Reference Manual

NCI/XF

Version 4.1

MVS Operating Environment

Publication number NCR0001.002

PassGo

© Copyright 2001 PassGo Technologies Ltd. All rights reserved.

Proprietary and Confidential Information of PassGo Technologies Ltd.

Published by:

PassGo Technologies Ltd, Horton Manor, Ilminster, Somerset, TA19 9PY, England.

Publication number NCR0001.002

Second Edition (September 2001)

This book refers to a number of hardware and software products that are produced by other companies. In most, if not all cases, the names of these products are claimed as trademarks by the companies that manufacture them. It is not our intention to claim either the products or their names or trademarks as our own.

Changes will be made periodically to the information contained in this book. If your book does not accurately reflect the level of product you are using, this may be due to a fix being applied to the product which has still to be released as a book update.

Preface

Purpose of this book

This book provides technical information relating to the NCI/XF programming language including a full description of NCI/XF language statements and variables.

The complete range of books associated with NCI/XF is as follows:

NCI/XF Reference Manual (this manual)

NCI/XF User Guide

NC-COMPILE Programming Reference Manual

NCI/XF Installation Manual

Who should read this book

This book is intended for use by NCI/XF programmers.

National characters

All references to national characters in this manual are in English (U.S.) format. The table below shows four commonly-used national characters with their English (U.K.), French and German equivalents:

Hex Value	English (U.S.)	English (U.K.)	French	German
X'4A'	¢	$	°	Ä
X'5B'	$	£	$	$
X'7B'	#	#	£	#
X'7C'	@	@	à	§

How to use this manual

This NCI/XF Reference Manual relates to NCI/XF version 4.1 running in an IBM MVS operating environment. It is intended for readers who are familiar with the contents of the NCI/XF User Guide.

The NCI/XF Reference Manual chapters:

Chapter 1 - NCI/XF language statements

Chapter 2 - NCI/XF defined system variables

Chapter 3 - NCI/XF defined user variables

Chapter 4 - NC-VSAM routines

should be read after familiarization with the NCI/XF User Guide chapters:

Chapter 1 - NCI/XF overview

Chapter 2 - How to code NCI/XF

The NCI/XF Reference Manual chapters:

Chapter 5 - NC-SAM routines

Chapter 6 - NC-MHO routines

Chapter 7 - NC-XMS routines and macros

Chapter 8 - NELOGON and NC-EDIT variables

should be read after familiarization with the NCI/XF User Guide chapters:

Chapter 6 - A guide to NC-SAM

Chapter 7 - A guide to NC-MHO

Chapter 8 - A guide to NC-XMS

Chapter 9 - A guide to NC-EDIT

Changes will be made periodically to all NCI/XF documents to reflect new facilities.

A reader's comment form is provided at the back of this manual. Any information supplied will be used and distributed by the authors in any way they consider appropriate without incurring any obligation whatsoever.

Table of contents

Preface
Chapter 1 NCI/XF language statements
Chapter 2 NCI/XF defined system variables
Chapter 3 NCI/XF defined user variables
Chapter 4 NC-VSAM routines
Chapter 5 NC-SAM routines
Chapter 6 NC-MHO routines
Chapter 7 NC-XMS routines and macros
Chapter 8 NELOGON and NC-EDIT variables
Chapter 9 NCI/XF panels and distributed routines
Chapter 10 Macros in (F)Call routines
Chapter 11 Messages and abend codes
Index

This page intentionally left blank

Chapter 1 - NCI/XF language statements

Introduction 1.4
* 1.5
A2E 1.6
ACQuire 1.8
ALARM 1.9
ALIAS 1.10
ASSIGN 1.11
ATTRibute 1.14
AUTOinput 1.18
BEEP 1.19
C2D 1.20
C2X 1.21
CALCulate 1.22
CALL 1.23
CHECKFix 1.24
CHECKPanel 1.25
COLumns 1.26
CONNect 1.27
CONNECTR 1.29
CURsor 1.32
D2C 1.35
D2P 1.37
DECRement 1.38
DECRYPTion 1.39
DEFINE 1.40
DELete 1.41
DEQueue 1.42
DIRECT 1.43
DISable 1.44
E2A 1.45
ELSE 1.47
ENAble 1.48
ENCRYPTion 1.49
ENQueue 1.50
EXECute 1.51
EXIT 1.52
FCALCulate 1.54
FCALL 1.56
GETLog 1.57
GLOBALV 1.58
GOTO 1.59
GRESET 1.61
IF 1.62
INCRement 1.71
INTerval 1.73
LABel 1.74
LENgth 1.75
LOAD 1.76
LOG 1.77
LOGLevel 1.79

LOGOFF 1.80
Message 1.81
MODStats 1.86
NOALARM 1.89
NOBEEP 1.90
OR 1.91
PADCenter 1.92
PADLeft 1.93
PADRight 1.94
PANel 1.95
PANStats 1.98
PARse 1.102
POST 1.104
PREFresh 1.105
QUERYTErm 1.110
QUEue 1.111
QUEue ADD 1.112
QUEue BLDINDEX 1.117
QUEue COPY 1.119
QUEue DELETE 1.124
QUEue FIND 1.126
QUEue GFIND 1.138
QUEue GPURGE 1.143
QUEue PURGE 1.147
QUEue REPLACE 1.148
QUEUE SORT 1.150
RESET 1.151
RESETAttr 1.152
RESETExec 1.153
RESTART 1.154
RETURN 1.155
REVerse 1.156
ROWs 1.157
SCAN 1.158
SET 1.159
SHUTDown 1.161
STATus 1.162
STRING 1.167
STRIPCent 1.168
STRIPEmbed 1.169
STRIPLeft 1.170
STRIPRight 1.171
SWITCHlog 1.172
TERMUPdate 1.173
TRACE 1.174
TRACEExec 1.178
TRACETerm 1.179
TRACEVar 1.182
TRACEVErb 1.183
TRANslate 1.184
TREFresh 1.185
TRMStats 1.190
TRUNCCent 1.193
TRUNCLeft 1.194
TRUNCRight 1.195
TURN 1.196

TYPe 1.199
UNLOAD 1.200
UNSTRING 1.201
UPDate 1.202
UPPERcase 1.204
UVBL 1.205
VALidate 1.206
VARiable 1.209
VARNAMEL 1.210
VERify 1.211
VEXECute 1.213
WAIT 1.214
WTO 1.215
X2C 1.216

Introduction

This chapter should be read only when you are familiar with the NCI/XF User Guide, *Chapter 2 - How to code NCI/XF*.

Note that on execution of a language statement, return codes are set in variable *&usrretc.*

Conventions used to describe language statements

In this chapter, capital letters in the language statement name at the top of each statement description show the minimum abbreviation for that statement. For example, the statement name QUEue FIND shows that the minimum abbreviation for the QUEUE FIND statement is QUE FIND.

Notational conventions

Each language statement has a section entitled *Format*. The following notations are used in the *Format* section for language statement syntax:

Square brackets ([ ])	are used to indicate that the enclosed parameters or keywords are optional. Default values are stated in the text as appropriate.
Ellipsis (....)	are used to indicate that the preceding item or group of items can be repeated more than once in succession.

Note: None of the above should be entered when the command is issued.

Uppercase letters	are used to represent values you must type without change.
Lowercase italic letters	are used to represent parameters which should be replaced by appropriate values prior to execution.

Syntax errors

It should be noted that in some cases a syntax error will result in the display of an error panel.

*

The COMMENT statement can be used in any of the panel sections and, because it uses only disk space, its overheads are nil. You are encouraged to document your panels thoroughly with the COMMENT statement.

Format

* *text of comment*

where

text of comment can be any characters.

Example

```
*This is a comment.
```

A2E

The A2E statement converts an ASCII character string to EBCDIC.

Format

A2E *operand1* = *operand2* [*table-suffix*]

where:

operand1 is the variable to receive the converted data. This may be a variable name or a variable variable. The converted data will be EBCDIC.

operand2 is the value or variable to be converted. This can be a literal, a variable, concatenated variables, a variable variable, a substring of a variable or any combination of these. The maximum length of data string for conversion is 32,767 bytes.

table-suffix is a literal or variable giving the suffix part of the name of a table of translation values from ASCII to EBCDIC. Available suffixes are:

N001 - US English (same as standard table)

N033 - French

N044 - UK English

N049 - German

For other suffixes see your system administrator.

If *table-suffix* is not coded, the standard table will be used (US English).

Note: The table used for translation should match the language set up on the system on which your NCI program will run.

Related statements

C2D

C2X

D2C

D2P

E2A

X2C

Return codes

After execution, variable *&usrretc* will be set to one of the following return codes:

0 translation successful.
24 *table-suffix* was coded but the corresponding module did not exist or was found to be in error.

Examples

```
X2C &ascii = 414243
A2E &ebcdic = &ascii
```

The X2C command converts the hexadecimal value 414243 to the ASCII characters ABC and sets the result in variable *&ascii*; the A2E command converts the characters in *&ascii* to EBCDIC and sets the result in variable *&ebcdic*.

```
X2C &ascii = 414243
A2E &ebcdic = &ascii tab2
```

The X2C command converts the hexadecimal value 414243 to the ASCII characters ABC and sets the result in variable *&ascii*; the A2E command converts the characters in *&ascii* to the three EBCDIC equivalent characters of ASCII literal ABC, according to the user supplied translation table assembled as module NCAETAB2.

