DBCS1

 .

 .

-* terminal supports IBM DBCS .

.DBCS2

 .

 . -* terminal supports Fujitsu DBCS.
```

Note: When operating with DBCS support disabled (SYSPARMS DBCS=NO), the &ZDBCS variable always returns a value of NO.

More information:

&ZSYSPARM (see page 727)

&ZDOMID

Returns the deletion identifier for a non-roll deletable message.

The system supports a class of messages classified as non-roll deletable. Such messages are normally used to communicate important information to the operator and when displayed remain visible until explicitly deleted by the operator or by the system.

Non-roll deletable messages is sourced externally, such as from the Automation Services component of the system when passing messages from the operating system, or from within the system by use of the &WRITE NRD=YES statement in an NCL procedure.

Messages generated by &WRITE NRD=YES are classified as non-roll deletable and remain displayed either until deleted by the operator, by placing the cursor on the message and pressing enter, or until the NCL procedure terminates.

The &NRDDEL statement allows an NCL procedure to delete a non-roll deletable message prior to the termination of the procedure. For example, the procedure can issue the &WRITE NRD=YES statement to advise the operator that recovery for a critical network resource is in progress. On successful recovery the procedure can use the &NRDDEL statement to cause the message to be deleted from the operators screen.

&ZDOMID is a system variable that is set following &WRITE NRD=YES and contains the identifier that must be used on the &NRDDEL statement to indicate which non-roll deletable message is to be deleted.

Example: &ZDOMID

```
& WRITE NRD=YES DATA=Production IMS system has abended.  +
            Restart in progress.
&SAVEDOMID=&ZDOMID
 .
 .                  -* Processing to restart IMS system
 .
&NRDDEL &SAVEDOMID  -* Delete earlier message.

&WRITE DATA=Production IMS restarted successfully.
```

Notes:

The &WRITE statement also supports the NRD=OPER operand which requests that a non-roll deletable message be generated which will not be deleted when the procedure terminates. &ZDOMID is not set for such messages as there is no deletion identifier for this type of message. Messages generated in this manner must be deleted manually by the operator as described above.

The NRDRET command is used by the operator to redisplay non-roll deletable messages deleted manually by positioning of the cursor.

Messages deleted by &NRDDEL cannot be recalled by the NRDRET command.

If multiple non-roll deletable messages are to be written it is the procedures responsibility to save the value of &ZDOMID in a user variable.

The actual format of the value of &ZDOMID depends on the origin of the domain ID. For MVS-generated messages, &ZDOMID contains the MVS-format domain ID for the message (8 hexadecimal characters). If the message originates from &WRITE NRD=YES, the domain ID has the format *dddd*/*nnnnnnnn*, where *dddd* is the domain ID of the originating product region and *nnnnnnnn* is a 1- to 8-digit hexadecimal number.

Note: For more information, see the NRDRET command description in the Online Help.

More information:

&WRITE (see page 695)
&NRDDEL (see page 518)

&ZDSNQLCL

Returns the value of the local data set qualifier as set in the DSNQLCL JCL parameter.

Note: For more information about the DSNQLCL JCL parameter, see the *Reference Guide*.

&ZDSNQSHR

Returns the value of the shared data set qualifier as set in the DSNQSHR JCL parameter.

Note: For more information about the DSNQSHR JCL parameter, see the *Reference Guide*.

&ZFDBK

Returns completion information following execution of selected NCL statements.

Many NCL verbs return completion information after execution that indicates the success or result of the operation. This completion information is returned in the &ZFDBK system variable, so that results are available from a consistent location.

Example: &ZFDBK

```
&GOTO .CODE&ZFDBK
 .
 .
 .
.CODE0
 .
 .-* FDBK = 0 = Successful operation
 .
 .CODE4
 .
 .-* FDBK = 4 = operation failed.
```

Notes:

&ZFDBK is set as a result of the execution of a variety of NCL statements, and is available for inspection on return from any statement that sets it. If the contents of the variable are to be used, remember that they should be inspected before any other statement is executed that might also set &ZFDBK.

The meaning of the values set in &ZFDBK depends upon the statement that sets the variable. Therefore, check that statement if you need any explanation of the &ZFDBK value returned.

&ZGDATEn

A set of system variables that return the date, in different formats, based on Greenwich Mean Time (GMT).

&ZGDATE1 to &ZGDATE17 (excluding &ZGDATE15) are system variables that supply the current date, based on GMT, in the following formats:

- &ZGDATE1—date as YY.DDD
- &ZGDATE2—date as DAY DD-MON-YYYY
- &ZGDATE3—date as DD-MON-YYYY
- &ZGDATE4—date as DD/MM/YY
- &ZGDATE5—date as MM/DD/YY
- &ZGDATE6—date as YY/MM/DD
- &ZGDATE7—date as YYMMDD
- &ZGDATE8—date as YYYYMMDD
- &ZGDATE9—date as *nnnnnn*
- &ZGDATE10—date as YYYYMMDDHHMMSSZ
- &ZGDATE11—date as YYYYMMDDHHMMSS.FFFFFFZ
- &ZGDATE12—date as DD/MM/YYYY
- &ZGDATE13—date as YYYY/MM/DD
- &ZGDATE14—date as MM/DD/YYYY
- &ZGDATE16—date as YYYY.DDD
- &ZGDATE17—date as YYYYDDD

where:

DAY

Is the day of the week as follows:

MON

Monday

TUE

Tuesday

WED

Wednesday

THU

Thursday

FRI

Friday

SAT

Saturday

SUN

Sunday

DD

Is the day of the month as a two-digit number.

DDD

is the Julian day within the year as a three-digit number.

MM

Is the month of the year as a two-digit number.

MON

Is the month of the year as follows:

JAN

January

FEB

February

MAR

March

APR

April

MAY

May

JUN

June

JUL

July

AUG

August

SEP

September

OCT

October

NOV

November

DEC

December

nnnnnn

Is the number of days from 1 January 0001 with no leading zeros.

YYYY

Is the current year as a four-digit number.

YY

Is the current year as a two-digit number.

Z

Is GMT.

FFFFFF

Is the time accurate to 10^{-6}.

Example: &ZGDATE*n*

```
&FILEDATE = &ZGDATE3
```

Note: If microsecond accuracy is not required, avoid using &ZGDATE11 to reduce unnecessary overhead. Each access to &ZGDATE11 causes the system to re-fetch and synchronize time with the operating system, to format the result to microsecond accuracy.

&ZGDAY

Returns the day of the week based on GMT.

&ZGDAY provides a system variable for the day of the week, based on GMT, in the form DDD, where DDD is set to one of the following values:

MON

Monday

TUE

Tuesday

WED

Wednesday

THU

Thursday

FRI

Friday

SAT

Saturday

SUN

Sunday

Example: &ZGDAY

```
&IF &ZGDAY EQ SUN &THEN -EXEC SUNDAY
&ELSE +
    -EXEC EVERYDAY
```

Notes:

The current date, based on the operating system time, is provided in different formats by the system variables &DATE1 to &DATE17.

The current date, based on GMT time, is provided in different formats by the system variables &ZGDATE1 to &ZGDATE17.

More information:

&DAY (see page 827)

&ZGOPS

Indicates the generic type of operating system.

When operating in mixed operating system environments it is desirable to structure NCL procedures in such a way that they are operating system independent. Although your product region supports most functions, regardless of the operating system, certain facilities is restricted in some cases. The &ZGOPS system variable allows a procedure to test the generic (as opposed to the specific) type of operating system and restrict functions accordingly.

The &ZGOPS variable returns one of the following values:

MVS

The operating system is z/OS, MSP, or VOS3.

VM

The operating system is z/VM.

Example: &ZGOPS

```
 .
 .
 .
&GOTO .&ZGOPS -* Branch to operating system dependent logic.
 .
 .
 .MVS
 .
 .
 .
 .VM
 .
 .
 .
```

Note: &ZGOPS simplifies coding when logic depends on the general type of operating system, as opposed to the specific operating system. For example, z/OS, MSP, and VOS3 are generically all MVS systems. When a function is to be coded that works on any MVS system, using &ZGOPS makes the coding easier than using multiple tests against &ZOPS, which indicates the specific operating system type.

More information:

&ZOPS (see page 952)
&ZOPSVERS (see page 954)

&ZGTIMEn

A set of system variables that return the time, in different formats, based on GMT.

&ZGTIME, &ZGTIME1, &ZGTIME2, &ZGTIME3, &ZGTIME10 and &ZGTIME11 are system variables that supply the time, based on GMT, in a variety of formats. &ZGTIME time as HH.MM.SS

&ZGTIME1

time as HH:MM:SS

&ZGTIME2

time as HH:MM:SS:TH

&ZGTIME3

time as *nnnnn*-an integer being hundredths of a second since midnight

&ZGTIME10

time as HHMMSS

&ZGTIME11

time as HHMMSS.FFFFFF

Example: &ZGTIME2

```
&FILETIME = &ZGTIME2
```

Note: Each access to &ZGTIME11 causes the system to re-fetch and synchronize time with the operating system, to format the result to microsecond accuracy. Use of this time should be avoided, to reduce unnecessary overheads, if such accuracy is not required.

More information:

&TIME (see page 863)
&ZTIMEn (see page 972)
&ZGTIMEZn (see page 899)

&ZGTIMEZn

A set of system variables that indicate the difference in time between local (operating system) time and Greenwich Mean Time (GMT).

&ZGTIMEZ, &ZGTIMEZ1, &ZGTIMEZ2, and &ZGTIMEZ3 are system variables that indicate the difference between local time and GMT.

&ZGTIMEZ

plus or minus HH.MM

&ZGTIMEZ1

plus or minus HH:MM

&ZGTIMEZ2

plus or minus HHMM

&ZGTIMEZ3

plus or minus *nnnnn*, where an integer is hundredths of a second since midnight

Example: &ZGTIMEZ

```
&TIMEDIFF = &ZGTIMEZ
```

Note: Each access to &ZGTIMEZ3 causes the system to re-fetch and synchronize time with the operating system, to format the result to microsecond accuracy. Unless you require this level of accuracy, you should avoid using the &ZUTIMEZ3 option, to reduce unnecessary overheads.

More information:

&TIME (see page 863)
&ZTIMEn (see page 972)
&ZGTIMEn (see page 898)
&ZUTIMEn (see page 981)
&ZUTIMEZn (see page 982)
&ZUTIMEZN (see page 982)

&ZINTYPE

(Message profile variable) Specifies whether an &INTREAD operation has been satisfied by a request message or a response message. Valid values are:

REQ

Request message

RESP

Response message

NONE

No message

Example: &ZINTYPE

```
.READLOOP
   &INTREAD ARGS TYPE=ANY
   &IF &ZINTYPE = REQ &THEN +
   &GOSUB .REQUESTS
   &ELSE +
   &IF &ZINTYPE = RESP &THEN +
      &GOSUB .RESPONSES
   &GOTO .READLOOP
```

More information:

&INTREAD (see page 375)
&ZMREQID (see page 944)
&ZMREQSRC (see page 945)

&ZIREQCNT

Returns the count of messages queued to an NCL process's dependent request queue.

The dependent request queue is the queue of messages sent to an executing NCL process from an external source using the INTQ command. Messages on this queue are removed by the NCL process by using the &INTREAD TYPE=REQ or TYPE=ANY statement. The process can check for the existence of messages on this queue that are waiting for processing by referencing the &ZIREQCNT variable.

Example: &ZIREQCNT

```
.REQLOOP
   &IF &ZIREQCNT GT 0 &THEN +
   &DO
      &INTREAD ARGS TYPE=REQ
      &GOTO .REQMSG   &DOEND
   &ELSE +
   &ENDAFTER &WRITE DATA=NO REQUEST MESSAGES
.REQMSG
   .
   -* Processing for requests.
   &GOTO .REQLOOP
```

More information:

&INTREAD (see page 375)
&ZIRSPCNT (see page 902)

&ZIRSPCNT

Returns the count of messages queued to an NCL process's dependent response queue.

The dependent response queue is the queue of messages returned in reply to a command issued using the &INTCMD statement, or sent to an executing NCL process from an external source using the INTQ command. Messages on this queue are removed by the NCL process by using the &INTREAD TYPE=RESP or TYPE=ANY statement. The process can check for the existence of messages on this queue that are waiting for processing by referencing the &ZIRSPCNT variable.

Example: &ZIRSPCNT

```
.RSPLOOP
   &IF &ZIRSPCNT GT 0 &THEN +
   &DO
      &INTREAD ARGS TYPE=REQ
      &GOTO .RSPMSG
   &DOEND
   &ELSE +
   &ENDAFTER &WRITE DATA=NO RESPONSE MESSAGES
.REPMSG
 .
 .
       -* Processing for responses
   &GOTO .RSPLOOP
```

More information:

&INTREAD (see page 375)
&ZIREQCNT (see page 901)

&ZJOBNAME

A system variable that returns the job name.

The variable allows an NCL procedure to determine the job name under which it is executing. In a VM/GCS system this is the GCS USERID, in other systems the job or started task name.

Example: &ZJOBNAME

```
&IF &ZGOPS = MVS &THEN +
 &WRITE DATA=Started task name is &ZJOBNAME
```

Use of this variable is especially useful on an ABENDCMD command specified during initialization. The following command:

```
ABENDCMD S &ZJOBNAME
```

would restart the system automatically in the event of an abend in a z/OS or MSP system, assuming the system was run as a started task.

&ZJOBNUM

A system variable that returns the JES2/3 job number for the last job submitted by NCL (OS/VS only).

NCL supports the output of records directly to the JES2/3 system spool. This is achieved using the standard file processing facilities of NCL. Direct submission to the system internal reader is also supported. In such cases an &FILE GET OPT=END statement is used to signal the end of a job stream and the &ZJOBNUM system variable will then be set to the job number allocated by JES to that job. This value remains intact until another job is submitted or the procedure terminates.

&ZJOBNUM will return a 1- to 5-character job number. If no jobs have been submitted, a null value is returned.

&ZJOBNUM is set only if the JES internal reader has been dynamically allocated using the ALLOCATE command. See notes below.

Example: &ZJOBNUM

```
&FILE OPEN ID=MYJES FORMAT=UNMAPPED
           -* Request unmapped mode.
&FILE PUT ID=MYJES VARS=CARD
           -* Put to internal reader.
&FILE GET ID=MYJES OPT=END
           -* Signal end of JCL stream.
&WRITE DATA=THE JOB NUMBER IS &ZJOBNUM.
            -* Display submitted job no.
&FILE CLOSE ID=MYJES-* Free resources.
```

Notes:

The use of &FILE GET OPT=END in such cases merely signals to NCL that the VSAM RPL is no longer required for processing. NCL in turn issues a VSAM ENDREQ request, which signals to JES the completion of the submission. JES then returns the number of the last job submitted.

Dynamic allocation of the internal reader is performed using the ALLOCATE command with the SYSOUT and PGM operands. For example:

```
ALLOC SYSOUT=A PGM=INTRDR DD=MYJES ID=*
```

More information:

&FILE PUT (see page 345)
&FILE GET (see page 333)

&ZJRNLACT

Returns the ddname of the active journal data set.

This system variable is used to determine which journal is currently active.

More information:

&ZJRNLALT (see page 905)

&ZJRNLALT

Returns the ddname of the alternate (or inactive) journal data set.

This system variable is used to determine which journal is currently inactive.

More information:

&ZJRNLACT (see page 905)

&ZLCLIPA

Returns the IP address of the local host for a TN3270 session.

Identifies the IP address of a TN3270 server where the session is a TN3270 session.

Note: This information is available to the SHOW USERS command only if the IPCHECK SYSPARMS operand has been set to REGISTER (the default) in the INIT procedure for your product region.

Example: &ZLCLIPA

```
&IF .&ZLCLIPA NE . &THEN +
   &SOCKET GETHOSTBYADDR ADDRESS=&ZLCLIPA
```

Note: For more information, see the SHOW USERS command description in the Online Help and the IPCHECK SYSPARMS operand description in the *Reference Guide*.

More information:

&ZLCLIPP (see page 906)
&ZREMIPA (see page 962)
&ZREMIPP (see page 963)

&ZLCLIPP

Returns the IP port of the TN3270 server for a TN3270 session.

Identifies the IP port of a TN3270 server.

Note: This information is available only if the IPCHECK SYSPARMS operand has been set to REGISTER (the default) in the INIT procedure for your product region.

Example: &ZLCLIPP

```
&IF .&ZLCLIPP NE . &THEN +
   &IF &ZLCLIPP NE 23 &THEN +
       &PORTDESC = &STR TELNET-&ZLCLIPP
   &ELSE +
       &PORTDESC = TELNET
```

Note: For more information about the IPCHECK SYSPARMS operand, see the *Reference Guide*.

More information:

&ZLCLIPA (see page 905)
&ZREMIPA (see page 962)
&ZREMIPP (see page 963)

&ZLOGMODE

&ZLOGMODE is set to the name of the VTAM logmode table entry used when the current terminal was connected.

Identifies the name of the logmode table entry used when the current terminal (that is, the terminal represented by the &LUNAME variable) was connected to your product region. Depending on installation naming and definition conventions, a procedure can use this value to determine the attributes of the terminal or to verify that a the correct logmode entry was used for a terminal with known characteristics.

Examples: &ZLOGMODE

```
   &GOTO .&ZLOGMODE
 .
 .
 .
.M2SNAQ       -* Terminal is bound as a model-2 SNA device
 .
 .
 .
.M2NSNA       -* Terminal is bound as a model-2 non-SNA device
 .
 .
 .
```

Notes:

On IBM and Fujitsu systems, the availability of &ZLOGMODE depends upon the release of VTAM that is being used and whether the terminal is connected on a same domain or cross domain session.

&ZLOGMODE is not available on Hitachi systems.

&ZLOGMODE is null if the terminal is reacquired as a model-2, following a bind failure due to mismatched bind parameters.

&ZLUNETID

Returns the network ID of the currently connected terminal.

Identifies the name of the network that owns the currently connected terminal. Allows procedures to determine the network to which the terminal belongs.

Example: &ZLUNETID

```
&IF .&ZLUNETID EQ .MYNET &THEN ...
```

Note: The availability of &ZLUNETID is dependent on the presence of the X'0E' control vector in the CINIT RU.

&ZLUTYPE

The &ZLUTYPE system variable indicates the type of device or region.

NCL procedures might need to vary processing depending on the type of device in use, or NCL environment the procedure is running in. For example, the EASINET procedure cannot use an &PANEL statement to display data on an LU1 type device. The procedure must therefore determine the type of device and process accordingly.

The &ZLUTYPE variable returns one of the following values:

3270

The device is a full-screen type device, either LU0 or LU2. The &ZROWS and &ZCOLS system variables is used to determine the device dimensions.

LU1

The device is a *typewriter* or *line-by-line* type terminal or printer.

ROF

The procedure is running under a Remote Operator Facility (ROF) session which has been logged on across an INMC session.

APPC

The procedure is running under an APPC user region.

OPER

The procedure is running under a system console region.

AOMP

The procedure is executing under the background AOM region.

PPOP

The procedure is executing under the PPOPROC region.

LOGP

The procedure is executing under the LOGPROC region.

CNMP

The procedure is executing under the CNMPROC region.

BMON

The procedure is executing under the background monitor region.

BLOG

The procedure is executing under the background logger region.

BSVR

The procedure is executing under the background server region.

BSYS

The procedure is executing under the background system region.

Example: &ZLUTYPE

```
& IF &ZLUTYPE EQ LU1 &THEN +
   &WRITE DATA=Enter Logon Request ==>

&IF &ZLUTYPE EQ OPER &THEN +
   &ENDAFTER &WRITE DATA=Invalid Request
```

Notes:

Apart from being used in the EASINET procedure, this variable offers an easy way to detect when a procedure is operating under one of the specialized system regions such as the system console.

The &ZNCLTYPE system variable is used to determine the type of the NCL process.

See Also: The &ZNCLTYPE system variable.

More information:

&ZNCLTYPE (see page 948)

&ZLU1CHN

Indicates the segment position of a message received from an LU1 device.

The &PROMPT statement allows an NCL process to receive messages from LU1 devices. Up to 256 bytes of a message is delivered to &PROMPT in a single invocation.

If the incoming message is more than 256 bytes long, the data is segmented into separate elements, each up to 256 bytes. These elements are delivered to successive &PROMPT statements.

If the NCL process needs to know the position of a message within its chain of elements, for example, to ensure that all the messages in a chain have been received before the messages are processed, then it can reference the &ZLU1CHN variable after completion of the &PROMPT statement.

The &ZLU1CHN value can also be used as a branching value to control the issuing of successive &PROMPT statements. The &ZLU1CHN variable will return one of the following values:

FIC

The message just received by &PROMPT is the first element in an incoming chain. At least one more element in the chain is expected.

MIC

The message just received is not the first element in a chain, nor is it the last. At least one more element is expected. MIC stands for middle in chain and in a multi-element chain all elements except the first and the last return this value.

LIC

The message just received is the last one of the incoming chain. No more elements will follow as part of the chain. The next element to arrive will be part of a separate chain.

OIC

The message received is in a chain all by itself. It is therefore both the first in a new chain and also the last. The next element to arrive will be part of a separate chain.

Example: &ZLU1CHN

```
   &PROMPT ARGS Please Enter Description ==>>
    &GOTO .&ZLU1CHN
.FIC
.MIC
    .
    . keep reading successive messages till entire chain is received
    .
   &PROMPT ARGS AUTONL=NO
   &GOTO .&ZLU1CHN
.OIC
.LIC
    .
    . process complete message
    .
```

Note: The chaining position indication is not related in any way to any SNA-level chaining associated with the receipt of the data from the device; &ZLU1CHN values indicate only whether there is more data associated with the message being delivered and whether another &PROMPT is required for the procedure to obtain the remaining part of the message.

More information:

&PROMPT (see page 567)

&ZMAIACT# or &ZMAIACTN

Returns the number of active sessions associated with the current window.

&ZMALARM

(Message profile variable) Indicates whether the message will cause the terminal alarm to sound. Value is YES or NO.

&ZMALLMSG

(Message profile variable) Indicates whether the message was generated by a MSG ALL command. Value is YES or NO.

&ZMAOMAU

Indicates whether or not the original WTO or WTOR issuer was authorized.

This system variable is set only for messages containing AOM data after &MSGREAD, &LOGREAD, or &INTREAD and indicates whether the original WTO or WTOR message issuer was authorized. It contains the value YES or NO.

Example: &ZMAOMAU

```
.LOOP
&MSGREAD SET&IF .&ZMAOMDTA = .YES AND .&ZMAOMAU = .NO &THEN &DO
   &MSGDEL
   &GOTO .LOOP
&DOEND
 .
 .
 .
```

Note: This system variable corresponds to the &AOMAUTH system variable available to AOMPROC.

See Also: &AOMAUTH.

More information:

&AOMAUTH (see page 753)

&ZMAOMBC

Indicates whether or not the current message has the AOM broadcast attribute. If the current message has AOM attributes, and is flagged as a console broadcast message, this system variable has the value YES. If it is not a broadcast message, the value is NO. If the current message has no AOM attributes, this system variable is null.

Example: &ZMAOMBC

```
-* Set broadcast msg to red.
&IF .&ZMAOMBC = .YES &THEN &MSGCONT COLOR=RED
```

Note: This variable is set only after &INTREAD, &MSGREAD, or &LOGREAD.

More information:

&AOMBC (see page 755)
&AOMMSGLV (see page 788)
&ZMAOMMLV (see page 920)

&ZMAOMDTA

Indicates whether or not the current message contains AOM data.

This system variable is set after an &LOGREAD, &MSGREAD, or &INTREAD statement to indicate whether or not the current message contains AOM data (attributes).

This system variable contains either YES or NO. It is always set when a message is current. If it is NO, the only other AOM system variable that has a non-null value is the &ZMAOMMSG system variable.

Example: &ZMAOMDTA

```
-* if no AOM data, just CONT.
&IF &ZMAOMDTA = NO &THEN &MSGCONT
```

Note: Messages can carry AOM data without being propagated to AOM receivers. The converse is true. The &ZMAOMDTA and &ZMAOMMSG system variables indicate which of the attributes (AOM message and/or AOM data) the message has.

More information:

&ZMAOMMSG (see page 921)

&ZMAOMID

Contains the AOM ID value.

The AOM screening table can set a twelve-character ID value. This ID is available to AOMPROC in &AOMID. If an AOM message is onward delivered, the same ID value is available in &ZMAOMID to message receivers.

This ID is used to trigger specific processing in, for example, a MSGPROC.

Example: &ZMAOMID

```
.LOOP
&MSGREAD SET
&IF .&ZMAOMID = .ID005 &THEN &MSGDEL
 .
 .
 .
```

Note: This variable is null if the message is not carrying AOM data (&ZMAOMDTA is NO).

More information:

&AOMID (see page 765)
&ZMAOMDTA (see page 913)

&ZMAOMJI

Contains the job ID of AOM messages sourced from MVS.

If the current message carries AOM data, and came from an MVS job, this system variable contains the JES JOBID in the format J/S/T*nnnnn*.

This corresponds to the &AOMJOBID system variable in AOMPROC.

Example: &ZMAOMJI

```
.LOOP
&MSGREAD SET
&IF .&ZMAOMJI = .J00999 &THEN &WRITE DATA=JOBID NEEDS RESET
 .
 .
 .
```

Note: This variable is null if the message is not carrying AOM data (&ZMAOMDTA is NO).

More information:

&AOMJOBID (see page 770)
&ZMAOMDTA (see page 913)

&ZMAOMJN

Contains the job name of AOM messages sourced from MVS.

If the current message carries AOM data, and came from an MVS job, this system variable contains the MVS job name.

This corresponds to the &AOMJOBNM system variable in AOMPROC.

Example: &ZMAOMJN

```
.LOOP
&MSGREAD SET
&IF .&ZMAOMJN = .PROD005 &THEN &WRITE DATA=PROD005 active.
 .
 .
 .
```

Note: This variable is null if the message is not carrying AOM data (&ZMAOMDTA is NO).

More information:

&AOMJOBNM (see page 771)
&ZMAOMDTA (see page 913)

&ZMAOMMID

Contains the AOM message ID.

If the current message carries AOM data, this system variable contains the AOM message ID. For multi-line WTO minor lines (&ZMAOMMIN=YES), the message ID is that of the major line.

This system variable is null if the current message contains no AOM data (&ZMAOMDTA=NO).

The maximum length of this system variable is 12 characters.

Example: &ZMAOMMID

```
.LOOP &MSGREAD SET
&GOTO .&ZMAOMMID
 :
 :
.$HASP150
 :
&GOTO .LOOP
```

Note: &ZMAOMMID corresponds to &AOMMSGID. The screening table and AOMPROC can alter the MSGID if, for example, messages are in a non-standard format or do not have their own identifiers.

More information:

&AOMMSGID (see page 787)
&ZMAOMDTA (see page 913)

&ZMAOMMIN

Indicates whether or not this is an AOM minor line.

If the current message carries AOM data, this system variable indicates whether or not this is a minor line from a multi-line WTO. It has the value YES or NO. It is null if the current message contains no AOM data (&ZMAOMDTA = NO).

Example: &ZMAOMMIN

```
.LOOP
&MSGREAD SET
&IF .&ZMAOMMIN = .YES &THEN &MSGDEL
 :
 :
```

Note: &ZMAOMMIN corresponds to the &AOMMINLN system variable available to an AOMPROC.

More information:

&AOMMINLN (see page 63)
&ZMAOMDTA (see page 913)

&ZMAOMMLC

(Message Profile Variable.) Indicates whether or not the current message is an MLWTO control line.

If the current message contains AOM data, this system variable indicates whether or not it is a multi-line WTO control line. Its value is YES or NO. It is null if the current line is not an AOM-sourced line.

Example: &ZMAOMMLC

```
& IF .&ZMAOMMLC = .YES &THEN +
   &GOSUB .CTL-LINE
```

More information:

&ZMAOMMLD (see page 918)
&ZMAOMMLE (see page 918)
&ZMAOMMLL (see page 919)
&ZMAOMMLT (see page 919)

&ZMAOMMLD

(Message Profile Variable.) Indicates whether or not the current message is an MLWTO data line.

If the current message contains AOM data, this system variable indicates whether or not it is a multi-line WTO data line. Its value is YES or NO. It is null if the current line is not an AOM-sourced line.

Example: &ZMAOMMLD

```
&IF .&ZMAOMMLD = .YES &THEN +
   &GOSUB .DAT-LINE
```

More information:

&ZMAOMMLC (see page 917)
&ZMAOMMLE (see page 918)
&ZMAOMMLL (see page 919)
&ZMAOMMLT (see page 919)

&ZMAOMMLE

(Message Profile Variable.) Indicates whether or not the current message is an MLWTO end line.

If the current message contains AOM data, this system variable indicates whether or not it is a multi-line WTO end line. Its value is YES or NO. It is null if the current line is not an AOM-sourced line.

Example: &ZMAOMMLE

```
& IF .&ZMAOMMLE = .YES &THEN +
   &GOSUB .END-LINE
```

More information:

&ZMAOMMLC (see page 917)
&ZMAOMMLD (see page 918)
&ZMAOMMLL (see page 919)
&ZMAOMMLT (see page 919)

&ZMAOMMLL

(Message Profile Variable.) Indicates whether or not the current message is an MLWTO label line.

If the current message contains AOM data, this system variable indicates whether or not it is a multi-line WTO label line. Its value is YES or NO. It is null if the current line is not an AOM-sourced line.

Example: &ZMAOMMLL

```
&IF .&ZMAOMMLL = .YES &THEN +
   &GOSUB .LAB-LINE
```

More information:

&ZMAOMMLC (see page 917)
&ZMAOMMLD (see page 918)
&ZMAOMMLE (see page 918)
&ZMAOMMLT (see page 919)

&ZMAOMMLT

(Message Profile Variable.) Indicates the type of MLWTO.

If the current message contains AOM data, this system variable indicates the type of multi-line WTO. Its value is one of the following letters:

C

indicates a control line

L

Indicates a label line

D

Indicates a data line

DE

Indicates a data end line (last line)

This system variable is null if the current line is not an AOM-sourced line.

Example: &ZMAOMMLT

```
&GOTO .ML-&ZMAOMMLT
```

More information:

&ZMAOMMLC (see page 917)
&ZMAOMMLD (see page 918)
&ZMAOMMLE (see page 918)
&ZMAOMMLL (see page 919)

&ZMAOMMLV

Contains the highest AOM message level of the current message.

Message levels is used to limit the messages that is delivered to AOM authorized receivers. The possible values in &ZMAOMMLV, in order of decreasing severity, are: WTOR, R, I, CE, E, BC, and IN. This system variable is null if the current message does not contain AOM data

(&ZMAOMDTA = NO).

Example: &ZMAOMMLV

```
&IF &ZMAOMMLV = I &THEN &MSGCONT COLOR=RED HLIGHT=REVERSE
```

Notes:

Message levels is modified in the AOM screening table. This system variable represents the final level that is used for message propagation to AOM receivers.

Authorized AOM receivers can profile their environment to receive one or more message levels.

This system variable corresponds to &AOMMSGLV.

Note: For more information, see the PROFILE command description in the Online Help.

More information:

&AOMMSGLV (see page 788)

&ZMAOMMSG

Indicates whether or not the current message was marked for propagation to eligible AOM receivers.

This system variable is set to YES if the current message was flagged as eligible for propagation to all eligible AOM receivers. If not, it is set to NO.

The value of this system variable is independent of the value of &ZMAOMDTA, which indicates whether or not the message carries AOM data.

Messages that also carry AOM data are subject to filtering on routing code and message level before delivery.

Example: &ZMAOMMSG

```
.LOOP
&MSGREAD SET
&IF &ZMAOMMSG = YES &THEN &GOTO .AOMMSG
...
...
```

Note: All standard AOM traffic has &ZMAOMMSG set to YES. SYSCMD command responses, however, while containing AOM data (&ZMAOMDTA = YES), are not normally broadcast to AOM receivers, but just delivered to the command issuer. Thus, they have &ZMAOMMSG = NO.

More information:

&ZMAOMDTA (see page 913)

&ZMAOMRC

Contains the AOM routing code(s) assigned to the current message.

If the current message contains AOM data (&ZMAOMDTA = YES), &ZMAOMRC contains the routing code(s) of the current message, enclosed in parenthesis, for example, (1,3,11).

&ZMAOMRC is set to null if there are no AOM attributes for the current message.

Example: &ZMAOMRC

```
.LOOP &MSGREAD SET
&GOTO .&ZMAOMID
 .
 .
 .
.PREPMSG
&WRITE RC=&ZMAOMRC NRD=OPER +
   DATA=PLEASE PREPARE PRINTER 1 +
     FOR SPECIAL PRINT - AWZ001
&MSGCONT
&GOTO .LOOP
```

As is seen from the example, &ZMAOMRC is formatted so that it is inserted directly into an &WRITE, &WTO, &WTOR, or &AOMALERT statement.

This system variable corresponds to the &AOMROUTC system variable available to an AOMPROC.

Authorized AOM receivers can use the PROFILE ROUTCDE= command to control the receipt of messages used on routing codes.

Note: For more information, see the PROFILE command description in the Online Help.

More information:

&AOMROUTC (see page 799)
&ZMAOMRCM (see page 923)
&ZMAOMRCX (see page 923)

&ZMAOMRCM

Contains the routing code(s) assigned to the current message, in &MASKCHK format.

If the current message contains AOM data, this system variable contains the 128 AOM routing codes, in MASKCHK format.

For example, ROUTCDE=(1,2,3,11) is represented by a 128-byte value of YYYNNNNNNNNYNNNNN....N.

This system variable is null if the current message has no AOM data (&ZMAOMDTA = NO).

Example: &ZMAOMRCM

```
&ROUTECODE = &MASKCHK YYYNNNNNNNNYNNNNN &ZMAOMRCM
```

Note: &ZMAOMRCM is used to identify invalid route codes.

More information:

&AOMRHEX (see page 798)
&ZMAOMRC (see page 922)
&ZMAOMRCX (see page 923)

&ZMAOMRCX

Contains the AOM routing code(s) assigned to the current message, in hexadecimal characters.

If the current message contains AOM data, then this system variable contains the 128 AOM routing codes, as sixteen hexadecimal digits. For example, &ZMAOMRCX contains E020000000000000 for ROUTCDE=(1,2,3,11).

This system variable is null if the current message has no AOM attributes (&ZMAOMDTA=NO).

Example: &ZMAOMRCX

```
& CALL STATPROG &ZMAOMRCX
```

Note: &ZMAOMRCX is hexadecimal packed before being passed to a user program via &CALL by using the built-in function &HEXPACK.

More information:

&AOMRHEX (see page 798)
&ZMAOMRC (see page 922)
&ZMAOMRCM (see page 923)

&ZMAOMSOS

Identifies the operating system type from which the current AOM message came.

A message that is originated by AOM can come from either a VM or MVS system. This system variable indicates the source of the message, for messages containing AOM data. It contains OS (meaning z/OS, MSP, or VOS3) or VM.

Example: &ZMAOMSOS

```
&MSGREAD SET
&GOTO .&ZMAOMSOS
 .
 .
 .
.VM -* process VM-sourced msg.
 .
 .
 .
.OS -* process MVS-sourced msg.
 .
 .
 .
```

Notes:

If the message has no AOM data, &ZMAOMSOS is set to null.

The value in this system variable indicates which of the operating-system-specific system variables will contain meaningful data. If MVS, then &ZMAOMJI and &ZMAOMJN are available. If VM, then &ZMAOMUI and &ZMAOMUN are available. In each case, the other variables are null.

More information:

&ZMAOMUI (see page 930)
&ZMAOMUN (see page 931)
&ZMAOMJI (see page 914)
&ZMAOMJN (see page 915)
&ZMAOMSYN (see page 925)

&ZMAOMSYN

Identifies the system name from which the current AOM message came.

MVS provides a system name that is unique within a sysplex. An AOM message originating from an MVS system carries this system name in its internal message flow.

If AOMPRFSN is set to YES (either in the PROFILE command or as a SYSPARM), then the system name is available as a prefix to the message when displayed on a terminal. If the message is being processed by a MSGPROC, then this system variable indicates the system name.

Example: &ZMAOMSYN

```
&MSGREAD SET
&GOTO .&ZMAOMSYN
 .
 .
 .
.OS -* process MVS-sourced msg
 .
 .
 .
```

Note: If the message has no AOM data, &ZMAOMSYN is set to null.

More information:

&ZMAOMUI (see page 930)
&ZMAOMUN (see page 931)
&ZMAOMJI (see page 914)
&ZMAOMJN (see page 915)
&ZMAOMSOS (see page 924)

&ZMAOMTM

Contains the AOM time stamp of the current message.

&ZMAOMTM is set to the time that the current message was generated, if the current message contains AOM data. This time is in the form hhmmss. If no AOM data is present, &ZMAOMTM is null.

Example: &ZMAOMTM

```
.LOOP
&MSGREAD SET &GOTO .&AOMID
 .
 .
 .
&GOTO .LOOP .PRODJOB3
&IF &ZMAOMTM GT &TIME3 +
   &WRITE NRD=OPER DATA=WARNING JOB &AOMJOBNM IS +
       RUNNING LATE.
 .
 .
 .
&GOTO .LOOP
```

Note: &ZMAOMTM corresponds to the &AOMTIME system variable available to an AOMPROC.

More information:

&AOMTIME (see page 811)

&ZMAOMTYP

Indicates the AOM type of a message.

For messages containing AOM data (&ZMAOMDTA=YES), this variable is set to either WTO, WTOR, or MSG, depending on the message type.

Example: &ZMAOMTYP

```
.LOOP
&MSGREAD SET
&GOTO .&ZMAOMTYP
 .
 .
 .
.WTO
  .
   process WTO
  .
. &GOTO .LOOP
.WTOR
 .
 .  process WTOR
 .
&GOTO .LOOP
.MSG
 .
 . process VM MSG
 .
```

Notes:

Using &ZMAOMTYP is one way for an NCL procedure (for example, a MSGPROC) to distinguish WTORs.

The values in this system variable correspond to the &AOMTYPE system variable.

More information:

&AOMTYPE (see page 812)

&ZMAOMUFM

Contains the eight AOM user flags in &MASKCHK format.

&ZMAOMUFM contains a string of eight characters that indicate the settings of the eight AOM user flags, if the current message contains AOM data. The eight characters are Y or N. For example:

YNNNYNNY

The built-in function &MASKCHK is used to check the settings.

Example: &ZMAOMUFM

```
&USRFLAG3 = &MASKCHK **Y***** &ZMAOMUFM
&IF .&USRFLAG = .EQ &THEN &FLAG3 = ON
```

Notes:

&ZMAOMUF1-8 are flags which is set in the screening table or by an AOMPROC.

This system variable corresponds to the &AOMUFLGS system variable available to an AOMPROC.

See Also: The &ZMAOMUF1-8 and &AOMUFLGS descriptions.

More information:

&ZMAOMUF1-8 (see page 929)
&AOMUFLGS (see page 813)

&ZMAOMUF1-8

These are eight system variables containing the AOM user defined flags, set in the screening table.

The default for the system variables &ZMAOMUF1....&ZMAOMUF8 is NO. This is set to YES by the screening table or reset via &AOMCONT or &AOMREPL.

Example: &ZMAOMUF1

```
.LOOP
&MSGREAD SET
&IF &ZMAOMUF1 = YES &GOTO .SPECPROC
 .
 .
 .
&GOTO .LOOP

 . -* Special message processing
 .
.SPECPROC
 .
 .
 .
&MSGCONT COLOR=YELLOW
&GOTO .LOOP
```

Note: These correspond to the eight AOMPROC-specific system variables &AOMUFLG1-8.

More information:

&AOMUFLG1-8 (see page 814)
&AOMUFLGS (see page 813)
&ZMAOMUFM (see page 928)

&ZMAOMUI

A system variable containing the originating user ID of an AOM message from a VM system.

If the current message contains AOM data (&ZMAOMDTA=YES), and the message originated from a VM system, this system variable contains the user ID from which the message came. For messages that originate from CP, the user ID is CP.

This user ID is the virtual machine name that issued the CP MSG command.

Example: &ZMAOMUI

```
.LOOP
&MSGREAD SET
&IF .&AOMVMUID = .USER1 &THEN +
   &SYSCMD DEST=GCS MSG USER1 WHATS WRONG?
 .
 .
 .
```

Notes:

The &ZMAOMUI system variable is useful for replying to the originating user when a problem is solved.

This system variable corresponds to the &AOMVMUID system variable available to an AOMPROC.

More information:

&AOMVMUID (see page 817)

&ZMAOMUN

Contains the VM RSCS node name that an AOM/VM message came from.

AOM/VM messages that carry AOM data (&ZMAOMDTA=YES) contain the RSCS node that the message originated from in this system variable. For CP-generated messages in the local system, this has a value of CP.

In a networked VM system, this field is useful for identifying the source of a message.

Example: &ZMAOMUN