ACQuire

The ACQUIRE statement allows a terminal to be remotely connected to NCI/XF. The ACQUIRE request is executed asynchronously from the issuing panel; execution of the panel is not halted and continues immediately with the statement that follows the ACQUIRE statement.

Format

ACQUIRE *termid* *result-var* [*queue-option*]

where

termid is the name of a terminal or variable containing the name of a terminal to be acquired.

result-var is a system variable name which will contain the result (success or failure) of the ACQUIRE statement. Specify the actual variable name without a leading ampersand (&) or the name of a variable with a leading ampersand (&) containing the required variable name. The result variable MUST be a system variable starting with SYS or an asterisk (*).

queue-option is either QUEUE or NOQUEUE, which specifies whether the ACQUIRE will be queued if the terminal is already in session with another application. In this situation the terminal will only be acquired when the current session, and any sessions queued ahead of the ACQUIRE, have ended. The default is NOQUEUE, in which case the terminal is not acquired if it is already in session with another application.

Return codes

After execution of an ACQUIRE statement, variable *&usrretc* is set to one of the following return codes:

0 The ACQUIRE has been accepted (passed to VTAM); *result-var* will subsequently contain a code to reflect the success or failure of the command.
8 The terminal *termid* is already connected to NCI/XF.
16 Syntax error. For example:
- two operands not specified.
- *termid* is not a valid terminal name.
- *result-var* is not a valid system variable name.

When *&usrretc* is zero, the return code set in variable *result-var* will be one of the following:

0 Terminal has been ACQUIREd successfully and has logged on to NCI/XF.
4 Terminal could not be ACQUIREd, for instance bind failure when logging on to NCI/XF.

Result-var is set asynchronously and might not be set immediately the ACQUIRE verb completes.

ALARM

The ALARM statement allows the terminal alarm to be sounded for every send, until a NOALARM command is encountered.

The feature can be incorporated into the security of the system, allowing the user to activate the alarm bell as a form of warning if, for instance, a password is wrongly entered or an application status changes.

Format

ALARM

Related statements

BEEP

NOALARM

NOBEEP

Example

```
If &1 ne ‘y’ then
  set &emsg = ‘Invalid option’
  ALARM
  panel
```

specifies that the error message ‘Invalid option’ will be displayed and the alarm will sound if variable &1 is not equal to ‘y’. The user will then be returned to the panel.

Note

Some terminals do not support an audible alarm.

ALIAS

The ALIAS statement is used in the)PANEL section to give every occurrence of a single ampersand (&) variable name coded in the)BODY section an assigned alias name. The alias name will take precedence over the normally assigned numeric variable name.

Refer to the NCI/XF User Guide, *Chapter 2 - How to code NCI/XF* and the section entitled *Introduction to variables* on page 2.10, for information regarding single character variable names.

Restrictions

The ALIAS statement can only be used in the)PANEL section.

Format

ALIAS &*name* .....

where

&*name*.....	is a series of alias names which will be assigned in sequence to every occurrence of a single ampersand (&) in the)BODY section of a panel. The number of alias names is restricted to 16 for each ALIAS statement. The names will be taken in sequence from successive statements.

Example

```
ALIAS &select &pcode &scode &usrtyp
```

assigns:

&select	to	*&1*
&pcode	to	*&2*
&scode	to	*&3*
&usertyp	to	*&4*

in the)BODY section.

ASSIGN

The ASSIGN statement allocates an attribute to NCI/XF variables. The attribute INDEXED provides for up to 15 variables on one statement. When INDEXED is used, the supplied variable name(s) are stored in a system index called the Variable Name Index (VNI).

When a variable name is stored in the VNI, there is no need for NCI/XF to hold the name in full for each terminal - only a 1-byte binary number to point to the variable name in the index. This reduces virtual storage requirements and will be most noticeable in NCI/XF applications with large user populations (ie terminals) where each user has a large number of variables similar to other users.

Variable names should be assigned to the VNI at system startup. (Refer to *Notes - ASSIGN varname CONF* on page 13).

Format

ASSIGN *varname* ..... *attribute*

where

varname..... is the name of the variable or variables, with leading ampersand (&), to have an attribute assigned. If *attribute* is CONF, only one variable name can be entered. If *attribute* is INDEXED, up to 15 variable names, each separated by a single space, can be entered. Variables must not start with *&usr* or *&sys* and indexed variables must not have an asterisk (*) in the second character position.

attribute is either CONF or INDEXED as follows:

CONF makes *varname* confidential by encrypting all internal use. This encryption is unique to the NCI/XF on which it is assigned (unlike the ENCRYPTION statement which encrypts in common with other NCI/XF's). Variable names starting with *&** will be made confidential to all users. All other variable names (except user and system variables) will be treated as local variables.

INDEXED stores the variable(s) supplied in *varname* in the Variable Name Index (VNI). Variables must not be more than 16 bytes long.

Related Statements

ENCRYPTION

DECRYPTION

Return codes

After execution of an ASSIGN statement, variable *&usrretc* is set to one of the following return codes:

0 successful completion.

4 invalid variable name supplied.

CONF - variable name begins with *&usr* or *&sys*.

INDEXED - variable name begins with *&usr* or *&sys*, or has an asterisk (*) in the second character position, or is longer than 16 bytes or shorter than 2 bytes. The invalid variable name was ignored, so if valid names were also supplied on the statement, they were still processed.

8 variable already assigned this attribute.

CONF - statement was ignored.

INDEXED - the variable name which was already assigned this attribute was ignored, so if non-indexed names were also supplied on the statement, they were still processed.

12 attribute not recognized; must be CONF or INDEXED.

16 (INDEXED only)
variable name index is full. The variable name index can hold up to 256 variable names. If more than one variable name was supplied on the statement, NCI/XF indexed each variable name in sequence, until the index became full.

Examples

```
ASSIGN &myspl CONF
```

will result in *&myspl* being encrypted for all internal use.

```
ASSIGN &apass CONF
SET &work = &apass
```

will result in *&work* containing the decrypted form of *&apass*.

```
ASSIGN &userid &passwd INDEXED
```

will result in variables *&userid* and *&passwd* being stored in the Variable Name Index.

Notes - ASSIGN *varname* CONF

ASSIGN *varname* CONF is not suitable for encrypting variables that are to be sent to another NCI/XF via MHO (since encryption is unique to the NCI/XF on which it is assigned).

When use is implied, for example as a result of tracing, the contents of the variable will be displayed on the log as N/A.

Notes - ASSIGN *varname* INDEXED

To make best use of the Variable Name Index:

1. A variable name must be at least 3 bytes long to benefit from indexing. Shorter variable names will not give any storage saving.
2. A list is built, for each terminal, of all the current variable name indexes when the device creates a session with NCI/XF. Names that are added to the index by any resource after a terminal has connected will be kept in full for that terminal (and all other previously connected terminals), rather than a cross-reference being made to the index. Therefore, the Variable Name Index must be built before any terminals connect to NCI/XF, ideally at system startup.
3. The biggest benefit to be gained from indexing will be from those variables held by every, or nearly every, terminal on the system (eg userids, terminal characteristics etc.).
4. Indexed variables carry a slight CPU overhead if they are referenced heavily, since the system must address another page in storage to retrieve the variable name. If CPU time is of more importance than storage, then bear this in mind.
5. Indexed variables which are likely to be null most of the time, will cause an increase in storage (and a slight decrease in CPU when referencing other variables on the same hash queue). This is because the system keeps a 'dummy' control block for null indexed variables, whereas it discards the control block for null non-indexed variables. Therefore, variables which are likely to be null most of the time for most terminals are not suitable for indexing.

ATTRibute

The ATTRIBUTE statement is used mainly in the)PANEL section, but can also be used in the)INIT,)UPDATE or)PROC sections.

In the)PANEL section it is used to define special characters, which, when coded in the)BODY section of a panel, will be translated to attribute bytes, before the panel is displayed at the terminal. In the)INIT,)UPDATE or)PROC sections it is used to modify the attributes previously assigned to a special character.

The ATTRIBUTE statement can be used to define an overall setting for a character, reset an attribute or allow specific occurrences of the character to have different settings. Although it is possible to define different attribute characters for every panel, it is better practice to use a standard character set.

Format

ATTRIBUTE *char*[*n*] [*properties* .....]

where

char is any alphabetic character which if placed in the body of the panel will be automatically translated to the field attribute property described by *properties*.

n allows the *n*th occurrence of *char* within the panel to be changed to the field attribute property described by *properties*, where *n* is any value from 0-9999 (the default is 0 for all occurrences). If specified, *n* must be adjacent to *char* with no intervening spaces.

If coded in the)INIT,)UPDATE or)PROC sections of a panel, and no *properties* are coded, the properties assigned to selected (*n*) occurrences of *char* are reset to the overall definition in the)PANEL section. (An asterisk (*) resets all occurrences).

properties ..... are the properties for *char*. If the ATTRIBUTE statement is coded in the)INIT,)UPDATE or)PROC sections of a panel, and no *properties* are coded, the properties assigned to selected occurrences of *char* are reset to the overall definition in the)PANEL section (an asterisk (*) in the *n* field resets all occurrences). The *properties* can be any combination of the options described on the next page.

ALPhanumeric	alphanumeric.
ASIS	trailing blanks are not to be removed from unprotected fields when entered.
CURsor	positions the cursor.
HIGhlight	highlighted.
NODisplay	non-display.
NOKEep	an implied reset (equivalent to ATTR *char* with no operands) is to be performed after the panel has been sent to the terminal. This is only valid in the)INIT,)UPDATE or)PROC sections.
NORmal	normal intensity.
NUMeric	numeric.
MDT	causes the particular data (normally a variable) that follows the attribute character to be sent with the 3270 MDT flag set.
PROTected	a display variable only: it is not possible to enter input data.
SEL	permits the field to be detected by a selector pen.
UNPRotected	a variable screen location which is unprotected and allows the variable to be set with input data.