```
.LOOP
&MSGREAD SET
&IF &ZMAOMUN NE VM1 &THEN &GOTO .REMOTEVM
 .
 .
 .
```

Note: This system variable corresponds to the &AOMVMUND system variable available to an AOMPROC.

More information:

&AOMVMUND (see page 818)

&ZMAPNAME

(Message profile variable) Indicates the name of the object in the message user MDO component, if present. For example, after an &INTREAD, it returns the map name for the $INT.USERMDO object.

&ZMCOLOR or &ZMCOLOUR

(Message profile variable) Indicates the color attribute of the message. Value is any one of NONE, RED, BLUE, GREEN, YELLOW, TURQUOISE, PINK, or WHITE.

&ZMDOCOMP

Indicates the last name segment of the fully qualified name for the MDO component involved in the last operation.

&ZMDOFDBK

The &ZMDOFDBK system variable shows the feedback code after a verb references an MDO. This feedback variable is used with &ZMDORC.

Note: For more information about the return code and feedback variables, see the *Network Control Language Programming Guide*.

The following table shows the possible values of the return code and feedback variables, and their meanings:

&ZMDORC	&ZMDOFDBK	Meaning
0	0	OK
4	0	Null: optional component present but empty, or null data assigned to optional component
	1	Null: optional component not present
	2	Null: mandatory component present but empty, or null data assigned to mandatory component
	3	Null: mandatory component not present
	4	String was truncated (applies to FIX offset or length components only)
8	0	Type check: data is invalid for type
	1	Data check: data is invalid structurally—a common cause is data too long or too short
	2	Length check: maximum MDO length exceeded
12	0	Name check: component not defined
	1	Name check: index position invalid or value is out of range
16	0	Map check: map not found
	1	Map check: map contains errors—load failed
	2	Map check: map/data mismatch

&ZMDOID

Shows the identifier of the MDO involved in the last operation.

&ZMDOM

(Message profile variable) Indicates whether the message is a delete operator message instruction. Value is YES or NO. This value is also dependent on the setting of the &ZMTYPE variable.

More information:

&ZMTYPE (see page 946)
&ZMMSG (see page 938)

&ZMDOMAP

Returns the map name for &ZMDOID.

&ZMDOMID

(Message profile variable) Contains the delete operator message identifier(DOMID) of the message read, provided the message has the non-roll delete message attribute (as determined by the setting of the &ZMNRD terminal). If &ZMNRD=NO, then &ZMDOMID is set to null.

More information:

&ZMNRD (see page 939)

&ZMDONAME

Indicates the fully qualified name of the MDO component involved in the last operation.

&ZMDORC

The &ZMDORC system variable shows the return code after any verb references an MDO. This return code variable is used in conjunction with &ZMDOFDBK.

Note: For more information about the return code and feedback variables, see the *Network Control Language Programming Guide*.

The following table shows the possible values of the return code and feedback variables, and their meanings:

&ZMDORC	&ZMDOFDBK	Meaning
0	0	OK
4	0	Null: optional component present but empty, or null data assigned to optional component
	1	Null: optional component not present
	2	Null: mandatory component present but empty, or null data assigned to mandatory component
	3	Null: mandatory component not present
	4	String was truncated (applies to FIX offset or length components only)
8	0	Type check: data is invalid for type
	1	Data check: data is invalid structurally—a common cause is data too long or too short
	2	Length check: maximum MDO length exceeded
12	0	Name check: component not defined
	1	Name check: index position invalid or value is out of range
16	0	Map check: map not found
	1	Map check: map contains errors—load failed
	2	Map check: map/data mismatch

&ZMDOTAG

Indicates the MDO tag value of the component involved in the last operation.

&ZMDOTYPE

Indicates the ASN.1 type of the &ZMDOCOMP.

&ZMEVONID

Contains the *nclid* of the procedure which issued an &EVENT.

&ZMEVONID is set when the incoming message was generated by an &EVENT verb. It contains the *nclid* of the procedure that issued the &EVENT.

&ZMEVPROF

Represents the EDS profile name which resulted in delivery of an event notification.

&ZMEVPROF is set for incoming event messages (N00102) and represents the EDS profile name which resulted in the delivery of the event notification.

A copy of the event notification is delivered for every profile whose attributes match the event attributes.

&ZMEVRCDE

Contains the route code of an incoming event message.

&ZMEVRCDE contains the route code of the incoming event message (N00102) if the ROUTECDE operand was specified on the originating &EVENT verb.

&ZMEVTIME

Contains the time that an event originated.

&ZMEVTIME is set for incoming event messages (N00102) to the time the event originated. It is in the format HH.MM.SS.THT.

&ZMEVUSER

Contains the user ID of a user who issued an &EVENT verb.

&ZMEVUSER is set when the incoming message was generated by an &EVENT verb.

This variable is also set for some system events and represents the user who was responsible for the event generation.

&ZMHLIGHT or &ZMHLITE

(Message profile variable) Indicates the display highlighting attribute of the message. Values are NONE, USCORE, REVERSE, or BLINK.

&ZMINTENS

(Message profile variable) Indicates the display intensity attribute of the message. Values are HIGH, LOW, or null if no message is processed.

&ZMLNODE

(Message profile variable) Indicates the terminal name of the user to whom the log message is to be attributed. Value is the name of a terminal (available to &LOGREAD only).

If the message originated in a dependant NCL environment, it contains the NCLID of the executing procedure.

For monitor class messages which are logged from a remote system, &ZMLNODE is set to *REMOTE* if no single user is the message receiver.

&ZMLNODE is NULL if the source terminal ID or NCLID is not available.

More information:

&LOGREAD (see page 400)

&ZMLOGCMD

(Message profile variable) Indicates whether a log message is an echo to the log of a command. Value is YES or NO. (Available to &LOGREAD only).

&ZMLSRCID

(Message profile variable) Contains the message prefix of the last handler for the message just received.

&ZMLSRCTP

(Message profile variable) Indicates the *type* of the last handler for the message just received. Values are:

null

If the message was generated within this system

ROF

If the message was delivered across a ROF session

MAI-OC

If the message was delivered across an MAI-OC session

&ZMLTIME

(Message profile variable) Contains the time stamp of a log message (available to &LOGREAD only).

The format of &ZMLTIME is HH.MM.SS.THT.

&ZMLUSER

(Message profile variable) Contains the user ID the log message came from (available to &LOGREAD only).

&ZMLUSER is NULL if the message is not attributed to a specific user.

$ZMMONMSG

(Message profile variable) Indicates whether the message received is a monitor class message. Value is YES or NO.

&ZMMSG

(Message profile variable) The value is YES or NO indicating whether or not the message received is a standard message.

The setting of &ZMMSG is always the opposite value to that for &ZMDOM, and is dependent on the setting of the &ZMTYPE variable.

More information:

&ZMDOM (see page 933)
&ZMTYPE (see page 946)

&ZMMSGCD

(Message profile variable) Indicates the hexadecimal message code attribute for this message. The message code dictates which user IDs are eligible to receive the message.

&ZMMDIDL

(Message profile variable) This is the domain ID of the previous product region to handle this message.

It is the same as the originating system, or different if the message came from a remote system and has been routed onwards by an intermediate system.

&ZMMDIDO

(Message profile variable) This is the domain ID for the product region where this message originated:

- If sourced from the local system, this is the local system domain ID.
- If sourced from a remote system, this variable carries the domain ID of the originating system, even though the message might have been routed onwards by intermediate systems.

&ZMNRD

(Message profile variable) Indicates whether the message carries the non-roll delete attribute. Values is:

NO

Not a non-roll delete message

YES

Message is non-roll delete and is deleted only by a delete operator message (DOM) instruction

OPER

The message is non-roll delete but is deleted only by the cursor delete function from an OCS window

More information:

&ZMDOMID (see page 933)

&ZMNRDRET

(Message profile variable) Indicates whether the message has been received as a result of a NRDRET command being issued by the user.

This flag allows an NCL procedure to ignore redisplayed messages when analyzing events. Value is YES or NO.

&ZMODFLD

Returns the name of a modified variable.

&ZMODFLD is used to determine the names of variables modified by &PANEL, &NDBGET or &SETVARS.

Example: &ZMODFLD

```
&CONTROL FLDCTL
&PANEL NAMEADDR -* Issue panel requesting name and address
  :
.VALIDATE
   &GOTO .&ZMODFLD
                  -* jump to field validation routine
   &GOTO .NEXTPANE
.NAME              -* process name input
  &IF .&NAME = .&THEN ...
  :
.ADDR              -* process address input
 :
&SETVARS PREFIX=K# KEYWORDS=(OPT,USER,FUNC) +
   MODFLD=YES PARMS ERROR=CONTINUE
```

Notes:

&ZMODFLD is processed as a stack. Each reference to &ZMODFLD returns the top element of the stack, until it is empty, and resets the MODFLD attribute of the returned field. &ASSIGN OPT=MODFLD with the NORESET option is used to access modified field names and still allow the subsequent use of the &ZMODFLD function.

When a panel is displayed, and the &CONTROL FLDCTL option is in force, NCL identifies all the fields on the panel which have been modified when the user causes input from the terminal. In addition, fields flagged as MODIFIED or ERRFLD by &ASSIGN OPT=SETMOD/SETERR before the panel display, or fields flagged by panel services field validation (when &CONTROL PANELRC is in effect) are also returned by the &ZMODFLD function.

For example, if a panel is displayed with ten input fields and the user changes three fields and presses Enter, successive references of &ZMODFLD return the names of the three modified fields.

The fields are returned in the order they appear on the screen, that is, they are processed on a left to right, top to bottom basis. This assists in the analysis and editing processes required in NCL procedures when handling multi-field input from terminals displaying NCL panels. Field validation is setup in an efficient manner to process only those fields modified rather than process all fields that exist on a panel.

The MODFLD attribute of a field is retained, unless cleared by &ZMODFLD, or by &ASSIGN OPT=MODFLD, or by a panel display. If a field still has a MODFLD attribute when a subsequent panel is displayed and it appears on that panel, then it is again treated as being modified by the user and appears in the &ZMODFLD stack. This is particularly useful when processing panels that contain selection lists. The user might enter several selections one of which is incorrect. The panel is redisplayed, highlighting the field in error. After the second display, the original correct selections can still be processed via &ZMODFLD.

The &SETVARS and &NDBGET statements also support the optional use of the MODFLD attribute.

More information:

&ASSIGN (see page 206)
&SETVARS (see page 604)
&NDBGET (see page 472)
&CONTROL (see page 266)

&ZMODSRCID

(Message profile variable) Contains the message prefix for the originator of the message just received.

&ZMOSRCTP

(Message profile variable) Indicates the type for the originator of the message just received. Values are:

null

If the message was generated within this system

ROF

If the message was delivered across a ROF session

MAIOC

If the message was delivered across an MAI-OC session

&ZMPPODTA

(Message profile variable) Indicates whether any PPO message profile information is available concerning this message.

Value is YES or NO. If YES, then other message profile variables are available containing information about certain PPO attributes of the message.

More information:

&ZMPPOSEV (see page 943)

&ZMPPOMSG

(Message profile variable) Indicates whether the message originated from PPO. Value is YES or NO.

&ZMPPOSCNT

A counter of remote domains to which a PPO message was delivered.

After an &PPOALERT LINK=* or &PPOALERT DOMAIN=* statement is executed, this variable is set to number of domains to which the message was sent across ISR.

More information:

&ZMPPODTA (see page 942)

&ZMPPOSEV

(Message profile variable) If &ZMPPODTA=YES, then this variable gives the severity level of the PPO message.

Values are:

U

Undeliverable

I

Information

W

Warning

N

Normal

S

Severe

More information:

&ZMPPODTA (see page 942)

&ZMPPOTM

(Message profile variable) If &ZMPPODTA=YES, this variable gives the time when the message was created.

More information:

&ZMPPODTA (see page 942)

&ZMPPOVNO

(Message profile variable) If &ZMPPODTA=YES, this variable contains the VTAM message number for the PPO message.

More information:

&ZMPPODTA (see page 942)

&ZMPREFXD

(Message profile variable) Indicates whether the message text has been prefixed with identifier values. For example, an MAI-OC session ID or a ROF message prefix. Value is YES or NO.

&ZMPTEXT

(Message profile variable) &ZMPTEXT is set to the entire message text, prefixed with any ROF or MAI-OC session identifiers.

&ZMREQID

(Message profile variable) If &ZINTYPE=REQ (that is, &INTREAD is satisfied by a request message), this variable is set to either of the following:

- The user ID for the user issuing the INTQ command that generated the request message, or
- The NCL ID for the NCL process that issued the INTQ command or &WRITE verb.

&ZMREQID depends on &ZINTYPE for its relevance; its setting is categorized by the &ZMREQSRC variable.

Note: &ZMREQID is available for &INTREAD only.

&ZMREQSRC

(Message profile variable) If &ZINTYPE=REQ (that is, &INTREAD is satisfied by a request message), this variable indicates whether the source of the INTQ command generating the message as either:

USER

The source is a user

NCL

The source is another NCL process

SYSTEM

The source is a system notification

Note: &ZMREQSRC is available for &INTREAD only.

&ZMSLEVEL

Indicates the version of System Services.

Use this variable to assist in structuring procedures that are to be run in environments with mixed versions of System Services. It returns a six-digit value that corresponds to the current version.

Example: &ZMSLEVEL

```
&IF &ZMSLEVEL LT 050000 &THEN +
 &WRITE DATA=System does not support shortcuts
```

&ZMSOLIC

(Message profile variable) Indicates whether the message was solicited or unsolicited. A solicited message is usually a command response. Values are YES (that is, solicited) or NO.

&ZMSOURCE

(Message profile variable) Indicates the verb that last set the values for the message profile variables. The suite of message profile variables remains set until changed by the execution of another verb which modifies that suite of variables.

Valid values of &ZMSOURCE are:

- INTREAD
- LOGREAD
- MSGREAD

&ZMTEXT

(Message profile variable) Contains the text of the message received. If the message is a delete operator message (DOM), a null value is returned.

After &LOGREAD, the text does not include the standard log message heading information of user ID, time and terminal name. These values are available from the &ZMLUSER, &ZMLTIME, and &ZMLNODE message profile variables that are set after &LOGREAD.

&ZMTYPE

(Message profile variable) Specifies the type of message received after execution of the &READ verb. Values are:

MSG

The message is a standard text message

DOM

The message is a delete operator message instruction

NONE

The message is a VTAM PPO command echo

REQ

The message is a request message that has satisfied &INTREAD TYPE=ANY, or &INTREAD TYPE=REQ

More information:

&ZMDOM (see page 933)
&ZMMSG (see page 938)

&ZNCLENV

Returns the execution environment of an NCL process. The execution environment is one of the following:

PRIMARY

The process is running under a primary environment.

DEPENDENT

The process is running under a dependent environment.

&ZNCLID

Returns the unique identifier of the NCL process.

Each NCL process in the system has a unique identifying number, which is used to identify individual processes and to allow communications from a terminal to a particular procedure. In an NCL environment in which many independent processes is executing concurrently, GO, INTQ, and FLUSH commands issued either from an OCS window or from an NCL process executing in the same environment, can use the NCL identifier to communicate directly with a specific process.

The &ZNCLID system variable allows an NCL procedure to determine its own NCL ID.

Example: &ZNCLID

```
&WRITE DATA=NCL Proc &ZNCLID recovering controller &PUNAME.
```

Notes:

Each NCL process in the system is allocated a unique identifier when it is started, and it retains this identifier until it ends. Multiple invocations of the same procedure therefore have the same name but different NCL IDs.

GO, INTQ, and FLUSH commands normally apply only to NCL procedures that are executing in the NCL environment associated with the terminal or procedure that issues the commands.

Note: For more information, see the GO, INTQ, FLUSH, and SHOW NCL command descriptions in the Online Help.

&ZNCLNEST

Returns the current procedure's EXEC nesting level within method level.

When an NCL process is invoked the first procedure executed is called the base procedure. If the base procedure issues an EXEC command to invoke another procedure, the second procedure is a nested procedure. This second procedure can EXEC other procedures. Any procedure can inspect the &ZNCLNEST system variable to determine its nesting level, which is the number of procedures deep from the base procedure. The value of &ZNCLNEST is a number in the range 1 to 250 (1 represents the nesting level of the base procedure).

Example: &ZNCLNEST

```
&IF &ZNCLNEST EQ 250 &SYSMSG = &STR MAXIMUM PROCEDURE DEPTH
```

A maximum of 250 nesting levels is allowed in any one process.

&ZNCLTYPE

Returns the type of the current procedure.

NCL processes is one of several different types. Most NCL processes are standard types, that is, they can use all the standard NCL statements, built-in functions, and system variables.

Some NCL processes, however, are running as a special type of process, and are thus authorized to use NCL statements, built-in functions, and system variables that are only available to that type of process.

As an example, the LOGPROC NCL process can use the &LOGREAD, &LOGDEL, and &LOGREPL NCL statements. No other NCL process can use these statements.

When writing generic NCL procedures, it might be necessary to determine the exact type of NCL process that the procedure is executing as. The &ZNCLTYPE system variable provides this.

The &ZNCLTYPE system variable provides the following values when referenced:

STD

The NCL process is a standard process. No special facilities are available. This type includes both non-full-screen and full-screen mode procedures.

AOMP

The NCL process is executing as the primary AOMPROC. The &AOM*xxx* NCL verbs are used. This value can only be returned if the AOM feature is installed.

AOMS

The NCL process is executing as a secondary AOMPROC. The &AOM*xxx* NCL verbs are used. This value can only be returned if the AOM feature is installed.

CNMP

The NCL process is executing a CNMPROC. The &CNM*xxx* NCL verbs are used. This value can only be returned if the NEWS feature is installed.

LOGP

The NCL process is executing as LOGPROC. The &LOG*xxx* NCL verbs are used.

MAIS

The NCL process is executing as an MAI script process. The &MAI*xxx* NCL verbs are used. This value can only be returned if the MAI/EF feature is installed.

MSGP

The NCL process is executing as a MSGPROC. The &MSG*xxx* NCL verbs are used.

PPOP

The NCL process is executing as PPOPROC. The &PPO*xxx* NCL verbs are used.

LOCK

The NCL process is running to handle a LOCKed session.

Example: &ZNCLTYPE

```
&IF &ZNCLTYPE NE MSGP &THEN +
    &ENDAFTER &WRITE DATA=ABORTING - NOT A MSGPROC
```

Note: The &ZLUTYPE system variable provides an indication of the region in which an NCL process is executing.

More information:

&ZLUTYPE (see page 908)

&ZNETID

Returns the value of the VTAM network identifier.

Identifies the network name of the VTAM in which the system is executing. The network name is as defined by the NETID operand specified in the VTAM initialization parameters.

Example: &ZNETID

```
&GOTO .&ZNETID
 :
 .CORPNET1
 .
 .-* Processing specific to this network
 .
.CORPNET2
 .
 .-* Processing specific to this network
 .
```

Notes:

There is one or more VTAM systems in the same network. This variable is used to control processing, based on which VTAM network the system is running under.

&NETID is only available on IBM systems. Other systems return a null value.

&ZNETNAME

Returns the network name of the primary ACB.

The &ZNETNAME variable returns the network name of the APPL definition used for the system's primary ACB. The network name might differ from the ACBNAME, depending upon how the APPL has been defined.

Some versions of VTAM might not support the interrogation of network name, in which case &ZNETNAME returns a null value.

Example: &ZNETNAME

```
& WRITE DATA=SYSTEM SERVICES OPERATING WITH +
 ACB=&ZNETNAME
```

Note: &NETNAME is only available on IBM systems. Other systems return a null value.

&ZNMDID

Returns the value of the system's domain identifier.

The NMDID system initialization parameter specifies a 1- to 4-character domain identifier. If not specified, a default of up to the first four characters of the primary ACBNAME is used. The domain identifier should be unique throughout all connected systems.

Example: &ZNMDID

```
&WRITE DATA=The system domain identifier is &ZNMDID.
```

Notes:

The SHOW DOMAINS command is used to determine the domain identifier of the local system and all currently connected systems.

The SHOW PARMS command is used to determine the settings of basic system initialization parameters.

More information:

&ZNMSUP (see page 951)

&ZNMSUP

Returns the value of the system user prefix.

The NMSUP system initialization parameter, as specified in the startup JCL.

It is a 1- to 4 -character system user prefix value. If not specified, the NMDID value, if specified, or the first four characters of the primary ACBNAME is used.

This value is used to construct the user IDs of the various background regions, for example, BSYS (where NMINIT and NMREADY execute) has a user ID of *dddd*BSYS, where *dddd* is the NMSUP value.

&ZOCS

Indicates whether the NCL process is associated with an OCS window.

This variable returns a value of YES if the process was invoked directly from an OCS window (via the START or EXEC command) or indirectly by another process executing in an OCS environment. A process executing in any other environment that is not associated with an OCS window (User Services, for example) sees the value of &ZOCS as NO.

Examples: &ZOCS

```
&IF &ZOCS EQ NO &THEN +
  &WRITE DATA=This must be run from OCS only
```

&ZOPS

Indicates the specific type of operating system.

When operating in mixed operating system environments it is desirable to structure NCL procedures in such a way that they are operating system independent. The &ZOPS system variable allows a procedure to test the specific type of operating system and modify processing accordingly.

The &ZOPS variable returns one of the following values:

MSP

The operating system is MSP.

MSPEX

The operating system is MSP/EX.

VMESA

The operating system is VM/ESA.

VMGCS

The operating system is VM/GCS.

VOS3

The operating system is VOS3.

z/OS

The operating system is z/OS.

Example: &ZOPS

```
& IF &ZOPS EQ VOS3 &THEN +
     &WRITE DATA=ABENDCMD COMMAND NOT AVAILABLE
```

Notes:

The system rejects commands that are not valid in certain operating systems. Use of &ZOPS allows an NCL procedure to tailor the operator interface so that certain options, perhaps on full-screen panels, do not appear when running in some environments.

&ZOPS returns the specific operating system type. If a generic test is adequate (for example, if you want to know whether you are operating under an MVS system of some sort), then using &ZGOPS might be simpler.

If you need to know the specific version of the operating system, then you need to use &ZOPSVERS as well as &ZOPS.

More information:

&ZGOPS (see page 897)
&ZOPSVERS (see page 954)

&ZOPSVERS

Indicates the version of the operating system under which your product region is operating.

When operating in mixed operating system environments it is desirable to structure NCL procedures in such a way that they are operating system independent. The &ZOPSVERS system variable allows a procedure to test the version of the operating system and modify processing accordingly.

The &ZOPSVERS variable returns version values in the following formats for the operating systems listed:

MSP

Format *nnnn* (for example, 1010)

MSPEX

Format *nnnn* (for example, 1030)

z/OS

Format *vv.rr.mm* (for example, 01.04.00 is z/OS V1R4)

VMESA

Format *L.svc* (for example, 1.000)

VMGCS

Format *L.svc* (for example, 1.000)

VOS3

Format *aa-bb* (for example, 10-20)

Examples: &ZOPSVERS

```
&IF &ZOPS EQ z/OS AND +
    &ZOPSVERS EQ 01.04.00 &THEN +
    &WRITE DATA=Running z/OS V1R4
```

Notes:

The system rejects commands that are not valid in certain versions of certain operating systems. Use of &ZOPSVERS allows an NCL procedure to tailor the operator interface so that certain options, perhaps on full-screen panels, do not appear when running in some environments.

&ZOPSVERS returns the specific version of an operating system type. If a test of the operating system type is adequate (for example, if you want to know whether you are operating on an MVS system of any version), then using &ZOPS might be simpler.

More information:

&ZGOPS (see page 897)
&ZOPS (see page 952)

&ZOUSERID

Provides the originating user ID for an NCL process that was submitted, or is executing as the result of a timer command.

Is used to determine the user ID of the original command.

Example: &ZOUSERID

```
&IF &ZOUSERID NE &USERID &THEN +
    &WRITE DATA=PROC SOURCED EXTERNALLY
```

Notes:

The originating user might no longer be signed on.

The originating user ID is tracked through any number and level of AT, EVERY, and SUBMIT commands.

The originating user ID is not tracked through ROF ROUTE commands to another system.

More information:

&USERID (see page 864)

&ZPERRORC

Contains the value of the standard panel field attribute COLOR (or COLOUR) for error fields and messages.

Specified in distributed panel definitions.

&ZPERRORH

Contains the value of the standard panel field attribute HLIGHT (or HLITE) for error fields.

Specified in distributed panel definitions.

&ZPINPHIC

Contains the value of the standard panel field attribute COLOR (or COLOUR) for mandatory input data fields and command fields.

Specified in distributed panel definitions.

&ZPINPLOC

Contains the value of the standard panel field attribute COLOR (or COLOUR) for optional input data fields.

Specified in distributed panel definitions.

&ZPINPUTH

Contains the value of the standard panel field attribute HLIGHT (or HLITE) for data input fields.

Specified in distributed panel definitions.

&ZPINPUTP

Contains the value of the standard panel field attribute PAD for data input fields.

Specified in distributed panel definitions.

&ZPLABELC

Contains the value of the standard panel field attribute COLOR (or COLOUR) for field labels and comments.

Specified in distributed panel definitions.

&ZPMTEXT1

Returns the text of the Primary Menu broadcast. To obtain the current Primary Menu broadcast text as set by the NSBRO command.

Examples: See the distributed panel $NMPMENU.

The NSBRO PM1 operand is used to set the Primary Menu text. This system variable allows the text to be used in NCL and substituted onto panels.

Note: For more information, see the NSBRO command in the Online Help and the *Reference Guide*.

&ZPOUTHIC

Contains the value of the standard panel field attribute COLOR (or COLOUR) for output data fields that are always present.

Specified in distributed panel definitions.

&ZPOUTLOC

Contains the value of the standard panel field attribute COLOR (or COLOUR) for output data fields that are not always present.

Specified in distributed panel definitions.

&ZPPKEYC

Contains the value of the standard panel field attribute COLOR (or COLOUR) for the output fields on the left and right of the panel title and the function key area.

Specified in distributed panel definitions.

&ZPPI

Indicates whether or not PPI is available.

To determine whether or not PPI is available in this system. The values that is returned are:

YES

Indicates that the PPI interface is initialized.

NO

Indicates that PPI is not available.

&ZPPI can thus be used as a test to determine whether a procedure can issue &PPI verbs.

&ZPPINAME

Contains the defined receiver ID of the current NCL process.

To determine the receiver ID of the current NCL process.

The value returned is the name specified on the ID= operand of the &PPI DEFINE, or the generated name if ID=* was specified. If the &PPI DEFINE was unsuccessful, or an &PPI DEACTIVATE is issued, &ZPPINAME returns a null value.

&ZPRINAME

Contains the name of the primary ACB or XNF UCE.

The value returned is the name specified on the PRI= JCL parameter. Use it in preference to &ZACBNAME for code executing on SSI or XNF.

&ZPRODNAM

Returns the product name.

The variable allows an NCL procedure to determine the product name of the system.

Example: &ZPRODNAM

```
&WRITE DATA=&ZPRODNAM will terminate in 5 minutes
```

&ZPSERVIC

Returns the value of the first four bytes of the PSERVIC field of the BIND for the current terminal.

The PSERVIC fields of the BIND parameters used when the current terminal was connected can be interrogated using the &ZPSERVIC variable. Certain terminal types might have non-standard PSERVIC values indicating particular device characteristics. NCL procedures can use &ZPSERVIC to gain access to these settings.

Example: &ZPSERVIC

```
&1 = &SUBSTR &ZPSERVIC 3 1
&IF &1 EQ 10 &THEN +
    &GOSUB .SPECIAL-DEV
```

Note: The PSERVIC field of the BIND parameters identifies the type of device that is connected, and provides information such as screen sizes. Additional bits within the PSERVIC field can also be used in Fujitsu installations to indicate special terminal attributes, such as DBCS or three-color support.

&ZPSKIP

Returns the next available segment of panel skip data.

The &ZPSKIP system variable is used to retrieve the next available segment of panel skip data. The stored panel skip string is automatically updated to remove this segment. The &ZPSKIP reference is thus performing a similar action to a panel display. The invoking NCL procedure should then use the returned data as if you had entered it as a menu option.

Example: &ZPSKIP

The following code illustrates typical &ZPSKIP usage:

```
&SELECT = &ZPSKIP
.PROCESS
   &GOTO .MENU &SELECT
   &SYSMSG = &STR MSG004 INVALID SELECTION

.MENU
   &PANEL USERPANEL
 ...
   &GOTO .PROCESS

.MENU1
 ...

.MENU2
 ...
```

Notes:

The system supports the ability to perform menu jumps or panel skips as a means of abbreviating panel navigation. Panel skips is used to move rapidly from one panel display to another without viewing each individual panel which would normally be displayed. The panel skip data is a variable length string wherein data for each panel is delineated by the use of a period (.). This data is normally used to automatically satisfy &PANEL statements.

The use of &ZPSKIP as a system variable is not required to implement panel skipping. The system automatically uses panel skip segments to satisfy &PANEL requests.

The &ZPSKIP system variable returns a null value if the current panel skip segment contains a field separator character.

When the &ZPSKIP variable returns a non-null value, the value of &INKEY is set to ENTER.

More information:

&ZPSKIP (see page 720)
&ZPSKPSTR (see page 961)

&ZPSKPSTR

Returns the current panel skip string in its entirety.

Allows the entire panel skip string to be stored and/or examined.

Example: &ZPSKPSTR

```
&SKPALL = &ZPSKPSTR
```

Notes:

&ZPSKPSTR does not alter the value of the current panel skip data. The primary menu procedure uses the following code to suppress panel skipping and then restore it if required:

```
&T#ZPSKIP = &ZPSKPSTR  -* extract value
&ZPSKIP  -* nullify current setting
...
...
&ZPSKIP = &T#ZPSKIP  -* restore panel skip The &ZPSKIP system variable and &ZPSKIP verb.
```

More information:

&ZPSKIP (see page 720)
&ZPSKIP (see page 959)

&ZPSUBTLC

Contains the value of the standard panel field attribute COLOR (or COLOUR) for sub-titles, headings and trailers.

Specified in distributed panel definitions.

&ZPTITLEC

Contains the value of the standard panel field attribute COLOR (or COLOUR) for the panel title. It is specified in distributed panel definitions.

&ZPTITLEP

Contains the value of the standard panel field attribute PAD for the panel title. Specified in distributed panel definitions.

&ZPWSTATE

Returns the state of a user's password when they log on.

Provides NCL with the ability to interrogate the state of a user's password. The following values are returned:

'Blank'

There is no password expiry outstanding for this user.

EXPIRED

The user's password has expired and needs to be changed.

&ZREMIPA

Returns the IP address of a remote host for a TN3270 session.

Identifies the IP address of a remote TN3270 server.

Note: This information is available only if the IPCHECK SYSPARMS operand has been set to REGISTER (the default) in the INIT procedure for your product region.

Example: &ZREMIPA

```
&IF .&ZREMIPA NE . &THEN +
   &DO
      &SOCKET PING ADDRESS=&ZREMIPA
```

Note: For more information, see the IPCHECK SYSPARMS operand in the *Reference Guide* and the SHOW USERS command description in the Online Help.

More information:

&ZLCLIPA (see page 905)
&ZLCLIPP (see page 906)
&ZREMIPP (see page 963)

&ZREMIPP

Returns the remote host IP port for a TN3270 session.

Identifies the remote IP port of a TN3270 connection.

This information is available only if the IPCHECK SYSPARMS operand has been set to REGISTER (the default) in the INIT procedure for your product region.

Note: For more information about the IPCHECK SYSPARMS operand, see the *Reference Guide*.

More information:

&ZLCLIPA (see page 905)
&ZLCLIPP (see page 906)
&ZREMIPA (see page 962)

&ZROWS

Indicates the number of rows available to the physical terminal.

NCL procedures might want to determine the number of rows (or lines) associated with the physical terminal. &LUROWS will return the number of lines in the current processing window. However, when operating in split screen mode, this might differ from the number of rows available on the actual screen. &ZROWS is the maximum number of rows available at the terminal regardless of the current processing window size.

While it is good practice to write procedures that cater for the largest screen size (for example: 62 lines for a 3290) and that automatically adjust if used on a smaller screen, it might offer performance advantages to format output to suit the maximum size required for the requesting terminal. This allows the terminal operator to alter screen dimensions as necessary, but also eliminates the formatting of unnecessary data.

Example: &ZROWS

```
& IF &CNT GT &ZROWS &THEN +
  &GOTO .NOMORE
```

Note: When processing with an LU1 type device, a value of 1 is returned. For VDU(screen) type devices, a value of 1 to 62 is returned. If tested from a procedure running under a system region, a value of 1 is returned. Therefore, &ZROWS always returns a value between 1 and 62.

More information:

&LUROWS (see page 842)
&ZCOLS (see page 885)

&ZSCOPE

Contains the scope of the server name if the current NCL process is registered as a server. Otherwise it is null.

&ZSECEXIT

Returns the type of security exit installed. Provides NCL with the ability to interrogate the system to determine what type of security exit the installation is using.

The following values are returned:

NONE

This system is running with local UAMS only.

PARTIAL

This system is running with a partial security exit.

FULL

This system is running with a full security exit.

Note: For more information about security exits, see the *Security Guide*.

&ZSERVER

Contains the server name if the current NCL process is registered as a server. Otherwise it is null.

&ZSNAMID

Returns an integer when using the &SNAMS verbs.

The integer returned is in the range 1 to 2,147,483,647 and is set following a successful &SNAMS REGISTER or &SNAMS SEND request.

It can subsequently be used on an &SNAMS RECEIVE request to indicate that only messages associated with that particular identifier (that is, the previous application registration, or the previous solicitation request) should satisfy the receive operation.

Its value is set following successful completion of various &SNAMS requests:

&SNAMS REGISTER

The integer returned is the registration identifier of the indicated application. This value is used on a subsequent &SNAMS DEREGISTER request to deregister the application. It may also be used on &SNAMS RECEIVE and RECEIVE_NOTIFY requests to target messages designated for the registered application.

&SNAMS SEND

&SNAMID is set only if the MDS-MU sent is a solicited request expecting one or more MDS-MU replies. The integer returned represents the request identifier. It may be used on subsequent &SNAMS_RECEIVE and RECEIVE_NOTIFY requests to correlate received responses.

&SNAMS RECEIVE

The integer returned is the message identifier. If the message received is a response to an outstanding request, it corresponds to the request identifier of the initial MDS request. Otherwise it is the registration identifier of the destination application.

Example: &ZSNAMID

```
&SNAMS REGISTER APPL=USERAPPL         -* Register an MS application
&APPLID=&ZSNAMID                      -* Save its registration ID
&SNAMS SEND MU=query                  -* Send an MDS request
&REQID=&ZSNAMID                       -* Save the request ID
&SNAMS RECEIVE MU=response ID=&REQID  -* Receive the response
&SNAMS RECEIVE MU=MSU ID=&APPLID      -* Receive an MDS-MU targeted
                                         at the registered application
```

&ZSOCCID

Returns the socket ID used by the interface; for example, when using IBM TCP/IP, the internal socket number (a small numeric value).

&ZSOCCID is used to identify an NCL socket or display produced by TCP/IP (for example, the NETSTAT command).

Example: &ZSOCCID

```
&SOCKET CONNECT HOSTNAME=&HOST PORT=&PORT
&IF &RETCODE EQ 0 &THEN +
    &WRITE DATA=SOCKET &ZSOCID CONNECTED USING INTERFACE ID &ZSOCCID
```

&ZSOCERRN

Returns the error number value associated with the last referenced socket.

&ZSOCERRN is set in combination with &ZFDBK and &ZSOCVERR. For more information, see the appendix "&SOCKET Verbs".

Example: &ZSOCERRN

```
&SOCKET SEND ID=&SOCKID DATA=&MSG
&IF &RETCODE NE 0 &THEN +
   &WRITE DATA=SEND FAILED ERRNO=&ZSOCERRN
```

More information:

&ZSOCVERR (see page 968)

&ZSOCFHNM

Returns the full host name of the host referenced by some requests (for example, &SOCKET GETHOSTBYNAME).

&ZSOCFHNM is set by the host lookup functions GETHOSTBYNAME and GETHOSTBYADDR.

It is also updated by &SOCKET calls with the name associated with the last referenced sockets.

Example: &ZSOCFHNM

```
&SOCKET GETHOSTBYADDR ADDRESS=198.4.58.4
&WRITE DATA=HOST &ZSOCFHNM
```

&ZSOCHADR

Returns the IP address of the host referenced by some requests (for example, &SOCKET GETHOSTBYADDR).

&ZSOCHADR returns the IP address of the remote host for a TCP socket. For a UDP socket it returns the IP address last sent to or received from. &ZSOCHADR is also set by the host lookup functions &SOCKET GETHOSTBYNAME and GETHOSTBYADDR.

Example: &ZSOCHADR

```
&SOCKET RECEIVE_FROM ID=&SOCKID VARS=D*
&WRITE DATA=RECEIVED FROM &ZSOCHADR
```

&ZSOCHNM

Returns the host name of the host referenced by some requests (for example, &SOCKET GETHOSTBYNAME).

&ZSOCHNM is set by the host lookup function GETHOSTBYNAME and GETHOSTBYADDR.

It is also updated by &SOCKET calls with the name associated with the last referenced socket.

Example: &ZSOCHNM

```
&SOCKET GETHOSTBYNAME HOSTNAME=&HOST
&WRITE DATA=HOST &HOST KNOWN AS &ZSOCHNM
```

&ZSOCID

Returns the socket ID of the last referenced socket.

Socket IDs are uniquely numbered in the range 1 to 999999. Use the SHOW SOCKETS command to display active sockets.

&ZSOCID returns a null value after &SOCKET CLOSE, &SOCKET DEALLOCATE, &SOCKET GETHOSTBYNAME, and &SOCKET GETHOSTBYADDR.

&ZSOCPRT

Returns the port number of the last referenced socket.

&ZSOCTYPE

Returns the type of the last referenced socket. Notes: The values returned is:

TCP

A TCP client created by using &SOCKET CONNECT/ACCEPT.

TCPLISTEN

A TCP server created by using &SOCKET REGISTER.

UDP

A UDP socket created by using &SOCKET OPEN.

&ZSOCVERR

Returns vendor error information from the last referenced socket.

&ZSOCVERR is set in combination with &ZFDBK and &ZSOCERRN.

Vendor error codes are mapped (normalized) into &ZSOCERRN values. Unmapped values result in an &ZSOCERRN value of 999.

More information:

&ZSOCERRN (see page 966)
&SOCKET Verbs (see page 1213)

&ZSSCPNAM

Returns the value of the VTAM SSCP name.

If supported by the version of VTAM, &ZSSCPNAM identifies the System Services Control Point name of the VTAM in which the system is executing as specified in the VTAM initialization parameters.

Example: &ZSSCPNAM

```
&WRITE DATA=VTAM SSCP=&ZSSCPNAM
```

If the level of VTAM in use does not support the provision of SSCP name, &ZSSCPNAM returns a default of VTAM.

&ZSYSNAME

Returns the MVS SYSNAME value. If the value is not available (if the operating system is non-MVS) a value of UNKNOWN is returned.

&ZTCP

Returns the status of the socket interface.

Check the status of the API and take the appropriate action.

Example: &ZTCP

```
& IF &ZTCP NE ACTIVE &THEN +
 &DO      -* Write the error message and return
    &WRITE THE TCP/IP INTERFACE IS INACTIVE
    &END
 &DOEND
.
.
```

Note: The value in this system variable is obtained by performing a SHOW TCPIP command.

More information:

&ZTCPHSTA (see page 970)
&ZTCPHSTF (see page 970)
&ZTCPHSTN (see page 971)

&ZTCPHSTA

Returns the value of the local host's IP address.

Example: IP address

```
192.168.2.66
```

Note: The value in this system variable is obtained by performing a SHOW TCPIP command. The value held in this system variable is only reliable if the socket interface is active; that is, if &ZTCP = ACTIVE.

More information:

&ZTCP (see page 969)
&ZTCPHSTF (see page 970)
&ZTCPHSTN (see page 971)

&ZTCPHSTF

Returns the value of the local host's full name.

Example: host full name

```
TESTMVS1.SYDNEY.TESTING.COM
```

Note: The value in this system variable is obtained by performing a SHOW TCPIP command. The value held in this system variable is reliable only if the socket interface is active; that is, if &ZTCP = ACTIVE.

More information:

&ZTCP (see page 969)
&ZTCPHSTA (see page 970)
&ZTCPHSTN (see page 971)

&ZTCPHSTN

Returns the value of the local host's short name.

Example: host short name

```
MERCURY
```

Note: The value in this system variable is obtained by performing a SHOW TCPIP command. The value held in this system variable is only reliable if the socket interface is active; that is, if &ZTCP = ACTIVE.

More information:

&ZTCP (see page 969)
&ZTCPHSTA (see page 970)
&ZTCPHSTF (see page 970)

&ZTIMEn

Provides different formats of the current time.

&ZTIME1, &ZTIME2, &ZTIME3, &ZTIME10, and &ZTIME11 are system variables which supply the current time in several different formats:

- &ZTIME1 time as HH:MM:SS
- &ZTIME2 time as HH:MM:SS.TH
- &ZTIME3 time as *nnnnnnn*
- &ZTIME10 time as HHMMSS
- &ZTIME11 time as HHMMSS.FFFFFF

where:

HH

Current hour of day (in 24-hour time).

MM

Current minute.

SS

The current second.

TH

Hundredths of a second.

nnnnnnn

The time since midnight in hundredths of a second. Leading zeros are suppressed.

FFFFFF

The time accurate to 10^{-6}

More information:

&TIME (see page 863)
&ZGTIMEn (see page 898)

&ZTSOUSER

Indicates if the user has connected to the product region through the TSO or TSS interface.

NCL procedures might want to determine if users have logged on through the TSO (or TSS) interface, and if so, perform appropriate processing.

&ZTSOUSER will return one of the following values:

YES

The user has logged on using the TSO interface.

NO

Access was not made using the TSO interface.

Example: &ZTSOUSER

```
&IF &ZTSOUSER EQ YES &THEN +
    &GOTO .TSOMENU
```

Note: &ZTSOUSER is used to provide TSO users with special processing.

&ZUCENAME

Returns the UCE name which your product region is using to communicate with XNF (used by Hitachi VOS3 systems only).

Example: &ZUCENAME

```
&GOTO .&ZUCENAME
 .
 .
 .
.ERROR
   &ENDAFTER &WRITE DATA=This procedure is restricted to +
                 production systems.
 .
 .
 .
.NMPROD1  -* Production system number 1
 .
 .
 .
.NMPROD2  -* Production system number 2
 .
 .
 .
```

Note: Is null if your product region is not using the XNF access method.

&ZUDATEn

A set of system variables that return the user's date, in different formats, time zone adjusted.

Use:&ZUDATE1 to &ZUDATE17 (excluding &ZUDATE15) are system variables that supply the current user's date, adjusted by time zone, in a variety of formats:

- &ZUDATE1—date as YY.DDD
- &ZUDATE2—date as DAY DD-MON-YYYY
- &ZUDATE3—date as DD-MON-YYYY
- &ZUDATE4—date as DD/MM/YY
- &ZUDATE5—date as MM/DD/YY
- &ZUDATE6—date as YY/MM/DD
- &ZUDATE7—date as YYMMDD
- &ZUDATE8—date as YYYYMMDD
- &ZUDATE9—date as *nnnnnn*
- &ZUDATE10—date as YYYYMMDDHHMMSSZ
- &ZUDATE11—date as YYYYMMDDHHMMSS.FFFFFFZ
- &ZUDATE12—date as DD/MM/YYYY
- &ZUDATE13—date as YYYY/MM/DD
- &ZUDATE14—date as MM/DD/YYYY
- &ZUDATE16—date as YYYY.DDD
- &ZUDATE17—date as YYYYDDD

where:

DAY

The day of the week as follows:

- MON Monday
- TU Tuesday
- WED Wednesday
- THU Thursday
- FRI Friday
- SAT Saturday
- SUN Sunday

DD

The day of the month as a 2-digit number

DDD

The Julian day within the year as a 3-digit number

MM

The month of the year as a 2-digit number

MON

The month of the year as follows:

- JAN January
- FEB February
- MAR March
- APR April
- MAY May
- JUN June
- JUL July
- AUG August
- SEP September
- OCT October
- NOV November
- DEC December

nnnnnn

The number of days from 1 January 0001 with no leading zeros

YYYY

The current year as a 4-digit number

YY

The current year as a 2-digit number

Z

Greenwich Mean Time (GMT)

FFFFFF

The time accurate to 10^{-6}

Example: &ZUDATE3

```
&FILEDATE = &ZUDATE3
```

Note: Use of &ZUDATE11 should be avoided, to reduce unnecessary overhead, if microsecond accuracy is not required. Each access to &ZUDATE11 causes the system to refetch and synchronize time with the operating system, to format the result to microsecond accuracy.

&ZUDAY

Returns the user's day of the week, time zone adjusted.

&ZUDAY provides a system variable for the user's day of the week, time zone adjusted. It is in the form DDD, where DDD is set to one of the following values:

MON

Monday

TUE

Tuesday

WED

Wednesday

THU

Thursday

FRI

Friday

SAT

Saturday

SUN

Sunday

Example: &ZUDAY

```
&IF &ZUDAY EQ SUN &THEN -EXEC SUNDAY
&ELSE +
 -EXEC EVERYDAY
```

Notes:

The current date, based on the operating system time, is provided in different formats by the system variables &DATE1 to &DATE17.

The current date, based on GMT, is provided in different formats by the system variables &ZGDATE1 to &ZGDATE17.

The user's current date, time zone adjusted, is provided in different formats by the system variables &ZUDATE1 to &ZUDATE17.

More information:

&DAY (see page 827)
&ZGDAY (see page 896)

&ZUNIQUE

To supply a unique value each time it is referenced.

The return value is an 8-character string that contains hexadecimal characters (0 to 9 and A to Z). There are approximately 4 billion (4,000,000,000) values available.

The first value is 00000001. The last is FFFFFFFF.

&ZUSERLC

Provides the language code for this user.

Your product region supports the specification of a language code for a user. This code is used to provide language dependent processing from NCL. Such processing might include variation of error messages or selection of alternative help panels for a particular user.

Example: &ZUSERLC

```
&IF &ZUSERLC EQ AL &THEN +
   &GOTO .GERMAN &PANEL HELP&ZUSERLC
```

When displaying panels using the &PANEL statement, the &SYSMSG system variable is primed by the system with the text of an appropriate message in the event of an error. The default system messages in such cases are in English. Use of the &CONTROL PANELRC option allows interception of the &SYSMSG variable by the procedure, thus facilitating translation, based on the &ZUSERLC system variable.

Note: For more information, see the *Network Control Language Programming Guide*.

More information:

&CONTROL (see page 266)
&ZUSERSLC (see page 979)

&ZUSERSLC

Returns the system recognized language code for a user.

Your product region supports the use of national language character sets, based on the user's language code, which is returned in the &ZUSERSLC system variable.

Example: &ZUSERSLC

```
&CALL PROC=GETMSG PARMS=(ID=&MSGID,LANG=&ZUSERSLC)
```

When the user's language code (&ZUSERLC) is a system defined value, this is the value returned in &ZUSERSLC. If &ZSERLC is not a system defined value, the language code of the system (which is set using the SYSPARMS command LANG operand) is returned. If the system language code is not a system defined value, the value UK is returned.

More information:

&ZUSERLC (see page 979)
Supported Language Codes for National Language Support (see page 1187)

&ZUSRMODE

Returns a value indicating special conditions of this signed on user.

To determine if the signed on user requires special processing.

The following values are returned:

'blank'

No special processing

NEW

New user processing

INSTALL

The user is the INSTALL user

Example: &ZUSRMODE

```
&IF .&ZUSRMODE = .NEW &THEN +
   &GOSUB .NEW_USER
```

&ZUTIMEn

Provides different formats of the user's current time, time zone adjusted.

&ZUTIME1, &ZUTIME2, &ZUTIME3, &ZUTIME10, and &ZUTIME11 are system variables which supply the user's current time, time zone adjusted, in several different formats:

- &ZUTIME1 time as HH:MM:SS
- &ZUTIME2 time as HH:MM:SS.TH
- &ZUTIME3 time as nnnnn
- &ZUTIME10 time as HHMMSS
- &ZUTIME11 time as HHMMSS.FFFFFF

where:

HH

is current hour of day (in 24-hour time)

MM

is current minute

SS

is current second

TH

is hundredths of a second

nnnnnnn

is the time since midnight in hundredths of a second. Leading zeros are suppressed.

FFFFFF

is the time accurate to 10 -6

Each access to &ZUTIME11 causes the system to refetch and synchronize time with the operating system, to format the result to microsecond accuracy. Unless you require this level of accuracy, you should avoid using the &ZUTIME11 option, to reduce unnecessary overheads.

More information:

&TIME (see page 863)
&ZGTIMEn (see page 898)
&ZUTIMEZn (see page 982)

&ZUTIMEZn

A set of system variables that indicate the difference in time between local (operating system) time and the user's time zone.

&ZUTIMEZ, &ZUTIMEZ1, &ZUTIMEZ2, and &ZUTIMEZ3 are system variables that indicate the difference between local time and the user's time zone.

&ZUTIMEZ plus or minus HH.MM

&ZUTIMEZ1 plus or minus HH:MM

&ZUTIMEZ2 plus or minus HHMM

&ZUTIMEZ3 plus or minus nnnnn-an integer being hundredths of a second since midnight

Example: &ZUTIMEZ1

```
&TIMEDIFF = &ZUTIMEZ1
```

Each access to &ZUTIMEZ3 causes the system to refetch and synchronize time with the operating system, to format the result to microsecond accuracy. Unless you require this level of accuracy, you should avoid using the &ZUTIMEZ3 option, to reduce unnecessary overheads.

See Also: The &TIME, &ZTIMEn, &ZGTIMEn, &ZUTIMEn, and &ZUTIMEZN system variables.

&ZUTIMEZN

Returns the user's time zone name.

Example: &ZUTIMEZN

```
&IF &ZUTIMEZN=AUSESTDT -EXEC STDT +
    &ELSE
 .
 .
 .
```

&ZVARCNT

Returns the number of non-null variables created or modified by the last NCL verb that used generic processing. After processing an MDO, &ZVARCNT is set to one.

A number of NCL verbs are capable of generating the number of variables necessary to hold the data that results from their operation. The procedure can inspect this variable to determine how many variables have been created or modified.

Example: &ZVARCNT

```
&INTREAD ARGS           -* read next command reply
&WORDS = &ZVARCNT       -* see how many words in the message.
&A1=X
&A2=X
&B1=                    -* (Null)
&B2=Y
&B3=                    -* (Null)
&ASSIGN VARS=A* FROM VARS=B*
&ASSIGN VARS=(X,Y,Z) FROM VARS=(B1,B2,B3)
                        -* After this statement &ZVARCNT=1
```

&ZVTAMLVL

Returns the VTAM release and version number, if available.

If supported by the version of VTAM, the system variable &ZVTAMLVL is set to the character value of the VTAM release level. The format is as follows:

```
X.Y.Z
```

X

The version number

Y

The release number

Z

The modification level.

Example: &ZVTAMLVL

```
&IF &ZVTAMLVL EQ 6.1.4 &THEN +
   &WRITE DATA=VTAM LEVEL ACCEPTABLE
```

Note: &ZVTAMLVL is only available on IBM systems. Other systems return a null value.

&ZVTAMPU

Returns the host PU name of VTAM.

If supported by the version of VTAM, the system variable &ZVTAMPU is set to the value of host PU name defined for this VTAM system. The value is as defined in the VTAM initialization parameters using the HOSTPU operand.

Example: &ZVTAMPU

```
&WRITE DATA=VTAM PUNAME=&ZVTAMPU
```

Note: If the level of VTAM in use does not support the provision of the host PU name this system variable will return a default value of 'ISTPUS'.

&ZVTAMSA

Returns the subarea number of VTAM.

If supported by the version of VTAM, the system variable &ZVTAMSA is set to the value of the VTAM subarea number.

Example: &ZVTAMSA

```
&WRITE DATA=VTAM SUBAREA=&ZVTAMSA
```

Note: &ZVTAMSA is only available on IBM systems. Other systems return a null value.

&ZWINDOW

Returns the identifier of the current window.

Each window is assigned a number-either 1 or 2. &ZWINDOW returns the number of the window under which the NCL process is executing.

Example: &ZWINDOW

```
&IF &ZWINDOW = 2 &THEN +
    &GOTO .SECOND-WIND
```

Note: The SPLIT and SWAP commands is used to open a second window. The window number is displayed on the supplied primary menu.

More information:

&ZWINDOW# (see page 985)

&ZWINDOW#

The &ZWINDOW# system variable returns the number of active windows.

To determine is the user has activated a second window. A value of 1 is returned if one window is active, 2 is returned if both windows are active.

Examples: &ZWINDOW#

```
&IF &ZWINDOW# = 2 &THEN +
    &TEXT = &STR TWO WINDOWS ACTIVE
```

Note: The SPLIT and SWAP commands is used to open a second window.

More information:

&ZWINDOW (see page 984)

&ZWSTATE

Returns a value indicating the current window's state. To determine window conditions for specific processing. The following values are returned: SIGNON The window has been created as a result of a signon NEW The window has been newly created by SPLIT/SWAP processing OLD The window has been initialized during earlier processing

Examples: &ZWSTATE

```
&IF &ZWSTATE = SIGNON &THEN +
     &GOSUB .LOGON_MSGS
```

Note: The value OLD is not returned when used in procedures other than the primary menu procedure.

&0

Returns the name of the procedure currently being executed.

&0 is used within a procedure where a message from the procedure makes reference to the name of the procedure itself. If this is done and the procedure is renamed, no modification of the message text is required.

Example: &0

```
&WRITE DATA=Error in input to &0 invoked from &00
```

Note: The value returned from &0 will be a 1 to 8 character name. If the procedure is not nested, &0 will return the name of the base procedure that was originally invoked by the user.

&00

Returns the name of the base procedure of the NCL process.

&00 makes it possible for a nested procedure to have error messages issued that correctly reference the name of the invoking procedure.

Example: &00

```
&WRITE DATA=Error in input to &0 invoked from &00
```

Note: The value returned from &00 is a 1- to 8-character name.

&000

&000 returns the value of the NCL global variable prefix.

NCL supports global variables that is set and shared by all NCL procedures throughout the system. Global variables are identified by a unique prefix. This is, by default, set to GLBL.

An installation can alter the default global variable prefix using the SYSPARMS NCLGLBL command. A change to this prefix would then require subsequent changes to all NCL procedures that referenced global variables. To eliminate this problem the &000 system variable is set to the current value of the global variable prefix. This makes it possible for global variables to be referenced using complex variable techniques that cause the name of the variable to be dynamically resolved to include the current global variable prefix.

For example, consider that the current global variable prefix is #$ and that a variable &#$Z1 has been created. This could be referenced in the following way:

```
&WRITE DATA=THE CURRENT VALUE OF GLOBAL Z1 IS &#$Z1
```

An alternative method using the &000 system variable would be:

```
&WRITE DATA=THE CURRENT VALUE OF GLOBAL Z1 IS &&000Z1
```

&000 takes advantage of the NCL parsing rule that delimits a numeric variable at the first non-numeric character. Thus, when resolving the value of &&000Z1, &000 is delimited at the character Z, and resolved to #$, which is the current value of the global variable prefix. Parsing then continues to resolve the value of &#$Z1.

Example: &000

```
&&000Z1 = &STR THIS IS DATA IN GLOBAL VARIABLE Z1
```

Note: Do not use a single character for the global variable prefix in your system-always use a combination of at least two characters, preferably a combination of the national characters, for example @#$. This minimizes the risk of the global variable prefix clashing with any other user variables. You should also set NCL coding standards for your installation which prevent naming conventions that conflict with your choice of global variable prefix.

Chapter 4: PSM Interface

This section describes how to use the Print Services Manager (PSM) NCL interface, the operands that can be specified when executing procedure $PSCALL, and the return codes that are set on completion.

This section contains the following topics:

About the PSM NCL Interface

Print Services Manager (PSM) has an NCL interface which enables you to add output to the spool and invoke other PSM facilities, from an installation-written NCL procedure. For example, your application could provide a facility that allows the user to print information, by first requesting PSM to present the Confirm Printer panel on which they can enter a printer name, number of copies, and hold and keep settings, and then sending the information to be printed to the spool. This means the application does not have to be concerned about how to send information to the printer or what to do if the printer is unavailable, as this is all handled by PSM. The application can then request PSM to present the Output Queue showing the status of the user's output.

The PSM NCL interface gives you the flexibility of invoking any of its facilities from anywhere within your NCL applications.

To call PSM from an NCL procedure you must execute NCL procedure $PSCALL.

$PSCALL Options

Following is a list of the options for which procedure $PSCALL can be invoked. These options are fully described on the following pages of this chapter.

$PSCALL OPT=BROWSE

browse a print request or the output for a print request

$PSCALL OPT=CANCEL

cancel an open print request

$PSCALL OPT=CLOSE

close an open print request

$PSCALL OPT=CONFIRM

confirm printer details

$PSCALL OPT=DELETE

delete a print request from the spool

$PSCALL OPT=HEADER

add header lines to an open print request

$PSCALL OPT=HOLD

hold a print request

$PSCALL OPT=INFO

return printer, form or setup information

$PSCALL OPT=MODIFY

modify a print request

$PSCALL OPT=OPEN

open a print request

$PSCALL OPT=PUT

add output (data that is to be printed) to a print request

$PSCALL OPT=QUEUE

present the Output Queue

$PSCALL OPT=RELEASE

release a print request

$PSCALL OPT=BROWSE

Browses a print request or the output for a print request.

```
&CONTROL NOSHRVARS
-EXEC $PSCALL  OPT=BROWSE
             [ BROWSE={ OUTPUT | REQUEST } ]
               REQ=n
```

Presents the Browse Output panel showing the output for a print request or the Print Request panel showing the request details. You cannot browse the output for a request whose status is BUILD-PRT or DIRECT-ERR.

When the Browse Output or Print Request panel is terminated by the user entering the EXIT or RETURN command (or pressing the Function key to which those commands are assigned), control is returned to the NCL procedure that executed $PSCALL.

Operands:

OPT=BROWSE

Specifies that a print request or the output for a print request is to be browsed.

BROWSE={ OUTPUT | REQUEST }

Specifies whether the request, or the output for the request, is to be browsed.

OUTPUT

Presents the Browse Output panel

REQUEST

Presents the Print Request panel

REQ=*n*

Specifies the number of the print request to be browsed.

Return Codes:

0

$PSCALL completed successfully. &$PSFDBK may be set to the following value:

1

RETURN command entered (or the function key to which it is assigned was pressed).

4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to one of the following values:

1

User not authorized for the request.

18

Print request not found on the spool.

19

No output found for the print request.

22

Status of the print request was BUILD-PRT or DIRECT-ERR.

8

An error occurred. &SYSMSG is set to an error message.

Examples: OPT=BROWSE

```
&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=BROWSE REQ=765
```

```
&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=BROWSE BROWSE=REQUEST REQ=9812
```

$PSCALL OPT=CANCEL

Cancels an open print request.

```
&CONTROL NOSHRVARS
-EXEC $PSCALL     OPT=CANCEL
                  REQ=n
```

Cancels an open print request. The request must have been opened by the same NCL process. The print request is closed and deleted from the spool.

Operands:

OPT=CANCEL

Specifies that an open print request is to be canceled.

REQ=*n*

Specifies the number of the print request to be canceled.

Return Codes:

0

$PSCALL completed successfully

4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to the following value:

17

Print request not open

8

An error occurred. &SYSMSG is set to an error message.

Examples: OPT=CANCEL

```
& CONTROL NOSHRVARS
-EXEC $PSCALL OPT=CANCEL REQ=898
```

$PSCALL OPT=CLOSE

Closes an open print request.

```
&CONTROL NOSHRVARS
-EXEC $PSCALL    OPT=CLOSE
                 REQ=n
```

Closes an open print request. The request must have been opened by the same NCL process. If the request was opened with the HOLD operand set to NO, the request is queued for printing, otherwise the status is set to HELD and the request is not printed until released.

Operands:

OPT=CLOSE

Specifies that an open print request is to be closed.

REQ=*n*

Specifies the number of the print request to be closed.

Return Codes:

0

$PSCALL completed successfully

4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to the following value:

17

Print request not open

8

An error occurred. &SYSMSG is set to an error message.

Example: OPT=CLOSE

```
& CONTROL NOSHRVARS
-EXEC $PSCALL OPT=CLOSE REQ=1256
```

$PSCALL OPT=CONFIRM

Presents the Confirm Printer panel and returns the entered details to the caller.

```
&CONTROL SHRVARS=($PS)
-EXEC $PSCALL      OPT=CONFIRM
                 [ USERID=userid ]
```

Confirms the printer name, number of copies, hold and keep settings required to satisfy a print request. The Confirm Printer panel is presented with the fields set to the values last used by the user.

You can modify any of the fields. The Printer Name field supports prompting. When you confirm the details by pressing Enter, the information is returned to the caller.

Operands:

OPT=CONFIRM

Specifies that the Confirm Printer panel is to be presented.

USERID=*userid*

Specifies the user ID of the user whose last used printing details are to be used to prime the fields on the Confirm Printer panel. The default is the user ID set in the system variable &USERID.

Return Codes:

0

$PSCALL completed successfully. The following variables are set:

&$PSPRTNAME

Printer name

&$PSCOPIES

Number of copies

&$PSHOLD

Hold setting, YES or NO

&$PSKEEP

Keep setting, YES or NO

4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an information message and &$PSFDBK is set to the following value:

23

CANCEL command was executed

8

An error occurred. &SYSMSG is set to an error message.

Examples: OPT=CONFIRM

```
& CONTROL SHRVARS =($PS)
-EXEC $PSCALL OPT=CONFIRM

&CONTROL SHRVARS =($PS)
-EXEC $PSCALL OPT=CONFIRM USERID=USER01
```

$PSCALL OPT=DELETE

Deletes a print request from the spool.

```
&CONTROL NOSHRVARS
-EXEC $PSCALL    OPT=DELETE
                 REQ=n
```

Deletes a print request from the spool. The request to be deleted must have a status of HELD, HELD-ERROR, DIRECT-ERR or QUEUED.

Operands:

OPT=DELETE

Specifies that a print request is to be deleted from the spool.

REQ=*n*

Specifies the number of the print request to be deleted.

Return Codes:

0

$PSCALL completed successfully.

4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to one of the following values:

1

User not authorized for the request.

18

Print request not found on the spool.

21

The print request is locked by the system or another user.

22

Status of the print request was not HELD, HELD-ERROR, DIRECT-ERR or QUEUED.

8

An error occurred. &SYSMSG is set to an error message.

Example: OPT=DELETE

```
& CONTROL NOSHRVARS
-EXEC $PSCALL OPT=DELETE REQ=915
```

$PSCALL OPT=HEADER

Adds header lines to an open print request.

```
&CONTROL SHRVARS=($PS)
-EXEC $PSCALL    OPT=HEADER
                 REQ=n
               [ SKIP={ 0 | 1 | 2 | 3 }  ]
               [ USCORE={ 1 | 2 } ]
               [ ALIGN={ LEFT | RIGHT | CENTER }  ]
               [ BOLD={ YES | NO } ]
               [ TOTAL=n ]
```

Prints a heading at the top of each page of output for a print request. The lines of text that are to be printed as the heading on each page need only be specified once, rather than each time a new page is requested. The header lines will be printed at the top of each new page.

Operands:

OPT=HEADER

Specifies that header lines are to be added to an open print request.

REQ=*n*

Specifies the number of the print request to which the header lines are to be added.

SKIP={ 0 | 1 | 2 | 3 }

Specifies the number of lines to be advanced before printing each header line. The default is 1.

USCORE={ 1 | 2 }

Specifies whether the text in each header line is to be underlined:

1

Specifies that text excluding the blanks between words is to be underlined.

2

Specifies that text including the blanks between words is to be underlined.

ALIGN={ LEFT | RIGHT | CENTER }

Specifies whether the text in each header line is to be aligned. The length used to align the text is the value defined in the Columns per Page field (width) for the printer.

BOLD={ YES | NO }

Specifies whether the text in each header line is to be bolded. The default is NO.

TOTAL=*n*

Specifies the number of variables containing header lines that are to be added to the print request. The valid range is 1 to 30. The default is 1.

Variables:

&$PSDATA*n*

Must be set to the header line of header text that is to be printed. n must be in the range 1 to 30. To print the page number in the heading, the variable must contain the characters &$PSP#.

Return Codes:

0

$PSCALL completed successfully.

4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to the following value:

17

The print request is not open.

8

An error occurred. &SYSMSG is set to an error message.

Example: OPT=HEADER

```
&CONTROL SHRVARS=($PS)
&TEMP = &CONCAT & $PSP#
&$PSDATA1 = &ASISTR Page=&TEMP
-EXEC $PSCALL OPT=HEADER REQ=342 SKIP=0 BOLD=NO TOTAL=1
&$PSDATA1 = &STR Title Line 1
&$PSDATA2 = &STR Title Line 2
-EXEC $PSCALL OPT=HEADER REQ=342 SKIP=1 BOLD=YES TOTAL=2
```

Notes:

When OPT=HEADER is specified, the header data passed is appended to the current header. If the previous call was not an OPT=HEADER, this will reset the header. This option allows a print request to contain numerous headers. The maximum number of text lines per header is 30.

A header is only physically printed when a form feed is done as the result of SKIP=NEWPAGE being specified on an OPT=PUT call, or when the lines per page defined for the printer is reached.

$PSCALL OPT=HOLD

Holds a print request.

```
&CONTROL NOSHRVARS
-EXEC $PSCALL    OPT=HOLD
                 REQ=n
```

Holds a print request so that it is not printed until released. The request to be held must have a status of BUILD or QUEUED.

Operands:

OPT=HOLD

Specifies that a print request is to be held.

REQ=*n*

Specifies the number of the print request to be held.

Return Codes:

0

$PSCALL completed successfully.

4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to one of the following values:

1

User not authorized for the request.

8

Print request not found on the spool.

21

The print request is locked by the system or another user.

22

Status of the print request was not BUILD or QUEUED.

8

An error occurred. &SYSMSG is set to an error message.

Example: OPT=HOLD

```
&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=HOLD REQ=2345
```

$PSCALL OPT=INFO

Returns printer, form or setup information. Optionally, presents a list of printers, forms, or setups from which a selection is made.

```
&CONTROL SHRVARS=($PS)
-EXEC $PSCALL   OPT=INFO
                INFO=PRINTER
                PRINTER={ printer name | prefix? | ? }
              [ TYPE={ JES | VTAM | EXIT | ALIAS } ]

-EXEC $PSCALL   OPT=INFO
                INFO=FORM
                FORM={ form name | prefix? | ? }

-EXEC $PSCALL   OPT=INFO
                INFO=SETUP
                SETUP={ setup name | prefix? | ? }
-EXEC $PSCALL   OPT=INFO
                INFO=USER
              [ USERID=userid ]
```

Validates a printer, form or setup name entered on a panel, or provides a list of valid values for a printer, form or setup field, from which a selection is made. This option also enables you to determine the name of a user's default printer. The information returned regarding the printer, form, or setup, could be used to determine how to format the output for a print request.

Operands:

OPT=INFO

Specifies that definition information is to be returned.

INFO={ PRINTER | FORM | SETUP | USER }

Specifies the type of information to be returned.

PRINTER

Returns information from a printer definition.

FORM

Returns information from a form definition.

SETUP

Returns information from a setup definition.

USER

Returns the name of the user's default printer.

PRINTER={ *printer name* | *prefix*? | ? }

Specifies the name of the printer. If a prefix followed by a question mark is specified, a Printer List is presented from which a selection is made. The list will contain all printers with names starting with the specified prefix. If a question mark is specified, a Printer List containing all the defined printers is presented.

TYPE={ JES | VTAM | EXIT | ALIAS }

Specifies the type of printers to be displayed in the Printer List. This operand is ignored if the PRINTER operand is not set to a question mark, or a prefix followed by a question mark.

FORM={ *form name* | *prefix*?| ? }

Specifies the name of the form. If a prefix followed by a question mark is specified, a Form List is presented from which a selection is made. This list will contain all forms with names starting with the specified prefix. If a question mark is specified, a Form List containing all the defined forms is presented.

SETUP={ *setup name* | *prefix*? | ? }

Specifies the name of the setup. If a prefix followed by a question mark is specified, a Setup List is presented from which a selection is made. This list will contain all setups with names starting with the specified prefix. If a question mark is specified, a Setup List containing all the defined setups is presented.

USERID=*userid*

Specifies the user ID for which the default printer is to be returned. If the INFO operand is set to USER and USERID is not specified, the default printer for the user ID set in the system variable &USERID, is returned.

Return Codes:

&RETCODE = 0

$PSCALL completed successfully. The variables returned are as follows:

&$PSPRTNAME

Printer name.

&$PSPRTRNAME

Name of the printer for which the printer is an alias, if the printer type is ALIAS.

&$PSPRTTYPE

Printer type: JES, VTAM, EXIT or ALIAS.

&$PSPRTRTYPE

Printer type of the printer for which the printer is an alias, if the printer type is ALIAS.

&$PSPRTCASE

Lower case flag: YES or NO.

&$PSPRTDESC

Description of the printer.

&$PSPRTLLIMIT

Line limit.

&$PSPRTDEST

JES remote destination name if the printer type is JES.

&$PSPRTCLASS

JES output class if the printer type is JES.

&$PSPRTLUNAME

VTAM defined network name for the printer if the printer type is VTAM.

&$PSPRTEXIT

Name of a printer exit if the printer type is EXIT.

&$PSPRTEXDATA

Printer exit data if the printer type is EXIT.

&$PSPRTLOGMOD

Name of an entry in the logmode table if the printer type is VTAM.

&$PSFORMNAME

Form name.

&$PSFORMDESC

Description of the form.

&$PSFORMLINES

Maximum number of lines to be printed per page.

&$PSFORMCOLS

Maximum number of columns to be printed per page.

&$PSFORMBANR

The banner page flag (*DEFAULT or *NONE), or the name of a banner exit NCL procedure.

&$PSSETUPNAME

Setup name.

&$PSSETUPDESC

Description of the setup.

&RETCODE = 4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to one of the following values:

2

Printer not defined

4

No printers defined with the specified prefix and/or type

5

Printer not selected from Printer List

6

Form not defined

8

No forms defined with the specified prefix

9

Form not selected from Form List

10

Setup not defined

12

No setups defined with the specified prefix

13

Setup not selected from Setup List

14

Default printer not defined for specified user ID

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Examples: OPT=INFO

```
&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=INFO INFO=PRINTER PRINTER=?

&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=INFO INFO=PRINTER PRINTER=FLOOR5

&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=INFO INFO=PRINTER TYPE=JES PRINTER=?

&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=INFO INFO=FORM FORM=AREA1?

&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=INFO INFO=SETUP SETUP=TPAPER

&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=INFO INFO=USER USERID=USER01

&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=INFO INFO=USER
```

Notes:

When the INFO operand is set to USER or PRINTER, all of the variables returned for return code 0 are set. When INFO is set to FORM, only the variables starting with the characters $PSFORM and $PSSETUP are set and the others are null. When INFO is set to SETUP, only the variables starting with the characters $PSSETUP are set and the others are null.

When variable &$PSPRTTYPE is set to JES, EXIT or VTAM, variables &$PSPRTRNAME and &$PSPRTRTYPE are null.

$PSCALL OPT=MODIFY

Modifies a print request that is on the spool.

```
&CONTROL NOSHRVARS
-EXEC $PSCALL   OPT=MODIFY
                REQ=n
```

Presents the print request panel allowing the request to be modified. You can modify the destination, number of copies, keep flag and priority, if authorized. You cannot modify a request with the status of WAIT, PRINTING, BUILD-PRT, or DIRECT-ERR.

When the print request panel is terminated by the user entering the FILE or CANCEL command (or pressing the function key to which those commands are assigned), control is returned to the NCL procedure that executed $PSCALL.

Operands:

OPT=MODIFY

Specifies that a print request is to be modified.

REQ=*n*

Specifies the number of the print request to be modified.

Return Codes:

&RETCODE = 0

$PSCALL completed successfully.

&RETCODE = 4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to one of the following:

1

User not authorized for the request

18

Print request not found on the spool

21

Print request locked by the system or another user

22

Status of the print-request was PRINTING, BUILD-PRT or DIRECT-ERR

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Example: OPT=MODIFY

```
&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=MODIFY REQ=23
```

$PSCALL OPT=OPEN

Opens a print request.

```
&CONTROL SHRVARS=($PS)
```

To open a print request for a printer that is defined to PSM:

```
-EXEC $PSCALL   OPT=OPEN
                DEST=PSM
              [ PRINTER=printer name ]
              [ USERID=userid ]
              [ JESCLASS=class ]
              [ VTAMLOG=logmode ]
              [ FORM=form name ]
              [ SETUP=setup name ]
              [ LOWCASE={ YES | NO } ]
              [ LINES=n ]
              [ COLS=n ]
              [ BANNER={ *DEFAULT | *NONE | proc name } ]
              [ JESFORM=JES Form name ]
              [ JESFCB=JES FCB name ]
              [ JESUCS=JES UCS code ]
              [ JESPGM=JES PGM name ]
              [ EXITDATA=c ]
              [ COPIES=n ]
              [ PRTY=n ]
              [ HOLD={ YES | NO } ]
              [ KEEP={ YES | NO } ]
```

To open a print request for a printer that is not defined to PSM and is a JES printer:

```
-EXEC $PSCALL   OPT=OPEN
                DEST=JES
              [ JESDEST=destid.userid ]
              [ JESCLASS=class ]
              [ FORM=form name ]
              [ SETUP=setup name ]
              [ LOWCASE={YES | NO } ]
              [ LINES=n ]
              [ COLS=n ]
              [ BANNER={ *DEFAULT | *NONE | proc name } ]
              [ JESFORM=JES Form name ]
              [ JESFCB=JES FCB name ]
              [ JESUCS=JES UCS code ]
              [ JESPGM=JES PGM name ]
              [ COPIES=n ]
              [ PRTY=n ]
              [ HOLD={YES | NO} ]
              [ KEEP={YES | NO} ]
```

To open a print request for a printer that is not defined to PSM and is a VTAM printer:

```
-EXEC $PSCALL   OPT=OPEN
                DEST=VTAM
                VTAMLU=luname
              [ VTAMLOG=logmode ]
              [ FORM=form name ]
              [ SETUP=setup name ]
              [ LOWCASE={ YES | NO } ]
              [ LINES=n ]
              [ COLS=n ]
              [ BANNER={ *DEFAULT | *NONE | proc name } ]
              [ COPIES=n ]
              [ PRTY=n ]
              [ HOLD={YES | NO} ]
              [ KEEP={YES | NO} ]
```

Allocates a print request number and opens it. You can then add output to the request using the PUT option.

Operands:

OPT=OPEN

Specifies that a print request is to be opened.

DEST={ PSM | JES | VTAM }

Specifies the type of output destination.

PSM

Indicates the printer is defined to PSM.

JES

Indicates the printer is not defined to PSM and is a JES printer.

VTAM

Indicates the printer is not defined to PSM and is a VTAM printer.

PRINTER=*printer name*

Specifies the name of the printer on which the request is to be printed.

USERID=*userid*

Specifies the user ID that is to be the owner of the print request. If not specified, the user ID set in the &USERID system variable will be the owner of the request. If the DEST operand is set to PSM and the PRINTER operand is not specified, this operand specifies the user ID whose default printer is the printer on which the request is to be printed. If a user ID is not specified, the default printer for the user ID set in the &USERID system variable is used.

JESDEST=*destid.userid*

Specifies the JES2 or JES3 remote destination of the printer on which the request is to be printed. *destid* is the remote destination name and *userid* is the remote user ID. *userid* is optional but, if specified, *destid* must also be specified.

JESCLASS=*class*

Specifies the JES2 or JES3 output class. If the DEST operand is set to JES and an output class is not specified, the output class defined in the defaults definition is used. If DEST is set to PSM, this class, if specified, is used instead of that defined in the printer definition.

VTAMLU=*luname*

Specifies the VTAM defined network name of the printer on which the request is to be printed.

VTAMLOG=*logmode*

Specifies the name of an entry in the LU's logmode table which is to be used for the session. If the DEST operand is set to VTAM and a logmode is not specified, the LU's default logmode entry is used. If DEST is set to PSM, this logmode, if specified, is used instead of that defined in the printer definition.

FORM=*form name*

Specifies the name of the form definition to be used. If the DEST operand is set to JES or VTAM, the default is that defined in the defaults definition. If DEST is set to PSM, this form name, if specified, is used instead of that defined in the printer definition.

SETUP=*setup name*

Specifies the name of the setup definition to be used. This setup name is used instead of that defined in the form definition.

LOWCASE={ YES | NO }

Specifies whether the printer supports lower case characters. If the DEST operand is set to PSM, the value of this operand is used instead of that defined in the printer definition. If DEST is set to JES or VTAM, the default is that defined in the defaults definition.

LINES=*n*

Specifies the maximum number of lines to be printed per page. The range is 0 to 999. If there is no limit to the number of lines per page (that is, the paper is continuous), 0 must be specified. If the DEST operand is set to PSM, the value of this operand is used instead of that defined in the form definition. If the DEST operand is set to JES or VTAM, the default is that defined in the defaults definition.

COLS=*n*

Specifies the maximum number of columns to be printed per page. The range is 1 to 256. If the DEST operand is set to PSM, the value of this operand is used instead of that defined in the form definition. If the DEST operand is set to JES or VTAM, the default is that defined in the defaults definition.

BANNER={ *DEFAULT | *NONE | *proc name* }

Specify *DEFAULT if the default banner page is to be printed, *NONE if no banner page is to be printed, or the name of an NCL procedure which is to be executed as a banner exit. If the DEST operand is set to PSM, the value of this operand is used instead of that defined in the form definition. If the DEST operand is set to JES or VTAM, the default is that defined in the defaults definition. The banner exit is described in the following pages of this chapter.

JESFORM=JES *form name*

Specifies the name of a JES form that is to be used when processing the SYSOUT data set. It the DEST operand is set to PSM, this JES form name is used instead of that defined in the form definition. This operand is ignored if the printer specified is not a JES printer.

JESFCB=JES FCB *name*

Specifies the name of a JES FCB (forms control buffer) that is to be used when processing the SYSOUT data set. If the DEST operand is set to PSM, this JES FCB name is used instead of that defined in the form definition. This operand is ignored if the printer specified is not a JES printer.

JESUCS=JES UCS *code*

Specifies the JES UCS (universal character set) code to be associated with the SYSOUT data set. If the DEST operand is set to PSM, this JES UCS code is used instead of that defined in the form definition. This operand is ignored if the printer specified is not a JES printer.

JESPGM=JES PGM *name*

Specifies the JES PGM (program) name to be associated with the SYSOUT data set. If the DEST operand is set to PSM, this JES PGM name is used instead of that defined in the form definition. This operand is ignored if the printer specified is not a JES printer.

EXITDATA=*c*

Specifies exit data that is to be passed to the printer exit NCL procedure instead of that defined in the printer definition. Exit data is from 1 to 120 characters long and must be quoted if it contains imbedded blanks. This operand is ignored if the DEST operand is set to JES or VTAM.

COPIES=*n*

Specifies the number of copies to be printed. The range is 1 to 255; the default is 1.

PRTY=*n*

Specifies a priority for the print request. The range is 1 to 99. The highest priority is 1 and the lowest is 99; the default is 50. Print requests for a printer will be printed in priority order starting with priority 1.

HOLD={ YES | NO }

Specifies whether the print request is to be assigned a status of HELD when closed (that is, will not be printed until released). The default is NO.

KEEP={ YES | NO }

Specifies whether the print request is to be kept after being printed. The default is NO.

Return Codes:

&RETCODE = 0

$PSCALL completed successfully. &$PSREQ# is set to the 4-digit request number allocated by the system.

&RETCODE = 4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to one of the following values:

2

Printer not defined

6

Form not defined

10

Setup not defined

14

Default printer not defined for specified user ID

20

JES printing not supported on operating system

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Example: OPT=OPEN

```
&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=OPEN DEST=PSM PRINTER=FLOOR5

&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=OPEN DEST=JES JESDEST=RMT15 JESCLASS=F +
       LOWCASE=NO LINES=60 COLS=80

&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=OPEN DEST=PSM USERID=USER01 COPIES=2 +
       PRTY=3

&CONTROL SHRVARS=($PS)
-EXEC $PSCALL OPT=OPEN DEST=VTAM VTAMLU=PLUA01 HOLD=YES
```

$PSCALL OPT=PUT

Adds output (data that is to be printed) to a print request.

```
&CONTROL SHRVARS=($PS)
-EXEC $PSCALL    OPT=PUT
                 REQ=n
               [ SKIP={ 0 | 1 | 2 | 3 | NEWPAGE } ]
               [ USCORE={ 1 | 2 }]
               [ ALIGN={ LEFT | RIGHT | CENTER } ]
               [ BOLD={ YES | NO } ]
               [ TOTAL=n ]
```

Adds output to an open print request. The output is added to the spool for the specified print request number.

Operands:

OPT=PUT

Specifies that lines of output are to be added to a print request.

REQ=*n*

Specifies the number of the print request to which the output is to be added.

SKIP={ 0 | 1 | 2 | 3 | NEWPAGE }

Specifies the number of lines to be advanced before printing each line of output. NEWPAGE specifies advance to a new page before printing each line of output. The default is 1.

USCORE={ 1 | 2 }

Specifies whether the text in each line of output is to be underlined:

1

Specifies that text excluding the blanks between words is underlined.

2

Specifies that text including the blanks between words is underlined.

{ LEFT | RIGHT | CENTER }

Specifies whether the text in each line of output is to be aligned. The length used to align the text is the value defined in the Columns per Page field (width) for the printer.

BOLD={ YES | NO }

Specifies whether the text in each line of output is to be bolded. The default is NO.

TOTAL=*n*

Specifies the number of variables containing lines of output that are to be printed. Range is 1 to 99999. The default is 1.

Variables:

&$PSDATA*n*

Must be set to the line of output to be printed. n must be in the range 1 to 99999. If the length of this variable is greater than the value defined in the Columns per Page field (width) for the printer, the line of output is truncated.

Return Codes:

&RETCODE = 0

$PSCALL completed successfully.

&RETCODE = 4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to the following value:

17 *Print request not open*

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Examples: OPT=PUT

```
&CONTROL SHRVARS=($PS)
&$PSDATA1 = &STR This is a line of print data
&$PSDATA2 = &STR This is another line of data
-EXEC $PSCALL OPT=PUT REQ=5 TOTAL=2

&CONTROL SHRVARS=($PS)
&$PSDATA1 = &STR This is the report heading
-EXEC $PSCALL OPT=PUT REQ=150 SKIP=NEWPAGE ALIGN=CENTER +
         BOLD=YES

&CONTROL SHRVARS=($PS)
&$PSDATA1 = &STR Date:&DATE4
-EXEC $PSCALL OPT=PUT REQ=5678 SKIP=2 ALIGN=RIGHT

&CONTROL SHRVARS=($PS)
&$PSDATA1 = &ASISTR  Name Description
-EXEC $PSCALL OPT=PUT REQ=2323 SKIP=3 USCORE=1
```

$PSCALL OPT=QUEUE

Presents the Output Queue.

```
&CONTROL NOSHRVARS
-EXEC $PSCALL    OPT=QUEUE
                 [ USERID={ userid | prefix* } ]
                 [ PRINTER={ printer name | prefix* } ]
```

Presents the Output Queue showing print requests that are currently on the spool. When the Output Queue is terminated by the user entering the EXIT or RETURN command (or pressing the Function key to which those commands are assigned), control is returned to the NCL procedure that executed $PSCALL.

Operands:

OPT=QUEUE

Specifies the Output Queue is to be presented.

USERID={ *userid* | *prefix }**

Specifies the user ID of the user whose print requests are to be displayed on the Output Queue. For the Output Queue to be displayed with print requests for users whose user ID starts with a particular prefix, specify the prefix followed by an asterisk (*).

PRINTER={ *printer name* | *prefix }**

Specifies the PSM printer name, JES remote destination, or VTAM defined network name (LU name), at which print requests to be displayed on the Output Queue are to be printed. For the Output Queue to be displayed with print requests for printers which the name, JES remote destination, or VTAM defined network name, starts with a particular prefix, specify the prefix followed by an asterisk (*).

Return Codes:

&RETCODE = 0

$PSCALL completed successfully. &$PSFDBK may be set to the following value:

RETURN command entered (or the Function key to which it is assigned was pressed)

&RETCODE = 4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to the following value:

User not authorized for the request

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Examples: OPT=QUEUE

```
&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=QUEUE

&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=QUEUE USERID=ZXPSD

&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=QUEUE USERID=ZXP*

&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=QUEUE PRINTER=FLOOR5

&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=QUEUE USERID=ZXPSD PRINTER=FLOOR5
```

$PSCALL OPT=RELEASE

Releases a held print request.

```
&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=RELEASE
              REQ=n
```

To release a print request for printing that is held on the spool. The request to be released must have a status of BUILD-HELD, HELD, or HELD-ERROR.

Operands:

OPT=RELEASE

Specifies that a print request is to be released for printing.

REQ=*n*

Specifies the number of the print request to be released.

Return Codes:

&RETCODE = 0

$PSCALL completed successfully.

&RETCODE = 4

$PSCALL completed successfully. The request was denied. &SYSMSG is set to an error message and &$PSFDBK is set to one of the following values:

1

User not authorized for the request

18

Print request not found on the spool

21

Print request locked by the system or another user

22

Status of the print request was not BUILD-HELD, HELD or HELD-ERROR

&RETCODE = 8

An error occurred. &SYSMSG is set to an error message.

Example: OPT=RELEASE

```
&CONTROL NOSHRVARS
-EXEC $PSCALL OPT=RELEASE REQ=4555
```

Banner Exit

A banner exit is an installation-written NCL procedure. The banner exit allows you to tailor the banner page printed at the front of the output for a print request. The banner exit is executed by the system before printing the output for a request.

It is passed information regarding the print request, and can pass back to the system the information that is to be printed on the banner page.

The variables passed to the banner exit by the system are as follows:

&$PSUSERID

The user ID of the user who generated the print request.

&$PSREQ#

The number of the print request.

&$PSREQDATE

The date the print request was generated in the format DD-MMM-YYYY.

&$PSREQTIME

The time the print request was generated in the format HH.MM.SS.

The variables the banner exit can set are as follows:

&$PSBANCNT

Must be set to the number of &$PSDATA*n* variables containing text that is to be printed on the banner page. The range is 1 to 999.

&$PSDATA*n*

May be set to the line of text to be printed on the banner page. n must be in the range 1 to 999.

&$PSSKIP*n*

Indicates the number of lines to be advanced before printing the line of text specified in the corresponding &$PSDATA*n* variable. Valid values are 0, 1, 2, 3. The default is 1. n must be in the range 1 to 999.

&$PSUSCOR*n*

Indicates whether the text specified in the corresponding &$PSDATA*n* variable is to be underlined. n must be in the range 1 to 999. Valid values are 1 and 2, where 1 specifies text excluding blanks between words is underlined, and 2 specifies the text including blanks between words is underlined.

&$PSALIGN*n*

Indicates where text specified in the corresponding &$PSDATA*n* variable is to be aligned. The variable n must be in the range 1 to 999. Valid values for this variable are LEFT, RIGHT, and CENTER. The line length used to align text is the value defined in the Columns per Page field (width), for the printer.

&$PSBOLD*n*

Indicates whether text specified in the corresponding &$PSDATA*n* variable is to be bolded. The variable n must be in the range 1 to 999. Valid values for this variable are NO and YES. The default is NO.

Printer Exit Interface

A printer exit is an installation-written NCL procedure. The printer exit enables the implementation of output destinations other than JES or VTAM, for example, a mailing system or a file.

When a print request that has a destination of a printer exit is closed or released, the printer exit is executed by the system to perform initialization processing, then once for each line of data and finally to perform termination processing.

The variables passed to the printer exit by the system are as follows:

&$PSACTION

This variable is set to indicate the processing that is to be performed. This variable is set to one of the following values:

INIT

Initialization processing is to be performed

DATA

Data line processing is to be performed

TERM

Termination processing is to be performed

&$PSTERM

This variable is set to indicate the type of termination when the &$PSACTION variable is set to TERM. This variable is set to one of the following values:

NORMAL

Normal termination

EXIT

Terminated by printer exit setting return code 8

CANCEL

Print request canceled by the procedure sending the output

&$PSEXITDATA

This variable is set to the exit data defined in the printer definition. Exit data in the printer definition is overridden on the OPEN call to $PSCALL.

&$PSUSERID

This variable is set to the user ID of the user who generated the print request.

&$PSREQ#

This variable is set to the number of the print request.

&$PSREQDATE

This variable is set to the date the print request was generated in the format DD-MMM-YYYY.

&$PSREQTIME

This variable is set to the time the print request was generated in the format HH.MM.SS.

&$PSDATA

This variable is set to the line of data that is to be processed.

&$PSSKIP

This variable is set to the skip amount specified for the line of data. Valid values are NEWPAGE, 0, 1, 2, and 3.

&$PSUSCORE

This variable is set to the underline setting specified for the line of data. Valid values are 1 and 2.

&$PSBOLD

This variable is set to the bold setting specified for the line of data. Valid values are YES and NO.

The variables that is set by the printer exit are as follows:

&$PSUSRD*c*

These variables is set to user data, where c is 1 to 5 alphanumeric and/or national characters. These variables are never set or cleared by the system, therefore must be completely managed by your installation.

The system variable &RETCODE is set by the printer exit as follows:

0

Continue processing.

8

An error occurred. Terminate processing and set the status of the print request on the spool to HELD-ERROR or DIRECT-ERR. &SYSMSG is set to an error message that is to be written to the activity log and stored in the error message field in the printer request.

Chapter 5: CA CCI Interface

This section describes how you can use NCL to connect to the CA Common Communications Interface (CA CCI). CA CCI is a CA Technologies program-to-program communication protocol. You can use CA CCI to communicate with other CA Technologies products that use CA CCI.

Note: For more information about CA CCI, see the CA Common Services documentation.

This section contains the following topics:

$CACCI OPT=INIT

Requests communication with CA CCI. After this call has successfully executed, &$CACCIID is set with a value to be used on subsequent calls to identify this CA CCI registration.

```
&CALL PROC=$CACCI PARMS=(OPT=INIT,
                         NAME=name)
```

Operands:

NAME=*name*

Defines the caller. This operand must be a unique name within a system for RECEIVE and within a product region for other calls, e.g. SEND or INQUIRE. The name is not visible on the CA SYSVIEW® Performance Management CCI Receivers panel until a RECEIVE is issued.

Limits: 1 to 20 mixed case characters, numbers, spaces, and special characters. Special characters require quotes.

Note: When the NCL process ends, registration with CCI for this name is automatically terminated if it is still active. The &$CACCIID value is only usable within the process, including procedures invoked by &CALL. It cannot be used by another process.

$CACCI OPT=INQUIRE

Requests that CA CCI provide a list of eligible receivers in an MDO mapped by $NMMPCCI.

```
&CALL PROC=$CACCI PARMS=(OPT=INQUIRE,
                         ID=cci_id,
                         NAME=name,
                         [SYSID=sysid,]
                         MDO=mdoname,
                         SHARE=(mdoname>))
```

Operands:

ID=*cci_id*

Specifies the identifier returned in &$CACCIID from an earlier CCI INIT call.

NAME=*name*

Specifies a mask that only returns receivers with names that match this value.

Limits: 1 to 20 mixed case characters, including numbers, spaces, special characters, and asterisks to allow for pattern matching. Special characters require quotes.

SYSID=*sysid*

Specifies the ID of the system from which to retrieve the receiver. The ID can be a mask that only returns receivers with IDs that match this value. If omitted, only receiver IDs on the local system are selected.

Limits: 1 to 8 characters, including asterisks to allow for pattern matching (see $CACCI Example)

MDO=*mdoname*

Specifies the name of the MDO to return the results, and is mapped by $NMMPCCI.

SHARE=(*mdoname*>)

Specifies MDO to be shared with the called procedure. The data in the MDO mdoname can be seen by the calling procedure.

Notes:

The map for the returned MDO is as follows:

```
--<LK0(2,1) TAGS(EXPLICIT)>--
$NMMPCCI DEFINITIONS ::= BEGIN
Inquire ::= SEQUENCE
{
  numReceivers    <<1>>  INTEGER,
  cciReceivers    <<2>>  SEQUENCE OF <<3>> Receiver OPTIONAL
}
Receiver ::= SEQUENCE
{
  receiverID <<4>> ID,
  remote <<5>> BOOLEAN,
  active <<6>> BOOLEAN,
  messagesQueued <<7>> INTEGER
}
ID ::= SEQUENCE
{
  name <<8>> GraphicString (SIZE(1..20)),
  sysID <<9>> GraphicString (SIZE(1..8)) OPTIONAL
}
END
```

$CACCI OPT=RECEIVE

Requests that CA CCI receive data. Data arrives attached to a NOTIFY event until either a CANCEL or TERM is issued, or the NCL procedure terminates. The data is sent to the process-dependent request queue and read by the &INTREAD verb.