The following extra options may be specified for extended data stream terminals

BLInking	flashing.
REVerse	reverse video.
UNDerscore	underline.

The following extra options may be specified for seven-color terminals

BLUe, GREen, PINk, RED, TURquoise, WHIte and YELlow

You may define a field as autoskip (ASKIP) by defining its attribute byte as PROTECTED and NUMERIC.

Examples

```
ATTR @ normal protected
```

sets @ with the operands specified, across an entire panel. This example, if coded in the)PANEL section, sets the overall definition for the character.

```
ATTR @1 high unprotected cursor
```

will define the first occurrence of @ according to the specified operands.

```
ATTR @1
```

if used in the)INIT,)UPDATE or)PROC sections will set the first occurrence of the @ character back to the overall definition.

```
ATTR @1 high unprotected cursor nokeep
```

if used in the)INIT,)UPDATE or)PROC sections will define the first occurrence of @ according to the specified operands, and will reset to the overall definition after the panel has been sent to the terminal.

```
ATTR @
```

if used in the)INIT,)UPDATE or)PROC sections will cause the attribute values assigned to @ to revert to the defaults given in the)PANEL section, but will retain any specific occurrence definitions.

```
ATTR @*
```

if used in the)INIT,)UPDATE or)PROC sections will return the attribute values assigned to @ to the defaults given in the)PANEL section and remove any specific occurrence definitions.

Notes

Extended datastreams

If a panel containing extended datastream attributes is sent to a terminal which does not support extended data streams, NCI/XF will automatically convert the attributes to a standard datastream. In a mixed network, therefore, panels can be designed for the most sophisticated terminal.

Four-color terminals

On four-color terminals, the following combination of attributes will produce the indicated colors:

HIGH PROTECTED	White
HIGH UNPROTECTED	Red
NORMAL PROTECTED	Blue
NORMAL UNPROTECTED	Green

Modifying attribute values

The initial value assigned in the)PANEL section may be changed in another section. The new values assigned will remain for the duration of this execution of the panel (until a 'PANEL name', RETURN or RESTART is executed) or until another ATTRIBUTE statement redefines the same character.

```
)panel
ATTR @ high protected
ATTR + normal protected
ATTR ! normal unprotected cursor
ATTR % nod protect numeric
)body
         +1. @Menu option 1
         +2. @Menu option 2

      Enter selected option@==>!&+

   %Invalid selection - please respecify+

)proc
If &1 eq 1
   panel menu1
If &1 eq 2
   panel menu2
*
ATTR % high protect numeric red blinking
Panel
```

defines a panel in which the user is invited to select either option 1 or 2 from a main menu. The message 'Invalid selection - please respecify' is preceded by the % attribute character which defines it as a non-display autoskip field. The user therefore does not see this message on first display.

In the)PROC section, the validation checks that the user has entered 1 or 2 and displays the appropriate menu panel. If the user enters anything other than 1 or 2, the % attribute character is changed so that the 'Invalid selection - please respecify' message flashes on the screen in red (if the terminal supports those attributes).

AUTOinput

The AUTOINPUT statement will cause the)PROC section of the panel being displayed next on the user's terminal to be driven immediately after the panel has been displayed. The user is not given the opportunity to enter any data. The AUTOINPUT specification will apply to the next panel display, regardless of whether it is a redisplay of the same panel or a new panel.

Format

AUTOINPUT [NOBID]

where

NOBID specifies that for SNA terminals the screen which is sent after the autoinput screen should not be sent with a preceding SNA BID command. This parameter is a performance related enhancement which can result in VTAM bracket error conditions, and should not be used unless advised by CKS.

Examples

```
)proc
If &count lt 5
   incr &count
   AUTO
   panel
```

will redisplay the same panel and immediately redrive this)PROC section, until *&count* exceeds 5.

```
)proc
AUTO
Panel panel5
```

will drive the)INIT section of panel PANEL5, display PANEL5 and immediately drive its)PROC section.

Notes

The AUTOINPUT statement meets certain specialized demands, for example:

- sending a dummy panel which sounds the terminal alarm before issuing a CONNECT to satisfy a queued logon request
- the use of certain hardware security devices which require a specific character sequence to be sent several times in quick succession at logon time
- creating animation on panels

Variable *&usraid* is set to AUTO when AUTOINPUT is in operation.

BEEP

The BEEP statement turns on the terminal alarm for the next send only. This is particularly useful when an error message is to be displayed.

A BEEP can be cancelled by NOBEEP.

Format

BEEP

Related statements

ALARM

NOALARM

NOBEEP

Example

```
If &1 ne ‘y’ then
  set &emsg = ‘Invalid option’
  BEEP
  panel
```

specifies that if variable *&1* is not equal to y, the Invalid option error message will be displayed and the alarm will sound. The user will then be returned to the panel.

Note

Some terminals do not support an audible alarm.

C2D

The C2D statement converts the binary numeric EBCDIC value of a character string to decimal.

(For decimal to binary conversion, refer to related statement D2C).

Format

C2D *operand1* = *operand2* [NEG]

where:

operand1 is the variable to receive the converted data.

operand2 is the value or variable to be converted. The same variable as *operand1* may be specified. The maximum length is 4 bytes. For positive numbers the maximum value is X'7FFFFFFF'; for negative numbers the maximum value is X'FFFFFFFF' and NEG must be coded to convert. (Note that in practice it is the binary number to be converted that is of interest, rather than the character string which it happens to represent).

NEG specifies that the number to be converted is to be treated as negative if its leftmost bit is turned on.

Related statements

A2E

C2X

D2C

D2P

E2A

X2C

Examples

```
C2D &dec = AZ            /*  X'C1E9'
```

will result in variable *&dec* being set to 49641.

```
C2D &dec = AZ NEG       /* X'41E9'
                        /* (X'C1' with leftmost
                           bit turned off = X'41')
```

will result in variable *&dec* being set to -16873.

C2X

The C2X statement converts character data to hexadecimal format.

Format

C2X *operand1* = *operand2*

where:

operand1 is the variable to receive the converted data.

operand2 is the value or variable to be converted. The same variable as *operand1* may be specified. The maximum data length is 16k characters.

Related statements

A2E

C2D

D2C

D2P

E2A

X2C

Example

```
C2X &HEX = ABC
```

will result in variable *&hex* being set to C1C2C3.

CALCulate

The CALCULATE statement allows NCI/XF to perform mathematical functions other than simple INCREMENT and DECREMENT. Brackets are resolved first, then processing continues from left to right.

Format

CALCULATE *&a = arithmetic-expression*

where

&a	is the name of a variable used to hold the results of the calculation.
arithmetic-expression	is the calculation to perform. Any combination of the following operators can be used:

+	addition
-	subtraction
*	multiplication
/	division
( )	brackets

Related statements

INCREMENT

DECREMENT

FCALCULATE

Return codes

After execution of a CALCULATE statement, the variable *&usrretc* is set to one of the following return codes:

0	calculation performed - variable set.
4	invalid parameters specified.
8	expression error.
12	unbalanced parentheses.
16	attempt to divide by zero.
20	overflow condition.

Example

```
)proc
If &usrretc = 0 then
  CALC &a = &usrpscr./2
```

if the condition specified is true, the contents of variable *&usrpscr* will be divided by 2 and the result placed in variable *&a*.

Note

Each embedded variable in the CALCULATE statement should be immediately followed by a period.

Substring variables are not permitted.

CALL

The CALL statement can be used to invoke a user-written exit routine to perform any installation specific functions not available in the NCI/XF language itself, for instance I/O to a dataset.

The module must reside in a STEPLIB library (MVS) or GLOBAL LOADLIB library (VM) and will be processed asynchronously to the NCI/XF mainline code.

The CALL statement can be used to call any program written in a high level language, for instance COBOL. (Refer to *COBOL* on page 12.3 of the NCI/XF User Guide).

Format

CALL *name* [*parm*.....]

where

name	is the name of a routine to be called.
parm.....	are parameters to be passed to the exit routine. These may be literals or NCI/XF variables. Up to 15 parameters can be passed on a single CALL.

Related statements

FCALL

LOAD

MODSTATS

UNLOAD

Return codes

After execution of a CALL statement, variable *&usrretc* is set to the contents of register 15 by the called user-written routine.

Example

```
CALL EXIT01 'ADD' &name &address
CALL DL1PROG 'LOCATE FIRST' &dbname &recid
```

will call exits EXIT01 and DL1PROG and pass the contents of the listed variables and literals to the respective routines.

Note

An abend will cause the NCI/XF ERROR-EXEC to be driven. This, by default, displays panel NCERR8.

CHECKFix

The CHECKFIX statement determines the fixes that have been applied to an NCI/XF PBLIB panel. After execution of the CHECKFIX statement, the number of fixes applied will be placed in variable *&usrf0* and the fix names will be placed in variables *&usrf1* through *&usrfNNN* (up to 999).

Format

CHECKFIX *name*

where

name is the name of the PBLIB panel for which fixes are to be determined.

Return codes

After execution of a CHECKFIX statement, variable *&usrretc* is set to one of the following return codes:

0 CHECKFIX statement processed successfully.
4 syntax error, or error opening file.
8 PBLIB panel not found.
16 more than 999 fixes found; only the first 999 are returned.

Example

```
CHECKF @XMSTART
```

will return the following information if fix XMB001 has been applied:

&usrf0 = 1
&usrf1 = XMB001
&usrretc = 0

CHECKPanel

The CHECKPANEL statement checks if a specified panel (or EXEC routine) exists. Panels exist in the following locations:

- in core (already used)
- in a prebuilt panel library
- in the normal panel library

Format

CHECKPANEL *name*

where

name is the name of the panel or routine to be checked.

Return codes

After execution of a CHECKPANEL statement, variable *&usrretc* is set to one of the following return codes:

0 panel exists.
4 panel not found.
8 invalid panel name supplied.

Example

```
CHECKP panname
If &usrretc eq 0 then
   panel panname
```

will check whether panel panname exists and if so, will execute it.

COLumns

The COLUMNS statement is used to establish the width of the PANEL to be displayed.

Restrictions

COLUMNS is valid in the)PANEL section only

Format

COLUMNS [*nnn*]

where

nnn is a numeric value specifying the width of the panel to be displayed. This should normally be set to the screen size of the terminal which will use the panel.

Permitted values are 40, 80 or 132, with a default of 80 if not specified.

Related statement

ROWS

Example

```
)panel
 interval 1
 rows    24
 COL     80
 attr     @ high protected
```

will set the width of the panel to be displayed to 80 characters.

CONNect

The CONNECT statement is used to pass a terminal from NCI/XF to an application.

Restrictions

CONNECT is not allowed in:

startup-panel	update-exec	losterm-exec
initial-exec	close-exec	lustat-exec
open-exec	final-exec	MODIFY

Format

CONNECT *applname* [*logmode* [*cinit*]]

where

applname is an application nodename or a variable set to an application nodename.

logmode specifies a logmode definition to be used. If this parameter is omitted, the terminal's default logmode is used. If a *logmode* operand is not required, the *cinit* operand can only be entered if *logmode* is entered as *&sysnull* or ' '.

cinit can be any data or a variable name set to the data which will be passed to the application as CINIT (logon) data when application connection is performed.

Related statement

CONNECTR

Return codes

After execution of a CONNECT statement, variable *&usrretc* is set to one of the following return codes:

0 connection successful.
4 application inactive/unknown/unavailable (variables *&usrrtcd* and *&usrsens* provide further information - see below).
16 application shutting down.
28 either the syntax is incorrect or a VTAM error occurred.
32 SIMLOGON recovery failure.

The VTAM return code and sense code are available in variables *&usrrtcd* and *&usrsens* respectively. When *&usrretc* is set to 4, the following codes apply:

&usrrtcd=1000 the application is not available (*&usrsens* will be 0000).

&usrrtcd=1453 the application name is unknown to VTAM or major node name not ACTIVE (*&usrsens* will be 0000).

&usrsens=non-zero the CONNECT worked but the target application rejected the terminal. The reason is defined by the codes in *&usrsens* and *&usrrtcd* (for instance, 0821 in *&usrsens* means the target application found the session parameters invalid). Note that in practice a bind reject (*&usrsens* 0821) has a *&usrrtcd* of 0000, but this can not be guaranteed.

Example

```
CONN TSO &usrlogm '&userid./&passwd'
```

will connect the terminal to TSO, using the logmode *&usrlogm* and CINIT data of *&userid./&passwd*.

Notes

Specifying logmodes

All operands are positional; if CINIT data is to be passed, the logmode operand must also be specified. The application session can be established using any logmode which exists in the logon mode table pointed to by the MODETAB operand of the terminal definition, or in the IBM default table ISTINCLM. This may or may not be the same logmode which was used to log on to NCI/XF.

To use the terminal's default logmode, omit the *logmode* operand or code *&sysnull*.

To use the same logmode which was used to establish the session with NCI/XF, code *&usrlogm*. See also *&usrlogt on page 3.4* and *&usrlogu on page 3.4*.

To use a totally different logmode, code the required name or the name of a variable which contains the required name.

Return to NCI/XF

After execution of a CONNECT statement, control always returns to the first statement after the CONNECT - this will be immediate if the session request fails, or many hours afterward when the user logs off the target application.

When the terminal is LOGAPPL'd to NCI/XF and the user logs off the target application, VTAM will automatically pass control of the terminal to NCI/XF and execution will immediately continue with the statement following the CONNECT.

Terminals not LOGAPPL'd to NCI/XF will return control to VTAM (or a controlling application) when the user logs off a target application. The next time the terminal logs on to NCI/XF, execution continues from the statement after the CONNECT.

You may insert any required logic after the CONNECT to handle specialized situations relevant to your own network, such as dial-up terminals, logical units connected via emulators etc.

CONNECTR

The CONNECTR statement is similar to the CONNECT statement when used to pass a terminal to an application. In addition, CONNECTR causes NCI/XF to wait for notification of the success of CLSDST PASS and then issues a SIMLOGON to itself.

Restrictions

CONNECTR is not allowed in:

startup-panel	update-exec	losterm-exec
initial-exec	close-exec	lustat-exec
open-exec	final-exec	MODIFY

Format

CONNECTR *applname* [*logmode* [*cinit*]]

where

applname is an application name or a variable set to an application name.

logmode specifies a logmode definition to be used. If this parameter is omitted, the default logmode as defined for this terminal is used. If a *logmode* operand is not required, the *cinit* operand can only be entered if *logmode* is entered as ' ' or *&sysnull*.

cinit can be any data, or a variable name to which the data is supplied, which will be passed to the application as CINIT data when application connection is performed.

Related statement

CONNECT

Return codes

After execution of a CONNECTR statement, variable *&usrretc* is set to one of the following return codes:

0 connection successful.

4 application inactive/unknown/unavailable (variables *&usrrtcd* and *&usrsens* provide further information - see below).

16 application shutting down.

28 either the syntax is incorrect or a VTAM error occurred.

32 connection successful followed by terminal lost.

The VTAM return code and sense code are available in variables *&usrrtcd* and *&usrsens* respectively. When *&usrretc* is set to 4, the following codes apply:

&usrrtcd=1000 means the application is not available (*&usrsens* will be 0000).

&usrrtcd=1453 means the application name is unknown to VTAM or major node name not ACTIVE (*&usrsens* will be 0000).

&usrsens=non 0000 the CONNECTR worked but the target application rejected the terminal. The reason is defined by the codes in *&usrsens* and *&usrrtcd* (for instance, 0821 in *&usrsens* means the target application found the session parameters invalid). Note that in practice a bind reject (*&usrsens* 0821) has a *&usrrtcd* of 0000, but this can not be guaranteed.

Example

```
CONNECTR TSO &logmode &cinit
```

will connect the terminal to TSO using logmode *&logmode* and CINIT data *&cinit*.

Notes

CONNECTR covers two specialized situations:

- terminals which are not LOGAPPL'd to NCI/XF
- terminals varied inactive while in session with the target application

Non-logappl'd terminals

If a terminal is not LOGAPPL'd to NCI/XF, control is returned to VTAM when the user logs off the target application. It may be desirable to return the terminal to NCI/XF at this time, using CONNECTR, to process statements after the CLSDST PASS request. It could, for instance, be required to decrement user counts, before internally issuing an NCI/XF LOGOFF command to return to VTAM.

```
If &usrnlog eq YES
   CONNECTR &appl &logmode &cinit
   exec testretc
   decr &applcnt
   panel menu1

Connect &appl &logmode &cinit
Exec testretc
Decr &applcnt
Logoff
```

If *&usrnlog* is YES, the terminal has been marked NOLOG in the Terminal Definitions Table (TDT) to specify that the terminal is not LOGAPPL'd to NCI/XF. CONNECTR will be used to return the terminal to VTAM before processing subsequent statements.

If *&usrnlog* is NO, the terminal is LOGAPPL'd to NCI/XF and CONNECT will be used before processing subsequent statements. The terminal is not returned to VTAM.

Dial-up lines

Where dial-up ports or emulators with port pooling and concatenation are in use, it is important that NCI/XF knows that the connection was lost while the terminal was in session with a target application. This is because the next session using that port may belong to a different user, possibly with different security authorization. NCI/XF should therefore do a RESTART to return to the top-level panel rather than continue with normal processing after the CLSDST PASS request. When a SIMLOGON is queued, NCI/XF is informed that a terminal loss occurred (*&usrretc* set to 32) and may react accordingly.

```
If &usrgrp ne 'DIALUP'
   goto notdial

CONNECTR &appl
Decr &applcnt
If &usrretc = 32
   restart
Exec testretc
Menu applmenu
```

WARNING: CONNECTR results in a Request Parameter List (RPL) remaining active until either the control unit is powered on or NCI/XF reacquires the terminal. Excessive use of this command will increase CSA usage.

CURsor

The CURSOR statement enables the cursor to be positioned anywhere within a panel, regardless of the field attributes in use on the panel.

Format

CURSOR [*varname*]

CURSOR [*row column*]

where

varname is the name of an NCI/XF variable displayed on the panel, or a variable containing the name. The cursor will be placed in the first position of the variable. When an explicit variable name is specified the leading ampersand (&) must be omitted (a leading ampersand (&) causes the variable to be resolved before the command is executed).

row is the line number, or a variable set to the line number in which the cursor is to be positioned. A *row* must have a corresponding *column* entry.

column is the column number, or a variable set to the column number, in which the cursor is to be positioned. A *column* must have a corresponding *row* entry.