```
&CALL PROC=$CACCI PARMS=(OPT=RECEIVE,
                          ID=cci_id,
                        [NAME=name,]
                        [SYSID=sysid])
```

Operands:

ID=*cci_id*

Specifies the identifier returned in $CACCIID from an earlier CCI INIT call.

NAME=*name*

Identifies the name of an eligible sender. If omitted, data from any sender is eligible for this request. The name is visible on the CA SYSVIEW Performance Management CCI Receivers panel in the SenderId column.

Limits: 1 to 20 mixed case characters, numbers, spaces, and special characters. Special characters require quotes.

SYSID=*sysid*

Specifies the system ID of the sender from which to receive data. The SYSID is visible on the CA SYSVIEW Performance Management CCI Receivers panel in the SSysId column.

Notes:

Use &INTREAD TYPE=REQ or TYPE=ANY to read the data. If a timeout occurs on &INTREAD, the RECEIVE is not canceled and a subsequent &INTREAD can still read the data.

Note: See the Notes in the description for &INTREAD for details of the mapped data object $INT.

The notify message for the RECEIVE is located in $INT.TEXT and the data is in $INT.USERMDO. The notify message format is as follows:

```
N00101 NOTIFY: CCI EVENT: RECEIVE RESOURCE: ID=cci_id
LCLNAME=local_name RSYSID=remote_sysid RMTNAME=remote_name
RC=return_code
```

If the return code is not zero, RECEIVE is terminated and must be reissued. The following fields are appended to the message:

- ERR=error code
- ERRX=extended error code
- DESC=error description

Always check the message text before reading the data because the INTQUE OCS command can also be used to satisfy the &INTREAD.

$CACCI OPT=SEND

Requests that CA CCI send data to a registered receiver.

```
&CALL PROC=$CACCI PARMS=(OPT=SEND,
                         ID=cci_id,
                         NAME=name,
                        [SYSID=sysid,]
                        [WAIT={YES | TARGET},]
                         DATA=data)
```

Operands:

ID=*cci_id*

Specifies the identifier returned in $CACCIID from an earlier CCI INIT call.

NAME=*name*

Specifies a receiver to send to.

Limits: 1 to 20 mixed case characters, numbers, spaces and special characters. Special characters require quotes.

SYSID=*sysid*

Specifies the system ID of a receiver to send to. This operand is not required if the receiver is active on the same system.

WAIT={YES | TARGET}

Specifies when the request is complete:

- YES means the SEND completes when the data is outbound from the local system.
- TARGET means that the SEND completes when the targeted receiver has received the data. TARGET is the same as YES for a local receiver.

DATA=*data*

Specifies the data to send.

Limits: 1 to 256 bytes

Note: If the data contains spaces or special characters, you can use the &ZQUOTE built-in function.

Examples:

```
&TEXT = &ZQUOTE This is a test
&CALL PROC=$CACCI PARMS=(OPT=SEND,+
ID=&MYTEST,+
NAME=FRED,+
SYSID=A11SENF,+
WAIT=TARGET,+
DATA=&TEXT)
```

$CACCI OPT={TERM | TERMINATE}

Requests that CA CCI terminate a registered receiver. When you request TERMINATE, you cannot use the $CACCIID value again, any RECEIVE open is terminated. You cannot RECEIVE, INQUIRE, or SEND until you issue an INIT call again.

```
&CALL PROC=$CACCI PARMS=(OPT={TERM | TERMINATE},
                         ID=cci_id)
```

Operands:

ID=*cci_id*

Specifies the identifier returned in $CACCIID from an earlier CCI INIT call.

$CACCI OPT=CANCEL

Requests that CA CCI cancel a previously registered receiver ID. When you CANCEL, you are only canceling a particular RECEIVE, but you can still issue INQUIRE, SEND or RECEIVE again.

```
&CALL PROC=$CACCI PARMS=(OPT=CANCEL,
                         ID=cci_id,
                         [NAME=name,]
                         [SYSID=sysid])
```

Operands:

ID=*cci_id*

Specifies the identifier returned in $CACCIID from an earlier CCI INIT call.

NAME=*name*

Identifies the sender's name, if one was specified on the RECEIVE. If omitted, any outstanding receive-any statement (RECEIVE with no NAME=) is canceled.

Limits: 1 to 20 mixed case characters, numbers, spaces, and special characters. Special characters require quotes.

SYSID=*sysid*

Specifies the system ID of the sender, if one was specified on the RECEIVE call.

Note: The optional NAME and SYSID operands should match those on the RECEIVE being canceled. If you issue RECEIVE with no NAME or SYSID specified (that is, RECEIVE any), then the CANCEL should be the same. If you requested to RECEIVE only from a named sender on a particular system, then the same name and system should be specified on CANCEL.

Return Codes and Variables

The following table shows the meanings of the return codes:

Return Code	Meaning	Variables Returned	Variable Contents
0	Successful	&$CACCID (from INIT call only)	ID to be used in subsequent CANCEL, INQUIRE, RECEIVE, SEND, or TERMINATE calls
4	CA CCI is inactive		
8	CA CCI logic error		
12	Abend has occurred, the system abend code is in the extended error code		
16	CA CCI unrecoverable problem		
20	CA CCI invalid PLIST		
32	CA CCI error		
72	Invalid OPT parameter		

The following table shows the range of error codes to which the returned variables apply and what variables are set, depending on the value of &RETCODE (return code):

Return Code	Meaning	Variables Returned	Variable Contents
0–32	CA CCI return code	&$CACCIFDBK	Feedback code
		&$CAERRORCODE	CCI error code
		&$CAERRCODEX	Extended error code (hexadecimal)
		&$CAERRORTEXT	Error description
64	OML error - note the variables and provide information for CA Technical Support	&$CAERRORCODE	OML error code
		&$CAERRORTEXT	OML error description
		&$CAERRORCOND	OML error condition code
		&$CAERRORKYWD	Keyword causing error
		&$CAERRORVAL	Value causing error
68	Invalid OPT parameter	&$CAERRORTEXT	Error description

Feedback Codes

The feedback codes set in &$CACCIFDBK are as follows:

Code	Option
1	INIT
2	TERM
3	SEND
4	RECEIVE
5	CANCEL
6	INQUIRE

$CACCI Example

The following NCL procedure shows the use of the INIT, INQUIRE, SEND, RECEIVE and TERMINATE functions to create a simple test of CA CCI within one system. The name of the procedure is MYCCIXMP.

```
-* Test $CACCI

&I = &SUBSTR &1 1 1
&IF .&I = .C &THEN +
  &GOSUB .CLIENT
&ELSE +
  &GOSUB .SERVER

&EXIT

-* Server side of CCI connection
.SERVER
-* Register with CCI
&CALL PROC=$CACCI PARMS=(OPT=INIT,NAME=MYSERVER)
&IF &RETCODE = 0 &THEN &DO
  &WRITE Server: INIT RC:+
   &RETCODE FB:&CACCIFDBK EC:&$CAERRORCODE ET:&$CAERRORTEXT
  &RETSUB
&DOEND
&SVID = &$CACCIID

-* Start a client to talk to us
&INTCMD -START MYCCIXMP CLIENT
```

```
-* Handshake with client
&DOUNTIL .&MSG = .READY OR .&MSG = .ABORT
  &INTREAD TYPE=ANY VARS=MSG
  &WRITE Client sent: &MSG
&DOEND

-* Bail out if client had problems
&IF &MSG = ABORT &THEN &DO
  &WRITE Server: Aborting as requested by client
  &RETSUB
&DOEND

-* Ask CCI for a list of all interested receivers
&CALL PROC=$CACCI
PARMS=(OPT=INQUIRE,ID=&SVID,NAME=MY**************,+
  MDO=MYMDO) SHARE=(MYMDO>)
&IF &RETCODE = 0 &THEN &DO
  &WRITE Server: INQUIRE RC:+
   &RETCODE FB:&CACCIFDBK EC:&$CAERRORCODE ET:&$CAERRORTEXT
  &RETSUB
&DOEND
```

```
-* Display receiver information
&ASSIGN VARS=NUMREC FROM MDO=MYMDO.NUMRECEIVERS
&WRITE Server: &NUMREC receivers found:
&WRITE Server: SYSID Name LclRmt State MsgsQd
&I = 1
&DOWHILE &I <= &NUMREC
  &ASSIGN VARS=SID FROM
MDO=MYMDO.CCIRECEIVERS.{&I}.RECEIVERID.SYSID
  &ASSIGN VARS=NAME FROM
MDO=MYMDO.CCIRECEIVERS.{&I}.RECEIVERID.NAME
 &ASSIGN VARS=REMOTE FROM MDO=MYMDO.CCIRECEIVERS.{&I}.REMOTE
 &ASSIGN VARS=ACTIVE FROM MDO=MYMDO.CCIRECEIVERS.{&I}.ACTIVE
  &ASSIGN VARS=QUEUED FROM
MDO=MYMDO.CCIRECEIVERS.{&I}.MESSAGESQUEUED
  &IF &REMOTE = 1 &THEN &REMOTE = REMOTE
  &ELSE &REMOTE = LOCAL*
  &IF &ACTIVE = 1 &THEN &ACTIVE = ACTIVE**
  &ELSE &ACTIVE = INACTIVE
  &WRITE Server: &SID &NAME &REMOTE &ACTIVE &QUEUED
  -* Send 2 messages to the receivers
&CONTROL NOUCASE
&DATA = &ZQUOTE (Hello from the CCI server)
&CONTROL UCASE
&CALL PROC=$CACCI
PARMS=(OPT=SEND,ID=&SVID,NAME=&NAME,SYSID=&SID,+
  DATA=&DATA)
&IF &RETCODE = 0 &THEN &DO
  &WRITE Server: SEND RC:+
  &RETCODE FB:&CACCIFDBK EC:&$CAERRORCODE ET:&$CAERRORTEXT
  &RETSUB
&DOEND
&CALL PROC=$CACCI PARMS=(DATA='Hello()from the CCI server
again',+
  OPT=SEND,ID=&SVID,NAME=&NAME,SYSID=&SID)
   &IF &RETCODE = 0 &THEN &DO
    &WRITE Server: SEND RC:+
     &RETCODE FB:&CACCIFDBK EC:&$CAERRORCODE ET:&$CAERRORTEXT
    &RETSUB
  &DOEND
  &I = &I + 1
&DOEND
&WRITE

-* Display all messages from the client
&DOUNTIL .&MSG = .FIN
  &INTREAD TYPE=ANY STRING=(MSG)
  &WRITE Client: &MSG
&DOEND
```

```
-* Terminate our CCI connection
&CALL PROC=$CACCI PARMS=(OPT=TERMINATE,ID=&SVID)
&IF &RETCODE = 0 &THEN &DO
  &WRITE Server: TERM gave:,
    &RETCODE &CACCIFDBK &$CAERRORCODE &$CAERRORTEXT
  &RETSUB
&DOEND
&RETSUB

-* Client side of CCI connection
.CLIENT

-* Register with CCI
&CONTROL NOUCASE
&CLIENT = &ZQUOTE MY Client .Do_Do
&CONTROL UCASE
&CALL PROC=$CACCI PARMS=(OPT=INIT,NAME=&CLIENT)
&IF &RETCODE = 0 &THEN &DO
  &WRITE Client: INIT RC:+
 &RETCODE FB:&CACCIFDBK EC:&$CAERRORCODE ET:&$CAERRORTEXT
&WRITE ABORT
&RETSUB
&DOEND
&CLID = &$CACCIID
&WRITE &CLID

-* Tell CCI to receive msgs from our server
&CALL PROC=$CACCI PARMS=(OPT=RECEIVE,ID=&CLID,NAME=MYSERVER)
&IF &RETCODE = 0 &THEN &DO
  &WRITE RECEIVE RC:+
   &RETCODE FB:&CACCIFDBK EC:&$CAERRORCODE ET:&$CAERRORTEXT
  &WRITE ABORT
  &RETSUB
&DOEND

-* Tell the server ready for messages
&WRITE READY

-* Receive and display all messages
&DOUNTIL &ZFDBK ¬= 0
    &INTREAD TYPE=REQ STRING=(*) WAIT=3
    &IF &ZFDBK = 0 &THEN &DO
      &ASSIGN VARS=MSG FROM MDO=$INT.USERMDO
      &ASSIGN VARS=NOTIFY FROM MDO=$INT.TEXT
      &WRITE Server sent: &MSG
      &WRITE Notify msg : &NOTIFY
    &DOEND
  &DOEND
```

```
-* Terminate our CCI connection
&CALL PROC=$CACCI PARMS=(OPT=TERMINATE,ID=&CLID)
&IF &RETCODE = 0 &THEN &DO
  &WRITE Client: TERM RC:+
   &RETCODE FB:&CACCIFDBK EC:&$CAERRORCODE ET:&$CAERRORTEXT
  &WRITE FIN
  &RETSUB
&DOEND

-* Tell the server finished
&WRITE FIN
&RETSUB
```

Chapter 6: Broadcast Services Interface

This section contains the following topics:

About Broadcast Services

Broadcast Services has an NCL external interface which enables you to issue broadcasts easily from installation-written NCL procedures. To use the interface, you execute the NCL procedure $BSCALL.

These topics describe the interface functions, the operands to specify when executing procedure $BSCALL and the return codes that are set on completion.

Note: For more information about the referenced Broadcast Services panels, see the *Reference Guide*.

$BSCALL OPT=SEND

Sends a broadcast.

```
&CONTROL NOSHRVARS NOVARSEG
-EXEC $BSCALL     OPT=SEND
                  TYPE={ALL | APPL | EASINET | TERMINAL |
                         MAIAPPL | NCLAPPL | USERID | NOTIFY }
                [ B1=textline1 ]
                [ B2=textline2 ]
                [ B3=textline3 ]
                [ B4=textline4 ]
                [ IMM={ YES | NO } ]
                [ MASK=usermask ]
                [ RETAIN={ PERM | VIEWED | NO } ]
                [ APPLPROC=procname ]
```

Operands:

OPT=SEND

Specifies a broadcast that is to be sent and which users are to receive it.

TYPE={ALL | APPL | EASINET | TERMINAL |MAIAPPL | NCLAPPL | USERID | NOTIFY }

Specifies the type of broadcast to send.

ALL

Broadcasts to all users. This includes EASINET users (where the &BROLINE1 to 4 variables will be updated) and all product region users who are currently logged on (executes the $NMBRO notification procedure).

APPL

Specifies that an NCL application wishes to broadcast a message to specific users. The broadcast will be actioned for signed on users whose user ID matches the string specified on the MASK operand. The APPLPROC operand is specified with this operand for a customized notification procedure.

Note: The RETAIN=PERM operand cannot be specified with TYPE=APPL.

EASINET

Broadcasts to specific EASINET terminals. This only updates the &BROLINE1 to 4 variables. The MASK operand must specify an LU name.

TERMINAL

Broadcasts to specific terminals, both EASINET and product region. This includes EASINET users (where the &BROLINE1-4 variables will be updated) and all product region users who are currently logged on (executes the $NMBRO notification procedure). The MASK operand must specify an LU name.

MAIAPPL

Broadcasts to MAI users of an application. The broadcast is sent based on the results of a SHOW MAI=ALL command. The MASK operand must specify a string that matches an MAI application name.

NCLAPPL

Broadcasts to users of NCL applications. This is based on the results of a SHOW NCL=ALL command. The MASK operand must specify a string that matches either the base or current procedure.

USERID

Broadcasts to specific users. The broadcast will be actioned for signed-on users whose user ID matches the string specified on the MASK operand.

If the mask contains a user ID which does not contain any wildcard characters, and RETAIN=PERM is specified, and the user ID is not currently logged on to a region, the broadcast will be retained until the broadcast is reviewed when the user logs on.

NOTIFY

Broadcasts to the user specified using the MASK operand, using their preferred method of notification. The method of notification is defined in the user's UAMS profile. (See the chapter “Working with UAMS” in the Security Guide.)

The default method is a standard broadcast (option U) with RETAIN=NO.

B1=*textline*1

Specifies the text for broadcast line 1.

B2=*textline*2

Specifies the text for broadcast line 2.

B3=*textline*3

Specifies the text for broadcast line 3.

B4=*textline*4

Specifies the text for broadcast line 4.

IMM={ YES | NO }

This operand is only used for broadcasts to product region users (that is, TYPE=ALL, ALLMASK, APPL, USERID, MAIAPPL and NCLAPPL). IMM=YES causes the $NMBRO notification procedure to be executed immediately, and the broadcast will interrupt the user's current work. This option is not recommended as data is lost if the user is currently typing.

IMM=NO means that the broadcast will not be actioned until the next user input, such as pressing Enter.

MASK= *usermask*

This operand is mandatory for the following types: NCLAPPL, MAIAPPL, APPL, USERID, ALLMASK, EASINET, and NOTIFY.

An asterisk is used as a wildcard character anywhere within the mask string for any type of broadcast except NOTIFY, which requires an exact user ID.

RETAIN={ PERM | VIEWED | NO }

Specifies the required retention of the broadcast.

PERM means that the broadcast that is sent shall be permanently retained, but not across region restarts.

VIEWED causes the broadcast to be retained until it has been viewed by all receivers. This is the default.

NO means that the broadcast will not be retained. That is, it will only be available to users who are currently logged on.

APPLPROC=*procname*

Specifies the name of the procedure that the $NMBRO notification procedure is to execute. The broadcast text is shared with the procedure in the variables &$BSB1-4. This is valid only with OPT=APPL.

Return Codes:

&RETCODE = 0

$BSCALL completed successfully.

&RETCODE = 8

An error occurred. &SYSMSG is set with an error message.

Example: OPT=SEND

```
&CONTROL NOSHRVARS NOVARSEG
&TEXT = &STR PRODUCTION SYSTEM IS NOW AVAILABLE
-EXEC $BSCALL OPT=SEND TYPE=ALL B1=&TEXT RETAIN=NO
```

Note: If the text entered in the B1-4 fields contains imbedded blanks, $BSCALL must be invoked with &CONTROL NOVARSEG in effect.

$BSCALL OPT=MENU

Displays the Broadcast Services : Primary Menu.

```
&CONTROL NOSHRVARS NOVARSEG
-EXEC $BSCALL OPT=MENU
```

Operands:

OPT=MENU

Specifies to display the Broadcast Services Primary Menu.

Return Codes:

&RETCODE = 0

$BSCALL completed successfully.

&RETCODE = 1

RETURN key pressed.

&RETCODE = 8

An error occurred. &SYSMSG is set with an error message.

Example: OPT=MENU

```
&CONTROL NOSHRVARS
-EXEC $BSCALL OPT=MENU
```

$BSCALL OPT=LISTALL

Lists all active broadcasts.

```
&CONTROL NOSHRVARS NOVARSEG
-EXEC $BSCALL OPT=LISTALL
```

Displays the Broadcast Services : List Broadcasts panel.

Operands:

OPT=LISTALL

Specifies that a list of active broadcasts is required.

Return Codes:

&RETCODE = 0

$BSCALL completed successfully.

&RETCODE = 1

RETURN key pressed.

&RETCODE = 8

An error occurred. &SYSMSG is set with an error message.

Examples: OPT=LISTALL

```
&CONTROL NOSHRVARS
-EXEC $BSCALL OPT=LISTALL
```

Note: This request is the same as selecting option L on the Broadcast Services : Primary Menu. This panel is used to view or delete the active broadcasts that are listed.

$BSCALL OPT=REVIEW

Displays the Broadcast Services : Review Broadcasts panel.

```
&CONTROL NOSHRVARS NOVARSEG
-EXEC $BSCALL OPT=REVIEW
            [ TYPE={ TERMINAL | MAIAPPL | NCLAPPL | USERID | * } ]
            [ MASK=usermask ]
```

To review outstanding broadcasts. The type of broadcasts to be displayed on the review panel is limited by specifying the TYPE operand.

Operands:

OPT=REVIEW

Specifies that a review of outstanding broadcasts is required.

TYPE={ TERMINAL |MAIAPPL | NCLAPPL | USERID | * }

Specifies the type(s) of broadcast to be displayed.

TERMINAL

Display broadcasts to specific product region terminals that were sent using TYPE=TERMINAL or using the TA option on the Broadcast Services Send panel.

MAIAPPL

Display permanent broadcasts to MAI users of an application.

NCLAPPL

Display permanent broadcasts to users of NCL applications.

USERID

Display broadcasts to specific users.

*****

Display broadcasts that have been sent to the calling user ID or LU name. A mask cannot be specified with this option. This is the default TYPE.

MASK= *usermaskl*

The type of broadcast to be displayed determines the type of mask that should be specified. The broadcast will be displayed if the mask specified when the broadcast was sent matches the mask specified on this call.

Return Codes:

&RETCODE = 0

$BSCALL completed successfully.

&RETCODE = 1

RETURN key pressed.

&RETCODE = 8

An error occurred. &SYSMSG is set with an error message.

Examples: OPT=REVIEW

```
&CONTROL NOSHRVARS
-EXEC $BSCALL OPT=REVIEW TYPE=NCLAPPL MASK=SDI*

&CONTROL NOSHRVARS
-EXEC $BSCALL OPT=REVIEW TYPE=MAIAPPL MASK=PRODCICS
```

$BSCALL OPT=DISCARD

Decrements the outstanding receivers counter by one.

```
&CONTROL NOSHRVARS NOVARSEG
-EXEC $BSCALL      OPT=DISCARD
                 [ ID=nnnnnnnn ]
```

This call is made by the distributed $LOGPROC procedure when the message N15577 is encountered. If the log proc procedure is customized, you must make sure that the code that processes the broadcast discarded message is present in the customized log proc procedure.

Operands:

OPT=DISCARD

Specifies that the outstanding receivers counter is to be decremented by one.

ID=*nnnnnnnn*

This is the user-supplied broadcast ID. If this is not specified, your product region will automatically generate a numeric broadcast ID, 8 digits in length.

Return Codes:

&RETCODE = 0

$BSCALL completed successfully.

&RETCODE = 8

An error occurred. &SYSMSG is set with an error message.

&RETCODE = 8

Example: OPT=DISCARD

```
&CONTROL NOSHRVARS
-EXEC $BSCALL OPT=DISCARD ID=123456
```

Notification Exit Interface

A notification exit is an installation written NCL procedure which performs the notification. The procedure is called with the following variables shared:

&$BSB1-*n*

These variables are set to the notification message lines.

&$BSBCNT

This variable is set to the number of lines of the notification.

&$BSUSER

This variable is set to the user ID specified in the notification rule. If it is blank, the UAMS user ID is used.

&$BSPARMS

This variable is set to the parameters specified in the notification rule.

The exit is invoked via an EXEC command and must set the system variable &RETCODE to indicate the following conditions:

0

Notification has been issued without errors.

8

Errors occurred. The notification might have been issued. The system variable &SYSMSG is set with a message that describes the error.

Chapter 7: Dataset Services Interface

This section contains the following topics:

About the Dataset Services Interface

The Dataset Services interface allows you to perform sequential file support functions in z/OS or MSP from product region applications. These functions include:

- Dynamically allocating sequential or partitioned data sets
- Reading and writing to sequential data sets and PDS members
- Manipulating members in PDS directories
- Performing data set maintenance
- Running utility functions
- Performing compound functions

When you use Dataset Services to create or update a partitioned data set (PDS) member, interactive system productivity facility (ISPF) statistics are generated. The DSSISPST system parameter controls the generation of these statistics.

Note: For information about the system parameter, see the *Reference Guide*.

To call the Dataset Services options from an NCL procedure, execute the NCL API $DSCALL, using keywords and shared variables.

Later topics describe the operands that are specified when executing $DSCALL options, and the return codes and variables that are set on completion.

Note: A basic knowledge of z/OS or MSP JCL and system utilities is assumed.

Exit Procedures

Note: For details of the exit procedure supplied with Dataset Services, NMDSSCHK, see the *Security Guide*.

The following table lists the $DSCALL options that can be invoked in alphabetical order.

Option	Function
OPT=ALIAS	Creates an alias for a member of a PDS.
OPT=ALLOC	Allocates an existing cataloged data set.
OPT=ALLOC STAT=NEW	Creates a new data set.
OPT=ALLOC SYSOUT=CLASS	Allocates a SYSOUT data set.

Option	Function
OPT=ALLOCINFO	Obtains allocation information by ddname or relative allocation number.
OPT=BROWSE	Opens a data set and displays a record.
OPT=CATLIST	Displays a selection list of data sets that begin with a specified high level qualifier.
OPT=CLOSE	Closes an open data set.
OPT=COMPRESS	Compresses a PDS.
OPT=CONCAT	Concatenates a set of data sets under a single ddname.
OPT=COPY	Copies sequential data sets or PDS members.
OPT=COPYPDS	Copies a PDS member from a source data set to a target data set.
OPT=COPYSEQ	Copies a sequential data set from a source data set to a target data set.
OPT=CREATE	Creates a data set.
OPT=DECONCAT	Deconcatenates a set of data sets.
OPT=DELETE	Deletes a cataloged data set.
OPT=DELMEM	Deletes a member of a PDS.
OPT=DEQ	Releases an SPF ENQ.
OPT=DSNLIST	Provides in variables a list of data sets concatenated to a DD.
OPT=DSNSPACE	Determines the amount of space in a data set.
OPT=EDIT	Opens a data set and allows a record to be edited.
OPT=ENQ	Performs an ENQ on a data set or member.
OPT=FCLOSE	Combines the CLOSE option and the UNALLOCATE or FREE option into a single call.
OPT=FINDMEM	Provides a visual display of where a member is in a concatenation.
OPT=FOPEN	Combines the ALLOC option for new or existing data sets, and the OPEN option into one call.

Option	Function
OPT=FREE	Deallocates a data set. See OPT=UNALL for a description of this option.
OPT=INFO	Combines the ALLOCATE option for an existing data set, and the UNALL or FREE option into a single call that retrieves data set information
OPT=LISTC	Obtains or displays a list of data sets that begin with a specified prefix.
OPT=MEMLIST	Obtains or displays a list of PDS members.
OPT=MOVE	Copies a sequential data set, then deletes the original data set.
OPT=MOVEPACK	Moves a data set to a specified volume.
OPT=OPEN	Opens a sequential data set or member of a PDS for input or output.
OPT=PRINT	Opens a data set and allows one or more records to be printed.
OPT=READ	Reads one or more records from an open data set.
OPT=RENAME	Renames a data set.
OPT=RENMEM	Renames a member of a PDS.
OPT=SHOWALLOC	Provides a full-screen display of allocated data sets.
OPT=SUBMIT	Submits JCL.
OPT=UNALL	Deallocates a data set.
OPT=UTILITY	Executes the IBM utilities: IEBGENER, IEBCOPY, IEHLIST, IEHMOVE and IDCAMS.
OPT=VOLSPACE	Determines the amount of space available on a volume.
OPT=WRITE	Writes one or more records to an open data set.

Return Codes

Return codes for each option are returned in the variables &$DSRC and &RETCODE. &$DSFDBK may also be set. The return codes are as follows:

RC=0

Function completed. &$DSFDBK will normally be 0 but can also be set to another value to provide additional information. This is documented under each specific function.

RC=4

A Dataset Services error has occurred and the function is incomplete. This is typically a run-time error. The specific error is indicated in &$DSFDBK and these are described in Feedback Codes in this chapter. The corresponding message for each code is N16C*xx* (where xx is the feedback code) and is returned in &SYSMSG.

RC=8

A Dataset Services Interface error has occurred and the function is incomplete. This is typically an application program specification error. The error is indicated in &SYSMSG with message number DS*nnnn*.

RC=12

A system utility has returned a non-zero return code. This may be an error or a warning. &$DSFDBK contains the return code from the system utility, and the SYSPRINT file contains the output from the system utility.

Feedback Codes

$DSCALL returns a feedback code that further qualifies the return codes 0 and 4 as described above. The feedback code is returned in variable &$DSFDBK.

Some feedback codes represent successful completion of the function and may be for information only. For example, feedback code 9 represents the normal end of file condition when a READ operation reaches the end of a file or member.

Note: A detailed description of the meaning of a feedback code is contained in the online help for the message. To display the online help, enter the message ID at the OCS command line and press the help function key (PF1).

The feedback codes (for return code 0 or 4) and their meanings are:

0

Function completed successfully

1

DDname not found

2

Data set not found

3

Member not found

4

Data set is not allocated

5

DYNALLOC failed

6

Data set is in use

7

Member replaced

8

Start of new directory

9

End of file or member

10

Write error on CLOSE

11

Stow error on CLOSE

12

Error on CLOSE

13

CLOSE ABEND occurred

14

Delete failure-not expired

15

Delete failure-SCRATCH failed

16

Rename failed

17

Rename failed, data set exists on more than one volume

18

Rename failed, data set is not on DASD

19

Reserved

20

Data set is migrated

21

Volume is not mounted

22

ENQ failed

23

Reserved

24

Data set is OPEN

25

Data set is VSAM

26

Reserved

27

Reserved

28

Reserved

29

Reserved

30

DCB OPEN failed

31

I/O error occurred

32

DCB ABEND occurred

33

Record supplied for WRITE is invalid length

34

Directory block length is invalid

35

OPEN mode is invalid

36

Userdata is invalid length

37

Number of userdata TTRs is invalid

38

PDS cannot be opened with MODE=EXTEND or DISP=MOD

39

Reserved

40

Invalid ddname

41

Invalid member name

42

Invalid second member name

43

Invalid data set name

44

Invalid second data set name

45

An invalid parameter was specified

46

DDLIST format is invalid

47

DDLIST must contain at least two ddnames

48

Invalid DDLIST, Duplicate DDNAME found

49

Reserved

50

Specified DSORG is invalid for this request

51

No member name has been specified

52

No member name may be specified

53

Spanned RECFM is not supported by Dataset Services

54

New member name already exists

55

PDS directory is full

56

Specified ENQ is already held

57

Specified ENQ is not held

58

Reserved

59

Reserved

60

Unable to determine allocation request type

61

ALLOC MDO operand has been omitted

62

ALLOC MDO operand is not allowed

63

ALLOC MDO operand is invalid

64

DYNALLOC request refused by SMS

65

Allocation relative request number reached

66

Last allocation relative request number does not exist

67

Data set already exists

68

DDNAME is already in use

69

Reserved

70

MDO update failed

71

ATTACH of system utility module failed

72

Obtain of data set information failed

73

RDJFCB failed

74

OBTAIN failed

75

Unexpected ENQ/DEQ return code

76

NOTE failed

77

Unexpected return code from function

78

Unexpected ABEND occurred in function

79

Reserved

80

Path name is invalid

81

Path name is already defined

82

Path name is not defined

83

Path name is wrong access class

84

Request is not valid on path now

85

Path has had a previous error

86

Subtask status is invalid

87

NMDSSCHK exit has failed

88

NMDSSCHK exit has refused request

89

Reserved

90

Request is not supported

91

An ABEND has occurred

92

Request has been canceled

93

A storage shortage has occurred

94

Reserved

95

Reserved

96

Reserved

97

Reserved

98

Reserved

99

Dataset Services has entered shutdown

$DSCALL OPT=ALIAS

Creates an alias for a member of a PDS.

```
&CONTROL SHRVARS=($DS)

EXEC $DSCALL OPT=ALIAS
             DSN=dataset_name
             MEMBER=member_name
             ALIAS=member_name
```

This call is used to create an alias entry for a member of a PDS. The alias entry inherits the SPF statistics of the original member. This call fails if a member with the same alias name already exists.

Operands:

OPT=ALIAS

Specifies that an alias be created for a member of a PDS.

DSN=*dataset_name*

Specifies the name of the PDS containing the member. No member name is specified.

MEMBER=*member_name*

Specifies the existing member for which an alias is to be created.

ALIAS=*member_name*

Specifies the alias name to be created. Any subsequent references to this name result in access to the member specified in the MEMBER operand.

Return Codes:

For information on &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes and Feedback Codes in this chapter.

Example: OPT=ALIAS

```
EXEC $DSCALL OPT=ALIAS DSN=CUSTOMER.DATA MEMBER=TEST01 + ALIAS=PROD01
```

$DSCALL OPT=ALLOC

Allocates an existing cataloged data set.

```
&CONTROL SHRVARS=($DS)

EXEC $DSCALL OPT=ALLOC
             DSN=dataset_name
           [ DD=DD_name ]
           [ DISP= { KEEP | DELETE } [, { KEEP | DELETE } ] ]
           [ MIGRATE= { YES | NO } ]
           [ MOUNT= { NO | YES } ]
           [ STAT= { OLD | MOD | SHR } ]
           [ INFO= { YES | WAIT | NOWAIT | NO } ]
```

This call is used to allocate an existing cataloged data set to your product region. The data set remains allocated to your product region until it is explicitly deallocated, or your product region terminates.

Operands:

OPT=ALLOC

Specifies that a data set is to be allocated.

DSN=*dataset_name*

Specifies the data set name to allocate.

DD=DD_*name*

Specifies the ddname for the allocation. If this operand is omitted, a ddname is generated by the operating system.

DISP={ KEEP | DELETE } [, { KEEP | DELETE }]

Specifies the Normal and Conditional Disposition of the data set. The disposition relates to the normal or conditional termination of your product region, not of the requesting NCL process. If KEEP is specified, the data set remains cataloged. If DELETE is specified, the data set is uncataloged and deleted.

MIGRATE={ YES | NO }

Specifies whether to allocate a data set if it has been migrated. If MIGRATE=NO is specified, Dataset Services checks that the data set has been migrated (for example, by DFHSM). If the data set has been migrated, Dataset Services rejects the allocation request. If MIGRATE=YES is specified, Dataset Services issues the allocation request without checking that the data set has been migrated.

MOUNT={ NO | YES }

Specifies whether a volume mount is allowed. If MOUNT=NO is specified, and an allocation request requires a volume which is off-line, the allocation request fails. If MOUNT=YES is specified and an allocation request requires a volume which is off-line, a MOUNT request for the volume is issued.

STAT= { OLD | MOD | SHR }

Specifies the data set status.

INFO= { YES | WAIT | NOWAIT | NO }

Allocation information is obtained by using the Dynamic Allocation (SVC 99) services, which require the SYSZTIOT system resource, which may not be available. The INFO= parameter specifies whether allocation information is required and, if so, the action required when SYSZTIOT is not available:

INFO=YES

Indicates that information is required. If the SYSZTIOT resource is not immediately available, then $DSCALL retries the allocation information request up to five times at two-second intervals. This is the default.

INFO=WAIT

Also indicates that information is required. If the SYSZTIOT resource is not immediately available, then $DSCALL retries the allocation information request at two-second intervals until the information is available.

INFO=NOWAIT

Also indicates that information is required. If the SYSZTIOT resource is not immediately available, then $DSCALL does not retry.

INFO=NO

Indicates that allocation information is not required.

If allocation information is requested (INFO=YES, INFO=WAIT, or INFO=NOWAIT), then the information is returned as described in the Return Variables section below. If allocation information is not requested (INFO=NO) or the SYSZTIOT resource is not available after any retries (INFO=YES or INFO=NOWAIT), then the return variables are undefined.

Return Codes:

$DSRC	$DSFDBK	Meaning
0	0	Data set was allocated; allocation details available in &$DS* variables as described below.
0	non-zero	Data set was allocated; however, allocation details are incomplete.

$DSRC	$DSFDBK	Meaning
4	non-zero	Data set was not allocated.
8	non-zero	A Dataset Services Interface error has occurred and the function is incomplete. This is typically an application program specification error. The error is indicated in &SYSMSG with message number DSnnnn.

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSSTAT

Status of data set; values are OLD, MOD, or SHR

&$DSDSN

Full data set name with member name omitted, and true name if an alias was entered

&$DSMEM

Member name if a member of a PDS

&$DSORG

Data set organization; values are PS, PO, PSU, POU, CX, CQ, MQ, GS, TX, TQ, TR, and VS

&$DSVOL

First volume

&$DSUNIT

Unit name for the volume

&$DSRECF

Record format; values are F,FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM, VBM, VSM, VBSM, U, UA, UM

&$DSRECL

Record length

&$DSRECA

Actual usable record length. If the record format is variable length, then the value is record length minus 4. In all other cases the value is the same as record length.

&$DSBLKS

Block size

&$DSDDNAME

DDname from the allocation, as supplied or as generated by the operating system

&$DSSTORCLS

SMS storage class if SMS is active for this data set

&$DSMGMTCLS

SMS management class if SMS is active for this data set

&$DSDATACLS

SMS data class if SMS is active for this data set

&$DSDYNEC

DYNALLOC error code

&$

DYNALLOC information code

Examples: OPT=ALLOC

```
EXEC $DSCALL OPT=ALLOC DSN=CUSTOMER.DATA STAT=SHR
```

Notes:

Only cataloged data sets are supported. Unit and volume parameters are not supported by this function.

If the data set has been migrated, the allocation may be delayed if a tape mount is required to recover the migrated data set.

Allocation of a data set uses the Dynamic Allocation (SVC 99) services, which require the SYSZTIOT system resource. An unconditional SVC 99 is used at first. This causes the allocation function to wait if SYSZTIOT is unavailable.

If allocation fails, then &$DSRC is set to 4 and specific information is made available in &$DSFDBK, &$DSDYNEC, &$DSDYNIC, and &SYSMSG.

If the data set is allocated, then &$DSRC is always set to zero. If allocation information was requested (INFO=YES, INFO=WAIT, or INFO=NOWAIT), then a conditional SVC 99 is used to obtain the allocation information which is returned in the &$DS* variables. If the information is available in the return variables, then &$DSFDBK is also set to zero. If INFO=YES or INFO=NOWAIT was specified, then failure of the allocation information request due to SYSZTIOT being unavailable is indicated by the following:

```
&$DSFDBK = 5

&$DSDYNEC = 0254
```

More information:

$DSCALL OPT=ALLOCINFO (see page 1071)

$DSCALL OPT=ALLOC STAT=NEW

Creates and allocates a new non-VSAM data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=ALLOC STAT=NEW
             FORMAT=dataset_format
             ORG=dataset_organization
             SPACE={ TRK | CYL } , pri,sec [ ,dir ]
           [ BLKSIZE=blocksize ]
           [ DATACLS=class ]
           [ DD=DD_name ]
           [ DISP={ KEEP | DELETE } [, { KEEP | DELETE } ] ]
           [ DSN=dataset_name ]
           [ DSNTYPE={ HFS | LIBRARY | PDSE } ]
           [ FREE={ UNAL | CLOSE } ]
           [ LRECL=logical_record_length ]
           [ MGMTCLS=class ]
           [ MOUNT={ NO | YES } ]
           [ RLSE={ NO | YES } ]
           [ STORCLS=class ]
           [ VOL=volser [ UNIT=unit ] ]
```

This call is used to create a new data set and allocate it to your product region. The data set remains allocated to your product region until it is explicitly deallocated, until a CLOSE is requested if allocated with FREE=CLOSE, or until your product region terminates. When the data set is deallocated or your product region terminates, the data set is deleted if the relevant disposition specified DELETE.

Operands:

OPT=ALLOC

Specifies that a data set is to be allocated.

STAT=NEW

Specifies that the status of the data set is NEW.

FORMAT=*dataset_format*

Specifies the data set format for the new data set. Valid values are: F, FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM VBM, VSM, VBSM, U, UA, UM.

ORG=*dataset_organization*

Specifies the data set organization for the new data set. Valid values are: PS, PO, PSU, and POU.

SPACE={ TRK | CYL } , *pri*,*sec* [,*dir*]

Specifies the space allocation in tracks or cylinders. The SPACE type and primary allocation quantity are required. The directory allocation quantity is required if the data set is a PDS.

BLKSIZE=*blocksize*

Specifies the block size for the new data set. The value must be an integer in the range 0 to 32,760.

DATACLS=*class*

Specifies the SMS data class.

DD=*DD_name*

Specifies the ddname for the allocation. If this operand is omitted, a ddname is generated by the operating system.

DISP={ KEEP | DELETE } , { KEEP | DELETE }

Specifies the Normal and Conditional Disposition of the data set. The disposition relates to the normal or conditional termination of your product region, not of the requesting NCL process. If KEEP is specified, the data set remains cataloged.

If DELETE is specified, the data set is uncataloged and deleted.

If a temporary data set is allocated (that is, the DSN operand is omitted) then DISP=DELETE, DELETE is forced. This ensures the data set is deleted when it is deallocated or when your product region terminates.

If the DSN operand is specified and the DISP operand is omitted, DISP defaults to KEEP. In this case the data set will be cataloged. If your product region terminates abnormally, the action taken is decided by the operating system.

DSN=*dataset_name*

Specifies the data set name to be created. If the DSN operand is omitted, a temporary data set is allocated. When a temporary data set is allocated, no disposition is allowed and DELETE, DELETE is forced.

DSNTYPE={ HFS | LIBRARY | PDSE }

Specifies whether you want to create a Hierarchical File System (HFS) file or a PDSE data set:

HFS

Specifies an HFS file

LIBRARY or PDSE

Specifies a PDSE data set

FREE={ UNAL | CLOSE }

Specifies whether the data set is to be deallocated by explicit request only (FREE=UNAL) or is to be deallocated when the file is closed (FREE=CLOSE).

LRECL=*logical_record_length*

Specifies the logical record length for the new data set. The value must be an integer in the range 1 to 32,760.

MGMTCLS=*class*

Specifies the SMS management class.

MOUNT={ NO | YES }

Specifies whether a volume mount is allowed. If MOUNT=NO is specified, and an allocation request requires a volume which is off-line, the allocation request fails. If MOUNT=YES is specified and an allocation request requires a volume which is off-line, a MOUNT request for the volume is issued.

RLSE={ NO | YES }

Specifies the secondary space release option.

STORCLS=*class*

Specifies the SMS storage class.

VOL=*volser* [UNIT=*unit*]

Specifies the volume serial number and unit name for the new allocation. If the VOL operand is omitted, the operating system determines if the allocation is allowed, and may choose to allocate the data set or any volume which the requesting user is authorized for.

Return Codes:

$DSRC	$DSFDBK	Meaning
0	0	Data set was allocated; allocation details available in &$DS* variables as described below.
0	non-zero	Data set was allocated; however, allocation details are incomplete.
4	non-zero	Data set was not allocated.
8	non-zero	A Dataset Services Interface error has occurred and the function is incomplete. This is typically an application program specification error. The error is indicated in &SYSMSG with message number DS*nnnn*.

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSSTAT

Status of data set; value is NEW

&$DSDSN

Full data set name with member name omitted, and true name if an alias was entered

&$DSDSNTYPE

Type of file or data set; value is HFS or LIBRARY

&$DSMEM

Member name if a member of a PDS

&$DSORG

Data set organization; values are PS, PO, PSU, and POU

&$DSVOL

First volume

&$DSUNIT

Unit name for the volume

&$DSRECF

Record format, values are F, FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM, VBM, VSM, VBSM, U, UA, UM

&$DSRECL

Record length

&$DSRECA

Actual usable record length. If the record format is variable length, then the value is record length minus 4. In all other cases the value is the same as record length.

&$DSBLKS

Block size

&$DSDDNAME

DDname from the allocation, as supplied or as generated by the operating system

&$DSSTORCLS

SMS storage class if SMS is active for this data set

&$DSMGMTCLS

SMS management class if SMS is active for this data set

&$DSDATACLS

SMS data class if SMS is active for this data set

&$DSDYNEC

DYNALLOC error code

&$DSDYNIC

DYNALLOC information code

Example: STAT=NEW

```
EXEC $DSCALL OPT=ALLOC DSN=CUSTOMER.DATA STAT=NEW +
   ORG=PO FORMAT=FB BLKSIZE=800 +
   LRECL=80 SPACE='TRK,5,1,10'
```

Note: Most non-VSAM data sets is allocated. However, IS and DA data sets are not supported.

More information:

$DSCALL OPT=ALLOC (see page 1060)

$DSCALL OPT=ALLOC SYSOUT=class

Allocates a SYSOUT data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=ALLOC
             SYSOUT=class
           [ BLKSIZE=blocksize ]
           [ COPIES=copies ]
           [ DD=DD_name ]
           [ DESTID=destination_ID ]
           [ FCB=fcb ]
           [ FOLD= { NO | YES } ]
           [ FORMAT=dataset_format ]
           [ FREE= { UNAL | CLOSE } ]
           [ HOLD= { NO | YES } ]
           [ LRECL=logical_record_length ]
           [ PGM=program_name ]
           [ UCS=ucs ]
           [ USERID=user_ID ]
```

This call is used to allocate a SYSOUT data set to your product region. The data set remains allocated to your product region until it is explicitly deallocated, until a CLOSE is requested if allocated with FREE=CLOSE, or until your product region terminates.

Operands:

OPT=ALLOC

Specifies that a data set is to be allocated.

SYSOUT=*class*

Specifies the SYSOUT data set class. The value must be a single alphabetic character which represents a valid SYSOUT class in the system in which your product region is executing.

BLKSIZE=*blocksize*

Specifies the block size for the data set. The value must be an integer in the range 1 to 32,760.

COPIES=*copies*

Specifies the number of SYSOUT copies. The value must be an integer in the range 1 to 255. If the operand is not specified, one copy is produced.

DD=*DD_name*

Specifies the ddname for the allocation. If this operand is omitted, a ddname is generated by the operating system.

DESTID=*destination*_ID

Specifies the Remote Workstation Identifier of the workstation to which the data set is to be routed. The value must be 1 to 8 characters.

FCB=*fcb*

Specifies the SYSOUT Forms Control Block (FCB) Image Identifier. The value must be 1 to 4 characters.

FOLD={ NO | YES }

Specifies the SYSOUT fold option.

FORMAT=*dataset_format*

Specifies the data set format. Valid values are: F, FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM VBM, VSM, VBSM, U, UA, UM.

FREE={ UNAL | CLOSE }

Specifies whether the data set is to be deallocated by explicit request only (FREE=UNAL) or is to be deallocated when the file is closed (FREE=CLOSE).

HOLD={ NO | YES }

Specifies the SYSOUT hold option.

LRECL=*logical_record_length*

Specifies the logical record length for the data set. The value must be an integer in the range 1 to 32,760.

PGM=*program_name*

Specifies the SYSOUT program name. The value must be a 1- to 8-character name that conforms to PDS member naming conventions. See the notes on $DSCALL OPT=ALLOC.

UCS=*ucs*

Specifies the SYSOUT Universal Character Set name. The value must be 1 to 4 characters.

USERID=*user*_ID

Specifies the user ID of the user at the remote workstation who will receive the data set.

Return Codes:

$DSRC	$DSFDBK	Meaning
0	0	Data set was allocated; allocation details available in &$DS* variables as described below.
0	non-zero	Data set was allocated; however, allocation details are incomplete.
4	non-zero	Data set was not allocated.
8	non-zero	A Dataset Services Interface error has occurred and the function is incomplete. This is typically an application program specification error. The error is indicated in &SYSMSG with message number DS*nnnn*.

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSRECF

Record format; values are F, FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM, VBM, VSM, VBSM, U, UA, UM

&$DSRECL

Record length

&$DSRECA

Actual usable record length. If the record format is variable length, then the value is record length minus 4. In all other cases, the value is the same as record length.

&$DSBLKS

Block size

&$DSDDNAME

DDname from the allocation, as supplied or as generated by the operating system

&$DSDYNEC

DYNALLOC error code

&$DSDYNIC

DYNALLOC information code

Examples: SYSOUT=class

```
EXEC $DSCALL OPT=ALLOC +
   SYSOUT=X BLKSIZE=121 LRECL=121 FORMAT=FBA
```

Note: If PGM=INTRDR is specified, the SYSOUT file is used to submit JCL for execution; however, it is recommended that the $DSCALL OPT=SUBMIT function is used.

More information:

$DSCALL OPT=ALLOC (see page 1060)

$DSCALL OPT=ALLOCINFO

Obtains allocation information by ddname or relative allocation number.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=ALLOCINFO
           { DD=DD_name | RELNUM=number }
```

This call is used to obtain information about a data set that is allocated to your product region. The data set to be queried is specified using the ddname of the data set, or by specifying the relative allocation number for the data set. If the ddname method is used, information can only be obtained about the first data set in the concatenation. If the relative allocation number method is used, information is obtained about any data set allocated to your product region.

Operands:

OPT=ALLOC

INFO Specifies that allocation information is to be obtained.

DD=DD_name

Specifies the ddname of the data set for which information is to be obtained.

RELNUM=number

Specifies the relative allocation number assigned by the operating system. The number must be an integer in the range 1 to 32,760. A full list of data sets allocated to your product region is obtained by requesting information using RELNUM=1, RELNUM=2, and so on. When the last data set allocated to your product region is reached, a feedback code of 65 (&$DSFDBK=65) is returned to indicate the last data set has been reached.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSDYNEC

DYNALLOC error code

&$DSDYNIC

DYNALLOC information code

&$DSFDBK

Feedback code

&$DSSTAT

Status of data set; values are NEW, OLD, MOD, or SHR

&$DSDSN

Full data set name or true name if an alias is used

&$DSMEM

Member name if a member of a PDS

&$DSORG

Data set organization; values are PS, PO, PSU, POU, CX, CQ, MQ, GS, TX, TQ, and TR

&$DSDDNAME

DDname for allocation. If you perform a query through RELNUM then this variable may be blank if the relative number is part of a concatenation.

&$DSSTORCLS

SMS storage class if SMS is active for this data set

&$DSMGMTCLS

SMS management class if SMS is active for this data set

&$DSDATACLS

SMS data class if SMS is active for this data set

&$DSCDISP

Conditional data set disposition

&$DSNDISP

Normal data set disposition

Example: OPT=ALLOCINFO

```
EXEC $DSCALL OPT=ALLOCINFO DD=SYSDD
```

Notes:

The ALLOCINFO function requires access to a system resource called SYSZTIOT that is also used by other functions such as ALLOCATE. The SYSZTIOT resource is come unavailable for a long period (for example, when waiting for a tape mount to be satisfied if a data set being allocated has been migrated to tape by DFHSM).

To avoid possible delays in the completion of Dataset Services functions, your product region checks to ensure that the SYSZTIOT resource is available. If it is unavailable, the request fails immediately rather than waiting for SYSZTIOT to become available. This should be taken into consideration when writing NCL procedures which use the Allocation Information Query function.

Failure of an allocation information request due to SYSZTIOT being unavailable is indicated by the following:

```
&$DSFDBK = 5

&$DSDYNEC = 0254
```

$DSCALL OPT=BROWSE

Displays a sequential data set or member of a PDS for browsing.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=BROWSE
             DATA={ }
             DD=DD_name or DSN=dataset_name
           [ MEMBER=member_name ]
           [ LIMIT=number ]
           [ TRUNCATE=number ]
```

Use this option to display the records in a member of a data set. This is a display only function.

Operands:

OPT=BROWSE

Specifies that a browse action is to be performed.

DATA={ }

Specifies the records to be read.

DD=DD_*name or* DSD=*dataset_name*

Specifies the ddname or data set name of the data set to be opened for browsing.

MEMBER=*member_name*

Specifies the member name. If the DD operand is used to identify a concatenated data set, the member is obtained from the first data set that it exists in.

LIMIT=*number*

Specifies the maximum number of records to be browsed in the file. This operand is only applicable if DATA=* is specified.

TRUNCATE=*number*

Specifies the length to which the records displayed from the file are truncated. The value TRUNCATE operand must be in the range 1 to 32,760.

$DSCALL OPT=CATLIST

Displays a selection list of data sets that begin with a specified name prefix.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=CATLIST
             QUAL= qualifier
```

This call allows users of your product region to obtain a full-screen selection list of cataloged data sets that have a specified name prefix. The list is obtained by your product region using an IDCAMS LISTCAT operation. When the list is displayed, the user can obtain data set information about data sets in the list.

Operands:

OPT=CATLIST

Specifies that a selection list of data sets be obtained.

QUAL=*qualifier*

The fully qualified name prefix for the list of data sets to be displayed. The qualifier can consist of more than one level.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=CATLIST

```
EXEC $DSCALL OPT=CATLIST QUAL=PROD1.MAIN
```

The CATLIST option returns a selection list such as the one shown in the first example that follows. Enter S beside a data set to select it and display data set information, as shown in second example that follows.

The following is an example of OPT=CATLIST Output:

```
USER01------------ Dataset Services : Listcat Output ---------------------
Command ===>                                                Scroll ===> PAGE

                              S/=Select B=Browse E=Edit P=Print M=Memlist
                                  D=Delete R=Rename SUB=Submit RC=Recall
  Dataset Name                            Catalog   Volume   Created
  PROD1.MAIN.ALERTCKP                     VMVS005            17-AUG-2008
  PROD1.MAIN.ALERTCKP.DATA                VMVS005   MVS007   17-AUG-2008
  PROD1.MAIN.ALERTCKP.INDX                VMVS005   MVS007   17-AUG-2008
S PROD1.MAIN.ALERTHST                     VMVS005            17-AUG-2008
  PROD1.MAIN.ALERTHST.DATA                VMVS005   MVS007   17-AUG-2008
  PROD1.MAIN.ALERTHST.INDX                VMVS005   MVS007   17-AUG-2008
  PROD1.MAIN.BASE.INSTALL                 VMVS005   PGM002   16-AUG-2008
  PROD1.MAIN.CMDLIB                       VMVS005   PGM002   16-AUG-2008
  PROD1.MAIN.CSI                          VMVS005            16-AUG-2008
  PROD1.MAIN.CSI.DATA                     VMVS005   PGM002   16-AUG-2008
  PROD1.MAIN.CSI.INDEX                    VMVS005   PGM002   16-AUG-2008
  PROD1.MAIN.HELP                         VMVS005   PGM002   16-AUG-2008
  PROD1.MAIN.ICOPANL                      VMVS005            17-AUG-2008
  PROD1.MAIN.ICOPANL.DATA                 VMVS005   MVS007   17-AUG-2008
  PROD1.MAIN.ICOPANL.INDX                 VMVS005   MVS007   17-AUG-2008
  PROD1.MAIN.INSTAL                       VMVS005   PGM002   16-AUG-2008
F1=Help      F2=Split     F3=Exit                   F5=Find       F6=Refresh
F7=Backward  F8=Forward   F9=Swap
```

The following is an example of Information for a Selected Data Set:

```
USER01------------------ PROD1.MAIN.ALERTHST -----------Columns 001 079
Command ===>                                             Scroll ===> PAGE
***************************** TOP OF DATA ******************************
Dataset name ........... PROD1.MAIN.ALERTHST
Dataset organization ... VSAM
Dataset record format ..
Logical record length ..
Dataset blocksize ......
Dataset volume ......... MVS007
Dataset unit ........... 3380
Storage class .......... DASD
Number of extents ...... 0
Create date ............ 17-AUG-2008
Expiration date ........ None
Last reference date .... Not available
Tracks per cylinder .... 15
Primary tracks ......... 0
Allocated tracks ....... 0
Used tracks ............ 0
Secondary allocation ... 0
**************************** BOTTOM OF DATA ****************************
F1=Help      F2=Split     F3=Exit      F4=Return    F5=Find
F7=Backward  F8=Forward   F9=Swap      F10=Left    F11=Right
```

Note: This function performs an IDCAMS LISTCAT to obtain the list of data sets. The SYSPRINT data set is automatically deleted. The only option available on the selection list is used to obtain data set information. The information is returned unless the data set is migrated.

$DSCALL OPT=CLOSE

Closes an open data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL     OPT=CLOSE
                 { DD=DD_name | ID=path_name }
```

Use this function to close an open data set.

Operands:

OPT=CLOSE

Specifies that a close action be performed.

DD=*DD_name*

Specifies the ddname of the data set to close. This operand is used if the path ID is not provided.

ID=*path_name*

Specifies the path ID of the data set to close.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=CLOSE

```
EXEC $DSCALL OPT=CLOSE DD=SYSDD
```

Note: For an OUTPUT data set, the final block is written. This may cause an ABEND message if the data set exceeds its size limit.

$DSCALL OPT=COMPRESS

Compresses a PDS.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=COMPRESS
             DSN=dataset_name
           [ DISPLAY= {YES | NO } ]
           [ STAT= { OLD | SHR } ]
           [ SYSPRINT= { DELETE | KEEP | DSN } ]
```

This call is used to compress a PDS using IEBCOPY. The compress function is used to reclaim unused space in the data set. This reduces fragmentation within the PDS and can result in more efficient use of the space allocated to the PDS.

Operands:

OPT=COMPRESS

Specifies that the data set be compressed.

DSN=*dataset_name*

Specifies the name of the data set to be compressed. No member name is specified.

DISPLAY={YES | NO }

Specifies whether to display the output on completion of the function. This operand is only valid if SYSPRINT=DSN is specified. If DISPLAY=YES is specified, a full-screen display of the SYSPRINT data set is presented after the compress operation has completed.

STAT={ OLD | SHR }

Specifies if the data set to be compressed is allocated exclusively for IEBCOPY (STAT=OLD) or is allowed to be shared with other users (STAT=SHR). It is recommended that STAT=OLD be specified to avoid the possibility of the PDS being corrupted if another user updates the PDS during the compress operation.

SYSPRINT={ DELETE | KEEP | DSN }

Specifies the type of data set to be allocated to the SYSPRINT DD for IEBCOPY.

- If SYSPRINT=KEEP is specified, your product region allocates a SYSOUT file with CLASS=A. After the compress operation has completed, the SYSOUT data set is deallocated with DISP=KEEP to cause it to be kept. The output from the compress operation is browsed through, for example, SDSF.
- If SYSPRINT=DELETE is specified, your product region allocates a SYSOUT file with CLASS=A. After the compress operation has completed, the SYSOUT file is deallocated with DISP=DELETE to cause it to be deleted. The output from the compress operation cannot be browsed.
- If SYSPRINT=DSN is specified, your product region allocates a temporary data set which is deleted when your product region terminates. The output from the compress operation is browsed through ISPF or by specifying DISPLAY=YES for this call. The data set is not deallocated from your product region on completion of the compress operation. The data set name, ddname, and volume for the data set are returned to the caller in variables. Use $DSCALL with OPT=FREE specifying the SYSPRINT ddname to remove the data set created by SYSPRINT.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSPRINTDSN

SYSPRINT data set name if SYSPRINT=DSN is specified, or SYSOUT=A if SYSPRINT=DELETE or SYSPRINT=KEEP

&$DSPRINTDD

SYSPRINT ddname

&$DSPRINTVOL

SYSPRINT data set volume

Example: OPT=CLOSE

```
EXEC $DSCALL OPT=CLOSE  DSN=SYS.WORK.DEV01 STAT=SHR +
   SYSPRINT=DSN DISPLAY=NO
```

Note: IEBCOPY is invoked internally.

$DSCALL OPT=CONCAT

Concatenates a set of data sets under a single ddname.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=CONCAT
             DDLIST=' ddname1, ddname2, ... ddnamen '
```

This call is used to concatenate two or more ddnames which are allocated to your product region under a single ddname. Each ddname in the list to be concatenated may be a single data set allocation or may itself be a concatenated DD.

Operands:

OPT=CONCAT

Specifies that data sets are to be concatenated.

DDLIST='*ddname*1, *ddname*2,...*ddnamen* '

Specifies the list of ddnames to be concatenated. The DDs are concatenated in the order in which they appear in the list. The resulting concatenated DD is assigned the first ddname in the list. All ddnames in the list other than the first cannot be directly referenced after concatenation has completed. An attempt to directly reference these ddnames will be rejected by the operating system. If the DD is subsequently deconcatenated, the original ddnames is directly referenced again. For more information, see $DSCALL OPT=DECONCAT.

Note: A maximum of 50 ddnames is specified in the DDLIST parameter.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSDYNEC

DYNALLOC error code

&$DSDYNIC

DYNALLOC information code

Example: OPT=CONCAT

```
EXEC $DSCALL OPT=CONCAT DDLIST='DD1,DD2'
```

This example concatenates all data sets allocated under the ddname 'DD2' to the 'DD1' DD data set list. The resulting DD is called 'DD1' and the ddname 'DD2' is no longer recognized by the operating system.

Note: Data sets that are concatenated in the JCL for your product region are deemed to be permanently concatenated, which means the DD cannot be deconcatenated. Data sets that are dynamically concatenated by your product region is deconcatenated.

$DSCALL OPT=COPY

Copies sequential data sets or PDS members.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=COPY
             FROMDSN=dataset_name
             TODSN=dataset_name
           [ DISPLAY= { YES | NO } ]
           { MEMBER=member_name |
             MEMBER={ * | member_name [, member_name,... ] } }
           [ REPLACE= { YES | NO } ]
           [ STAT= { SHR | OLD } ]
           [ SYSPRINT= { KEEP | DELETE | DSN } ]
```

This call is used to copy a sequential data set to another sequential data set, a sequential data set to a PDS member, a PDS member to a sequential data set, or PDS members to another PDS.

Operands:

OPT=COPY

Specifies that the sequential data set or PDS members be copied.

FROMDSN=*dataset_name*

Specifies the name of the sequential data set or PDS to be copied.

- If the value of the FROMDSN operand is a sequential data set, the value of the TODSN operand is a sequential data set or a PDS with a single member name specified.
- If the value of the FROMDSN operand is a PDS with no member name specified, the value of the TODSN operand must be a PDS and cannot have a member name specified. A selection list is presented to allow selection of individual members to be copied.
- If the value of the FROMDSN operand is a PDS with an asterisk (*) specified as the member name, the value of the TODSN operand must also be a PDS, and all members in the PDS are copied.
- If the value of the FROMDSN operand is a PDS with a member name containing a mask (for example ABC*), the value of the TODSN data set must also be a PDS, and all member names that fit specified mask are copied. No selection list is presented.
- If the value of the FROMDSN operand is a PDS with a single member name specified, the value of the TODSN operand is a sequential data set or a PDS. If the value of the TODSN operand is a PDS and has no member name specified, the PDS member specified in the FROMDSN operand is copied with no name change. If the value of the TODSN operand is a PDS and does specify a member name, that member name is given to the PDS member specified in the FROMDSN operand.

The following table summarizes the results of the various copy operations.

From	To	Member	Result	Utility
SEQ	SEQ	NULL	Data set copied	IEBGENER
SEQ	PDS (mem)	NULL	Copied to member	IEBGENER
PDS	PDS	NULL	Selection List	IEBCOPY
PDS(*)	PDS	NULL	All members copied	IEBCOPY
PDS (mask)	PDS	NULL	Selected members copied	IEBCOPY

From	To	Member	Result	Utility
PDS (mem)	PDS	NULL	Member copied to same name	IEBGENER
PDS (mem)	PDS (newmen)	NULL	Member copied and renamed	IEBGENER
PDS (mem)	SEQ	NULL	Member copied to sequential data set	IEBGENER
PDS	PDS	NON-BLANK	As specified in the Member parameter	IEBCOPY

TODSN=*dataset_name*

Specifies the name of the target data set.

DISPLAY={ YES | NO }

Specifies whether to display the output on completion of the function. This operand is only valid if SYSPRINT=DSN is specified. If DISPLAY=YES is specified, a full-screen display of the SYSPRINT data set is presented after the copy operation has completed.

MEMBER={ *member_name* | MEMBER={* | *member_name* [, *member_name*...] } }

Specifies the names of the members to be copied. If MEMBER=* is specified, all members in the PDS are copied.

If the value of the MEMBER operand is a member name containing a mask (for example, ABC*), all members starting with the specified prefix are copied.

If the value of the MEMBER operand is a list of single member names, only the specified members are copied.

REPLACE={YES | NO }

Specifies whether to replace an existing data set of the same name and is only applicable where the values of the FROMDSN and TODSN operands are both PDS data sets.

If REPLACE=YES is specified, the member name specified in the FROMDSN operand is copied and given the member name specified in the TODSN operand even if that member name already exists.

If REPLACE=NO is specified, the member specified in the FROMDSN operand is not copied if the member name specified in the TODSN operand already exists.

STAT={ OLD | SHR }

Specifies whether the data set specified in the TODSN operand is allocated exclusively for the copy operation (STAT=OLD), or is shared with other users (STAT=SHR). It is recommended that STAT=OLD be used as data set integrity cannot be guaranteed if another user attempts to access the data set while the copy operation is in progress.

The data set specified in the FROMDSN operand is always allocated with STAT=SHR.

SYSPRINT={ DELETE | KEEP | DSN }

Specifies the type of data set to be allocated to the SYSPRINT DD for the copy operation.

If SYSPRINT=KEEP is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed the SYSOUT data set is deallocated with DISP=KEEP to cause it to be kept. The output from the copy operation is browsed through, for example, SDSF.

If SYSPRINT=DELETE is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed, the SYSOUT file is deallocated with DISP=DELETE to cause it to be deleted. The output from the copy operation cannot be browsed.

If SYSPRINT=DSN is specified, your product region allocates a temporary data set, which is deleted when your product region terminates. The output from the copy operation is browsed through ISPF or by the use of the DISPLAY=YES operand on this call. The data set is not deallocated from your product region on completion of the copy operation. The data set name, ddname and volume for the data set are returned in variables to the caller. Removing the data set is the responsibility of the caller-the caller should deallocate the data set to delete it.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSPRINTDSN

SYSPRINT data set name

&$DSPRINTDD

SYSPRINT ddname

&$DSPRINTVOL

SYSPRINT data set volume

Example: OPT=COPY

```
EXEC $DSCALL OPT=COPY FROMDSN=SYS1.DEV.SAMPLE +
   TODSN=SYS1.WORK.TEST01
```

$DSCALL OPT=COPYPDS

Copies a PDS member from a source data set to a target data set.

```
EXEC $DSCALL OPT=COPYPDS
             FROMDSN=source_dsn TODSN=target_dsn
             MEMBER=member_name
             REPLACE={ YES | NO }
             STAT={ SHR | MOD | OLD }
             SYSPRINT={ KEEP | DELETE | DSN }
             DISPLAY={ YES | NO }
```

This call is used to copy a PDS member from a source data set to a target data set.

Operands:

OPT=COPYPDS

Indicates that a copy of a PDS member is requested.

FROMDSN=*source_dsn*

Specifies the data set which contains the member to be copied.

TODSN=*target_dsn*

Specifies the data set to be copied to.

MEMBER=*member_name*

Specifies the names of the members to be copied. If MEMBER=* is specified, all members in the PDS are copied.

REPLACE={ YES | NO }

Specifies whether to replace an existing data set of the same name and is only applicable where the values of the FROMDSN and TODSN operands are both PDS data sets.

If REPLACE=YES is specified, the member name specified in the FROMDSN operand is copied and given the member name specified in the TODSN operand even if that member name already exists.

If REPLACE=NO is specified, the member specified in the FROMDSN operand is not copied if the member name specified in the TODSN operand already exists.

STAT={ SHR | MOD | OLD }

Specifies whether the data set specified in the TODSN operand is allocated exclusively for the copy operation (STAT=OLD), or is shared with other users (STAT=SHR). It is recommended that STAT=OLD be used as data set integrity cannot be guaranteed if another user attempts to access the data set while the copy operation is in progress.

The data set specified in the FROMDSN operand is always allocated with STAT=SHR.

You can use STAT=MOD to modify a sequential data set by adding new records after the last record in the data set.

SYSPRINT={ DELETE | KEEP | DSN }

Specifies the type of data set to be allocated to the SYSPRINT DD for the copy operation.

- If SYSPRINT=KEEP is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed the SYSOUT data set is deallocated with DISP=KEEP to cause it to be kept. The output from the copy operation is browsed through, for example, SDSF.
- If SYSPRINT=DELETE is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed, the SYSOUT file is deallocated with DISP=DELETE to cause it to be deleted. The output from the copy operation cannot be browsed.
- If SYSPRINT=DSN is specified, your product region allocates a temporary data set, which is deleted when your product region terminates. The output from the copy operation is browsed through ISPF or by the use of the DISPLAY=YES operand on this call. The data set is not deallocated from your product region on completion of the copy operation. The data set name, ddname and volume for the data set are returned in variables to the caller. Removing the data set is the responsibility of the caller-the caller should deallocate the data set to delete it.

DISPLAY= YES | NO

Specifies whether to display the output on completion of the function. This operand is only valid if SYSPRINT=DSN is specified. If DISPLAY=YES is specified, a full-screen display of the SYSPRINT data set is presented after the copy operation has completed.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Examples: OPT=COPYPDS

```
EXEC $DSCALL OPT=COPYPDS FROMDSN=dsn1_name
   TODSN= dsn2_name MEMBER=member_name STAT=SHR
   SYSPRINT=DSN DISPLAY=YES
```

This example copies *member_name* from *dsn1_name* to *dsn2_name* and displays the sysprint output to the screen.

```
EXEC $DSCALL OPT=COPYPDS FROMDSN=dsn1_name
  TODSN=dsn2_name MEMBER=member_name STAT=SHR
  SYSPRINT=DELETE
```

This example copies *member_name* from *dsn1_name* to *dsn2_name* and deletes the sysprint output.

$DSCALL OPT=COPYSEQ

Copies a sequential data set from a source data set to a target data set.

```
EXEC $DSCALL OPT=COPYSEQ
             FROMDSN=source_dsn TODSN=target_dsn
             REPLACE={ YES | NO }
             STAT={ SHR | MOD | OLD }
             SYSPRINT={ KEEP | DELETE | DSN }
             DISPLAY={ YES | NO }
```

This call is used to copy a sequential data set from a source data set to a target data set.

Operands:

OPT=COPYSEQ

Indicates that a copy of a sequential data set is requested.

FROMDSN=*source_dsn*

Specifies the data set which contains the member to be copied.

TODSN=*target_dsn*

Specifies the data set to be copied to.

REPLACE={ YES | NO }

Specifies whether to replace an existing data set of the same name and is only applicable where the values of the FROMDSN and TODSN operands are both PDS data sets.

If REPLACE=YES is specified, the member name specified in the FROMDSN operand is copied and given the member name specified in the TODSN operand even if that member name already exists.

If REPLACE=NO is specified, the member specified in the FROMDSN operand is not copied if the member name specified in the TODSN operand already exists.

STAT={ SHR | MOD | OLD }

Specifies whether the data set specified in the TODSN operand is allocated exclusively for the copy operation (STAT=OLD), or is shared with other users (STAT=SHR). It is recommended that STAT=OLD be used as data set integrity cannot be guaranteed if another user attempts to access the data set while the copy operation is in progress.

The data set specified in the FROMDSN operand is always allocated with STAT=SHR.

You can use STAT=MOD to modify a sequential data set by adding new records after the last record in the data set.

SYSPRINT= DELETE | KEEP | DSN

Specifies the type of data set to be allocated to the SYSPRINT DD for the copy operation.

- If SYSPRINT=KEEP is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed the SYSOUT data set is deallocated with DISP=KEEP to cause it to be kept. The output from the copy operation is browsed through, for example, SDSF.
- If SYSPRINT=DELETE is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed, the SYSOUT file is deallocated with DISP=DELETE to cause it to be deleted. The output from the copy operation cannot be browsed.
- If SYSPRINT=DSN is specified, your product region allocates a temporary data set, which is deleted when your product region terminates. The output from the copy operation is browsed through ISPF or by the use of the DISPLAY=YES operand on this call. The data set is not deallocated from your product region on completion of the copy operation. The data set name, ddname and volume for the data set are returned in variables to the caller. Removing the data set is the responsibility of the caller-the caller should deallocate the data set to delete it.

DISPLAY= YES | NO

Specifies whether to display the output on completion of the function. This operand is only valid if SYSPRINT=DSN is specified. If DISPLAY=YES is specified, a full-screen display of the SYSPRINT data set is presented after the copy operation has completed.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Examples: OPT=COPYSEQ

```
EXEC $DSCALL OPT=COPYSEQ FROMDSN=dsn1_name
    TODSN= dsn2_name STAT=SHR
    SYSPRINT=DSN DISPLAY=YES
```

This example copies *dsn1_name* to *dsn2_name* and displays the sysprint output to the screen.

```
EXEC $DSCALL OPT=COPYPDS FROMDSN=dsn1_name
    TODSN=dsn2_name MEMBER=member_name STAT=SHR
    SYSPRINT=DELETE
```

This example copies *dsn1_name* to *dsn2_name* and deletes the sysprint output.

$DSCALL OPT=CREATE

Creates a new data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=CREATE
             DSN=dataset_name
             FORMAT=format
             ORG=dataset_organization
             RLSE={ NO | YES }
             SPACE= { TRK | CYL } , pri, sec [ ,dir ]
           [ BLKSIZE=blocksize ]
           [ DATACLS=class ]
           [ FREE= { UNAL | CLOSE } ]
           [ LRECL= logical_record_length ]
           [ MGMTCLS=class ]
           [ MOUNT= { NO | YES } ]
           [ STORCLS=class ]
           [ VOL=volser [ UNIT=unit ] ]
```

This call is used to create a new data set on DASD. This call is similar to the $DSCALL OPT=ALLOC call except the data set must not be a temporary or SYSOUT data set. The data set is cataloged and does not remain allocated to your product region on completion of the operation.

Operands:

OPT=CREATE

Specifies that a data set is to be created.

DSN=*dataset_name*

Specifies the data set name to create.

FORMAT=*format*

Specifies the format of the created data set. Valid values are: F, FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM VBM, VSM, VBSM, U, UA, UM.

ORG=*dataset_organization*

Specifies the data set organization for a new data set. Valid values are: PS, PO, PSU, and POU.

RLSE={ NO | YES }

Specifies whether to have a secondary space release option.

SPACE={ TRK | CYL } , *pri*, *sec* [,*dir*]

Specifies the space allocation in tracks or cylinders. The SPACE type and primary allocation quantity are required. The directory allocation quantity is required if the data set is a PDS.

BLKSIZE=*blocksize*

Specifies the block size for a new data set. The value must be an integer in the range 1 to 32,760.

DATACLS=*class*

Specifies the SMS data class option.

FREE={ UNAL | CLOSE }

Specifies whether the data set is to be deallocated by explicit request only (FREE=UNAL) or is to be deallocated when the file is closed (FREE=CLOSE).

LRECL=*logical_record_length*

Specifies the logical record length for a new data set. The value must be an integer in the range 1 to 32,760.

MGMTCLS=*class*

Specifies the SMS management class option.

MOUNT={ NO | YES }

Specifies whether a volume mount is allowed. If MOUNT=NO is specified, and an allocation request requires a volume which is offline, the allocation request fails. If MOUNT=YES is specified and an allocation request requires a volume which is offline, a MOUNT request for the volume is issued.

STORCLS=*class*

Specifies the SMS storage class option.

VOL=*volser* [UNIT=*unit*]

Specifies the volume serial number and unit name for a new allocation. If the VOL operand is omitted, the operating system determines if the allocation is allowed, and may choose to allocate the data set or any volume which the requesting user is authorized for.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSDSN

Full data set name with member name omitted, and true name if an alias was entered

&$DSORG

Data set organization; values are PS, PO, PSU, POU, CX, CQ, MQ, GS, X, TQ, and TR

&$DSVOL

First volume

&$DSRECF

Record format; values are F,FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM, VBM, VSM, VBSM, U, UA, UM

&$DSRECL

Logical record length

&$DSRECA

Actual usable record length. If the record format is variable length, then the value is record length minus 4. In all other cases, the value is the same as record length.

&$DSBLKS

Block size

&$DSSTORCLS

SMS storage class if SMS is active for this data set

&$DSMGMTCLS

SMS management class if SMS is active for this data set

&$DSDATACLS

SMS data class if SMS is active for this data set

&$DSDYNEC

Dynamic allocation error code

&$DSDYNIC

Dynamic allocation information code

Example: OPT=CREATE

```
EXEC $DSCALL OPT=CREATE DSN=DEV.SAMPLE.PDS.WORK02 +
   SPACE='TRK,10,5,1' ORG=PO FORMAT=FB +
   LRECL=80, BLKSIZE=4000, STAT=NEW
```

$DSCALL OPT=DECONCAT

Deconcatenates a DD that is allocated to your product region.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=DECONCAT
             DD=DD_name
```

This call is used to deconcatenate a DD into its original DD allocations. The DD can represent a single data set allocation, or be a concatenation of two or more allocated data sets. Deconcatenation results in each data set in the concatenation returning to its original allocation state, where the ddname each data set was assigned when allocated is available. DDs which are permanently concatenated remain concatenated.

Operands:

OPT=DECONCAT

Specifies that a DD is to be deconcatenated.

DD=DD_*name*

Specifies the DD to be deconcatenated.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSDYNEC

DYNALLOC error code

&$DSDYNIC

DYNALLOC information code

Example: OPT=DECONCAT

```
EXEC $DSCALL OPT=DECONCAT DD=DD1
```

This example deconcatenates DD1. If the DD consists of data sets which were originally allocated under the ddnames of DD1 and DD2 (as in the example for $DSCALL OPT=CONCAT) the deconcatenation of DD1 results in DD1 and DD2 being returned to their original state.

$DSCALL OPT=DELETE

Deletes a cataloged data set.

```
&CONTROL SHRVARS=($DS)

EXEC $DSCALL    OPT=DELETE
                DSN=dataset_name
                CONFIRM = { YES | NO }
```

This call is used to delete a cataloged data set. The data set is allocated to your product region, then deleted and uncataloged. This call cannot be used to delete a PDS member. To delete a specific member, use the $DSCALL OPT=DELMEM call.

Operands:

OPT=DELETE

Specifies that the data set be deleted.

DSN=*dataset_name*

Specifies the name of the data set to be deleted. No member name is specified.

CONFIRM={ YES|NO }

Specifies whether to display a confirmation query before deleting the data set. If CONFIRM=YES is specified (or left to default) a full-screen panel is displayed to request confirmation of the DELETE action. The DELETE action is confirmed or canceled.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=DELETE

```
EXEC $DSCALL OPT=DELETE DSN=SYS.WORK.DEV01 CONFIRM=NO
```

$DSCALL OPT=DELMEM

Deletes a member of a PDS.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=DELMEM
             DSN=dataset_name
             MEMBER=member_name
             CONFIRM={ YES | NO }
```

This call is used to delete a member of a PDS. This call fails if the specified member is in use by another user.

Operands:

OPT=DELMEM

Specifies that a member be deleted.

DSN=*dataset_name*

Specifies the name of the data set containing the member. No member name is specified.

MEMBER=*member_name*

Specifies the member to be deleted. A member name must be specified.

CONFIRM={ YES | NO }

Specifies whether to display a confirmation query before deleting the member. If CONFIRM=YES is specified (or left to default) a full-screen panel is displayed to request confirmation of the DELMEM action. The DELMEM action is confirmed or canceled.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=DELMEM

```
EXEC $DSCALL OPT=DELMEM DSN=CUSTOMER.DATA MEMBER=TEST01
```

$DSCALL OPT=DEQ

Releases an ENQ on a data set or member of a PDS.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL     OPT=DEQ
                 { DSN= dataset_name | DD=DD_name }
                 [ MEMBER=member_name ]
```

This call releases an ENQ on a data set or a member of a PDS which was obtained using a $DSCALL OPT=ENQ call. The operands specified on this call should specify the same values as the OPT=ENQ call specified when the ENQ was obtained. After the DEQ request has completed, the data set or PDS member is edited by another user.

Operands:

OPT=DEQ

Specifies that an ENQ is to be released.

DSN=*dataset_name*

Specifies the data set name.

DD=DD_*name*

Specifies the ddname under which the data set is allocated to your product region. If the ddname represents a concatenated DD, the name of the first data set in the concatenation is used.

MEMBER=*member_name*

Specifies the member name where the release of the ENQ is for a specific member of a PDS.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=DEQ

```
EXEC $DSCALL OPT=DEQ DD=SYSDD
```

$DSCALL OPT=DSNLIST

Provides in variables the number and name of data sets concatenated to a ddname.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=DSNLIST
             DD=dd_name
```

Used to provide a list of all the data sets concatenated to a ddname.

Operands:

OPT=DSNLIST

Indicates that a list of data sets concatenated to a ddname is requested.

DD=DD_*name*

Specifies the ddname under which the data sets are allocated to your product region.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSCONCAT#

Number of data sets returned

&$DSDSN1-*n*

Name of the data set found in the concatenation

Example: OPT=DSNLIST

```
EXEC $DSCALL OPT=DSNLIST DD=COMMANDS
```

$DSCALL OPT=DSNSPACE

Computes the number of allocated tracks in a data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL     OPT=DSNSPACE
                 DSN=dataset_name
```

Used to find how much space is allocated to a data set.

Operands:

OPT=DSNSPACE

Indicates that the number of allocated tracks on the specified data set are to be found.

DSN=*dataset_name*

The name of the data set for which the number of allocated tracks will be found.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSDYNEC

DYNALLOC error code.

&$DSALCTRKS

The number of allocated tracks.

&$DSUNIT

Unit of the volume.

&$DSORG

The organization of the data set.

Example: OPT=DSNSPACE

```
EXEC $DSCALL OPT=DSNSPACE DSN=SYS1.DEV.SAMPLE
```

$DSCALL OPT=EDIT

Displays a sequential data set or member of a PDS for editing.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL    OPT=EDIT
                DD=DD_name | DSD=dataset_name
                [MEMBER=member_name]
                DATA={ }
                [LIMIT=number]
                [TRUNCATE=number]
```

Use this option to display and edit the records in a member of a data set.

Operands:

OPT=EDIT

Specifies that an EDIT is to be performed.

DD=DD_*name* *or* DSD=*dataset_name*

Specifies the ddname or data set name of the data set to be opened for editing.

MEMBER=*member_name*

Specifies the member name. If the DD operand is used to identify a concatenated data set, the member is obtained from the first data set that it exists in.

DATA={ }

Specifies the records to be edited.

LIMIT=*number*

Specifies the maximum number of records to be edited in the file. This operand is only applicable if DATA=* is specified.

TRUNCATE=*number*

Specifies the length to which the records displayed from the file are truncated. The value of the TRUNCATE operand must be in the range 1 to 32,760.

$DSCALL OPT=ENQ

Performs an ENQ on a data set or member.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL     OPT=ENQ
                 { DSN=dataset_name | DD=DD_name }
                 [ MEMBER=member_name ]
```

This call is used to serialize access to a data set or member of a PDS in the same way ISPF does. It is used to prevent another user editing the data set or member while it is being used by your product region.

Operands:

OPT=ENQ

Specifies that an ENQ be issued.

DSN=*dataset_name*

Specifies the data set name to issue the ENQ for. If the data set is a PDS, a member name may be specified using the MEMBER operand.

DD=DD_*name*

Specifies the ddname of the data set to issue the ENQ for. If the ddname represents a concatenated DD, the ENQ is issued for the first data set in the concatenation. If the data set is a PDS, a member name may specified using the MEMBER operand.

MEMBER=*member_name*

Specifies a member name for the ENQ but is only valid if the ENQ is for a PDS.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=ENQ

```
EXEC $DSCALL   DD=DDSYS1
```

Note: Issue an ENQ before opening a data set when you intend to update the data set. The ENQ option does not protect the data set from simultaneous physical writing, however an internal ENQ is issued on the OPEN to protect the physical data set. If the ENQ function is not explicitly released, it is released when the owning NCL process terminates.

$DSCALL OPT=FCLOSE

Combines the deallocating and closing data set functions into a single call.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL     OPT=FCLOSE
                 { DD=DD_name | ID=path_name }
                 DISP={ KEEP | DELETE }
```

This call is used as a more simple interface than the combination of OPT=CLOSE and OPT=UNALL. It allows a single call to close and deallocate a data set.

Operands:

OPT=FCLOSE

Specifies that a data set is to be closed and deallocated.

DD=DD_*name*

Specifies the ddname to be closed and deallocated.

ID=*path_name*

Specifies the path name that allows multiple paths to the same data set. If no path ID is specified, the ddname is used as the path ID.

You must specify a ddname. If the ddname specified is other than ID, you also need to specify the ID parameter; otherwise you do not.

Note: Any combination of OPEN, FOPEN, CLOSE, FCLOSE, READ, and WRITE requests must be done in a single procedure.

DISP={ KEEP | DELETE }

Specifies the override of the Normal Disposition that was specified when the data set was allocated to your product region.

Note: If the data set name was generated by the system, then it is deleted, regardless of the disposition specified.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSDYNEC

DYNALLOC error code

&$DSDYNIC

DYNALLOC information code

Example: OPT=FCLOSE

```
EXEC $DSCALL OPT=FCLOSE DD=&Q$DD
```

Notes:

For an OUTPUT data set, the final block is written. This may cause an ABEND message if the data set exceeds its size limit.

A temporary data set, whose name was generated at allocation, is deleted when it is deallocated-regardless of the disposition value that was specified.

If a concatenated data set is freed, the first data set in the concatenation is deallocated and the other ddnames in the concatenation become visible. The net effect is the same as a deconcatenation followed by a deallocation of the first ddname in the concatenation.

$DSCALL OPT=FINDMEM

Provides a selection list of all data sets in the DD_name concatenation and an entry to indicate if the member name is found in this data set or not.

```
EXEC $DSCALL OPT=FINDMEM
             DD=DD_name
             MEMBER=member_name
```

This call is used to show which data sets contain the specified member name from a ddname.

Operands:

OPT=FINDMEM

Finds a member from a concatenated list of data sets.

DD=DD_*name*

Specifies the ddname. If this operand is omitted, the COMMANDS ddname is used.

MEMBER=*member_name*

Specifies the member name.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=FINDMEM

```
EXEC $DSCALL OPT=FINDMEM DD=COMMANDS MEMBER=proc_name
```

This example produces results that indicate that *proc_name* was found in the data sets SYSA.NM.PROD.SOURCE and SYSA.NM.BASE.SOURCE.

$DSCALL OPT=FOPEN

Combines the ALLOC option for new or existing data sets and the OPEN option into one call.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=FOPEN
             DD=DD_name [ ID=path_name ]
             DSN=dataset_name
             FORMAT=dataset_format
             ORG=dataset_organization
             RLSE= { NO | YES }
             SPACE= { TRK | CYL } , pri ,sec [ ,dir ]
           [ BLKSIZE=blocksize ]
           [ DATACLS=class ]
           [ DISP= { KEEP | DELETE } [ , { KEEP | DELETE } ] ]
           [ DSNTYPE= { LIBRARY | PDSE } ]
           [ FREE= { UNAL | CLOSE } ]
           [ LRECL=logical_record_length ]
           [ MEMBER=member_name ]
           [ MGMTCLS=class ]
           [ MIGRATE= { YES | NO } ]
           [ MODE= { INPUT | OUTPUT } ]
           [ MOUNT= { NO | YES } ]
           [ STAT= {  NEW | OLD | MOD | SHR } ]
           [ STORCLS=class ]
           [ VOL=volser [ UNIT=unit ] ]
```

This call is used as a more simple interface than the combination of OPT=ALLOC and OPT=OPEN. It allows a single call to be made to allocate and open a new or existing cataloged data set.

Operands:

OPT=FOPEN

Specifies that a data set is to be allocated and opened.

DD=*DD_name*

Specifies the ddname. If this operand is omitted, a ddname is generated by the operating system.

ID=*path_name*

Specifies the ID of a path to be opened for access to the data set. The use of a path ID lets a procedure access more than one file, with the unique path ID used to identify which file is being accessed. The path ID must be specified on subsequent requests such as READ, WRITE, and CLOSE. If this operand is omitted, the ddname specified in the DD operand is used as the path ID.

DSN=*dataset_name*

Specifies the data set name to allocate and open. If the DSN operand is omitted, a temporary data set is allocated. When a temporary data set is allocated, no disposition is allowed and DELETE, DELETE is forced.

FORMAT=*dataset_format*

Specifies the data set format for the new data set. Valid values are: F, FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM VBM, VSM, VBSM, U, UA, UM.

ORG=*dataset_organization*

Specifies the data set organization. Valid values are: PS, PO, PSU, POU, CX, CQ, MQ, GS, TX, TQ, TR.

RLSE={ NO | YES }

Specifies whether to use the secondary space release option.

SPACE={ TRK | CYL } , *pri,sec* { ,*dir* }

Specifies the space allocation in tracks or cylinders. The SPACE type and primary allocation quantity are required. The directory allocation quantity is required if the data set is a PDS.

BLKSIZE=*blocksize*

Specifies the block size for the data set. The value must be an integer in the range 0 to 32,760.

DATACLS=*class*

Specifies the SMS data class.

DISP={ KEEP | DELETE } [, { KEEP | DELETE }]

Specifies the Normal and Conditional Disposition of the data set. The disposition relates to the normal or conditional termination of your product region, not of the requesting NCL process. If KEEP is specified, the data set remains cataloged. If DELETE is specified, the data set is uncataloged and deleted.

DSNTYPE={ LIBRARY | PDSE }

Specifies whether you want to allocate and open a PDSE data set. The two values mean the same thing: a PDSE data set.

MIGRATE={ YES | NO }

Specifies whether to allocate a data set if it has been migrated. If MIGRATE=NO is specified, Dataset Services checks that the data set has been migrated (for example, by DFHSM). If the data set has been migrated, Dataset Services rejects the allocation request. If MIGRATE=YES is specified, Dataset Services issues the allocation request without checking that the data set has been migrated.

MODE={ INPUT | OUTPUT }

Specifies whether the data set is open for input or output.

MOUNT={ NO | YES }

Specifies whether a volume mount is allowed. If MOUNT=NO is specified, and an allocation request requires a volume which is off-line, the allocation request fails. If MOUNT=YES is specified and an allocation request requires a volume which is off-line, a MOUNT request for the volume is issued.

FREE={ UNAL | CLOSE }

Specifies whether the data set is to be deallocated by explicit request only (FREE=UNAL) or is to be deallocated when the file is closed (FREE=CLOSE).

LRECL=*logical_record_length*

Specifies the logical record length for the data set. The value must be an integer in the range 1 to 32,760.