Specifying CURSOR with no operands cancels a previous CURSOR statement and positions the cursor as determined by the screen attribute bytes.

Examples

```
CUR seltn
```

places the cursor in position 1 of the variable *&seltn*.

```
CUR input
Calc &work = 3+&usrcurc
CUR &usrcurr &work
```

places the cursor in position 4 of variable *&input* (variables *&usrcurc* and *&usrcurr* are described *on page 1.34*).

```
Set &name = seltn
CUR &name
```

places the cursor in position 1 of the variable *&seltn*.

```
Set &row = 5
Set &col = 25
CUR &row &col
```

places the cursor in line 5 column 25 of the panel.

```
CUR 12 20
```

places the cursor in line 12 column 20 of the panel.

Notes

Determining cursor position

NCI/XF automatically maintains the following variables on behalf of each terminal user:

&usrcurr the row currently containing the cursor
&usrcurc the column currently containing the cursor

After execution of a CURSOR statement these variables will be updated to reflect the new cursor position as specified by the command.

To redisplay a panel with the cursor positioned where the user left it, the following statement may be used:

```
CUR &usrcurr &usrcurc
```

Determining variable position

To determine the position of an NCI/XF variable on the screen, use the explicit form of CURSOR statement as follows:

```
CUR varname
```

This sets variables *&usrcurr* and *&usrcurc* to the required values and is particularly useful when the cursor is being placed somewhere other than the beginning of a field (such as when validating user input). For instance, assuming that positions 1-2 of *&account* contain *ab* and position 6 must be numeric, the following code will place the cursor under position 6 and redisplay the panel:

```
Set &¢prefix = &account(1:2)
If  &¢prefix = AB
    verify &account = '*****N'
    If &usrretc ne 0
       CUR account
       calc &¢col = 5+&usrcurc
       CUR &usrcurr &¢col
       set &¢emsg = 'Invalid item'
       panel
```

D2C

The D2C statement converts a signed or unsigned decimal number to binary.

(For binary conversion to decimal, refer to related statement C2D).

Format

D2C *operand1* = *operand2*

where:

operand1 is the variable to receive the converted data. The converted data will be binary, representing a numeric EBCDIC value for a character string. (Note that in practice it is the binary number that is of interest rather than the character string which it happens to represent). If the decimal number to be converted is negative, the leftmost bit of the binary result will be turned on; if the bit is already on, the length of the result will be increased by appending, to the left, one byte of all binary zeros except for a 1 in the leftmost bit.

operand2 is the value or variable to be converted. This can be a signed or unsigned decimal number which must be between +2,147,483,647 and -2,147,483,647. The same variable as *operand1* may be specified.

Related statements

A2E

C2D

C2X

D2P

E2A

X2C

Examples

```
D2C &char = 193
```

will result in variable *&char* being set to X'C1'= A.

```
D2C &char = 2
```

will result in variable *&char* being set to X'02'.

```
D2C &char = -1
```

will result in variable *&char* being set to X'81'.

```
D2C &char = -127
```

will result in variable *&char* being set to X'FF'.

```
D2C &char = -255
```

will result in variable *&char* being set to X'80FF'.

D2P

The D2P statement converts signed or unsigned decimal data to signed packed format.

Format

D2P *operand1* = *operand2*

where:

operand1 is the variable to receive the converted data.

operand2 is the value or variable to be converted. The same variable as *operand1* may be specified. The maximum data length is 16 characters.

Related statements

A2E

C2D

C2X

D2C

E2A

X2C

Example

```
D2P &packed = -123
```

will result in variable *&packed* being set to X'123D'.

DECRement

The DECREMENT statement is used to reduce the value of a previously defined variable by 1. Any attempt to decrement a variable already set to zero will be ignored.

The DECREMENT statement can be used in conjunction with the INCREMENT statement to restrict numbers of users to applications. For instance when a terminal logs on to an application, INCREMENT a counter. At logoff, DECREMENT the counter. When the counter reaches a preset value send the user an informational message and refuse the session. The same kind of technique can be used for application and CPU load balancing.

Format

DECREMENT *&var*

where

&var is the variable to be decremented by 1.

Related statements

CALCULATE

INCREMENT

FCALCULATE

Example

```
)proc
If &usraid = pf03 then
   if &systsoc eq &systsom then
      goto maxusers
   increment &systsoc
   connect TSO
   DECR &systsoc
   exec testretc
```

will decrement the contents of variable *&systsoc* by one when the TSO session has ended.

Note

Any attempt to decrement a variable not previously set to a numeric value or a value greater than 9999999 will result in a run-time error and the NCI/XF ERROR-EXEC will be driven. This will, by default, display panel NCERR5.

DECRYPTion

The DECRYPTION statement converts a previously encrypted variable back to its original value.

Use of the ENCRYPTION/DECRYPTION statements is strongly recommended for installations which have security systems and pass userids and passwords to NCI/XF for validation. It prevents security exposures in the event of NCI/XF storage being examined.

Format

DECRYPTION *&var1* = *&var2*

where

&var1	is the receiving variable.
&var2	is the variable to be decrypted.

Related statement

ENCRYPTION

ASSIGN

Example

```
DECRYPT &usrpass = &garbage
```

will decrypt the contents of the encrypted variable *&garbage*, and place the result in variable *&usrpass*.

DEFINE

The DEFINE statement is used to dynamically update the Terminal Definitions Table (TDT), allowing terminal definitions to be added or updated on-line.

Format

DEFINE *name*

where

name is a member or file containing a list of terminals and associated initial panels that NCI/XF is to use. It will be validated for syntax before the update.

Return codes

After execution of a DEFINE statement, variable *&usrretc* is set to one of the following return codes:

0 TDT successfully reloaded.
4 TDT reload failed.

Example

```
DEFINE NCITDT
```

Notes

As soon as a successful DEFINE has been completed, all the variables related to the TDT (such as *&usrgrp*, *&usrrest*, *&usrlogt*, *&usrnlog*) are automatically updated for all users.

If a terminal is removed from the TDT, the NCI/XF ERROR-EXEC will be driven. This, by default, will display panel NCERR9 at that terminal before logging the terminal off.

DELete

The DELETE statement is used to delete a panel from NCI/XF storage. This is very useful during panel design and testing because NCI/XF will dynamically load a new copy from disk when the panel is next referenced. The storage held by the panel is recovered when the panel user count reaches zero.

Format

DELETE *panels*.....

where

panels..... are the names of panels, or variables set to the names of panels, to be deleted. The maximum number of panel names that can be entered is 16.

Generic panel names can be defined by using the two special mask characters, asterisk (*) and plus (+). Asterisk (*) specifies a set of panel names based on the characters before it (for instance pan* specifies all panel names beginning pan).

A plus sign (+) specifies that the character matching its position in the variable name(s) will be masked (for instance test+pan specifies panel names that begin with test, followed by any character and end with pan). Multiple plus signs (+) can be used.

Return codes

After execution of a DELETE statement, variable *&usrretc* is set to one of the following return codes:

0 all specified panels deleted.
4 one or more of the panel names supplied did not match any of the panels in storage.

Example

```
)proc
If &1 eq 3 then
   if &member ne &sysnull then
      set &¢msg = 'panel &member deleted'
      DEL &member
      If &usrretc ne 0 then
         set &¢msg = 'Delete &member failed'
      panel
```

if the conditions specified are true, panel *&member* will be deleted from storage.

DEQueue

The NCI/XF ENQUEUE and DEQUEUE statements provide facilities equivalent to the operating system ENQ and DEQ macros. You can use ENQUEUE and DEQUEUE to assign control of a serially reusable resource to your panel. You can choose your own enqueue names to control sharing of user defined resources within NCI/XF or you can enqueue on external resources to control the sharing of resources between NCI/XF and other system facilities (such as datasets).

Refer to the descriptions of the ENQ and DEQ macros in the appropriate IBM operating system manual, for further details.

Format

DEQUEUE *majorname minorname* [*scope*]

where

majorname	is the major name used to issue the original resource enqueue, or a variable containing the major name.
minorname	is the minor name used to issue the original resource enqueue, or a variable containing the minor name.
scope	specifies SYSTEMS, SYSTEM or STEP, as on the original resource enqueue (MVS only). If omitted the default is SYSTEMS.

Return codes

After execution of a DEQUEUE statement, *&usrretc* will contain the return code supplied in register 15 by the operating system DEQ macro.

A return code of 7 indicates a parameter error.

Related statement

ENQUEUE

Example

```
Set  &major = 'sysdsn'
Set  &minor = &dataset
Padr &minor 44
DEQ &major &minor systems
```

will dequeue the resource(s) in *&dataset* which were previously enqueued using the major name of sysdsn. On regaining control from the resource(s), the control program will assign control to the next queued resource(s), if any.

DIRECT

The DIRECT statement is used to update the NCI/XF Directory of Users on-line. The NCI/XF Directory of users is an optional NCI/XF security system which can be used if you want to restrict system access by means of userids and passwords. It should only be used if you have no other security system in use at your installation and you have no other means of providing this facility. Refer to *Creating the directory of users* on page 3.20.

Format

DIRECT *name*

where

name is the name (or a variable containing the name) of a member or file containing a list of userids and passwords for users authorized to use NCI/XF. Its syntax will be checked before the update.

Return codes

After execution of a DIRECT statement, variable *&usrretc* is set to one of the following return codes:

0 update successful.
4 update failed.