MEMBER=*member_name*

Specifies the member name. To open a PDS that was not allocated with a member name, specify the member name in the call. If the allocation was to a member name, the member name specified in the MEMBER operand overrides it.

MGMTCLS=*class*

Specifies the SMS management class.

STAT={ NEW | OLD | MOD | SHR }

Specifies the status of the data set.

STORCLS=*class*

Specifies the SMS storage class.

VOL=*volser* [UNIT=*unit*]

Specifies the volume serial number and unit name for the new allocation. If the VOL operand is omitted, the operating system determines if the allocation is allowed, and may choose to allocate the data set or any volume which the requesting user is authorized for.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSSTAT

Status of data set; values are OLD, MOD, or SHR

&$DSDSN

Full data set name with member name omitted, and true name if an alias was entered &$DSDSNTYPE Type of data set; value is LIBRARY

&$DSMEM

Member name if a member of a PDS &$DSID Path ID

&$DSORG

Data set organization; values are PS, PO, PSU, and POU

&$DSVOL

First volume

&$DSUNIT

Unit name for the volume

&$DSRECF

Record format; values are F, FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM, VBM, VSM, VBSM, U, UA, UM

&$DSRECL

Record length

&$DSRECA

Actual usable record length. If the record format is variable length, then the value is record length minus 4. In all other cases, the value is the same as record length.

&$DSBLKS

Block size

&$DSDDNAME

DDname from the allocation, as supplied or as generated by the operating system

&$DSSTORCLS

SMS storage class if SMS is active for this data set

&$DSMGMTCLS

SMS management class if SMS is active for this data set

&$DSDATACLS

SMS data class if SMS is active for this data set

&$DSDYNEC

DYNALLOC error code

&$DSDYNIC

DYNALLOC information code

Example: OPT=FOPEN

```
EXEC $DSCALL OPT=FOPEN DSN=SYS.WORK.DEV01 MEMBER=TEST01 +
   STAT=OLD MODE=OUTPUT
```

Notes:

Only cataloged data sets are supported. Unit and volume parameters are not supported by this function.

Most non-VSAM data sets are allocated. IS and DA data sets are not supported. If MEMBER= was specified, the member is opened. If the allocation was for a sequential data set, the data set is opened.

$DSCALL OPT=INFO

Retrieves data set information.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=INFO
             DSN=dataset_name [ DD=DD_name ]
           [ MOUNT= { NO | YES } ]
           [ MIGRATE= { YES | NO } ]
```

This call is used to obtain data set information for a data set. The information is returned to the calling procedure in variables.

Operands:

OPT=INFO

Specifies the data set information that is to be returned.

DSN=*dataset_name*

The name of the data set for which information is to be retrieved.

DD=DD_*name*

Specifies the ddname of a data set allocated to your product region for which information is to be retrieved.

MOUNT={ NO | YES }

Specifies whether a volume mount is allowed. If MOUNT=NO is specified, and an allocation request requires a volume which is off-line, the allocation request fails. If MOUNT=YES is specified and an allocation request requires a volume which is off-line, a MOUNT request for the volume is issued. You cannot use this operand if DD= is specified.

MIGRATE={ YES | NO }

Specifies whether to allocate a data set if it has been migrated. If MIGRATE=NO is specified, Dataset Services checks that the data set has been migrated (for example, by DFHSM). If the data set has been migrated, Dataset Services rejects the allocation request. If MIGRATE=YES is specified, Dataset Services issues the allocation request without checking that the data set has been migrated.

Note: If you specify a ddname, the MIGRATE and MOUNT parameters are ignored.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSSTAT

Status of data set; values is OLD, MOD, SHR, or NEW

&$DSDSN

Full data set name with member name omitted, and true name if an alias was entered

&$DSDSNTYPE

Type of file or data set; value is HFS or LIBRARY

&$DSMEM

Member name if a member of a PDS

&$DSORG

Data set organization; values are PS, PO, PSU, POU, CX, CQ, MQ, GS, TX, TQ, TR, and VS

&$DSVOL

First volume &$DSUNIT Unit name for the volume

&$DSRECF

Record format; values are F, FB, FS, FBS, FA, FBA, FSA, FBSA, FM, FBM, FSM, FBSM, V, VB, VS, VBS, VBSA, VM, VBM, VSM, VBSM, U, UA, UM

&$DSRECL

Record length

&$DSRECA

Actual usable record length. If the record format is variable length, then the value is record length minus 4. In all other cases, the value is the same as record length.

&$DSBLKS

Block size

&$DSDDNAME

DDname from the allocation, as supplied or as generated by the operating System

&$DSNUMEXT

Number of extents

&$DSCDATE

Create date

&$DSEDATE

Expire date

&$DSLREF

Last referenced date

&$DSTPCYL

Tracks per cylinder

&$DSPRITRKS

Primary allocation in tracks

&$DSALCTRKS

Allocated tracks for data set

&$DSUSETRKS

Used tracks

&$DSSECQTY

Secondary space allocation

Examples: OPT=INFO

```
EXEC $DSCALL OPT=INFO DSN=SYS1.LINKLIB MOUNT=YES
```

Note: Only cataloged data sets are supported. Unit and volume parameters are not supported by this function.

$DSCALL OPT= LISTC

Obtains or displays a list of data sets that begin with a specified name prefix.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT= LISTC QUAL=qualifier
           [ SYSPRINT= { KEEP | DSN | DELETE } ]
           [ DISPLAY= { YES | NO } ]
           [ PARM={ NAME | VOLUME | ALL } ]
```

This call allows users of your product region to perform an IDCAMS LISTCAT operation and receive the results. This is used to obtain a list of cataloged data sets which have a specified name prefix.

Operands:

OPT=LISTC

Specifies that a list of data sets be obtained or displayed.

QUAL=*qualifier*

The fully qualified name prefix for the list of data sets to be displayed.

SYSPRINT={ KEEP | DSN | DELETE }

Specifies the type of data set to be allocated to the SYSPRINT DD for the copy operation.

- If SYSPRINT=KEEP is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed the SYSOUT data set is deallocated with DISP=KEEP to cause it to be kept. The output from the copy operation is browsed through, for example, SDSF.
- If SYSPRINT=DSN is specified, your product region allocates a temporary data set, which is deleted when your product region terminates. The output from the copy operation is browsed through ISPF or by the use of the DISPLAY=YES operand on this call. The data set is not deallocated from your product region on completion of the copy operation. The data set name, ddname, and volume for the data set are returned in variables to the caller.
- If SYSPRINT=DELETE is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed, the SYSOUT file is deallocated with DISP=DELETE to cause it to be deleted. The output from the copy operation cannot be browsed.

DISPLAY={ YES | NO }

Specifies whether to display the SYSPRINT output. The display parameter is only valid if SYSPRINT=DSN.

PARM={ NAME | VOLUME | ALL }

Specifies the level of detail of the returned catalog information. The value is any valid parameter used in the IBM LISTCAT utility after the LEVEL parameter.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSPRINTDSN

Name of SYSPRINT file

&$DSPRINTVOL

Volume where SYSPRINT file is located

&$DSPRINTDD

SYSPRINT ddname

Examples: OPT=LISTC

When $DSCALL is called with the LISTC option and no PARM specified:

```
EXEC $DSCALL OPT=LISTC QUAL=SYS1 DISPLAY=YES SYSPRINT=DSN
```

it produces the following output:

```
USER01---------- SYS95229.T172448.RA000.PROD1.R0000102 ----Columns 001 079
Command ===>                                                Scroll ===> PAGE

****************************** TOP OF DATA ******************************
1IDCAMS SYSTEM SERVICES                                        TIME: 17:24:
0
    LISTCAT LEVEL(SYS1) NAME
0NONVSAM ------- SYS1.AADFMAC1
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSCLI0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSKEL0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSMSG0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSPNL0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSTBL0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABMFMOD0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABNJMISC
 F1=Help      F2=Split     F3=Exit      F4=Return    F5=Find
 F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right
```

When $DSCALL is called with the LISTC option and PARM=ALL specified:

```
EXEC $DSCALL OPT=LISTC QUAL=PROD.V3R2M0 DISPLAY=YES PARM=ALL
+       SYSPRINT=DSN
```

it produces the following output:

```
USER01---------- SYS95229.T172834.RA000.PROD1.R0000106 ----Columns 001 079
Command ===>                                                Scroll ===> PAGE

****************************** TOP OF DATA ********************************
1IDCAMS SYSTEM SERVICES                                      TIME: 17:28:
0
    LISTCAT LEVEL(SYS1) ALL
0NONVSAM ------- SYS1.AADFMAC1
      IN-CAT --- CATALOG.MCAT.VMVS006
      HISTORY
        DATASET-OWNER-----(NULL)     CREATION--------1991.310
        RELEASE----------------2     EXPIRATION------0000.000
      VOLUMES
        VOLSER------------MVS005     DEVTYPE------X'3010200E'    FSEQN----
      ASSOCIATIONS
        ALIAS----TARGSYS.SYS1.AADFMAC1
0NONVSAM ------- SYS1.ABLSCLI0
      IN-CAT --- CATALOG.MCAT.VMVS006
      HISTORY
        DATASET-OWNER-----(NULL)     CREATION--------1991.310
        RELEASE----------------2     EXPIRATION------0000.000
      VOLUMES
 F1=Help      F2=Split     F3=Exit      F4=Return    F5=Find
 F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right
```

When $DSCALL is called with the LISTC option and PARM=VOLUME specified:

```
EXEC $DSCALL OPT=LISTC QUAL=SYS1 DISPLAY=YES PARM=VOLUME +
      SYSPRINT=DSN
```

it produces the following output:

```
USER01--------- SYS98265.T154724.RA000.PROD41.R0105116 -----Columns 001 079
Command ===>                                                 Scroll ===> PAGE

****************************** TOP OF DATA ********************************
1IDCAMS SYSTEM SERVICES TIME: 17:30:
0
0NONVSAM ------- SYS1.AACBCNTL
      IN-CAT --- CATALOG.VOS3SCT
      HISTORY
        DATASET-OWNER-----(NULL)     CREATION--------1997.167
        RELEASE----------------2     EXPIRATION------0000.000
      VOLUMES
        VOLSER------------OS3SDL     DEVTYPE------X'3010200F'    FSEQN----
0NONVSAM ------- SYS1.AADFMAC1
      IN-CAT --- CATALOG.VOS3SCT
      HISTORY
        DATASET-OWNER-----(NULL)     CREATION--------1997.167
        RELEASE----------------2     EXPIRATION------0000.000
      VOLUMES
        VOLSER------------OS3SDL     DEVTYPE------X'3010200F'    FSEQN----
0NONVSAM ------- SYS1.AADRLIB
 F1=Help      F2=Split     F3=Exit      F4=Return    F5=Find
 F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right
```

When $DSCALL is called with the LISTC option and PARM=VOLUME specified:

```
EXEC $DSCALL OPT=LISTC QUAL=SYS1 DISPLAY=YES PARM=VOLUME +
      SYSPRINT=DSN
```

it produces the following output:

```
USER01---------- SYS95229.T172448.RA000.PROD1.R0000102 ----Columns 001 079
Command ===>                                                Scroll ===> PAGE

****************************** TOP OF DATA ********************************
1IDCAMS SYSTEM SERVICES                                      TIME: 17:24:
0
    LISTCAT LEVEL(SYS1) NAME
0NONVSAM ------- SYS1.AADFMAC1
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSCLI0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSKEL0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSMSG0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSPNL0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABLSTBL0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABMFMOD0
      IN-CAT --- CATALOG.MCAT.VMVS006
0NONVSAM ------- SYS1.ABNJMISC
 F1=Help      F2=Split     F3=Exit      F4=Return    F5=Find
 F7=Backward  F8=Forward   F9=Swap     F10=Left     F11=Right
```

Note: This is the same as the output produced when $DSCALL is called with the LISTC option and no PARM specified (see the first example).

$DSCALL OPT=MEMLIST

Obtains or displays a list of PDS members.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=MEMLIST DSN=dataset_name
           [ DISPLAY={ YES | NO } ]
           [ MEMBER=pattern ]
```

This call is used to obtain a list of members in a PDS and return information about the members to the calling procedure in variables. It is also used to obtain a full-screen selection list of members in a PDS.

Operands:

OPT=MEMLIST

Specifies that information about the members be obtained.

DSN=*dataset_name*

Specifies the name of the PDS containing the members. No member name is specified.

DISPLAY={YES | NO }

Specifies whether to display the selection list of PDS member names. When DISPLAY=NO is specified, the member information is returned in variables &$DSMEM#, &$DSMEM*n*, and &$DSSTAT*n*.

MEMBER=*pattern*

Specifies the name of a member pattern which specifies a subset of the members in the PDS. If not specified all members are searched. You can specify a particular pattern name, or use a mask to indicate more than one pattern name.

A pattern can contain the question mark (?) and asterisk (*) characters. The ? character is used to match any single character in more than one member name. The * character is used to match any number of characters in more than one member name. For example:

ABCD

matches only the pattern name ABCD

A?D

matches any three character pattern name that begins with A and ends with D, such as, AID, AFD, AZD

A*D

matches member names of any length that begin with A and ends with D, such as, AD, ABCD, ABD

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSMEM#

Number of members returned

&$DSMEM*n*

Member names in ascending order

&$DSSTAT*n*

SPF statistics for the associated member name

&$DSLIB*n*

The concatenation level of the data set where the member resides

Examples:

```
EXEC $DSCALL OPT=MEMLIST DSN=SYS.WORK.DEV01 DISPLAY=YES
```

The selection list displays ISPF member information as shown in the following figure. The BROWSE, EDIT, DELETE, PRINT, RENAME, and SUBMIT actions can be applied to the members on the list.

This example produces the following output:

```
PROD----------- Dataset Services : PDS Member SPF Statistics ------------------
Command ===>                                                  Scroll ===> CSR

                          B/=Browse E=Edit D=Delete P=Print R=Rename SUB=Submit
    Member   VV.MM    Created        Changed      Size  Init   Mod ID       Lib
    $$$AW    01.05  11-MAY-2007 14-MAY-2007 20:41  418   378     0 SYSIP01    1
    $$$AW1   01.07  11-MAY-2007 15-MAY-2007 02:29  531   378     0 SYSIP01    1
    $$$AW3   01.03  15-MAY-2007 15-MAY-2007 21:48  570   570     0 SYSIP01    1
    $$$SPACE
    $$COPYRI 01.00  26-JAN-2008 26-JAN-2008 22:16    0     0     0 SYSIP02    7
    $$NMLIC  01.00  26-JAN-2008 26-JAN-2008 22:16    0     0     0 SYSIP02    7
    $$QASMSG 01.01  23-AUG-1995 23-AUG-1995 18:44    0     0     0 SYSIP14    8
    $$SL     01.00  11-FEB-2008 11-FEB-2008 19:27    0     0     0 SYSIP02    6
    $$SYSPRD 01.00  18-MAR-2008 18-MAR-2008 00:04    0     0     0 SYSIP02    3
    $$SYSPRO 01.00  11-FEB-2008 11-FEB-2008 19:27    0     0     0 SYSIP02    6
    $@COPY   01.02  04-MAY-1992 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $@DELETE 01.02  04-MAY-1992 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $@E      01.02  04-MAY-1992 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $@L      01.02  04-MAY-1992 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $@M      01.02  04-MAY-1992 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $@NMFTS  01.02  04-MAY-1992 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $@NOTIFY 01.02  04-MAY-1992 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $@R      01.02  04-MAY-1992 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $@REPRO  01.03  05-JUN-1994 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $@S      01.02  04-MAY-1992 17-JUL-2004 05:44    0     0     0 SYSIP02    8
    $ACBERFL 01.01  04-MAY-1992 04-MAY-1992 20:36    0     0     0 SYSIP24    8
```

$DSCALL OPT=MOVE

Copies a sequential data set, then deletes the original data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL     OPT=MOVE
                 FROMDSN=dataset_name
                 TODSN=dataset_name
```

This call is used to copy a sequential data set to another sequential data set using the IEBGENER utility and delete the original data set after the copy has completed.

Operands:

OPT=MOVE

Specifies that the data set be copied then deleted.

FROMDSN=*dataset_name*

Specifies the name of the data set to be moved.

TODSN=*dataset_name*

Specifies the name of the data set to copy the data set specified in FROMDSN to.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=MOVE

```
EXEC $DSCALL OPT=MOVE FROMDSN=SYS1.WORK.GO +
   TODSN=PROD1.CHANGE.GO
```

Note: Both the FROMDSN and TODSN data sets must be sequential and have the same record length and record format. Both data sets must exist prior to the call. If the copy function fails the FROMDSN data set is not deleted. This does not apply to PDSs.

$DSCALL OPT=MOVEPACK

Moves a data set to another disk pack.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL     OPT=MOVEPACK
                 DSN=dataset_name VOL=volume_name
                 SYSPRINT={ KEEP | DELETE | DSN }
                 DISPLAY={ YES | NO }
```

Use this call to move a data set to a specified disk pack. This option is only useful for moving cataloged data sets.

The new data set is cataloged, with the same name, on the specified target disk pack-the data set name cannot be changed. If the target volume is the same as the source disk pack, no action is taken.

For a VSAM data set the IDCAMS utility is used to perform the move. The VSAM data set is exported and then imported on the target volume. The IEHMOVE utility is used for non-VSAM data sets. The data set is moved to the target volume and deleted from the current volume by the IEHMOVE utility. The utility catalogs the data set on the target volume.

Operands:

OPT=MOVEPACK

Indicates that a data set is to be moved between disk packs.

DSN=*dataset_name*

The name of the PDS or sequential data set to be moved.

VOL=*volume_name*

The name of the disk pack to which the data set will be moved.

SYSPRINT={ KEEP | DELETE | DSN }

Specifies the type of data set to be allocated to the SYSPRINT DD for the copy operation.

- If SYSPRINT=KEEP is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed the SYSOUT data set is deallocated with DISP=KEEP to cause it to be kept. The output from the copy operation is browsed through, for example, SDSF.
- If SYSPRINT=DELETE is specified, your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed, the SYSOUT file is deallocated with DISP=DELETE to cause it to be deleted. The output from the copy operation cannot be browsed.
- If SYSPRINT=DSN is specified, your product region allocates a temporary data set, which is deleted when your product region terminates. The output from the copy operation is browsed through ISPF or by the use of the DISPLAY=YES operand on this call. The data set is not deallocated from your product region on completion of the copy operation. The data set name, ddname, and volume for the data set are returned in variables to the caller.

DISPLAY={ YES | NO }

Specifies whether to display the output on completion of the function. This operand is only valid if SYSPRINT=DSN is specified. If DISPLAY=YES is specified, a full-screen display of the SYSPRINT data set is presented after the copy operation has completed.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=MOVEPACK

```
EXEC $DSCALL OPT=MOVEPACK DSN=MYDATASET VOL=MVS009
```

$DSCALL OPT=OPEN

Opens a sequential data set or member of a PDS for input or output.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=OPEN
             DD=DD_name
           [ ID=path_name ]
           [ MEMBER=member_name ]
           [ MODE= { INPUT | OUTPUT } ]
```

This call is used to open a data set or a member of a PDS in preparation for reading (specify MODE=INPUT) or writing (specify MODE=OUTPUT).

Operands:

OPT=OPEN

Specifies that an OPEN action be performed on a data set or member of a PDS.

DD=DD_*name*

Specifies the ddname of the data set to be opened. If the ddname represents a concatenated DD, the first data set in the concatenation is opened.

ID=*path_name*

Specifies the ID of a path to be opened for access to the data set. The use of a path ID lets a procedure access more than one file, with the unique path ID used to identify which file is being accessed. The path ID must be specified on subsequent requests such as READ, WRITE, and CLOSE.

If this operand is omitted, the ddname is used as the path ID. If a path with that name already exists, an error is indicated. The path ID is returned in the shared variable &$DSID.

MEMBER=*member_name*

Specifies the member name. To open a PDS that was not allocated with a member name, specify the member name in the call. If the allocation was to a member name, the member name specified in the MEMBER operand overrides it.

MODE={ INPUT | OUTPUT }

Specifies whether the data set is open for input or output.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSID

Path ID

Example: OPT=OPEN

```
EXEC $DSCALL OPT=OPEN DD=SYSDD MODE=OUTPUT +
   MEMBER=TEST01
```

Note: If the original allocation was to a PDS or a member of a PDS, the member is opened. If the allocation was for a sequential data set, the data set is opened.

$DSCALL OPT=PRINT

Prints one or more records from a data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=PRINT DD=DD_name
           [ MEMBER=member_name ]
             DATA={ }
           [ LIMIT=number ]
           [ TRUNCATE=number ]
```

Use this option to print the records in a member of a data set.

Operands:

OPT=PRINT

Specifies that a print action is to be performed.

DD=DD_*name*

Specifies the ddname of the data set to be opened for printing.

MEMBER=*member_name*

Specifies the member name. If the DD operand is used to identify a concatenated data set, the member is obtained from the first data set that it exists in.

DATA={ }

Specifies the records to be printed.

LIMIT=*number*

Specifies the maximum number of records to be printed in the file. This operand is only applicable if DATA=* is specified.

TRUNCATE=*number*

Specifies the length to which the records printed from the file are truncated. The value of the TRUNCATE operand must be in the range 1 to 32,760.

$DSCALL OPT=READ

Reads one or more records from a data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=READ
             DATA= { * | variable_name [ , variable_name,... ] | prefix*}
           { DD=DD_name | ID=path_name }
           [ LIMIT= number ]
           [ TRUNCATE= number ]
```

This option is used to read records from a data set or a member of a PDS into variables. The procedure must have an open path to the data set or PDS member to be read (use $DSCALL OPT=OPEN to open a path).

Operands:

OPT=READ

Specifies that a read action be performed.

DATA={ * | *variable_name* [, *variable_name*...] | *prefix}**

Specifies the records to be read.

- If DATA=* is specified, the records are returned in a shared variable, &$DSDATA*. *All records are read* until the LIMIT value is reached or the end of file (EOF) is reached. The records are placed in variables called &$DSDATA1, &$DSDATA2, and so on as required. If the LIMIT operand is omitted, a value of 1 is assumed.
- If DATA=*variable_name* is specified, a single record is read. Multiple variables need to be specified (*variable_name1,variable_name2,...,variable_namen*) when the record is longer than 250 bytes and must be divided into 250-byte segments. In this case, the relevant variables must be included in SHRVARS with the $DSCALL procedure. The variable name cannot have prefix $DS; if it does, it is reset to null.

  Note: This version of the DATA operand reads only one record into the data set. It must be used to read records containing more than 250 bytes.
- If DATA=*prefix** is specified, the records are returned in shared variables *&prefix1*, *&prefix2*, and so on, as required. All records are read until the LIMIT value is reached or the end of file (EOF) is reached. If the LIMIT operand is omitted, a value of 1 is assumed.

DD=DD_*name*

Specifies the ddname of the file to be read. This operand is used as the path ID if the ID parameter is not set. The path ID specified must already have been created with an OPT=OPEN or OPT=FOPEN call.

ID=*path_name*

Specifies the path ID of the file to be read.

Note: Any combination of OPEN, FOPEN, CLOSE, FCLOSE, READ, and WRITE requests must be done in a single procedure.

LIMIT=*number*

Specifies the maximum number of records to be read from the file. The value of the LIMIT operand must be in the range 1 to 9,999. This operand is applicable only if DATA=* or DATA=prefix* is specified.

TRUNCATE=*number*

Specifies the length to which records read from the file are truncated. The value of the TRUNCATE operand must be in the appropriate range for the value specified for the DATA operand:

- If DATA=* or DATA=*prefix** is specified, then the TRUNCATE value must be in the range 1 to 250.
- If DATA=*variable_name1,variable_name2,...,variable_namen* is specified, then the TRUNCATE value must be in the range 1 to 32,760.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

$DSFDBK is set to 9 and $DSRC set to 4 if the number of records specified by the LIMIT operand is not satisfied.

Return Variables:

&$DSDATA*n*

Contains the records for the DATA=* option, where n is a number to identify each return. If the DATA=variable_name operand is used, the nominated variables contain the data

&$DSDATA#

Contains the count of records read. For DATA=* it contains the number of records read on this call.

&$DSEOF

End of file indicator. The value is null unless the end of file is encountered. If end of file is encountered, this variable is set to the value EOF. If DATA=variable_name is specified, no data is returned if the end of file is encountered.

Examples: OPT=READ

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=READ ID=SYSPATH DATA=* +
   LIMIT=9999

&CONTROL SHRVARS=($DS, ABC)
EXEC $DSCALL OPT=READ ID=SYSPATH DATA=ABC

&CONTROL SHRVARS=($DS, ABC)
EXEC $DSCALL OPT=READ ID=SYSPATH +
   DATA=ABC* LIMIT=9999
```

$DSCALL OPT=RENAME

Renames a cataloged data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL    OPT=RENAME
                DSN=dataset_name
                NEWNAME=dataset_name
```

This call is used to rename a cataloged data set. Your product region issues an ENQ for both the old and new data set names before attempting to perform the request. If the new data set name already exists, or if the data set is in use, the request fails.

Operands:

OPT=RENAME

Specifies that the data set be renamed.

DSN=*dataset_name*

Specifies the name of the data set to be renamed. No member name is specified.

NEWNAME=*dataset_name*

Specifies the new name for the data set. No member name is specified.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=RENAME

```
EXEC $DSCALL OPT=RENAME DSN=SYS1.WORK.DEV01 +
   NEWNAME=SYS1.WORK.TEST12
```

$DSCALL OPT=RENMEM

Renames a member of a PDS.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL     OPT=RENMEM
                 DSN=dataset_name
                 MEMBER=member_name
                 NEWNAME=member_name
```

This call is used to rename a member of a PDS. This call fails if the specified member is in use by another user, or if a member with the new name already exists.

Operands:

OPT=RENMEM

Specifies that a member be renamed.

DSN=*dataset_name*

Specifies the name of the PDS containing the member. No member name is specified.

MEMBER=*member_name*

Specifies the member to be renamed. A member name must be provided.

NEWNAME=*member_name*

Specifies the new name for the member.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=RENMEM

```
EXEC $DSCALL OPT=RENMEM DSN=CUSTOMER.DATA MEMBER=DEV01 +
   NEWNAME=TEST01
```

$DSCALL OPT=SHOWALLOC

Provides a full-screen display of allocated data sets.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=SHOWALLOC
           [ DD=DD_name ]
```

This call is used to display a full-screen selection list of data sets that are allocated to your product region. The selection list can contain all data sets allocated to your product region or only show data sets that are allocated to a specific DD.

Operands:

OPT=SHOWALLOC

Specifies display of allocated data sets.

DD=DD_*name*

Specifies that the display is to contain only data sets which are allocated to the specified DD. If this operand is omitted, all data sets that are allocated to your product region are displayed.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Examples:

To display information about a single ddname, SYSDD:

```
EXEC $DSCALL OPT=SHOWALLOC DD=SYSDD
```

To display information about all ddnames beginning with S:

```
EXEC $DSCALL OPT=SHOWALLOC DD=S*
```

To display information about all ddnames:

```
EXEC $DSCALL OPT=SHOWALLOC
```

Note: The data sets are displayed on a selection list that provides two actions, UNALLOCATE (which requires confirmation) and INFO (that displays information about a selected data set) as shown in the following figure, which shows an example of $DSCALL output when OPT=SHOWALLOC is specified:

```
USER01------------ Dataset Services : Allocated Files -------------------
Command ===>                                              Scroll ===> PAGE

                                                      S/=Info U=Unallocate
    DDname   Status     User     Dataset name
    AOMDB    SHR        DE1NBSYS AUDE0.DENM1.AOMDB
    AOMDB1   SHR        DE1NBSYS AUDE0.DENM1.AOMDB
    CG#00119 SHR        DE1NBSYS SYS3.BIGBLOCK
             SHR        DE1NBSYS AUDE0.NMV2SMS.WORK.CG#00119.SOURCE
             SHR        DE1NBSYS AUDE0.NMV2SMS.TEST.SOURCE
             SHR        DE1NBSYS AUDE0.NMV2SMS.LODG.SOURCE
             SHR        DE1NBSYS AUDE0.NMV2SMS.PROD.SOURCE
             SHR        DE1NBSYS AUDE0.NMV2SMS.BASE.SOURCE
             SHR        DE1NBSYS AUDE0.IHS.PROCS
             SHR        DE1NBSYS AUDE0.LIB.NCL
    CG#00165 SHR        DE1NBSYS SYS3.BIGBLOCK
    CG#00168 SHR        DE1NBSYS SYS3.BIGBLOCK
             SHR        DE1NBSYS AUDE0.NMV2SMS.WORK.CG#00168.SOURCE
             SHR        DE1NBSYS AUDE0.NMV2SMS.TEST.SOURCE
             SHR        DE1NBSYS AUDE0.NMV2SMS.LODG.SOURCE
             SHR        DE1NBSYS AUDE0.NMV2SMS.PROD.SOURCE
             SHR        DE1NBSYS AUDE0.NMV2SMS.BASE.SOURCE
F1=Help      F2=Split     F3=Exit                 F5=Find     F6=Refresh
F7=Backward  F8=Forward   F9=Swap
```

$DSCALL OPT=SUBMIT

This call submits JCL to the JES internal reader for execution. The JCL is contained in a sequential data set or PDS member, or is passed to $DSCALL in variables.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=SUBMIT
           { DSN=dataset_name |
             VARS=prefix
             RANGE=(a , b) }
```

Operands:

OPT=SUBMIT

Submits the JCL.

DSN=*dataset_name*

Specifies the name of the PDS or sequential data set containing the JCL. If the data set name is a PDS, a member name must be specified.

VARS=*prefix*

The variable name prefix of the variables that contain the JCL.

Note: If VARS= is specified, the relevant variables must be shared to $DSCALL, for example, using &CONTROL SHRVARS. The variable names are assumed to be *prefixa*, *prefixb*,..., *prefixn*.

RANGE= (*a* , *b*)

The numeric range (of the variable name suffix) for the variables containing the JCL. This operand is valid, and is required, only when the VARS operand is specified.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSJOB#

Contains JOB*nnnnn* where *nnnnn* is the job number assigned.

Example: OPT=SUBMIT

This example submits JCL contained in a data set:

```
EXEC $DSCALL OPT=SUBMIT DSN=CUSTOMER.DATA.JOBS(REORG)
```

This example submits JCL using a range of variables:

```
&CONTROL SHRVARS=($DS,$TEST)
&$TEST1=&STR //FREDX JOB 'SAMPLE',CLASS=A,MSGLEVEL=(1,1),MSGCLASS=T,
&$TEST2=&STR // NOTIFY=FRED
&$TEST3=&STR //STEP EXEC PGM=IEFBR14
&$TEST4=&STR //
EXEC $DSCALL OPT=SUBMIT VARS=$TEST RANGE=(1,4)
```

Notes:

This function allocates a SYSOUT data set with CLASS=A and PGM=INTRDR. The JCL is obtained from the specified data set or variables and written to the SYSOUT file. When the SYSOUT file is closed, the job number assigned to the submitted job is returned. This value is placed in variable $DSJOB# for return to the caller.

This function does not work if your product region is running under the master scheduler instead of JES.

This function is designed to submit only a single job.

$DSCALL OPT=UNALL

This call deallocates a data set from your product region.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT={ UNALL | FREE }
             DD=ddname
           [ DISP={ KEEP | DELETE } ]
```

Operands:

OPT={ UNALL | FREE }

Specifies the deallocation of a data set.

DD=*ddname*

Specifies the ddname to deallocate.

DISP={ KEEP | DELETE }

Specifies an override of the Normal Disposition which was specified when the data set was allocated to your product region.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSDYNEC

DYNALLOC error code

&$DSDYNIC

DYNALLOC information code

Example: OPT=UNALL

```
EXEC $DSCALL OPT=UNALL DD=DD1
```

Notes:

If a concatenated data set is freed, the first data set in the concatenation is deallocated. The other ddnames in the concatenation return to their original allocation state. The net effect is the same as a deconcatenation followed by a deallocation of the first ddname in the concatenation.

A temporary data set whose name was generated at allocation is deleted when deallocated, regardless of the disposition value specified.

$DSCALL OPT=UTILITY

Executes the IBM utilities: IEBGENER, IEBCOPY, IEHLIST, IEHMOVE, and IDCAMS.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=UTILITY
             UTILITY=program
           [ DISPLAY= { YES | NO } ]
           [ IEBUPDTE= { YES | NO } ]
           [ SYSIN= DD_name ]
           [ SYSPRINT= { KEEP | DELETE | DSN } ]
           [ SYSUT1= DD_name ]
           [ SYSUT2= DD_name ]
```

This call allows users of your product region to execute some IBM utility programs. If any SYS* data sets are required for the specified utility, then the data sets must be allocated by the user before issuing the $DSCALL call and the ddnames must be passed as parameters to $DSCALL.

Operands:

OPT=UTILITY

Specifies that an operating system utility is to be executed. If you are executing in an MSP or VOS environment, then the actual utility called may be different from the IBM utility name. See the Notes section on the following page.

UTILITY=*program*

Specifies the name of the utility to be executed.

DISPLAY={ YES | NO }

Specifies whether to display the output on completion of the function. This operand is valid only if SYSPRINT=DSN is specified. If DISPLAY=YES is specified, then a full-screen display of the SYSPRINT data set is presented after the copy operation has completed.

IEBUPDTE={ YES | NO }

Specifies whether PDS updates is performed.

SYSIN=DD_*name*

Specifies a SYSIN ddname. This operand is required for all IBM utilities except IEBGENER. If no SYSIN ddname is provided for IEBGENER, then the default SYSIN file (/*) is used.

SYSPRINT={ DELETE | KEEP | DSN }

Specifies the type of data set to be allocated to the SYSPRINT DD for the copy operation.

- If SYSPRINT=KEEP is specified, then your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed the SYSOUT data set is deallocated with DISP=KEEP to cause it to be kept. The output from the copy operation is browsed through, for example, SDSF.
- If SYSPRINT=DELETE is specified, then your product region allocates a SYSOUT file with CLASS=A, and after the copy has completed the SYSOUT file is deallocated with DISP=DELETE to cause it to be deleted. The output from the copy operation cannot be browsed.
- If SYSPRINT=DSN is specified, then your product region allocates a temporary data set, which is deleted when your product region terminates. The output from the copy operation is browsed through ISPF or by the use of the DISPLAY=YES operand on this call. The data set is not deallocated from your product region on completion of the copy operation. The data set name, ddname, and volume for the data set are returned in variables to the caller.

SYSUT1=DD_*name*

Specifies the SYSUT1 ddname. This operand is required for IEHMOVE and IEBGENER.

SYSUT2=DD_*name*

Specifies the SYSUT2 ddname. This operand is required for IEBGENER only.

Return Codes:

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSPRINTDSN

Name of SYSPRINT file

&$DSPRINTVOL

Volume where SYSPRINT file is located

&$DSPRINTDD

SYSPRINT ddname

Example: OPT=UTILITY

```
EXEC $DSCALL OPT=UTILITY UTILITY=IDCAMS DISPLAY=YES +
   SYSIN=SYSIN SYSUT1=DDSYS1 SYSUT2=DDSYS2
```

Notes:

It is the caller's responsibility to allocate the SYSUT1, SYSUT2, and SYSIN data sets prior to the OPT=UTILITY call if they are required for the utility. This is also necessary for other ddnames used on control statements in SYSIN. For example, for IEBCOPY where SYSIN is COPY INDD=DD1 OUTDD=DD2. In this case, DD1 and DD2 must be allocated prior to the OPT=UTILITY call.

The following table shows the IBM utilities that are called by the OPT=UTILITY call and the names of corresponding utilities that are used in other operating environments.

Utility Name	IBM	Fujitsu	Hitachi
IEBCOPY	IEBCOPY	JSECOPY	JSDCOPY
IEBGENER	IEBGENER	JSDGENER	JSDSCOPY
IEHMOVE	IEHMOVE	JSGMOVE	JFSMOVE

Utility Name	IBM	Fujitsu	Hitachi
IDCAMS	IDCAMS	KQCAMS	JSCVSUT
IEHLIST	IEHLIST	?	?

$DSCALL OPT=VOLSPACE

Finds the number of free tracks on a volume (disk pack).

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=VOLSPACE
             VOL=volume_name
```

Use this function to find how much space is available on a volume.

Operands:

OPT=VOLSPACE

Specifies that free space on a volume be found.

VOL=*volume_name*

The name of the volume on which free space is to be found.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Return Variables:

&$DSFREETRKS

The number of free tracks on the volume

&$DSUNIT

Unit of volume disk pack

Example: OPT=VOLSPACE

```
EXEC $DSCALL OPT=VOLSPACE VOL=MOS001
```

$DSCALL OPT=WRITE

Writes one or more records to an open data set.

```
&CONTROL SHRVARS=($DS)
EXEC $DSCALL OPT=WRITE
             DATA= { * | variable_name [ , variable_name,... ] | prefix*}
           { DD=DD_name | ID=path_name }
           [ COUNT= number ]
           [ TRUNCATE= number ]
```

This call is used to write data contained in variables to a data set or member of a PDS. The procedure must have an open path to the data set or PDS member to write to it (use $DSCALL OPT=OPEN to open a path).

Operands:

OPT=WRITE

Specifies that a write action be performed.

DATA={ * | *variable_name* [, *variable_name*,...]| | *prefix }**

Specifies the records to be written to the data set.

- If DATA=* is specified, All records are written until the COUNT value is reached. The records to be written are contained in variables &$DSDATA1, &$DSDATA2, and so on as required.
- If DATA=*variable_name* is specified, only a single record is written. If multiple variables are specified, the data contained in the variables is concatenated to form a single record which is then written to the data set. In this case the relevant variables must be included in SHRVARS with the $DSCALL procedure. The variable name cannot have prefix $DS; if it does, it will be reset to null.
- If DATA=*prefix** is specified, the records to be written are contained in the shared variables &*prefix1*, &*prefix2*, and so on as required.

DD=DD_*name*

Specifies the ddname of the file to write to. This operand is used as the path ID if the ID operand is not specified.

ID=*path_name*

Specifies the Path ID of the file to write to.

Any combination of OPEN, FOPEN, CLOSE, FCLOSE, READ, and WRITE requests must be done in a single procedure.

COUNT=*number*

Specifies the number of records to be written to the data set. This is a required operand if DATA=* or DATA=*prefix** is specified.

TRUNCATE=*number*

Specifies the length the written records will be truncated to. The maximum value of the TRUNCATE operand is 250.

Return Codes:

Note: For more information about &SYSMSG, &$DSRC, and &$DSFDBK, see Return Codes (see page 1051) and Feedback Codes (see page 1051).

Example: OPT=WRITE

```
&$DSDATA1 = REC1
&$DSDATA2 = REC2
&$DSDATA3 = REC3
&$DSDATA4 = REC4
EXEC $DSCALL OPT=WRITE ID=SYSPATH COUNT=4 DATA=*
```

Chapter 8: MVS System Symbols Interface

This section contains the following topics:

Accessing MVS Static System Symbols

The standard MVS static system symbols include the following:

- &SYSALVL
- &SYSCLONE
- &SYSNAME
- &SYSPLEX
- &SYSR1

Other symbols is defined through the SYMDEF argument in the SYS1.PARMLIB(IEASYM*xx*) member.

You can use the following operating system command to display these symbols:

```
D SYMBOLS
```

These symbols are made available to any NCL application.

To access the static system symbols from a NCL procedure, use $CAPKBIF procedure. The procedure supports the following call types:

- $CAPKBIF PARMS=(PLEXSUB...)
- $CAPKBIF PARMS=(PLEXSYM,,COUNT)
- $CAPKBIF PARMS=(PLEXSYM,*symbol*,NEXT)
- $CAPKBIF PARMS=(PLEXSYM,*symbol*,VALUE)

Return Variables:

If &RETCODE is set to 0, the &$CAVALUE variable contains the returned value.

If &RETCODE is not zero, an error has occurred and &SYSMSG will contain the error description.

If &RETCODE is set to 8, an error has occurred. Details of the error are contained in variables as follows:

- &$CAERRORTEXT contains the error description.
- &$CAERRORCODE contains the error code.
- &$CAERRORCOND contains the error condition.
- &$CAERRORSUBN contains the substituted variable name.

$CAPKBIF PLEXSUB

Substitutes system and user symbols into a string.

```
&CALL PROC=$CAPKBIF PARMS=(PLEXSUB,string,subchar,static,dynamic,
usymbol1,uvalue1, ...,usymbol50,uvalue50)
```

Operands:

string

Specifies the string with variables to be substituted.

subchar

Specifies the prefix that designates a variable (the variable designator). This allows you to alter the variable designator to a value other than the default value of the ampersand (&). By altering the variable designator, you prevent NCL from substituting variables with its values when defining the string, which would lead to incorrect results.

Note: You can use more than one character, for example,. '%^'.

static

Specifies whether static system symbols are used. Valid values are Y or N.

Default: Y

dynamic

Specifies whether dynamic system symbols are used. Valid values are Y or N.

Default: Y

usymbol1,uvalue1,...,usymbol50,uvalue50

Specifies optional pairs of up to 50 user-defined symbols and their values. The symbol name is up to 16 characters long, with the first character being an alphabetic or national character and the remaining alphanumeric or national characters. The value must not increase the string length when substituted, that is, it can be up to one character longer than the symbol name.

Examples:

The following sample NCL code creates a unique data set name identified by system and product region job name.

```
&TMPDSN = &CONCAT & SYSNAME.. & JOBNAME..# & UNIQUE
&UNIQSTR = &SUBSTR &ZUNIQUE 2 7
&CALL PROC=$CAPKBIF PARMS=(PLEXSUB,&TMPDSN,,,,+
JOBNAME,&ZJOBNAME,UNIQUE,&UNIQSTR)
&IF &RETCODE = 0 &THEN +
&TMPDSN = &$CAVALUE
```

Where:

```
&SYSNAME = SYS1
&ZJOBNAME = REGION01
&ZUNIQUE = 0000043B
```

The value returned in &#CAVALUE is as follows:

```
SYS1.REGION01.#000043B
```

The following code shows how using the exclamation mark as the substitution character (subchar) can simplify the previous code sample to create the same result.

```
&UNIQSTR = &SUBSTR &ZUNIQUE 2 7
&CALL PROC=$CAPKBIF PARMS=(PLEXSUB,+
!SYSNAME..!JOBNAME..#!UNIQUE,!,,,+
JOBNAME,&ZJOBNAME,UNIQUE,&UNIQSTR)
&IF &RETCODE = 0 &THEN +
&TMPDSN = &$CAVALUE
```

Where:

&SYSNAME = SYS1

&ZJOBNAME = REGION01

&ZUNIQUE = 0000043B

The value returned in &#CAVALUE is as follows:

```
SYS1.REGION01.#000043B
```

The following example shows the use of the PLEXSUB function in an initialization file to set a unique name within the sysplex for the name of a PSM Spool File data set by using the &SYSNAME system symbol and a user symbol for the region's job name.

```
-*-------------------------------------------------------------*
-* $PSPSMSPOOL - PSM Spool File Specification                  *
-*                                                             *
-* THE FOLLOWING PARAMETERS ARE SET IN THE FOLLOWING ORDER:    *
-*   1) INIT: PSSD1 - Spool File Dataset Name                  *
-*   2) INIT: PSSO1 - Spool File VSAM Options                  *
-*   3) INIT: PSSI1 - File Disposition 1                       *
-*-------------------------------------------------------------*
.PSMSPOOL
  &CALL PROC=$CAPKBIF PARMS=(PLEXSUB,+
   AUDE0.!SYSNAME..!JOBNAME..PSPOOL,!,,,+
   JOBNAME,&ZJOBNAME)
  &$IAPPSSD1 = &$CAVALUE
```

Where:

&SYSNAME = SYS1

&ZJOBNAME = REGION01

The value returned in &#CAVALUE is as follows:

AUDE0.SYS1.REGION01.PSPOOL

$CAPKBIF PLEXSYM COUNT

Returns the number of static system symbols.

```
&CALL PROC=$CAPKBIF PARMS=(PLEXSYM,,COUNT)
&CALL PROC=$CAPKBIF PARMS=(PLEXSYM)
```

Both statements are equivalent.

$CAPKBIF PLEXSYM symbol NEXT

Returns the name of the next static system symbol after a specified symbol. If you supply a null symbol, the call returns the name of the first symbol.

```
&CALL PROC=$CAPKBIF PARMS=(PLEXSYM,symbol,NEXT)
&CALL PROC=$CAPKBIF PARMS=(PLEXSYM,,NEXT)
```

$CAPKBIF PLEXSYM symbol VALUE

Returns the value of the specified static system symbol.

```
&CALL PROC=$CAPKBIF PARMS=(PLEXSYM,symbol,VALUE)
&CALL PROC=$CAPKBIF PARMS=(PLEXSYM,symbol)
```

Both statements are equivalent.

Example:

The following sample NCL procedure displays all the static system symbols on a given system.

```
&CALL PROC=$CAPKBIF PARMS=(PLEXSYM,,COUNT)
&Count = &$CAVALUE
&WRITE Number of PLEXSYM variables = &Count
&SYM = &I = 1
&DOWHILE &I LE &Count

    &CALL PROC=$CAPKBIF PARMS=(PLEXSYM,&SYM,NEXT)

    &SYM = &$CAVALUE

    &CALL PROC=$CAPKBIF PARMS=(PLEXSYM,&SYM,VALUE)

    &WRITE &I &SYM=&$CAVALUE

    &I = &I + 1

&DOEND
```

Chapter 9: Timer Services Interface

This section contains the following topics:

About the Timer Services NCL Interface.

Timer Services has an NCL external interface, which enables you to easily set timers to run specific commands from installation-written NCL procedures. To do this, you must execute the NCL procedure $TICALL.

The functions that is accessed using $TICALL are:

- Add a timer—$TICALL FUNC=ADD
- Retrieve a timer definition—$TICALL FUNC=GET
- Add or update a timer—$TICALL FUNC=PUT
- Delete a timer definition—$TICALL FUNC=DEL
- List a set of timer definitions—$TICALL FUNC=LIST
- Start timer processing—$TICALL FUNC=START
- Query timer processing status—$TICALL FUNC=STATUS
- Stop timer processing—$TICALL FUNC=STOP
- Execute timer and setup for next timer—$TICALL FUNC=NEXT

$TICALL FUNC=ADD

Adds a timer definition.

```
&CONTROL SHRVARS
-EXEC $TICALL FUNC=ADD
            [ CLASS=classname ]
            [ SUBCLASS=subclassname ]
              NAME=name
            [ SDATE=startdate ]
            [ EDATE=enddate ]
            [ STIME=starttime ]
            [ ETIME=endtime ]
              FREQ={ nnnn | HH.MM.SS }
            [ FTYPE={ MONTHS | DAYS | TIME } ]
            [ LIMIT={ nnnn | 1 } ]
            [ DAYLIST=(MON,TUE,…..) ]
            [ CATCHUP={ YES | NO } ]
            [ DELEXP={ YES | NO } ]
            [ STATUS={ ACTIVE | INACTIVE } ]
            [ SAVE={ YES | NO } ]
            [ ROUTE={ * | userid | MON | LOG | SYS } ]
            [ KEEP={ MON | LOG | SYS } ]
              COMMAND=command
            [ DESC=description ]
```

Adds a new timer definition. If a same named definition already exists, then this call will fail.

Operands:

FUNC=ADD

Adds a timer definition.

CLASS=*classname*

Specifies a class name, maximum 8 characters. Timer definitions are categorized by class, subclass and a name. The name is mandatory but the class and subclass names are optional.

SUBCLASS=*subclassname*

Specifies a sub-class name, maximum 8 characters.

NAME=*name*

Specifies a name for this definition, maximum 12 characters. This is a required operand.

SDATE=*startdate*

Specifies the start date for this timer. Specify as a DATE8 format (YYYYMMDD). The default is the current system date.

EDATE=*enddate*

Specifies the end date for this timer. Specify as a DATE8 format (YYYYMMDD). There is no default.

STIME=*starttime*

Specifies the start time for this timer. Specify as HH.MM.SS. The default is the current system time.

ETIME=*endtime*

Specifies the end time for this timer. Specify as HH.MM.SS. There is no default.

FREQ={ *nnnn* | HH.MM.SS }

Specifies the timer interval between successive instances. Either a number of intervals or an hour/minute/second value is specified. The interval is determined by the FTYPE value. If a hour/minute/second value is specified, then FTYPE=TIME must be specified. Both the FREQ and FTYPE operands are only applicable if LIMIT is greater than one.

FTYPE={ MONTHS | DAYS | TIME }

Specifies the interval between successive instances. The actual interval is determined by the FREQ and FTYPE values. The TIME value represents seconds. The default is TIME. For example, FREQ=60 indicates every 60 seconds and FREQ=2 FTYPE=DAYS indicates every 2 days.

LIMIT = 0-9999

Specifies the number of instances that this timer will be activated. The frequency and interval between successive timer executions is determined by the FREQ and FTYPE operands. The default for LIMIT is 1 (one), indicating that this is a one-off timer. A value of 0 (zero) indicates that there is no limit and that there will be an infinite number of instances when the timer will be scheduled.

DAYLIST = (MON,TUE,WED,THU,FRI,SAT,SUN)

Specifies a list of days for which this timer is applicable. The default is all days are applicable.

CATCHUP = { YES | NO }

Specifies whether catchup is to be performed. This occurs if the next scheduled time for this timer has already passed. Catchup processing occurs at region initialization - any timers that were scheduled to occur whilst the region was down are scheduled for immediate execution at region initialization. The default is no catchup is performed.

DELEXP = { YES | NO }

Specifies whether this timer definition is to be deleted after expiry—that is, after the specified end date (EDATE) has passed. If EDATE was not specified, then the definition does not expire. The default is not to delete the timer definition at expiry.

STATUS = { ACTIVE | INACTIVE }

Specifies the actual status to be assigned to this timer definition. The status is changed to ACTIVE at a later time via a 'PUT' function call.

SAVE = { YES | NO }

Specifies whether the timer definition should be saved to the VFS file. SAVE=YES allows for timer definitions to be retained across system restarts.

ROUTE = { * | *userid* | MON | LOG | SYS }

Specifies the user ID under which this command is to be performed. The default ROUTE=* specifies the user ID executing the $TICALL.

KEEP = { MON | LOG | SYS }

Specifies the user ID under which this command is to be performed if the user ID specified by ROUTE has signed off. The default is LOG.

COMMAND = *command*

Specifies the command to be executed when the timer is scheduled to run. The command is up to 256 characters. If the command contains blanks, then it must be enclosed in quotes.

DESC = *description*

Specifies a description for this timer definition. The description is up to 256 characters. If the description contains blanks, then it must be enclosed in quotes.

Return Codes:

&RETCODE = 0

$TICALL completed successfully. Data is returned in the following variables:

$TICATCHUP

CATCHUP

$TICLASS

CLASS

$TICDATE

Create date

$TICOMMAND

COMMAND

$TIDELEXP

DELEXP

$TIEDATE

EDATE

$TIETIME

ETIME

$TIFREQ

FREQ

$TIFTYPE

FTYPE

$TIKEEP

KEEP

$TILIMIT

LIMIT

$TINAME

NAME

$TIROUTE

ROUTE

$TISDATE

SDATE

$TISTATUS

STATUS

$TISTIME

STIME

$TISUBCLASS

SUBCLASS

$TIUSER

Create user ID

$TIAUTH

User authority of $TIUSER

&RETCODE > 0

An error occurred. &SYSMSG is set with an error message.

Example: FUNC=ADD

```
&DOCMD = &STR 'SHOW USER'
&TOMORROW = &DATECONV DATE8 &DATE8 DATE8 +1
&DESC = &STR 'Show user at 2:00pm and 2:01pm tomorrow'
&CALL PROC=$TICALL SHARE=($TI>) PARMS=(FUNC=ADD CLASS=USER +
 NAME=CMD1 FREQ=60 LIMIT=2 SDATE=&TOMORROW STIME=14.00.00 +
 DESC=&DESC COMMAND=&DOCMD)
```

Note: If the text entered in the Description and Command fields contains imbedded blanks, they must be enclosed in quotes.

$TICALL FUNC=GET

Retrieves an existing timer definition.

```
&CONTROL SHRVARS=($TI)
-EXEC $TICALL    FUNC=GET
                [ CLASS=classname ]
                [ SUBCLASS=subclassname ]
                  NAME=name
                [ OPT={ file get option | KEQ } ]
```

Retrieves an existing timer definition.

Operands:

FUNC=GET

Retrieves an existing timer definition.

CLASS=*classname*

Specifies a class name, maximum 8 characters. Timer definitions are categorized by class, subclass and a name. The name is mandatory but the class and subclass names are optional.

SUBCLASS=*subclassname*

Specifies a sub-class name, maximum 8 characters.

NAME=*name*

Specifies a name for this definition, maximum 12 characters. This is a required operand.

OPT = { *file get option* | KEQ }

Specifies the FILE GET OPTION. This option is passed to the VARTABLE GET and FILE GET verb calls. Valid values include:

- KEQ (Key Equal)
- KGT (Key Greater Than)
- KGE (Key Greater Than or Equal)
- KLT (Key Less Than)
- KLE (Key Less Than or Equal)

Note: Refer to &FILE GET and &VARTABLE GET documentation for more information on these values. The default is KEQ.

Return Codes:

&RETCODE = 0

$TICALL completed successfully. Data is returned in the following variables:

$TICATCHUP

CATCHUP

$TICLASS

CLASS

$TICDATE

Create date

$TICOMMAND

COMMAND

$TIDELEXP

DELEXP

$TIEDATE

EDATE

$TIETIME

ETIME

$TIFREQ

FREQ

$TIFTYPE

FTYPE

$TIKEEP

KEEP

$TILIMIT

LIMIT

$TINAME

NAME

$TIROUTE

ROUTE

$TISDATE

SDATE

$TISTATUS

STATUS

$TISTIME

STIME

$TISUBCLASS

SUBCLASS

$TIUSER

Create/modify user ID

$TIAUTH

User authority of $TIUSER

$TIMDATE

Modified date

$TIMTIME

Modified time

&RETCODE > 0

An error occurred. &SYSMSG is set with an error message.

Examples FUNC=GET

```
&CALL PROC=$TICALL SHARE=($TI>) PARMS=(FUNC=GET CLASS=USER +
 NAME=CMD1)
* Timer &$TICLASS/&$TISUBCLASS/&$TINAME data returned
```

$TICALL FUNC=PUT

Updates an existing timer definition.

```
&CONTROL SHRVARS=($TI)
-EXEC $TICALL FUNC=PUT
            [ CLASS=classname ]
            [ SUBCLASS=subclassname ]
              NAME=name
            [ SDATE=startdate ]
            [ EDATE=enddate ]
            [ STIME=starttime ]
            [ ETIME=endtime ]
            [ FREQ={ nnnn | HH.MM.SS } ]
            [ FTYPE={ MONTHS | DAYS | TIME } ]
            [ LIMIT= nnnn ]
            [ DAYLIST=(MON,TUE,…..) ]
            [ CATCHUP={ YES | NO } ]
            [ DELEXP={ YES | NO } ]
            [ STATUS={ ACTIVE | INACTIVE } ]
            [ SAVE={ YES | NO } ]
            [ ROUTE={ * | userid | MON | LOG | SYS } ]
            [ KEEP={ MON | LOG | SYS } ]
            [ COMMAND=command ]
            [ DESC=description ]
```

Updates an existing timer definition. If a same named definition does not exist, then the $TICALL FUNC=PUT will fail.

Operands:

FUNC=PUT

Updates an existing timer definition.

The description of all the other operands is the same as $TICALL FUNC=ADD (see page 1146), except that there are no defaults for the PUT function. If an operand is not specified, then that particular field will not be updated in the timer definition.

Return Codes:

&RETCODE = 0

$TICALL completed successfully. Data is returned in the following variables:

$TICATCHUP

CATCHUP

$TICLASS

CLASS

$TICDATE

Create date

$TICOMMAND

COMMAND

$TIDELEXP

DELEXP

$TIEDATE

EDATE

$TIETIME

ETIME

$TIFREQ

FREQ

$TIFTYPE

FTYPE

$TIKEEP

KEEP

$TILIMIT

LIMIT

$TINAME

NAME

$TIROUTE

ROUTE

$TISDATE

SDATE

$TISTATUS

STATUS

$TISTIME

STIME

$TISUBCLASS

SUBCLASS

$TIUSER

Modify user ID

$TIMDATE

Modified date

$TIMTIME

Modified time

$TIAUTH

User authority of $TIUSER

&RETCODE > 0

An error occurred. &SYSMSG is set with an error message.

Example: FUNC=PUT

```
&TOMORROW = &DATECONV DATE8 &DATE8 DATE8 +1
 &DESC = &STR 'Updating the start date and description'
&CALL PROC=$TICALL SHARE=($TI>) PARMS=(FUNC=PUT CLASS=USER +
 NAME=CMD1 SDATE=&TOMORROW DESC=&DESC)
```

Note: If the text entered in the Description and Command fields contains imbedded blanks, they must be enclosed in quotes.

$TICALL FUNC=DEL

Deletes an existing timer definition.

```
&CONTROL NOSHRVARS
-EXEC $TICALL FUNC=DEL
            [ CLASS=classname ]
            [ SUBCLASS=subclassname ]
              NAME=name
```

Deletes an existing timer definition.

Operands:

FUNC=DEL

Deletes an existing timer definition.

CLASS=*classname*

Specifies a class name, maximum 8 characters. Timer definitions are categorized by class, subclass and a name. The name is mandatory but the class and subclass names are optional.

SUBCLASS=*subclassname*

Specifies a sub-class name, maximum 8 characters.

NAME=*name*

Specifies a name for this definition, maximum 12 characters. This is a required operand.

Return Codes:

&RETCODE = 0

$TICALL completed successfully.

&RETCODE > 0

An error occurred. &SYSMSG is set with an error message.

Example: FUNC=DEL

```
&CALL PROC=$TICALL PARMS=(FUNC=DEL CLASS=USER NAME=CMD1)
```

$TICALL FUNC=LIST

Lists a set of existing timer definition.

```
&CONTROL NOSHRVARS
-EXEC $TICALL    FUNC=LIST
                [ CLASS=classname ]
                [ SUBCLASS=subclassname ]
                [ NAME=name ]
```

Lists a set of existing timer definitions.

Operands:

FUNC=LIST

Lists a set of existing timer definition.

CLASS=*classname*

Specifies a class name, maximum 8 characters. Timer definitions are categorized by class, subclass and a name.

SUBCLASS=*subclassname*

Specifies a sub-class name, maximum 8 characters.

NAME=*name*

Specifies a name for this definition, maximum 12 characters.

Return Codes:

&RETCODE = 0

$TICALL completed successfully.

&RETCODE > 0

An error occurred. &SYSMSG is set with an error message.

Example: FUNC=LIST

```
&CALL PROC=$TICALL PARMS=(FUNC=LIST)
```

$TICALL FUNC=START

Starts timer processing.

```
&CONTROL NOSHRVARS
-EXEC $TICALL     FUNC=START
```

Starts timer processing. This is issued automatically at region startup.

Operands:

FUNC=START

Starts timer processing.

Return Codes:

&RETCODE = 0

$TICALL completed successfully.

&RETCODE > 0

An error occurred. &SYSMSG is set with an error message.

Example: FUNC=START

```
&CALL PROC=$TICALL PARMS=(FUNC=START)
```

$TICALL FUNC=STOP

Stops timer processing.

```
&CONTROL NOSHRVARS
-EXEC $TICALL FUNC=STOP
```

Stops timer processing.

Operands:

FUNC=STOP

Stops timer processing.

Return Codes:

&RETCODE = 0

$TICALL completed successfully.

&RETCODE > 0

An error occurred. &SYSMSG is set with an error message.

Example: FUNC=STOP

```
&CALL PROC=$TICALL PARMS=(FUNC=STOP)
```

$TICALL FUNC=STATUS

Status request of Timer processing.

```
&CONTROL SHRVARS=($TISTATUS)
-EXEC $TICALL FUNC=STATUS
```

Returns the status of timer processing.

Operands:

FUNC=STATUS

Request the status of timer processing.

Return Codes:

&RETCODE = 0

$TICALL completed successfully.

&RETCODE > 0

An error occurred. &SYSMSG is set with an error message.

Example: FUNC=STATUS

```
&CALL PROC=$TICALL SHARE=($TISTATUS) PARMS=(FUNC=STATUS)
&WRITE DATA=Timer Status is &$TISTATUS
```

$TICALL FUNC=NEXT

Execute timer and setup for next timer.

```
&CONTROL NOSHRVARS
-EXEC $TICALL FUNC=NEXT
```

Execute timer and setup for next timer. Any commands that are due at this time will be executed. The time of the next timer due will be calculated and a timer set to execute the 'NEXT' function at that time (or 1 second past midnight tomorrow morning).

The NEXT function call is issued internally automatically. Generally, there is no need for this call to be issued explicitly.

Operands:

FUNC=NEXT

Setup for next timer.

Return Codes:

&RETCODE = 0

$TICALL completed successfully.

&RETCODE > 0

An error occurred. &SYSMSG is set with an error message.

Example: FUNC=NEXT

```
&CALL PROC=$TICALL PARMS=(FUNC=NEXT)
```

Chapter 10: Persistent Global Variables Interface

Note: For information about how to preserve data between region restarts, see the *Administration Guide*.

This section contains the following topics:

$CAGLBL OPT=LOAD

Create a persistent global variable.

```
&CALL PROC=$CAGLBL PARMS=(OPT=LOAD
                  [,NAME={gname | (gname,….)}]
                  [,DEBUG={N | Y}])
```

Note: Optional parameters that are not required are ignored.

Operands:

OPT=LOAD

Loads persistent global variables. If NAME= is omitted or not specified, all persistent global variables in the external repository are loaded with their corresponding values.

Note: A message is issued to the log listing all GLBLs loaded and the values assigned to them.

If NAME= is specified, specified persistent global variables are loaded. A message is issued to the log listing all global variables loaded and the values assigned to them.

Note: If no persistent variables are found, a message is issued to the log.

NAME=*gname*

(Optional) Specifies the persistent global variable to load. Do not include the global variable prefix (&&000).

Note: No wildcards are supported.

DEBUG={N|Y}

When set to Y, initiates internal debugging of the $CAGLBL procedure. Should only be used with assistance from CA Support.

Return Codes:

RETCODE

0 - Successful execution

8 - Unsuccessful execution

Examples:

```
&CALL PROC=$CAGLBL PARMS=(OPT=LOAD)

&CALL PROC=$CAGLBL PARMS=(OPT=LOAD,NAME=MYVAR)
```

$CAGLBL OPT=SAVE

Save a persistent global variable.

```
&CALL PROC=$CAGLBL    PARMS=(OPT=SAVE,
                      NAME=gname,DATA=data |
                      NAME=gname | (gname,...) |
                      NAME=gname | (gname,...), VARS=vname | (vname,.)
                     [,ENV=MACRO]
                     [,DEBUG=N | Y])
```

Operands:

OPT=SAVE

Specifies the SAVE function and creates a persistent global variable (PGV). Values of all global variables whose *gname*s are provided as arguments of NAME are saved in the external repository. Global variables without the current value (null) are not saved and existing external storage records related to them are purged.

- When single *gname* and data is provided, the data is assigned to the global variable *gname* and is saved.
- When a list of *gname*s is accompanied by the list of *vname*s, values from each *vname* are assigned to global variable *gname* and are saved.

A message is issued to the log for each variable saved with the value provided.

Note: The maximum number of persistent global variables that can be saved is limited to 999.

DATA=*data*

Specifies the data to be assigned to the global variable and saved.

VARS=*vname*

Specifies a name of the variable whose value will be assigned to the corresponding global variable gname.

ENV=Macro

Specifies that all the messages should be sent to an internal process log instead of OCS or Activity logs.

DEBUG=N|Y

When set to Y, initiates internal debugging of the $CAGLBL procedure. Should only be used with assistance from CA Technical Support.

Return Codes:

RETCODE

0 - Successful execution

8 - Unsuccessful execution

Examples:

```
&NAME = &STR ( $VARA , $VAR1 , VAR6, VARL )
&CALL PROC=$CAGLBL   PARMS=(OPT=SAVE,DEBUG=N,NAME=&NAME)

&&000LONG = ABCDEFGHIJKLMNOPQRSTUVWXYZ

&CALL PROC=$CAGLBL   PARMS=(OPT=SAVE,DEBUG=Y,NAME=LONG)
```

$CAGLBL OPT=PURGE

Purge a persistent global variable.

```
&CALL PROC=$CAGLBL PARMS=(OPT=PURGE [,DEBUG={N | Y}])
```

Operands:

OPT=PURGE

Specifies the PURGE function. The values of all the global variables with corresponding records in the external repository (PGVs) are nullified and their respective records in the external repository are deleted.

Note: An audit message will be issued.

DEBUG={N | Y}

When set to Y, initiates internal debugging of the $CAGLBL procedure. Should only be used with assistance from Technical Support.

Return Codes:

RETCODE

0—Successful execution

8—Unsuccessful execution

Examples:

```
&CALL PROC=$CAGLBL PARMS=(OPT=PURGE)
```

$CAGLBL OPT=LIST

List persistent global variables.

```
&CALL PROC=$CAGLBL PARMS=(OPT=LIST [,DEBUG={N | Y}])
```

Operands:

OPT=LIST

Specifies the LIST function. Names and values of all the global variables current and present in the external storage are listed to the OCS terminal and activity log.

DEBUG={N|Y}

When set to Y, initiates internal debugging of the $CAGLBL procedure. Should only be used with assistance from Technical Support.

Return Codes:

RETCODE

0 - Successful execution

8 - Unsuccessful execution

Examples:

```
&CALL PROC=$CAGLBL PARMS=(OPT=LIST)
```

$CAGLBL OPT=SHGLBL

Creates SH NCLGLBL output for a persistent global variable.

```
&CALL PROC=$CAGLBL PARMS=( OPT=SHGLBL
                       [ ,NAME={ pattern | * } ]
                       [ ,DEBUG={ N | Y } ] )
```

Operands:

OPT=SHGLBL

Specifies the SHGLBL function. The output from the SH NCLGLBL command is processed and the following actions are taken prior to repeating it to the output, depending on the specification of the NAME parameter:

If NAME= is not specified

Lists the names of global variables (as with SH NCLGLBL). The names will be sorted, names of global variables having persistence will be marked with '#' before them.

If NAME=*

Lists the names, value lengths and values of all global variables in sorted order (as with SH NCLGLBL=). Names of global variables with persistence are prefixed with # , and an additional line is added that lists the length and value of the persistent variable. If the first 230 characters of global variable value differ from the persistent value, the global variable value appears in blue.

If NAME=*pattern*

Lists the names, value lengths and values of global variables with the name starting with pattern in sorted order (as with SH NCLGLBL=*pattern*). Names of global variables with persistence are prefixed with #, and an additional line is added that lists the length and value of persistent variable. If the first 230 characters of global variable value differ from the persistent value, the global variable appears in blue.

Note: You can use the EQUATE command to simplify your process, for example:

```
equate showglbl+ start $caglbl
opt=shglbl +
```

You can define the EQUATE in the $NM EQUATES parameter group (enter /PARMS at the prompt to list parameter groups).

NAME=*pattern*

Specifies a full name or starting characters of the global variables names.

DEBUG={ N | Y }

When set to Y, initiates internal debugging of the $CAGLBL procedure and should only be used with assistance from Technical Support.

Return Codes

RETCODE

0—Successful execution

8—Unsuccessful execution

Examples:

```
&CALL PROC=$CAGLBL PARMS=(OPT=SHGLBL)
```

Appendix A: Event Distribution Services

Event Distribution Services (EDS) is a feature that lets you control event notification to your system.

This section contains the following topics:

Sample Code

The following example shows the use of &INTREAD for receiving EDS notification. It also contains an example of an &EVENT verb.

VTAM node failure events will be generated if PPO is active and the standard DEFMSG commands have been executed, either in the READY procedure or some time prior to execution of this procedure, that is:

```
DEFMSG MSGID=(129,259,526,822) DELIVER=ALL +
    EVENTNAME=NODE.FAILED
```

If the node is critical, the event is reissued, and can then be picked up by a network recovery server which may be executing somewhere else in your product region.

```
&CONTROL
   &EVNAME = &STR $SNA.NODE.FAILED      -* Node failure event name
   &EVTYPE = &STR CONFIGURATION
-*
-*  Issue PROFILE command for $SNA.NODE.FAILED event receipt
-*

   &INTCMD -PROFILE EDS ENABLE=&0 +
     NAME=&EVNAME +
     TYPE=&EVTYPE

   &WAITSECS = 600          -* 10 mins
.MONEVENT
   &INTREAD TYPE=RESP +
      MDO=PPOMDO +
      WAIT=&WAITSECS
   &RC = &ZFDBK             -* Copy feedback info.
   &IF &FDBK = 4 &THEN +
    -* Nothing arrived
      &DO
          &WRITE LOG=YES TERM=NO &0 &EVNAME MONITOR ACTIVE, +
          * No node failures detected in past 10 mins.
          &GOTO .MONEVENT -* Go to wait again
      &DOEND
```

```
&IF &RC > 4 &THEN +
 &DO
    &WRITE MON=YES LOG=Y COLOR=RED DATA=MDO FAILURE +
    OCCURRED ON &0 MONITOR INTREAD. +
    FDBK=&ZFDBK,&ZMDORC,&ZMDOFDBK
 &DOEND
  &ASSIGN VARS=EVENTEXT FROM MDO=&STEM.TEXT
  &PARSE ARGS DATA=&EVENTEXT
&IF .&1 EQ .STOP &THEN +
 -* Stop if STOP written
      &GOTO .ENDUP          -* to response queue
   &IF .&1 NE .N00102 &THEN +
  -* GOBACK if this is not an
      &GOTO .MONEVENT       -* EDS notification
-* Extract event details from &MSG MDO
   &ASSIGN VARS=PRIRES  FROM MDO=&STEM.RESOURCE.PRIMARY
   &ASSIGN VARS=SECRES  FROM MDO=&STEM.RESOURCE.SECONDARY
   &ASSIGN VARS=NAME  FROM MDO=&STEM.EVENT.NAME
   &ASSIGN VARS=REF  FROM MDO=&STEM.EVENT.REFERENCE
   &ASSIGN VARS=ROUTCD  FROM MDO=&STEM.EVENT.ROUTCDE
   &ASSIGN VARS=CLASS  FROM MDO=&STEM.EVENT.TYPE
   &ASSIGN VARS=IST#  FROM MDO=PPOMDO.PPOCNTL.VTAMNUM
   &ASSIGN VARS=ISTTEXT  FROM MDO=PPOMDO.TEXT
-*
-*  Log the failure.  Pass the PPO MDO in case LOGPROC
-*  wants to do something with it.
-*
   &WRITE TERM=NO LOG=YES MDO=PPOMDO +
      DATA=&0 IST&IST# FAILURE DETECTED FOR +
 NODE &PRIRES,&SECRES PPOREF=&REF -*   &WRITE TERM=NO LOG=YES +
 -* Log the VTAM msg text
      DATA = &ISTTEXT -* -* For the purposes of this example, re-issue the event
 -* and go back to wait for further failure notification.
 -* Additional filtering or recovery code could be placed
 -* here, depending on installation requirements.
-*
   &EVENT NAME=SNA.RECOVERY.REQUIRED +
      RESOURCE=(&PRIRES,&SECRES) +
      TYPE=APPLICATION +
      MDO=PPOMDO
   &GOTO .MONEVENT
-* Await next event
-*
.ENDUP
 &END
```

System Event Names

Important! Do not generate system events (using &EVENT) because doing so could potentially cause instability to your region.

This section lists the attributes of internally-generated events. The attributes is specified on the PROFILE EDS command to define those events that are of interest to a process.

The event name is the primary EDS identifier and has been chosen to describe what the event is. All internally-generated event names begin with a $$ prefix.

For information about the event attributes, see the associated operand descriptions for the &EVENT verb in the chapter "Verbs and Built-in Functions".

If the attribute operand is shown in upper case, the operand is the literal value for the attribute. If the operand appears in italics or lower case, the operand represents a variable attribute value.

$$AOM.ABENDED

TYPE=SERVICEABILITY

REF=SYS *or* USER *abend-code*

$$AOM.MESSAGES.LOST

TYPE=SERVICEABILITY

REF=N83K01 *or* N83201 DATA=*ref msg-text*

$$AOM.PAUSED

TYPE=SERVICEABILITY

REF=N85747

$$AOM.RUNNING

TYPE=SERVICEABILITY

REF=N85749

$$AOM.SHUTDOWN.COMPLETE

TYPE=SERVICEABILITY

REF=N83302

$$AOM.STARTED

TYPE=SERVICEABILITY

REF=N85722

$$FTS.RECEIVE.START

TYPE=PROCEDURAL

RESOURCE=(*request-name*,*origin*)

REF=*msg-id*

$$FTS.RECEIVE.END.OK

TYPE=PROCEDURAL

RESOURCE=(*request-name*,*origin*)

REF=ACK (*if* ACK *required*)

DATA=*request-type to-dsn*

$$FTS.RECEIVE.END.WARN

TYPE=PROCEDURAL

RESOURCE=(*request-name*,*origin*)

REF=ACK (*if* ACK *required*)

DATA=*request-type to-dsn*, *warning-msg*

$$FTS.RECEIVE.STATS

TYPE=UTILIZATION

RESOURCE=(*request-name*,*origin*)

DATA=N44802 *msg-text*

$$FTS.RECEIVE.FAIL

TYPE=SERVICEABILITY

RESOURCE=(*request-name*,*origin*)

DATA=N44804 *msg-text*

$$FTS.TRANSMIT.START

TYPE=PROCEDURAL

RESOURCE=(*request-name*,*destination*)

REF=*msg-id*

$$FTS.TRANSMIT.END.OK

TYPE=PROCEDURAL

RESOURCE=(*request-name*,*destination*)

REF=ACK (*if* ACK *required*)

DATA=*request-type from-dsn*

$$FTS.TRANSMIT.END.WARN

TYPE=PROCEDURAL

RESOURCE=(*request-name*,*destination*)

REF=ACK (*if* ACK *required*)

DATA=*request-type from-dsn*, *warning-msg*

$$FTS.TRANSMIT.STATS

TYPE=UTILIZATION

RESOURCE=(*request-name*,*destination*)

DATA=N44302 *msg-text*

$$FTS.TRANSMIT.FAIL

TYPE=SERVICEABILITY

RESOURCE=(*request-name*,*destination*)

DATA=N44304 *msg-text*

$$NTS.SESSION.START

TYPE=CONFIGURATION

RESOURCE=(*session-pair*)

OBJECT=SESSION

$$NTS.SESSION.END

TYPE=CONFIGURATION

RESOURCE=(*session-pair*)

OBJECT=SESSION

$$NTS.SESSION.FAIL

TYPE=SERVICEABILITY

RESOURCE=(*session-pair*)

OBJECT=SESSION

$$NTS.RTM.OBJ.EXCEEDED

TYPE=SERVICEABILITY

RESOURCE=(*session-pair*)

OBJECT=SESSION

DATA=RTM *data*

$$SNAMS.APPL.REGISTER

TYPE=CONFIGURATION

RESOURCE=(*global-routing-name*,*netid*.*nauname*)

OBJECT=SNA

$$SNAMS.APPL.DEREGISTER

TYPE=CONFIGURATION

RESOURCE=(*global-routing-name*,*netid*.*nauname*)

OBJECT=SNA

$$SNAMS.EP.ACTIVE

TYPE=CONFIGURATION

RESOURCE=(*category-name*,*netid*.*nauname*)

OBJECT=SNA

$$SNAMS.EP.INACTIVE

TYPE=CONFIGURATION

RESOURCE=(*category-name*,*netid*.*nauname*)

OBJECT=SNA

$$SNAMS.FP.ACTIVE.ASSIGNED

TYPE=CONFIGURATION

RESOURCE=(*category-name*,*netid*.*nauname*.*appl*) O

BJECT=SNA

$$SNAMS.FP.ACTIVE.LOCAL

TYPE=CONFIGURATION,

RESOURCE=(*category-name*,*netid*.*nauname*.*appl*)

OBJECT=SNA

$$SNAMS.FP.INACTIVE.ASSIGNED

TYPE=CONFIGURATION

RESOURCE=(*category-name*,*netid*.*nauname*.*appl*)

OBJECT=SNA

$$SNAMS.FP.INACTIVE.LOCAL

TYPE=CONFIGURATION

RESOURCE=(*category-name*,*netid*.*nauname*.*appl*)

OBJECT=SNA

$$SYS.FILE.OPEN

TYPE=SERVICEABILITY

RESOURCE=*ddname*

OBJECT=DDNAME

$$SYS.FILE.CLOSE

TYPE=SERVICEABILITY

RESOURCE=*ddname*

OBJECT=DDNAME

$$SYS.FILE.ASSIGN

TYPE=SERVICEABILITY

RESOURCE=(*file-id*,*ddname*)

OBJECT=FILEID

$$SYS.FILE.RESET

TYPE=SERVICEABILITY

RESOURCE=*ddname*

OBJECT=DDNAME

$$SYS.FILE.STOP

TYPE=SERVICEABILITY

RESOURCE=(*file-id*,*ddname*)

OBJECT=FILEID

$$SYS.FILE.ALLOC

TYPE=CONFIGURATION

RESOURCE=*ddname* REF=1*st* 7 *bytes of* DATA OBJECT=DDNAME

DATA=*data-set-name*

$$SYS.FILE.UNALLOC

TYPE=CONFIGURATION

RESOURCE=*ddname* REF=1*st* 7 *bytes of*

DATA OBJECT=DDNAME DATA=*data-set-name*

$$SYS.FILE.EOV

TYPE=SERVICEABILITY

RESOURCE=*ddname*

OBJECT=DDNAME

$$SYS.FILE.CA.SPLIT

TYPE=SERVICEABILITY

RESOURCE=*ddname*

OBJECT=DDNAME

$$SYS.FILE.FULL

TYPE=SERVICEABILITY

RESOURCE=*ddname*

OBJECT=DDNAME

$$SYS.FILE.SHORTAGE

TYPE=SERVICEABILITY

RESOURCE=*ddname*

OBJECT=DDNAME

DATA=N28802 *msg-text*

$$SYS.FILE.ERROR

TYPE=SERVICEABILITY

RESOURCE=*ddname*

OBJECT=DDNAME

DATA=N28803 *or* N28804 *msg-text*

$$SYS.INMCAM.ACTIVE

TYPE=SERVICEABILITY

RESOURCE=(*group-name*,*lu-name*)

OBJECT=INMCAM

REF=*msg-id*

$$SYS.INMCAM.INACTIVE

TYPE=SERVICEABILITY

RESOURCE=(*group-name*,*lu-name*)

OBJECT=INMCAM

REF=*msg-id*

$$SYS.LINK.ACTIVE

TYPE=SERVICEABILITY

RESOURCE=(*link-name*,*lu-name*)

REF=N35002 *or* N47A03

OBJECT=INMC *or* APPC

$$SYS.LINK.INACTIVE

TYPE=SERVICEABILITY

RESOURCE=(*link-name*,*lu-name*)

REF=N35001 *or* N47A02

OBJECT=INMC *or* APPC

$$SYS.LOG.SWAP

TYPE=SERVICEABILITY

RESOURCE=(*new-dd*,*old-dd*)

REF=N16109

$$SYS.MAI.DISCONNECT

TYPE=CONFIGURATION

RESOURCE=(*lu-name*,*acb-name*)

REF=06

$$SYS.MAI.LOGOFF

TYPE=CONFIGURATION

RESOURCE=(*lu-name*,*acb-name*)

REF=05

$$SYS.MAI.LOGON

TYPE=CONFIGURATION

RESOURCE=(*lu-name*,*acb-name*)

REF=01

$$SYS.MAI.RECONNECT

TYPE=CONFIGURATION

RESOURCE=(*lu-name*,*acb-name*)

REF=02

$$SYS.MAI.SESSION.START

TYPE=CONFIGURATION

RESOURCE=(*lu-name*,*acb-name*)

REF=09

$$SYS.MAI.SESSION.STOP

TYPE=CONFIGURATION

RESOURCE=(*lu-name*,*acb-name*)

REF=0A

$$SYS.MAI.SKIP.TO.MENU

TYPE=CONFIGURATION

RESOURCE=(*lu-name*,*acb-name*)

REF=04

$$SYS.MAI.SKIP.TO.SESS

TYPE=CONFIGURATION

RESOURCE=(*lu-name*,*acb-name*)

REF=03

$$SYS.NCL.ENDED

TYPE=SERVICEABILITY

RESOURCE=(*proc-name*,*ncl-id*)

REF=*msg-id* (&ZMEVUSER *is set*)

$$SYS.NCL.ERROR

TYPE=SERVICEABILITY

RESOURCE=(*proc-name*,*ncl-id*)

REF=*msg-id* (&ZMEVUSER *is set*)

$$SYS.NETSPY.ACTIVE

TYPE=SERVICEABILITY

RESOURCE=(*conn-name*,*lu-name*)

OBJECT=NETSPY

REF=NT7001

$$SYS.NETSPY.INACTIVE

TYPE=SERVICEABILITY

RESOURCE=(*conn-name*,*lu-name*)

OBJECT=NETSPY

REF=NT7002

$$SYS.NMINIT.COMPLETE

TYPE=SERVICEABILITY

RESOURCE=(*system-id*)

REF=N00515

$$SYS.NMREADY.COMPLETE

TYPE=SERVICEABILITY

RESOURCE=(*system-id*)

REF=N00516

$$SYS.SHUTDOWN.ACKNOWLEDGED

TYPE=SERVICEABILITY

RESOURCE=*nm-dom-id*

REF=N11601 (&ZMEVUSER *is set to* ID *of command requester*)

$$SYS.SHUTDOWN.CANCELLED

TYPE=SERVICEABILITY

RESOURCE=*nm-dom-id*

REF=N11603 (&ZMEVUSER *is set to* ID *of command requester*)

$$SYS.SHUTDOWN.COMMENCED

TYPE=SERVICEABILITY

RESOURCE=*nm-dom-id*

REF=N11701

$$SYS.TCPIP.ACTIVE

TYPE=SERVICEABILITY

REF=*type* (IBM, TCPAXS, VM, XNFTCP, *or* TISP)

$$SYS.TCPIP.FAILED

TYPE=SERVICEABILITY

REF=*type* (IBM, TCPAXS, VM, XNFTCP, *or* TISP)

$$SYS.TCPIP.INACTIVE

TYPE=SERVICEABILITY

REF=*type* (IBM, TCPAXS, VM, XNFTCP, *or* TISP)

$$SYS.TCPIP.QUIESCING

TYPE=SERVICEABILITY

REF=*type* (IBM, TCPAXS, VM, XNFTCP, *or* TISP)

$$SYS.TCPIP.RETRY

TYPE=SERVICEABILITY

REF=*type* (IBM, TCPAXS, VM, XNFTCP, *or* TISP)

$$SYS.TCPIP.STARTING

TYPE=SERVICEABILITY

REF=*type* (IBM, TCPAXS, VM, XNFTCP, *or* TISP)

$$SYS.TCPIP.STOPPING

TYPE=SERVICEABILITY

REF=*type* (IBM, TCPAXS, VM, XNFTCP, *or* TISP)

$$SYS.USER.LOGON

TYPE=CONFIGURATION

RESOURCE=(*user-id*,*lu-name*)

REF=N20E05

DATA=N20E05 *msg-text*

$$SYS.USER.LOGOFF

TYPE=CONFIGURATION

RESOURCE=(*user-id*,*lu-name*)

REF=N20E07

DATA=N20E07 *msg-text*

$$SYS.USER.FAIL

TYPE=SERVICEABILITY

RESOURCE=(*user-id*,*lu-name*) REF=*reason*

DATA=N20E08 *msg-text*

$$SYS.USER.CONTEXT.SWITCH

TYPE=CONFIGURATION

RESOURCE=(*lu-name*,*application*)

OBJECT=MAISESS *or* SOLVE DATA=MAI *session-id*,*acb-name*

Notes:

- User ID is available in &ZMEVUSER.
- For a jump to an MAI application, the OBJECT name will be MAISESS; otherwise, it will be SOLVE.

$$SYS.USER.DISCONNECT

TYPE=CONFIGURATION

RESOURCE=(*user-id*,*lu-name*)

REF=*disconnect-id*

$$SYS.USER.RECONNEC

TTYPE=CONFIGURATION

RESOURCE=(*user-id*,*lu-name*)

REF=*disconnect-id*

$$SYS.USER.TIMEOUT

TYPE=SERVICEABILITY

OBJECT=MAISESS

RESOURCE=(*lu-name*,*mai-appl*)

REF=N54T01

$$SYS.VTAM.ACB.OPEN

TYPE=SERVICEABILITY

RESOURCE=*acb-name*

OBJECT=PPO, CNM, *or* PRI

$$SYS.VTAM.ACB.CLOSE

TYPE=SERVICEABILITY

RESOURCE=*acb-name*

OBJECT=PPO, CNM, *or* PRI

$$SYS.VTAM.PPO.ENABLED

TYPE=SERVICEABILITY

RESOURCE=*ppo-acb-name*

$$SYS.VTAM.PPO.DISABLED

TYPE=SERVICEABILITY

RESOURCE=*ppo-acb-name*

$$SYS.VTAM.RCVCMD.FAIL

TYPE=SERVICEABILITY

OBJECT=PPO *or* SPO

REF=R0 *return-code*

DATA=N09B02 *msg-text*

$$SYS.VTAM.SPO.ENABLED

TYPE=SERVICEABILITY

RESOURCE=*primary-acb-name*

$$SYS.VTAM.SPO.DISABLED

TYPE=SERVICEABILITY

RESOURCE=*primary-acb-name*

$$SYS.VTAM.SPO.REDIRECTED

TYPE=SERVICEABILITY

RESOURCE=(*domain-id*)

Extended Data

For some events, when sysparm FTSFTM is YES, the data is extended by adding USER-ID: *userid* FROMDSN: *from*-DSN TODSN: *to*-DSN, where:

userid

Shows the user ID who issued the transmission request

***from*-DSN**

Shows the name of the data set being transmitted

***to*-DSN**

Shows the name of the data set receiving the transmitted data.

This applies to the following events:

- $$FTS.RECEIVE.START
- $$FTS.RECEIVE.STATS
- $$FTS.RECEIVE.FAIL

- $$FTS.TRANSMIT.START
- $$FTS.TRANSMIT.STATS
- $$FTS.TRANSMIT.FAIL

The extended data has extraneous spaces removed.

Appendix B: Supported Language Codes for National Language Support

The following language codes are supported:

Language	Language Code	Code Page
Belgian	BE	500
French Canadian	CF	037
Danish	DK	277
Austrian/German	GR	273
Spanish	SP	284
French	FR	297
Italian	IT	280
Japanese (Fujitsu)	NF	-
Japanese (Hitachi)	NH	-
Japanese (IBM)	NI	290
Dutch	NL	037
Norwegian	NO	277
Portuguese	PO	037
Swiss/German	SG	500
Swiss/French	SF	500
Swedish	SV	278
UK English	UK	285
US English	US	037

Appendix C: Supported Character Sets

This section describes the meaning of the TRANSLATE operand on the &DECODE and &ENCODE verbs.

This section contains the following topics:

Code Page Selection

When TRANSLATE=DEC is specified, the DEC character code page (see page 1190) is used for translation.

When TRANSLATE=ASCII is specified, the ASCII character code page (see page 1192) is used for translation.

When TRANSLATE=ISO is specified, the ISO character code page (see page 1193) is used for translation.

DEC Character Code Page

On an &DECODE operation, if the TRANSLATE=DEC operand is specified, all character strings are assumed to be from the DEC character set shown in the following table and are translated to their equivalents in the EBCDIC character set. An &ENCODE operation, however, assumes the character strings to be from the EBCDIC character set, and translates them from their equivalents in the DEC character set shown in the following table.

	0	1	2	3	4	5	6	7	A	B	C	D	E	F
-0	NUL	DLE	SP	0	@	P	'	p	nnbsp		À	*1	à	*2
-1	SOH	DCI	!	1	A	Q	a	q	¡	”	Á	Ñ	á	ñ
-2	STX	DC2	”	2	B	R	b	r	¢	2	Â	Ò	â	ò
-3	ETX	DC3	#	3	C	S	c	s	£	3	Ã	Ó	ã	ó
-4	EOT	DC4	$	4	D	T	d	t	¤	'	Ä	Ô	ä	ô
-5	ENQ	NAK	%	5	E	U	e	u	¥	*3	Å	Õ	å	õ
-6	AC	SYN	&	6	F	V	f	v	*4	¶	Æ	Ö	æ	ö
-7	BEL	ETB	'	7	G	W	g	w	§	.	Ç	x	ç	
-8	BS	CA	(	8	H	X	h	x	¨z	*5	È	Ø	è	ø
-9	HT	EM	)	9	I	Y	i	y	E	*6	É	Ù	é	ù
-A	LF	SUB	*	:	J	Z	j	z	ª	º	Ê	Ú	ê	ú
-B	VT	ESC	+	;	K	[	k	{	«	»	Ë	Û	ë	û
-C	FF	FS	,	<	L	\	l	|	:	*7	Ì	Ü	ì	ü
-D	CR	GS	-	=	M	]	m	}	shy	*8	Í	*9	í	*10
-E	SO	RS	.	>	N		n	~	®	*11	Î	*12	î	*13
-F	SI	US	/	?	O	_	o	DEL	¯	¿	Ï	b	ï	ÿ

Items marked with an asterisk (*) in the table are as follows:

*1	Capital letter D with stroke	*8	Vulgar fraction one half
*2	Small Icelandic letter eth	*9	Capital letter Y with acute accent
*3	Small Greek letter mu (micron sign)	*10	Small letter y with acute accent
*4	Broken bar	*11	Vulgar fraction three quarters
*5	Cedilla	*12	Capital Icelandic letter thorn
*6	Superscript one	*13	Small Icelandic letter thorn
*7	Vulgar fraction one quarter		

Columns 0 to 1 (code points 00 to 1F) contain a set of 32 control characters. On an &DECODE operation, those characters without an equivalent in the EBCDIC character set are translated to NULLS (X'00'). Similarly, on an &ENCODE operation, EBCDIC control characters without an equivalent in this code page are translated to the DEC NULL character (X'00').