Example

```
If &1 eq 2 then
 If &member ne &sysnull then
   set &¢msg = 'Update successful'
   DIRECT &member
   if &usrretc ne 0 then
      set &¢msg = 'Update failed,RC=&usrretc'
panel
```

will update the member or file whose name is contained in variable *&member*.

DISable

The DISABLE statement temporarily prevents a terminal from connecting to NCI/XF. This is useful in an 8100 DPCX environment where terminals are defaulted to connect to HOST applications but occasionally work in 8100 mode.

Format

DISABLE

Related statement

ENABLE

Note

When the terminal next attempts to connect to NCI/XF, the logon will be rejected and the terminal will be enabled.

E2A

The E2A statement converts an EBCDIC character string to ASCII.

Format

E2A *operand1* = *operand2* [*table-suffix*]

where:

operand1 is the variable to receive the converted data. This may be a variable name or a variable variable. The converted data will be ASCII.

operand2 is the value or variable to be converted. This can be a literal, a variable, concatenated variables, a variable variable, a substring of a variable or any combination of these. The maximum length of data string for conversion is 32,767 bytes.

table-suffix is a literal or variable giving the suffix part of the name of a table of translation values from EBCDIC to ASCII. Available suffixes are:

N001 - US English (same as standard table)

N033 - French

N044 - UK English

N049 - German

For other suffixes see your system administrator.

If *table-suffix* is not coded, the standard table will be used (US English).

Note: The table used for translation should match the language set up on the system on which your NCI program will run.

Related statements

A2E

C2D

C2X

D2C

D2P

X2C

Return codes

After execution, variable *&usrretc* will be set to one of the following return codes:

0 translation successful.
24 *table-suffix* was coded but the corresponding module did not exist or was found to be in error.

Examples

```
E2A &ascii=ABC
```

will result in variable *&ascii* being set to X'414243'.

```
E2A &ascii=ABC tab1
```

will result in the variable *&ascii* being set to the three ASCII equivalent characters of literal ABC, according to the user supplied translation table assembled as module NCAETAB1.

ELSE

The ELSE statement allows users to process alternative commands when an IF condition is not satisfied.

Format

ELSE

Related statement

IF

Example

```
If &parm1 = y
  set &select = 1
ELSE
  log 'no option selected'
```

if *&parm1* is not equal to y this will log the message 'no option selected'.

Notes

The ELSE command must be coded in the same column as the IF command to which it refers.

Commands to be executed when the IF condition is false, should be indented following the ELSE statement.

ENAble

The ENABLE statement allows the connection of a previously DISABLED terminal to NCI/XF.

Format

ENABLE

Related statement

DISABLE

ENCRYPTion

The ENCRYPTION statement allows fields such as passwords to be held in encoded format to prevent security exposures.

Use of the ENCRYPTION/DECRYPTION statements is strongly recommended for installations which have security systems, and pass userids and passwords to NCI/XF for validation. It prevents security exposures in the event of NCI/XF storage being examined.

Format

ENCRYPTION *&var1* = *&var2*

where

&var1 is the receiving variable.

&var2 is the variable to be encrypted.

Related statement

DECRYPTION

ASSIGN

Example

```
ENCRYPT &garbage = &usrpass
Reset &usrpass
```

will encrypt the contents of variable *&usrpass* and place the result in variable *&garbage*. Variable *&usrpass* will be reset.

ENQueue

The NCI/XF ENQUEUE and DEQUEUE statements provide facilities equivalent to the operating system ENQ and DEQ macros. You can use ENQUEUE and DEQUEUE to assign control of a serially reusable resource to your panel. You can choose your own enqueue names to control sharing of user defined resources within NCI/XFs or you can enqueue on external resources to control the sharing of resources between NCI/XF and other system facilities (for instance datasets).

For further details refer to the descriptions of the ENQ and DEQ macros in the appropriate IBM operating system manual.

Format

ENQUEUE *majorname minorname* [*scope*]

where

majorname	is the major name to enqueue on, or a variable containing the major name.
minorname	is the minor name to enqueue on, or a variable containing the minor name.
scope	specifies SYSTEMS, SYSTEM or STEP to define the scope of the enqueue (MVS only). If omitted the default is SYSTEMS.

Related statement

DEQUEUE

Return codes

After execution of an ENQUEUE statement, variable *&usrretc* will contain the return code supplied in register 15 by the operating system ENQ macro.

A return code of 7 indicates a parameter error.

Example

```
Set  &major = 'sysdsn'
Set  &minor = &dataset
Padr &minor 44
ENQ &major &minor systems
```

will enqueue the resource(s) in *&dataset* using the major name of sysdsn. The control program will assign control to the resource(s) in *&dataset* if they are available.

EXECute

The EXECUTE statement allows standard subroutines to be invoked so that repeated processes can be kept to a minimum across panels.

Nesting of subroutines is allowed to any depth.

Format

EXECUTE *name* [*parm1*.....]

where

name is the name of the subroutine required.

parm1.... are any parameters which are to be passed to the subroutine. The parameters can be literals, variables (with substrings), or variable variables. Up to 15 parameters can be passed on a single EXECUTE.

Variables *&usrexN* must not be used. For example, the following entry is invalid:

EXEC B FRED &USREX1

Related statement

EXIT

Example

```
If &usraid = PF03 then
   connect TSO
   EXEC testretc
```

will EXEC routine testretc after the CONNECT statement has completed.

Notes

On entry to the routine, the variable *&usrex0* is set to the number of parameters and the variables *&usrex1* to *&usrexn* are set to each parameter. The variables *&usrexn* can be altered by the user, and stored before another EXEC is run (or reset before an EXIT).

The subroutine must exist in the NCI/XF panel library.

When control is returned from the subroutine, as the result of an EXIT or)END statement in the routine, execution continues from the statement after the EXEC.

If the requested subroutine is not found in the NCI/XF panel library then the NCERR1 error panel will be displayed.

EXIT

The EXIT statement has two distinct purposes:

- to return control from a subroutine to the statement after the EXEC which invoked the subroutine.
- to exit from an)UPDATE section without redisplaying the panel.

Restrictions

When used in either the)INIT or)PROC section, the EXIT statement is only valid if used in a subroutine.

Format

EXIT [*code*]

where

code is an optional return code for this subroutine, and must be numeric in the range 0 - 4095. The variable *&usrretc* will be set to this value. If *code* is unspecified, *&usrretc* will remain unaltered.

Examples

```
)update
*
* Test for a change in TSO status
*
Set &oldstat = &tsostat
Status tso
Exec setstat
If &tsostat ne &oldstat
   panel
EXIT
```

This example shows EXIT used in an)UPDATE section. This will exit the)update section when a change in TSO status is detected.

The following example shows EXIT used in a routine.

```
)routine setstat
*
* Set status based on return code.
*
Goto lab&usrretc
*
Label lab0
Set &tsostat = AVAILABLE
EXIT 0
Label lab4
Label lab8
Label lab12
Label lab16
Set &tsostat = UNAVAILABLE
EXIT &usrretc
Label lab20
Label lab24
Label lab28
Set &tsostat = UNKNOWN
EXIT &usrretc
)end
```

FCALCulate

The FCALCULATE statement provides the facility of a floating point calculator, accurate to eight decimal places. Brackets are resolved first, then processing continues from left to right.

Format

FCALCULATE *&a = arithmetic-expression*

where

&a	is the name of a variable used to hold the results of the calculation.
arithmetic-expression	is the calculation to perform, including integer or floating point numbers. Any combination of the following operators can be used: + addition - subtraction * multiplication / division ( ) brackets

Related statements

CALCULATE

INCREMENT

DECREMENT

Return codes

After execution of an FCALC statement, variable *&usrretc* is set to one of the following return codes:

0	calculation performed - variable set.
4	invalid parameters specified.
8	expression error.
12	unbalanced error.
16	attempt to divide by zero.
20	overflow condition.

Example

```
FCALC &var = 1/3        <---sets &var to 0.33333333
FCALC &var = &var.*6    <---sets &var to 1.99999998
FCALC &var = 1.23*10    <---sets &var to 12.3
FCALC &var = 6/(2*3)    <---sets &var to 1
```

Notes

Initial and final values greater than

999999999.99999999

and intermediate values greater than or equal to

1*10E50

will cause an overflow.

Initial values which have more than eight decimal places will be truncated to eight decimal places. Intermediate values less than or equal to

1*10E-50

will be rounded to 0, and all final values rounded to the nearest

0.00000001

FCALL

The FCALL statement provides the same function as the CALL statement, except that NCDIALOG macro requests issued by called user written exit routines will be expedited by the NCDIALOG component of NCI/XF. This fast call mechanism provides a more efficient technique of executing call routines.

The following two rules must be observed by the exit routine if it is to be invoked by FCALL:

6. An NCDIALOG macro must not be issued if there is an outstanding STIMER REAL issued by the exit.
7. If the exit routine triggers an IRB, then the IRB exit must not issue an NCDIALOG macro.

Format

FCALL *name* [*parm*.....]

where

name	is the name of a routine to be called.
parm.....	are parameters to be passed to the exit routine. These may be literals or NCI/XF variables. Up to 15 parameters may be passed on a single FCALL.

Related statements

CALL

LOAD

MODSTATS

UNLOAD

Return codes

Variable *&usrretc* is set to the contents of register 15 by the called user-written routine.

Example

```
FCALL EXIT01 'ADD' &name &address
FCALL DL1PROG 'LOCATE FIRST' &dbname &recid
```

will call exits EXIT01 and DL1PROG and pass the contents of the listed variables and literals to the respective routines.

Note

If an abend occurs, the NCI/XF ERROR-EXEC will be driven. This, by default, will cause panel NCERR8 to be displayed.

GETLog

The GETLOG statement retrieves a line from the NCI/XF log buffer.

Format

GETLOG *linenum*

where

linenum is the number of the print line to be retrieved from the in-core buffer. The number is relative to the first line (not a buffer position).

Related statements

LOG

Related variables

&syslold

&syslnew

&usrlogl

Return codes

After execution of a GETLOG statement, variable *&usrretc* is set to one of the following return codes:

0 Record still in buffer (*&usrlogl* contains line).
4 Record not yet entered or no longer in buffer.
8 Invalid line number.

Example

```
GETL 2000
```

If the requested line was still in the buffer, the variable *&usrlogl* will contain the requested line.

GLOBALV

The GLOBALV statement maintains a pool of variables which can be shared between two or more users. These variables are protected from a RESET command. Variables can be copied into, retrieved from, or deleted from the pool.

Format

GLOBALV *process poolname* [*varname* .....]

where

process indicates the process to be performed on the variable(s). This can be one of the following:

- PUT copies the named variable(s) into the pool; existing values are overwritten.
- GET retrieves the named variable(s) from the pool.
- DELETE deletes the named variable(s) from the pool.

poolname specifies the name of the pool of variables. The pool is global and can be accessed by all users. The name can be 1 through 16 characters.

varname.... specifies the names of up to 14 variables, without the leading ampersand (&). System variable names cannot be specified. If the variable name is omitted from a GET or DELETE, all variables in the pool are processed. Using PUT with a null variable has no effect, and using GET for a variable that is not in a pool will result in a null value being set for that variable.

Return codes

After execution of a GLOBALV statement, variable *&usrretc* is set to one of the following return codes:

0 Command completed successfully.
4 Syntax error; for instance variable name omitted on PUT.
8 Pool does not exist (for GET and DELETE operations only).

Examples

```
GLOBALV PUT APOOL USER PSWD
RESET
............
............
GLOBALV GET APOOL USER PSWD
```

saves variables *&user* and *&pswd* from being reset by the RESET command.

```
GLOBALV GET APOOL
GLOBALV DELETE APOOOL
```

retrieves all variables in APOOL and deletes the pool.

GOTO

The GOTO statement is used to branch to a specific LABEL in the same section within a panel.

Format

GOTO *label* [*errorlab*]

where

label is a specific point in the section indicated by a LABEL statement.

errorlab is a label to branch to, if *label* is not found in the current section.

Variables may be used as part of the label name to branch to. The variable will be resolved when the GOTO is executed and the label name will be built using the resolved value.

Related statement

LABEL

Example

```
GOTO lab&usraid invkey

Label labpa01
Label labpa02
Label labpa03
Set &msg = 'You pressed a <PA> key'
Panel

Label labentr
Set &msg = 'You pressed the <Enter> key'
Panel

Label invkey
Set &emsg = 'Invalid key'
Panel
```

will branch to the label invkey unless the variable *&usraid* contains pa01, pa02, pa03 or entr.

Note

Restrictions in label specification

The GOTO statement can only specify a label within the same panel section, for instance control cannot pass from the)UPDATE section to the)PROC section. In addition, the label must exist within the same library member; a GOTO cannot jump into or out of a subroutine.

Error panel

If the MAXGOTO limit is exceeded (refer to *Startup options* on page 3.6 of the NCI/XF Installation Manual) the NCI/XF ERROR EXEC will be driven. This, by default, will display panel NCERR4.

If the GOTO label is missing, the NCI/XF ERROR-EXEC will again be driven and this, by default, will display panel NCERR6.

GRESET

The GRESET statement is used to delete sets of variables. All storage occupied by the variables is freed. After a GRESET, each variable is considered to have a value equal to *&sysnull*.

Format

GRESET *&varname*.....

where

&varname..... are the names of the variable sets to be deleted. The number of variable names is restricted to 16 for each GRESET statement. A group of variable names is defined by using the two special mask characters asterisk (*) and plus (+).

Asterisk (*) specifies a set of variable names based on the characters before it. (This means that only those characters to the left of the asterisk are used as part of the search data). However, *&** should not be used since this denotes a system variable. (*&*** can be used to reset all system variables beginning with *&**.)

A plus sign (+) specifies that this character should be ignored when the variable name is being matched with the mask. A set of variable names which match the masked *&varname* will be deleted. Multiple plus signs (+) can be used.

Examples

```
GRESET &ab+d* &*ab*
```

resets two variables as follows:

- all variables with ab as the first two characters, any character as the third character and d as the fourth character.
- all system variables beginning with asterisk (*) that have ab as the second and third characters.

```
GRESET &**
```

resets all system variables.

IF

The IF statement is used to test for TRUE and FALSE conditions and for successful event completion, such as application connection, password validation, etc.

Format

IF *operand1* *function* *operand2* [*type*]
action-statements

where

operand1 is the first operand to be compared. This can be a variable which will be resolved before the IF is performed.

function can be EQ, NE, GE, GT, LE, LT or symbols =, >=, >, <=, <.

operand2 is the second operand to be compared. This can be a variable which will be resolved before the IF is performed.

type is the type of comparison to be made between *operand1* and *operand2* and can be one of the following:

THEN if the comparison between *operand1* and *operand2* is true, *action statements* will be performed. THEN does not have to be entered in the statement (optional for readability). If *type* is omitted THEN is the default.

PADNULL will pad *operand1*, if null, with blanks to the same length as *operand2* for comparison. For instance:

```
IF &var = '  ' PADNULL
```

will pad *&var*, if it is null, with a blank and hence treat null variables as equal to blanks.

SHORT comparison using a shorter operand length. For instance:

```
IF &a = &b SHORT
```

causes the comparison to be made for the shorter of the two lengths, for instance:

```
IF &var = &sysnull SHORT
```

always produces an equal condition because zero length data is compared.

ASIS ensures that both operands are treated literally, for instance:

```
IF 'ABCDE' = 'ABCDE ' ASIS
```

will not pad, etc, the operands which will therefore be unequal. Neither operand is uppercased so the IF becomes case sensitive.

action-statements may be any valid NCI/XF language statements.

Related statements

OR

ELSE

Notes

Comparison

If both operands are numeric, a true numeric comparison is done.

If one or both operands are non-numeric, comparison proceeds byte for byte from left to right. Both operands are internally uppercased before comparison. If the operands are of unequal length, the shorter operand is treated as padded with blanks to the length of the longer.

If one or both operands are null, operand length only is compared, (null=0 length).

ENDIF

There is no ENDIF construct within the NCI/XF language. The scope of an IF construct is delimited by the starting column of following statements, described for TRUE, FALSE and OR conditions in the following paragraphs.

TRUE and FALSE conditions

For a true condition, statements immediately after an IF statement starting in a column to the right of the IF are processed. At the end of the indented TRUE statements, processing will drop through to the next sequential statement unless an explicit branch (GOTO) is specified. The TRUE processing may itself include IF statements (nested IFs).

A false condition will cause an automatic branch to the next statement which starts in the same column as, or to the left of, the IF statement (except in the case of an IF immediately after another IF in the same column - see the note on OR processing on on page 1.64). Comment statements are ignored for the purpose of delimiting IF processing. Refer also to the ELSE statement.

```
If &var1 = 123
   set &var2 = true    <---- process if true
   goto lab1
Set &var2 = false      <---- process if false
Label lab1
.
.
.
```

OR construct

When an IF statement is immediately followed by either another IF statement starting in the same column or an OR statement starting in the same column, this is then treated as an OR condition. If either of the two contiguous statements yield true, processing will proceed as described for TRUE condition above.

```
If &sysdayn eq 6
Or &sysdayn eq 7
   set &msg = 'Overtime rates apply today'
```

```
If &sysdayn eq 6
If &sysdayn eq 7
   set &msg = 'Overtime rates apply today'
```

AND construct

The AND construct is supported by the use of nested IFs, an IF statement immediately followed by another IF statement starting in a column to its right.

```
If &sysdayn eq 1
   if &systime gt '12:00:00'
      set &msg = 'It is Monday afternoon'
```

Using the IF verb

Processing of an IF statement splits into three categories:

NULL, NUMERIC and CHARACTER

which are handled differently depending on comparison type as shown below and on the following pages.