Columns 2 to 7 (code points 20 to 7F) and columns A to F (code points A0 to AF) contain graphic characters. On an &DECODE operation, those characters without an equivalent in the EBCDIC character set are translated to SPACEs (X'40'). On an &ENCODE operation, EBCDIC graphic characters without an equivalent in this code page are translated to the DEC SPACE character (X'20').

Columns 8 to 9 (code points 80 to 9F) are control character code points not defined for this character code page. If they appear on an &DECODE operation, they are translated to the EBCDIC NULL character (X'00').

ASCII Character Code Page

On an &DECODE operation, if the TRANSLATE=ASCII operand is specified, all character strings are assumed to be from the ASCII character set as shown in the following table and are translated to their equivalents in the EBCDIC character set. An &ENCODE operation, however, assumes the character strings to be from the EBCDIC character set and translates them to their equivalents in the ASCII character set shown in the following table.

	0	1	2	3	4	5	6	7
-0	NUL	DLE	SP	0	@	P	'	p
-1	SOH	DC1	!	1	A	Q	a	q
-2	STX	DC2	”	2	B	R	b	r
-3	ETX	DC3	#	3	C	S	c	s
-4	EOT	DC4	$	4	D	T	d	t
-5	ENQ	NAK	%	5	E	U	e	u
-6	AC	SYN	&	6	F	V	f	v
-7	BEL	ETB	'	7	G	W	g	w
-8	BS	CAN	(	8	H	X	h	x
-9	HT	EM	)	9	I	Y	i	y
-A	LF	SUB	*	:	J	Z	j	z
-B	VT	ESC	+	;	K	[	k	{
-C	FF	FS	,	<	L	\	l	|
-D	CR	GS	-	=	M	]	m	}
-E	SO	RS	.	>	N		n	~
-F	SI	US	/	?	O		o	DEL

Columns 0 to 1 (code points 00 to 1F) contain a set of 32 control characters. On an &DECODE operation, those characters without an equivalent in the EBCDIC character set are translated to NULLS (X'00'). On an &ENCODE operation, EBCDIC control characters without an equivalent in this code page are translated to the ASCII NULL character (X'00').

Columns 2 to 7 (code points 20 to 7F) contain a set of 94 graphic characters, plus the SPACE and DELETE characters. On an &ENCODE operation, EBCDIC graphic characters without an equivalent in this code page are translated to the ASCII SPACE character (X'20').

Columns 8 to 9 (code points 80 to 9F) and columns A to F (code points A0 to AF) are respectively, control character and graphic character code points not defined for this code page. If they appear on an &DECODE operation, they are translated to the EBCDIC NULL (X'00') and SPACE (X'40') characters respectively.

ISO Character Code Page

On an &DECODE operation, if the TRANSLATE=ISO operand is specified, all character strings are assumed to be from the ISO character set shown below and are translated to their equivalents in the EBCIDIC character set. An &ENCODE operation, however, assumes the character strings to be from the EBCDIC character set and translates them to their equivalents in the ISO character set shown in the following table.

	0	1	2	3	4	5	6	7
-0	NUL	DLE	SP	0	@	P	'	p
-1	SOH	DC1	!	1	A	Q	a	q
-2	STX	DC2	”	2	B	R	b	r
-3	ETX	DC3	#	3	C	S	c	s
-4	EOT	DC4	*	4	D	T	d	t
-5	ENQ	NAK	%	5	E	U	e	u
-6	ACK	SYN	&	6	F	V	f	v
-7	BEL	ETB	'	7	G	W	g	w
-8	BS	CAN	(	8	H	X	h	x
-9	HT	EM	)	9	I	Y	i	y
-A	LF	SUB	*	:	J	Z	j	z
-B	VT	ESC	+	;	K	[	k	{
-C	FF	FS	,	<	L	\	l	|
-D	CR	GS	-	=	M	]	m	}
-E	SO	RS	.	>	N		n	¯
-F	SI	US	/	?	O	_	o	DEL

Columns 0 to 1 (code points 00 to 1F) contain a set of 32 control characters. On an &DECODE operation, those characters without an equivalent in the EBCDIC character set are translated to NULLS (X'00'). On an &ENCODE operation, EBCDIC control characters without an equivalent in this code page are translated to the ISO NULL character (X'00').

Columns 2 to 7 (code points 20 to 7F) contain a set of 94 graphic characters, plus the SPACE and DELETE characters. On an &ENCODE operation, EBCDIC graphic characters without an equivalent in this code page are translated to the ISO SPACE character (X'20').

Columns 8 to 9 (code points 80 to 9F) and columns A to F (code points A0 to AF) are respectively, control character and graphic character code points not defined for this code page. If they appear on an &DECODE operation, they are translated to the EBCDICNULL (X'00') and SPACE (X'40') characters respectively.

Appendix D: Processing Double Byte Character Set Data

This section contains the following topics:

About Double Byte Characters

Many Asian languages are written using symbols rather than letters. Because of the large number of symbols used, these languages require two bytes to represent each symbol (rather than one byte as used in languages such as English). These language representations are known as double byte character sets, or DBCS.

DBCS data is normally delineated by special characters known as shift characters. A string of data can contain a mixture of single byte (SBCS) and double byte data. A shift out character is used to mark the start of DBCS data and a shift in character marks the return to SBCS data.

Manipulating a DBCS string requires special care to preserve its integrity. The NCL language provides extensive support for DBCS data manipulation. This support is available in product regions executing with the SYSPARMS DBCS operand set to YES, IBM, or FUJITSU.

DBCS Support in NCL

Support for DBCS data manipulation in NCL is activated by the following &CONTROL options:

- &CONTROL DBCS
- &CONTROL DBCSN
- &CONTROL DBCSP

All three of these options alter the way in which NCL operates, to ensure that DBCS data is recognized and preserved. When any of these options are in effect, the string manipulation functions of NCL check for the presence of DBCS data. If a string is padded or truncated, the shift characters are automatically preserved-without the shift characters it is impossible to identify the DBCS

Note: On Hitachi systems, the character X'40' is considered to be a neutral character which can appear in both SBCS or DBCS data. When NCL calculates character boundaries in DBCS data, single X'40' characters are considered valid if the NCL procedure is executing on behalf of a user logged-on from a Hitachi 560/20 terminal, or if the procedure is executing in a background region (for example, BSYS region) and the operating system on which the region is executing is VOS3.

Terminals capable of displaying DBCS data exhibit differing display characteristics. Terminals such as the IBM 5550 family display shift characters as blank fields on the screen. On Fujitsu and Hitachi terminals the shift characters take no screen position.

NCL provides for these differing characteristics according to a control setting of the &CONTROL verb:

- &CONTROL DBCS treats shift characters as significant, including them in calculations of length and offset
- &CONTROL DBCSN considers shift characters are not significant; they are ignored in length and offset calculations
- &CONTROL DBCSP chooses between these options depending on the processing environment
- &CONTROL NODBCS treats all data as SBCS (valid mixed DBCS/SBCS data strings could be corrupted)

NCL Function Changes with &CONTROL DBCS Options

The remainder of this chapter describes the NCL functions that are affected by the &CONTROL DBCS, &CONTROL DBCSN, and &CONTROL DBCSP options. Those NCL functions are:

- &ASISTR
- &CONCAT
- &FNDSTR
- &LENGTH
- &OVERLAY
- &REMSTR
- &SELSTR
- &SETLENG
- &STR
- &SUBSTR

In the examples supplied for each function, the following symbols are used:

- A less than sign (<) is used to represent a shift out character
- A greater than sign (>) is used to represent a shift in character
- DBCS characters are represented as a period (.) followed by a single byte character (for example, <.A> represents a double byte A)
- An underscore (_) is used to represent a X'40' in a DBCS string or a single byte blank

&ASISTR

The &ASISTR built-in function is used to assign data to a variable. If an &CONTROL DBCS, DBCSN or DBCSP option is in effect, and the data ends with a DBCS string which does not contain a shift in character to identify the end of the DBCS string, the &ASISTR function adds a shift in character to the end of the data before assignment takes place. This is useful for avoiding syntax errors which occur due to processing incomplete DBCS strings.

Example: &ASISTR

```
&A contains the value <.A.B.C
&CONTROL DBCS&A = &ASISTR &A
```

results in

```
&A = <.A.B.C>
```

&CONCAT

The &CONCAT built-in function is used to concatenate several pieces of data to form a single string. With an &CONTROL DBCS, DBCSN or DBCSP option in effect, the concatenation of DBCS data removes any consecutive shift in or shift out characters in the resultant data string.

Example: &CONCAT

```
&CONTROL DBCS
&A = &CONCAT <.A> <.B> ABC D<.C> E
```

results in

```
&A = <.A.B>ABCD<.C>E
```

&FNDSTR

The &FNDSTR built-in function is used to locate a given data string (called the search argument) in another data string. This function requires special processing for DBCS data, to avoid a single byte character being located in a double byte string. For example, if searching for a single byte A (X'C1'), the second byte of a Kanji symbol represented by X'45C1' contains the X'C1' code, but it should not be considered a match.

The search argument is single byte data only, or double byte data only, or a mixture of single byte data and double byte data.

The search argument is stripped of consecutive shift out/shift in sequences before the scan is performed. For example, if the search argument is AA<>BB, the actual string searched for would be AABB. If the resulting string is null, a value of 0 will be assigned to the target variable.

If the search argument begins with a double byte character, the shift out is removed from the search argument before the scan is performed. If the search argument ends with a double byte character, the shift in character is removed from the search argument before the scan is performed. This allows a DBCS string to be located within another DBCS string.

If the search argument begins with a single byte character, a match only occurs in the case where the first character at the location at which the data was found, is a single byte character. If the search argument begins with a double byte character, a match only occurs where the first character at the location at which the data was found is a double byte character.

When using &CONTROL DBCS, DBCSN, or DBCSP, if the search argument is found in a section of data which is not the first section of data, a value of 999 is returned. This is the same as the processing with &CONTROL NODBCS.

Examples: &FNDSTR

```
&CONTROL DBCS
&A = &FNDSTR <> AAA<>BBB
```

results in

```
&A = 0
```

After the shift out and shift in, characters are removed from the search argument. No data remains so the value returned is 0.

```
&CONTROL DBCS
&A = &FNDSTR A<>B AAABBB
```

results in

```
&A = 3
```

After the shift out and shift in, characters are removed from the search argument. The search argument becomes AB.

```
&CONTROL DBCS
&A = &FNDSTR <.B> AAA<.A.B.C>
```

results in

```
&A = 7
```

After the shift out and shift in, characters are removed from the search argument The search argument becomes a double byte character .B, which exists at offset 7.

```
&CONTROL DBCSN
&A = &FNDSTR <.B> AAA<.A.B.C>
```

results in

```
&A = 6
```

After the shift out and shift in, characters are removed from the search argument. The search argument becomes double byte character .B, which exists at offset 6. The shift out character after AAA is not included in the calculation of the offset.

```
&CONTROL DBCS
&A = &FNDSTR A <.A.B.C>
```

results in

```
&A = 0
```

Although the value for single byte A and the value for the second byte of the double byte .A are the same, no match occurs because the search argument is a single byte character and .A is a double byte character.

```
&CONTROL DBCS&
A = &FNDSTR <.C>D <.A.B.C>DEF
```

results in

```
&A = 6
```

```
&A = &FNDSTR A<.B> AAA<.B.C.D>
```

results in

```
&A = 3
```

The search argument is a mixture of double byte data and single byte data.

```
&CONTROL DBCS
&A = &FNDSTR <.A> AAAA BBBB CC<.A>DD EEEE
```

results in

```
&A = 999 (string found but not in first piece of data)
```

&LENGTH

The &LENGTH built-in function is used to determine the length of a piece of data. If an &CONTROL DBCSN option is in effect (or a DBCSP option which is functioning as DBCSN), the length returned will be adjusted to exclude any shift characters from the length.

Examples: &LENGTH

```
&CONTROL DBCS
&A = &LENGTH AA<.A.B>BB
```

results in

```
&A = 10
```

Note: The shift characters are included in the length.

```
&CONTROL DBCSN
&A = &LENGTH AA<.A.B>BB
```

results in

```
&A = 8
```

Note: The shift characters are not included in the length.

&OVERLAY

The &OVERLAY built-in function is used to replace data, at a specified offset and for a specified length, with some other data. In the case where the data is single byte data only, this is a simple function. When DBCS strings are included, it is difficult to ensure that the integrity of DBCS strings in the data is preserved. The &OVERLAY process must consider the contents of the data at the start and end of the area which is being overlaid, to determine if any shift characters need to be added or removed to maintain valid DBCS strings.

With &CONTROL DBCS, DBCSN, or DBCSP in effect, the offset and length is subject to variation due to the contents of the data. For example, if the original string contains shift characters, and &CONTROL DBCSN is in effect, the shift characters are not included in the calculation of the offset or length of the data which is to be overlaid.

If additional shift out or shift in characters must be added to the resultant string to preserve the integrity of the DBCS data, and shift characters are included in the length of the string, the string could be truncated to ensure that the length of the overlaid area is not greater than the length specified. If any truncation occurs, it occurs at the right hand side of the data to be overlaid, regardless of the alignment option specified.

If the data at the start of the overlaid area is the second byte of a DBCS character, the offset is adjusted by one, to exclude the character from the overlaid area. To ensure that data following the overlaid area is at the same logical offset relative to the start of the data, after the operation, as it was before the operation, the length of the overlaid area will be reduced by one. If the data at the end of the overlaid area is the first byte of a DBCS character, the length of the overlaid area is reduced by one to ensure that the DBCS character remains complete.

If the data at the start of the overlaid area is SBCS data, and the data to be placed in the string starts with DBCS data, a *shift out* character will be added to the start of the data when it is inserted. If shift characters are counted in the length of the string, the length of the overlaid area will be reduced by one to compensate for the shift out character. If the data at the start of the overlaid area is DBCS, and the data to be placed in the string starts with SBCS data, a shift in character will be added to the start of the data when it is inserted. If shift characters are counted in the length of the string, the length of the overlaid area will be reduced by one to compensate for the shift character.

If the data at the end of the overlaid area is SBCS data, and the data to be placed in the string ends with DBCS data, a *shift in* character will be added to the end of the data when it is inserted. If shift characters are counted in the length of the string, the length of the overlaid area will be reduced by one to compensate for the shift out character. If the data at the end of the overlaid area is DBCS data, and the data to be placed in the string ends with SBCS data, a shift out character will be added to the front of the data when it is inserted. If shift characters are counted in the length of the string, the length of the overlaid area will be reduced by one to compensate for the shift character.

If the data to be inserted starts with a shift out character, it is removed to allow the data to start with a DBCS character, but the shift out character is not included in the length of the data being inserted, even if &CONTROL DBCS is in effect. Similarly, if the data ends with a shift in character, the shift in character is removed, but is not included in the length of the data. In this case, the shift out and shift in characters are present only to identify the data as DBCS data. This makes it possible to overlay DBCS data onto DBCS data, while &CONTROL DBCS is in use, without adjusting the overlay length for the shift out and shift in characters.

Alignment (left, right, and center) is supported under all circumstances. However, the pad characters can only be single byte characters.

Examples: &OVERLAY

```
&CONTROL DBCS
&A = &OVERLAY AAAA <.A> 2 1
```

results in

```
&A = AAAA
```

Note: The data has been truncated because the length of the area (1) is not sufficient for any DBCS data.

```
&CONTROL DBCS
&A = &OVERLAY AAAA <.A> 2 2
```

results in

```
&A = A<>A
```

An attempt has been made to insert a double byte character in the middle of a single byte string. However, the length of the area (2) is not large enough to contain a DBCS character as well as shift characters, so only the required shift out and shift in have been inserted.

```
&CONTROL DBCSN
&A = &OVERLAY AAAA <.A> 2 2
```

results in

```
&A = A<.A>A
```

Note: The shift out and shift in characters are not included in the length of the overlaid area.

```
&CONTROL DBCS
&A = &OVERLAY <.A.B.C.D.E> A 6 1
```

results in

```
&A = <.A.B.C.D.E>
```

There is not sufficient room in the overlay area for the data, due to the requirement to insert shift out and shift in characters, so the resulting string is unchanged.

```
&CONTROL DBCS
&A = &OVERLAY <.A.B.C.D.E> A 6 2
```

results in

```
&A = <.A.B><.D.E>
```

An attempt has been made to insert a single byte character in the middle of a double byte string. However the length of the area (2) is only enough for the insertion of shift in and shift out characters, so the data has been truncated.

```
&CONTROL DBCSN
&A = &OVERLAY <.A.B.C.D.E> A 5 2 ALIGNL-
```

results in

```
&A = <.A.B>A-<.D.E>
```

The single byte A has replaced the double byte character .C and one pad character has been required to maintain the total length of the string. The shift in and shift out characters are not included in the offset or length of the overlaid area.

```
&CONTROL DBCS
&A = &OVERLAY <.A.B.C.D.E> A 6 4 ALIGNR-
```

results in

```
&A = <.A.B>-A<.E>
```

Four bytes of DBCS data have been overlaid. Some shift in and shift out characters have been required, reducing the length of the data to 2, and the data has been inserted in a right aligned manner with padding to the left.

```
&CONTROL DBCS
&A = &OVERLAY <.A.B.C.D.E> <.Z.Z> 4 4
```

results in

```
&A = <.A.Z.Z.D.E>
```

The shift out and shift in characters in the data being inserted have not been included in the data length.

&REMSTR

The &REMSTR built-in function is used to split a section of data into two parts and assign the second part of the data to a target variable. The location at which the data is split is determined by the specification of a single character, which acts as a delimiter for the split operation. The only functional difference with an &CONTROL DBCS, DBCSN or DBCSP option in effect, is that the designated split character is a single byte character or a double byte character. With the &CONTROL NODBCS option in effect, only a single byte character is specified.

If the designated split character is a single byte character, the split can only occur where the character exists in a single byte section of the data. If the designated split character is a double byte character, the shift out and shift in characters are removed from the character, and the split can only occur where the split character is found in a double byte section of the data.

If the split occurs in a double byte section of data, a shift out character is added to the start of the resultant data, if required, to preserve the integrity of the DBCS string.

Examples: &REMSTR

```
&CONTROL NODBCS &A = &REMSTR (C) ABCDEF
```

results in

```
&A = DEF
```

```
&CONTROL DBCS&A = &REMSTR (C) ABCDEF
```

results in

```
&A = DEF
```

The &CONTROL NODBCS operation is identical to the &CONTROL DBCS operation.

```
&CONTROL NODBCS&A = &REMSTR (<.D>) ABC<.C.D.E>
```

This operation results in a syntax error, because a double byte character can only be specified if an &CONTROL DBCS, DBCSN or DBCSP option is in effect.

```
&CONTROL DBCS&A = &REMSTR (<.C>) ABC<.C.D.E>
```

results in

```
&A = <.D.E>
```

Note: A shift out has been added before the double byte character .D to maintain integrity of the DBCS string.

&SELSTR

The &SELSTR built-in function is used to split a section of data into two parts and assign the first part of the data to a target variable. The location at which the data is split is determined by the specification of a single character, which acts as a delimiter for the split operation. The only functional difference with an &CONTROL DBCS, DBCSN or DBCSP option in effect, is that the designated split character is a single byte character or a double byte character. With the &CONTROL NODBCS option in effect, only a single byte character is specified.

If the designated split character is a single byte character, the split can only occur where the character exists in a single byte section of the data. If the designated split character is a double byte character, the shift out and shift in characters are removed from the character, and the split can only occur where the split character is found in a double byte section of the data.

If the split occurs in a double byte section of data, a shift in is added to the end of the resultant data, if required, to preserve the integrity of the DBCS string.

Examples: &SELSTR

```
&CONTROL NODBCS
&A = &SELSTR (C) ABCDEF
```

results in

```
&A = AB
```

```
&CONTROL DBCS
&A = &SELSTR (C) ABCDEF
```

results in

```
&A = AB
```

The &CONTROL NODBCS operation is identical to the &CONTROL DBCS operation.

```
&CONTROL NODBCS
&A = &SELSTR (<.D>) ABC<.C.D.E>
```

This operation results in a syntax error, because a double byte character can only be specified if an &CONTROL DBCS, DBCSN or DBCSP option is in effect.

```
&CONTROL DBCS
&A = &SELSTR (<.D>) ABC<.C.D.E>
```

results in

```
&A = ABC<.C>
```

Note: A shift in has been added after the double byte character .C to maintain integrity of the DBCS string.

&SETLENG

The &SETLENG built-in function is used to assign data of a specific length to a target variable. If the original data is longer than the specified length, it is truncated. If the original data is shorter than the specified length, the data is padded with blanks before assignment takes place. If &SETLENG truncates DBCS data, the shift in character could be removed, causing an incomplete DBCS string to be produced. This could lead to syntax errors in later processing or undisplayable data being presented on the screen. With an &CONTROL DBCS, DBCSN, or DBCSP option in effect, a shift in is added to the end of the data if required.

When shift characters do not occupy a screen position, formatting tabular displays, such as selection lists, becomes difficult. This is because the columns of data are misaligned, due to the difference in the displayable length of the data and the length of the data contained in an NCL variable. With &CONTROL DBCSN or DBCSP in effect, &SETLENG overcomes this problem. It adjusts the length of data assigned to the target variable to ensure that if the shift characters are present and they do not occupy a position on the screen. The displayable length of the data will always be exactly the same as the length specified on the &SETLENG statement.

Examples: &SETLENG

```
&CONTROL NODBCS
&A = &STR <.A.B.C.D>
&A = &SETLENG 6
```

results in

```
&A = <.A.B.
```

No special consideration is given to the DBCS string.

```
&A = &STR  <.A.B.C.D>
&CONTROL DBCS
&A = &SETLENG 6
```

results in

```
&A = <.A.B>
```

Note: A shift in has been added to the end of the data. The length of the resultant data is 6.

```
&A = &STR <.A.B.C.D>
&CONTROL DBCSN
&A = &SETLENG 6
```

results in

```
&A = <.A.B.C>
```

Note: A shift in has been added to the end of the data. The length of the resultant data is 6. The shift characters are not counted in the final data length.

```
&A = &STR <.A.B>
&CONTROL DBCS
&A = &SETLENG 8
```

results in

```
&A = <.A.B>__
```

The length of the resultant data is 8. Two blanks have been added to the end of the data to ensure the resultant length is correct.

```
&A = &STR <.A.B.C.D>
&CONTROL DBCSN
&A = &SETLENG 8
```

results in

```
&A = <.A.B>____
```

The length of the resultant data is 8. The shift characters are not counted in the final data length. Four blanks have been added to the end of the resultant data to ensure that the resultant length is correct.

```
&A = &STR <.A.B>
&CONTROL DBCSP
&A = &SETLENG 8
```

results in

```
&A = <.A.B>__
```

if the terminal is an IBM DBCS terminal, or results in

```
&A = <.A.B>_____
```

if the terminal is a Fujitsu or Hitachi DBCS terminal.

The displayable length of the resultant data is 8 in both cases. Blanks have been added to the end of the resultant data in both cases to ensure that the resultant length is correct.

```
&COL1 = &STR data ....
&COL2 = &STR data ....
&COL3 = &STR data ....
&CONTROL DBCSP
&COL1 = &SETLENG 20
&COL2 = &SETLENG 40
&COL3 = &SETLENG 10
&CONTROL NODBCS
&LINE1 = &CONCAT  &COL1 &COL2 &COL3
```

This example is creating a variable called &LINE1, which is part of a tabular display in which the first column starts at column 1, the second starts at column 21, and the third starts at column 61. Regardless of the contents of the data, and regardless of the terminal on which the data will be displayed, the columns will always be at the correct offset. This is because &SETLENG always ensures that the displayable length of the data is exactly as requested.

&STR

The &STR built-in function is used to assign data to a variable. If an &CONTROL DBCS, DBCSN, or DBCSP option is in effect, and the data ends with a DBCS string that does not contain a shift in character to identify the end of the DBCS string, the &STR function adds a shift in character to the end of the data before assignment takes place. This is useful for avoiding syntax errors that occur due to processing incomplete DBCS strings.

Example: &STR

```
&A contains the value <.A.B.C
&CONTROL DBCS
&A = &STR &A
```

results in

```
&A = <.A.B.C>
```

&SUBSTR

The &SUBSTR built-in function is used to extract data from within a string. If the data is single byte data only, this is a simple function. When DBCS strings are included, the standard &SUBSTR function can cause problems if the data within the part of the string which is defined by the offset and length operands contains DBCS data. The data that is extracted can contain shift characters that do not occur in shift out/shift in pairs. This can lead to problems, such as syntax errors, in later processing.

With &CONTROL DBCS, DBCSN, or DBCSP in effect, the offset and length is subject to variation due to the contents of the data. For example, if the original string contains shift characters, and &CONTROL DBCSN is in effect, the shift characters are not included in the calculation of the offset or length of the data that is to be extracted. If additional shift characters need to be added to the resultant string to preserve the integrity of the DBCS data, and they are included in the length of the string, the string could be truncated to ensure that the length of the extracted data is not greater than the length requested.

If the data at the specified offset is the second byte of a DBCS character, the character is replaced by a shift out character if shift characters are included in the string length, or by a single byte blank and a shift out character if shift characters are not included in the string length. This ensures that the following DBCS character extracted remains at the same logical offset in the resultant string.

If the data at the end of the extracted string is the first byte of a DBCS character, the character is replaced by a shift in character if shift characters are included in the string length, or by a shift in character and a single byte blank if shift characters are not included in the string length.

If the extracted data begins with a DBCS character, a shift out character is added to the front of the resultant string, to preserve the integrity of the DBCS string. If the extracted data ends with a DBCS character, a shift in character is added to the end of the resultant string, to preserve the integrity of the DBCS string.

If the original data string is shorter that the sum of the offset and length operands, no blank padding is added to the end of the resultant data string. However, a shift in is added if necessary.

If the data to be extracted is part of a DBCS string and the length specified is not large enough to allow a whole DBCS character to be extracted, the target variable is set to null.

Examples: &SUBSTR

```
&CONTROL DBCS
&A = &SUBSTR AA<.A.B.C.D> 1 5
```

results in

```
&A = AA<>_
```

Note: The data was truncated to ensure that the final data length did not exceed the requested length. The shift characters are included in the length of the string. The resultant data has one trailing blank.

```
&CONTROL DBCSN
&A = &SUBSTR AA<.A.B.C.D> 1 5
```

results in

```
&A = AA<.A>_
```

Note: The data would have ended on the first byte of a DBCS character (.B) which would be invalid, so the data has been truncated to remove the first byte of this character. The shift characters are not counted in the length of the string. The resultant data has one trailing blank.

```
&CONTROL DBCS
&A = &SUBSTR AA<.A.B.C.D> 4 5
```

results in

```
&A = _<.B>
```

Note: The shift characters are counted in the offset and length of the string. A single blank has been added to the start of the data because the offset specified was the first byte of a DBCS character. Because of the requirement to add a shift in character, the data has been truncated.

```
&CONTROL DBCSN
&A = &SUBSTR AA<.A.B.C.D> 5 2
```

results in

```
&A = <.B>
```

```
&A = &SUBSTR AA<.A.B.C.D> 5 4
```

results in

```
&A = <.B.C>
```

Note: The shift characters are not counted in the offset and length of the string. No data truncation is required, because the shift characters are not included in the length of the string.

```
&CONTROL DBCS
&A = &SUBSTR AA<.A.B.C.D> 6 1
```

results in &A being null.

```
&A = &SUBSTR AA<.A.B.C.D> 6 2
```

results in &A being null.

In both cases, the data being extracted is from a DBCS string, and the length specified is not large enough for a DBCS character, so the target variable is set to null.

Appendix E: &SOCKET Verbs

The &SOCKET verb set enables NCL processes to use the TCP/IP socket interface.

This section contains the following topics:

About the Socket Interfaces

A socket is an end point for interprocess communication over a network running TCP/IP. The socket interfaces support a number of underlying transport mechanisms. Sockets can simultaneously transmit and receive data from another process, using methods that depend on the type of socket being used. Sockets is of the following types, each representing a different type of communications service:

- TCP sockets
- UDP sockets

Note: &SOCKET verbs only support IPv4 address types.

TCP Sockets

TCP (Transmission Control Protocol) sockets provide reliable, connection-based communications. In the case of a sockets interface, the two processes must establish a logical connection with each other. The data is a stream of bytes that is sent without errors or duplication, and is received in the same order in which it was sent.

The following sections describe the various types of TCP socket applications and the &SOCKET verbs you would use for each type.

TCP Server Verb Set

&SOCKET ACCEPT

Accepts connection from clients

&SOCKET CLOSE

Closes the client connection

&SOCKET RECEIVE

Receives data from clients

&SOCKET REGISTER

Registers a server

&SOCKET SEND

Sends data to clients

&SOCKET TRANSFER_REQUEST

Transfers a socket ID from one NCL process to another

&SOCKET TRANSFER_ACCEPT

Accepts a socket ID from a donor NCL process

Note: For details of these verbs, see Verbs and Built-in Functions (see page 29).

TCP Client Verb Set

&SOCKET CLOSE

Closes the server connection

&SOCKET CONNECT

Connects to the server

&SOCKET RECEIVE

Receives data from the server

&SOCKET SEND

Sends data to the server

Note: For details of these verbs, see Verbs and Built-in Functions (see page 29).

UDP Sockets

UDP (User Datagram Protocol) sockets communicate by way of discrete messages called datagrams, which are sent as packets. UDP sockets are connectionless. Communication processes do not have a logical connection with each other and therefore the delivery of their data is unreliable. The datagrams is lost or duplicated, or they might not arrive in the same order in which they were sent.

The following sections describe the UDP socket application and the &SOCKET verbs you would use.

UDP Sockets Verb Set

&SOCKET CLOSE

Closes the communication socket

&SOCKET OPEN

Opens the communication socket and port

&SOCKET RECEIVE_FROM

Receives datagrams

&SOCKET SEND_TO

Sends datagrams

Note: For details of these verbs, see Verbs and Built-in Functions (see page 29).

Host Verb Set

&SOCKET PING

Pings the host

&SOCKET TRACEROUTE

Traces the route to the host

The PING and TRACEROUTE verbs directly access the lower layer protocols such as Internet Protocol (IP) and Internet Control Message Protocol (ICMP).

The PING and TRACEROUTE verbs may not be supported by all interfaces.

Note: For details of these verbs, see Verbs and Built-in Functions (see page 29).

Name Services Verb Set

&SOCKET GETHOSTBYNAME

Obtains IP address for a specified host name

&SOCKET GETHOSTBYADDR

Obtains host name information for a specified IP address

The GETHOSTBYNAME and GETHOSTBYADDR verbs request Domain Name Services (DNS) functions that translate an IP address into a host name and conversely.

Note: For details of these verbs, see Verbs and Built-in Functions (see page 29).

Socket Built-in Functions

The socket interfaces support built-in functions that you can use to obtain information about socket processes.

- To determine if a function is supported, use the &ZTCPSUPP function.
- To obtain information about the local host, use the &ZTCPINFO function.
- To obtain information about a specific socket, use the &ZSOCINFO function.
- To obtain information about error codes, use the following functions:
  - &ZTCPERNM for the logical name of a TCP/IP error code
  - &ZTCPERDS for a short message about a TCP/IP error code.

Note: For details of these verbs, see Verbs and Built-in Functions (see page 29).

System Variables

The following system variables are available within the TCP/IP &SOCKET verb set.

&ZTCP

Indicates the status of the socket interface

&ZTCPHSTA

Contains the value of the local host's IP address

&ZTCPHSTF

Contains the value of the local host's full name

&ZTCPHSTN

Contains the value of the local host's short name

&ZSOCID

Contains the socket ID of the last referenced socket

&ZSOCHNM

Contains the host name of the host referenced by some requests, such as &SOCKET GETHOSTBYADDR

&ZSOCFHNM

Contains the full host name of the host referenced by some requests, such as &SOCKET GETHOSTBYADDR

&ZSOCHADR

Contains the IP address of the host referenced by some requests, such as &SOCKET GETHOSTBYNAME

&ZSOCCID

Contains the socket ID used by the interface; for example, for IBM TCP/IP the internal socket number (a small number). Is used to identify an NCL socket or display produced by TCP/IP; for example, the NETSTAT command.

&ZSOCERRN

Contains the last ERRNO value returned from an &SOCKET request

&ZSOCPRT

Contains the port number of the last referenced socket

&ZSOCTYPE

Indicates the socket type of the last referenced socket

&ZSOCVERR

Returns vendor error information-format is specific to the TCP/IP interface in use

Note: For details of these verbs, see System Variables (see page 733).

Sample Code for TCP and UDP &SOCKET Verbs

Sample code for TCP and UDP &SOCKET verbs is available online. It is located in the CC2DEXEC file under the names given for the following examples.

Examples of Using TCP &SOCKET Verbs

Following are examples of using TCP &SOCKET verbs.

$NMSATC1-TCP Socket Server

This sample server procedure accepts connections and transfers them to new NCL processes that are started by the server to service the connections.

To invoke the server, you must specify a port number.

```
$NMSATC1 PORT=port_number
```

This procedure starts the procedure $NMSATC2 as a new process to service each new connection.

This sample also demonstrates the use of an asynchronous &SOCKET verb.

This sample works in conjunction with the following sample procedures:

- $NMSATC2 - Command Processor
- $NMSATC3 - Command Client

$NMSATC2 - Command Processor

This sample procedure is started by $NMSATC1 (the server) and has a connection transferred to it. This procedure accepts the connection, receives a command, and issues the command. It then reads the responses from the command and sends them back to the requester. The requests and responses are received and sent in ASCII to demonstrate how to do ASCII/EBCDIC translation.

Any errors that this procedure encounters are written to the log.

This sample works in conjunction with the following sample procedures:

- $NMSATC1 - TCP Socket Server
- $NMSATC3 - Command Client

You can also use Telnet instead of $NMSATC3 as the client program.

$NMSATC3 - Command Client

This sample client procedure sends a command and receives responses using the &SOCKET verb. It works in conjunction with the following sample server application procedures:

- $NMSATC1
- $NMSATC2

This procedure demonstrates ASCII/EBCDIC conversion, because all data sent and received is in ASCII.

```
$NMSATC3 ipAddr=
hostName=
port=
command="command to be executed"
```

Note: The command must be in quotes.

You can use a Telnet client instead of this program to send commands to the server (procedures $NMSATC1 and $NMSATC2).

$NMSATC4 - SMTP Client

This sample SMTP client procedure sends mail to a user. The contents of the mail are hard-coded in this procedure, but the procedure can easily be modified so that the text to be sent is passed to it.

```
$NMSATC4 SENDER=USERID@COMPANY.COM
RECIPIENT=USERID@COMPANY.COM
SMTPSVR=SMTP_SERVER_NAME_OR_ADDR
```

You should read RFC821 for an understanding of the SMTP protocol. This RFC is available at the following web address: http://www.ietf.org

Socket Interface Feedback and Error Codes

The error codes that relate to problems encountered when attempting to use the &SOCKET verbs are:

- Feedback codes (&ZFDBK)
- Socket error codes (&ZSOCERRN)
- Vendor-specific codes (&ZSOCVERR)

Note: There are also return codes used by the socket interfaces. These are documented with the verbs.

More information:

TCP/IP Feedback Codes (&ZFDBK) (see page 1220)
TCP/IP Socket Errors (&ZSOCERRN) (see page 1223)
Interpreting Vendor-specific Error Codes (&ZSOCVERR) (see page 1227)

TCP/IP Feedback Codes (&ZFDBK)

It is possible that an unlisted return code is an obsolete return code from a previous release of your TCP/IP product.

1

storage failure

Determine why the address space is experiencing storage problems

2

socket failed

See SOCERRN (see page 1223) code

3

sendto failed

See SOCERRN (see page 1223) code

4

recvfrom failed

See SOCERRN (see page 1223) code

7

bind failed

See SOCERRN (see page 1223) code

9

listen failed

See SOCERRN (see page 1223) code

10

accept failed

See SOCERRN code

12

getclientid failed

Contact Technical Support.

13

send failed

See SOCERRN (see page 1223) code

14

receive failed

See SOCERRN (see page 1223) code

15

socket_ID invalid

Check the ID= parameter on the &SOCKET verb

16

connect failed

See SOCERRN (see page 1223) code

17

invalid data

Contact Technical Support

18

attach failed

Contact Technical Support

21

gethostbyname failed

See SOCERRN (see page 1223) code

22

subtask terminated or abended

Check whether a TCPIP STOP has been done; otherwise contact Technical Support

23

setsockopt failed

See SOCERRN (see page 1223) code

24

invalid IP address specified

Specify valid IP address

25

MDO error: see &ZMDORC and &ZMDOFDBK

Note: For more information, see the *Network Control Language Programming Guide*.

26

Request has been flushed

Investigate why the NCL procedure was flushed and restart if required

27

getsockname failed

Contact Technical Support

29

hopnumber too low

Increase the number of hops specified on the Trace TCP/IP Route panel

30

givesocket failed

Contact Technical Support

31

takesocket failed

Contact Technical Support

34

gethostbyaddress failed

See SOCERRN (see page 1223) code

35

Invalid NCL ID

Specify valid NCL ID

36

Ping request not accepted

See SOCERRN (see page 1223) code

37

Traceroute request not accepted

See SOCERRN (see page 1223) code

38

Buffer overflow occurred

Supply smaller amount of data

39

MDO too large

Decrease number of pings or hops

96

TCP/IP QUIESCE command entered

No action

97

socket being closed

No action-socket closed or process flushed

98

invalid request

Contact Technical Support

99

socket interface not initialized

Issue the TCPIP START command from OCS

TCP/IP Socket Errors (&ZSOCERRN)

It is possible that an unlisted return code is an obsolete return code from a previous release of your TCP/IP product.

3—ENOMEM

Out of memory

Determine why the address space is experiencing storage problems

7—EUNSUPP

Unsupported I/O operation

Contact CA Technical Support.

23—EACCES

User or program lacks adequate permission to access this socket. Permission denied

Ensure that the user ID of the address space is in the OBEY list of IBM TCP/IP in the TCPIP.PROFILE.TCPIP file.

28—EDESTADDRREQ

Socket operation requires a destination address

Contact CA Technical Support.

29—EMSGSIZEA

UDP socket could not accommodate a message as large as this one

Contact CA Technical Support.

34—EOPNOTSUPP

The call does not support this type of socket

Contact CA Technical Support.

37—EADDRINUSE

The given address is already in use

Find the process which has the port registered

39—ENETDOWN

Cannot talk to the networking software on this local machine, or the host's network is down.

Check your network and/or TCP/IP stack

40—ENETUNREACH

This host cannot find a route to the specified destination network

Check your network and/or TCP/IP stack

41—ENETRESET

The remote host is not communicating over the network at this time

Check the status of the remote host

42—ECONNABORTED

The local communications software aborted the connection

Check the status of the TCP/IP address space

43—ECONNRESET

The peer process has reset the connection

Check the peer and determine the reason for its termination

44—ENOBUFS

The operating system did not have enough memory to perform the requested operation

Check your network and/or TCP/IP stack

47—ESHUTDOWN

The connection has been shutdown

Contact CA Technical Support

48—ETIMEDOUT

The destination host did not respond to a connection request

49—ECONNREFUSED

The destination host refused the socket connection

Check the status of the remote host

50—EHOSTDOWN

Socket operation failed because the destination host was down

Ensure that the destination host is active and retry

51—EHOSTUNREACH

Socket operation failed because the destination host is unreachable

Ensure that the destination host is active and retry

52—EPIPE

The peer process closed its socket while the local task was still writing data to the connection

Check the status of the remote host

63—EIO

I/O error occurred

Check the TCP/IP region for messages

64—ECONNCLOSED

Connection closed by peer

Determine why peer closed connection

65—ESOCKCLS

Socket closed

No action-socket closed by process

66—ENOPIDS

No process IDs available for ping or traceroute

Retry operation

67—ENOPORTS

No ports available for traceroute

Retry operation

68—ESHPORTCLS

Shared port closed

No action-shared port closed by process; will be opened later in processing

69—EUNKSERVER

Unknown server

Check the server name and then retry.

70—EINVPORTNUM

Invalid port number

Check the port number and then retry.

71—ESERVERNAMEINUSE

The given server is already in use

Find the process which is using the port.

72—EDNRNOTFND

Host name or address not found for SOLVE DNR

Check the host name and then retry.

73—EDNRNORSP

SOLVE DNR timed out

Retry operation

74—EDNRBADNAME

Invalid domain name for SOLVE DNR

Check the domain name and then retry.

75—EDNRERROR

SOLVE DNR error (for example, send/receive error or storage error)

Check descriptive text returned with error code

999—EOTHER

Vendor-specific error

See ZSOCVERR (see page 1227) and vendor TCP/IP error codes

Interpreting Vendor-specific Error Codes (&ZSOCVERR)

Vendor-specific errors have an error number of 999 and an additional VERRIN (vendor-specific error) code. The interpretation of this error code is different depending on the vendor of the TCP/IP software. These error codes appear in messages and in the NCL system variable &ZSOCVERR.

Interpreting CA TCPaccess CS Systems Error Codes

For CA TCPaccess CS, the &ZSOCVERR (VERRIN) system variable is in *one* of the following formats:

- Format A: 04/*aa-bb-cccc*
- Format B: *nn-xxxx*

Decoding Format A

The A-format 04/*aa-bb-cccc* is decoded as follows:

04

A TPL function received a return code of 4

aa

Recovery action code (the value of TPLACTCD in hexadecimal notation)

bb

Specific error code (the value of TPLERRCD in hexadecimal notation)

cccc

Diagnostic code (that is, the value of TPLDGNCD in hexadecimal notation)

For more information, see the *CA TCPaccess Communications Server Unprefixed Messages and Codes* guide and perform the following steps:

1. Locate the chapter “API Return Codes”.
2. Among the RTNCD *aabb* page titles, locate the first two sets of digits (*aa-bb*).
3. For each of these titles, locate the table that contains the various diagnostic code values *(cccc*).

Decoding Format B

The B-format *nn-xxxx* is decoded as follows:

nn

General return code (the R15 value from a TPL function request)

xxxx

Diagnostic code (the low half of R0 from the TPL function request, hexadecimal expanded)

For more information, see the *CA TCPaccess Communications Server Unprefixed Messages and Codes* guide and perform the following steps:

1. Locate the chapter “API Return Codes”.
2. Locate TPL-Based General Return Codes.
3. For each return code value (*nn*), locate the table that contains the diagnostic code values *(xxxx*).

Interpreting IBM Systems Error Codes

The &ZSOCVERR (VERRIN) system variable contains the IBM TCP/IP socket ERRNO value. This is translated into the &ZSOCERRN value.

The IBM TCP/IP socket ERRNO value is displayed as a decimal number.

Note: For z/OS V1.2 or later, see *IBM Communications Server IP and SNA Codes* (SC31-8791) for the meaning of the value.

TCP/IP Vendor Interface Restrictions and Limitations

This section describes the restrictions and limitations of each TCP/IP vendor interface. It covers the following vendor interfaces:

- CA TCPaccess Communications Server
- IBM Communications Server

Vendor interface restrictions and limitations are particularly relevant to NCL &SOCKET programming.

CA TCPaccess CS

The CA TCPaccess interface uses the assembler macro TLI interface to connect to CA TCPaccess CS.

This interface has the following restrictions:

- The CA TCPaccess CS configuration parameters limit the maximum size of a UDP datagram that is sent or received. As distributed, this limit is 9000 bytes.
- By default, the system uses global DNS, except for obtaining the local host name, when the system requests local DNS.

IBM Communications Server

The system interfaces to IBM's Communications Server using the TCP/IP macro-level interface, which uses the HPNS (High Performance Native Sockets) facility.

This interface has the following restrictions:

- To use the PING and TRACEROUTE functions, the region RACF user ID must be in the TCP/IP OBEY list.
- The user ID must have an OMVS segment with a UID of 0.