THEN - *the default comparison*

Null

If one comparate (values) is null, then the comparison is carried out on lengths only.

```
$sysnull eq &sysnull
$sysnull lt notnull
$sysnull lt &sysblnk
$sysnull lt &sysxc00
```

Numeric

If both comparates are numeric, then the comparison is carried out numerically.

```
2 eq 2
2 eq 0002
2 gt '01'
```

Character

If neither comparate is null, and either is non-numeric, then EBCDIC character comparison is carried out. Both comparates are uppercased and any X'00' characters are translated to spaces (x'40'). The shorter of the two is right padded with spaces to the length of the longer. The EBCDIC value of the two equal length strings is now compared.

```
b eq B
b eq 'b '
b gt ' b'
b gt ' A'
b gt A
```

PADNULL - *as THEN, except null operands are padded with spaces*

Null

If either comparate is null the comparison is done as for character. The null comparate is padded with spaces to the length of the other. This is uppercased and all its x'00' are translated to spaces.

```
$sysnull eq &sysnull
$sysnull lt notnull
$sysnull eq &sysblnk
$sysnull eq &sysxc00
```

Numeric

As THEN.

```
2 eq 2
2 eq 0002
2 gt '01'
```

Character

As THEN.

```
b eq B
b eq 'b '
b gt ' b'
b gt ' A'
b gt A
```

SHORT - *as THEN, except the comparison is always made on the length of the shorter comparate only*

Null

If either comparate is null, the comparison is done on a zero length and will always yield equal.

```
$sysnull eq &sysnull
$sysnull eq notnull
$sysnull eq &sysblnk
$sysnull eq &sysxc00
```

Numeric

If both comparates are numeric, the comparison is carried out numerically on the length of the shorter comparate.

```
2 eq 2
2 gt 0002
2 gt '01'
```

Character

If neither comparate is null, and either is non-numeric, EBCDIC character comparison is carried out for the length of the shorter comparate. Both comparates are uppercased and any X'00' characters are translated to spaces (x'40'). The longer of the two is truncated (to the right) to the length of the shorter. The EBCDIC value of the two equal length strings is now compared.

```
b eq B
b eq 'b '
b gt ' b'
b gt ' A'
b gt A
```

ASIS - *as THEN, except that the comparates are not altered at all*

Null

As THEN.

```
$sysnull eq &sysnull
$sysnull lt notnull
$sysnull lt &sysblnk
$sysnull lt &sysxc00
```

Numeric

Even if both comparates are numeric, then EBCDIC string comparison is still carried out.

```
2 eq 2
2 gt 0002
2 gt '01'
```

Character

If neither comparate is null, and even if both are numeric, EBCDIC character comparison is still carried out, without altering the comparates at all.

```
b lt B
b lt 'b '
b gt ' b'
b gt ' A'
b lt A
```

Logic flow

If the comparison yields true, then any following statements are processed. If the comparison yields false, then all following statements are skipped, until one is found starting in the same column as the original IF. So, for

```
If &x eq &y then
   statement 1     <----------- true only
  statement 2      <----------- true only
   statement 3     <----------- true only
statement 4        <----------- either
 statement 5       <----------- either
```

if *&x* and *&y* are equal, statements 1 onwards will be processed. If they are not equal, only statements 4 onwards will be processed.

When the first statement found in the same column as the IF is an ELSE, the code following it is skipped or processed, depending on the original IF. So, for

```
If &x eq &y then
   statement 1     <----------- true only
   statement 2     <----------- true only
Else
   statement 3     <----------- false only
   statement 4     <----------- false only
statement 5        <----------- either
statement 6        <----------- either
```

if *&x* equals *&y*, statements 1, 2 and 5 onwards will be processed. If *&x* does not equal *&y*, statements 3, 4 and 5 onwards will be processed. An OR function is provided by following the IF with any number of OR statements (same syntax as IF) starting in the same column. When a comparison yields false, the following OR statement is processed instead. When a comparison yields true, all following OR statement are skipped. For instance:

```
If &x eq &y
Or &x eq &z
Or &y eq &z then
   statement 1
Else
   statement 2
```

would result in statement 1 being executed, if any two of *&x*, *&y* and *&z* were equal. If all three were different, then statement 2 would be processed.

An AND function is provided by following the IF with any number of IF statements starting in a column to the right of the IF. When a comparison yields false, the following IF statements are skipped. When a true comparison is made the following IF statement is processed. For instance:

```
If &x eq &y then
   if &x eq &z then
      statement 1
   else
      statement 2
Else
   statement 3
```

would result in statement 1 being executed, if all of &x, *&y* and *&z* were equal. If *&x* and *&y* were equal, but different from *&z*, then statement 2 would be processed. If *&x* and *&y* were different, then statement 3 would be processed.

Examples

```
If &company = a
   IF &dept = 1
      set &msg = ‘company is a and dept is 1’
      goto finish
   IF &dept = 2
      set &msg = ‘company is a and dept is 2’
      goto finish
   set &msg = ‘company is a and dept unknown’
   goto finish
IF &company = b
   IF &dept = 1
      set &msg = ‘company is b and dept is 1’
      goto finish
   IF &dept = 2
      set &msg = ‘company is b and dept is 2’
      goto finish
   Set &msg = ‘company is b and dept unknown’
   goto finish
Set &msg = ‘Company code not recognized’
Label finish
Panel
```

```
If &sysdayn=1
   set &msg = ‘it is Monday morning’
   If &systime gt ‘12:00:00’
      set &msg = ‘it is Monday afternoon’
      If &systime gt ‘17:00:00’
         set &msg = ‘It is Monday evening’
```

```
If &company = a
Or &company = b
   set &msg = ‘company a or b’
```

```
If &userid = &sysnull then
   set &emsg = ‘please enter your userid’
   panel
If &pass eq &sysnull then
   set &emsg = ‘please enter your password’
   panel
```

INCRement

The INCREMENT statement is used to increase a previously defined variable by 1. Only variables containing whole numbers up to the value of 999,999,999,999,999 can be incremented.

Format

INCREMENT *&var* [ZPAD]

where:

&var	is the variable to be incremented by 1.
ZPAD	maintains any existing zero padding. Padding will only extend as far as the number of zeros specified in the original SET statement.

Related statements

CALCULATE

DECREMENT

FCALCULATE

Examples

```
)proc
If &usraid = PF03
   if &systsoc = &systsom
      goto maxusers
   INCR &systsoc
   connect TSO
   exec testretc
   decrement &systsoc
   panel
```

If the specified conditions are met, this will increment the contents of variable *&systsoc* before connecting to TSO.

```
SET &count = 9
INCR &count ZPAD
```

sets *&count* to 10 and padding is maintained.

```
SET &count = 00009
INCR &count ZPAD
```

sets *&count* to 00010 and padding is maintained.

Note

An attempt to increment a variable not previously set to a numeric value or a value greater than 999,999,999,999,999 will result in a run-time error and the NCI/XF ERROR-EXEC will be driven. This will, by default, display panel NCERR5.

INTerval

The INTERVAL statement defines the number of terminal timer units which must finish, without any other terminal activity, before the)UPDATE code will be invoked.

Restrictions

INTERVAL is valid in the)PANEL section only.

Format

INTERVAL *n*

where

n is the number of whole timer units to expire without any terminal activity, in addition to the remainder of the current interval, before the)UPDATE section of the panel can be driven. The value specified must be numeric in the range 0 to 999,999.

Notes

An NCI/XF terminal timer unit is the number of seconds defined by the UPDATE-INTERVAL startup parameter multiplied by the TERMINAL-UPDATE value. Startup values may be overridden by an UPDATE or TERMUPDATE language statement.

Timer intervals are controlled by NCI/XF mainline code and are unrelated to the display of a particular panel. For instance, if the timer interval is 30 seconds at the time a panel is displayed, the current interval may have from 1 to 30 seconds before the next timer 'pops'. An understanding of this is fundamental to the correct use of the INTERVAL statement.

If INTERVAL 0 is specified, the)UPDATE section will be driven, even if terminal activity occurs, at every timer pop. This may be almost immediately after the original display of the panel and will be a maximum of the current timer interval less 1 micro-second, and every timer interval thereafter. If INTERVAL is omitted, this is the default.

If INTERVAL 1 is specified, the)UPDATE section will be driven, only if no terminal activity occurs, at the completion of 1 whole timer interval plus the remainder of the interval active at the time the panel is displayed. For instance, if the panel is displayed 10 seconds into a 30 second timer interval, the)UPDATE section will be driven, assuming no terminal activity occurs, 50 seconds later (30-10+30) and every 30 seconds thereafter.

If INTERVAL 2 is specified and the panel is first displayed 22 seconds into a 30 second interval, the panel will be eligible for)UPDATE processing 68 seconds later ((30-22)+30*2) and every 60 seconds (30*2) thereafter, assuming no terminal activity occurs.

LABel

The LABEL statement is used to indicate a position that can be branched to via the GOTO statement.

Format

LABEL *name*

where

name is a unique name indicating a possible branch position.

LABEL names are automatically uppercased on both LABEL and GOTO statements.

Related statement

GOTO

Example

```
)proc
If &usraid = PF03
   If &systsoc = &systsom
      goto maxusers
   increment &systsoc
   connect TSO
   decrement &systsoc
   exec testretc
   panel
LAB maxusers
Set &emsg = 'Maximum users already logged on'
Panel
```

LENgth

The LENGTH statement is used to establish the length of data supplied by the user.

Format

LENGTH *&len1* = *&var1*

where

&len1 is the variable to contain the length of supplied data.

&var1 is the variable to be tested.

Example

```
)proc
If &usraid eq pak1 then
   restart
LEN &LEN1 = &input
If &len1 = 1 then
   goto lab1
If &len1 gt 1 then
   panel &input
Label lab1
```

checks the length of the data in variable *&input* and performs functions depending on the result.

LOAD

The LOAD statement allows a user to load a copy of any re-entrant module into core.

Format

LOAD *modules.....*

where

modules are the module names to process. Up to 16 module names can be provided in one instruction.

Related Statements

CALL

FCALL

MODSTATS

UNLOAD

Return codes

If all modules are successfully loaded variable *&usrretc* is set to return code 0, otherwise the return code is set to the position in the list of the first module that failed to load.

Notes

Statistics at NCI/XF shutdown concerning the number of times a module has been called and loaded can be produced by coding the startup option

CALL-STATS=YES

Each module which has been called is then displayed with a count of the number of times it was called, and the number of times it required loading from disk. The same type of module-related statistics can be obtained during NCI/XF execution by either the MODSTATS verb or the @UTSTATS exec routine.

LOG

The LOG statement sends a user defined message to the NCI/XF log dataset (the log dataset is defined by the NCILOG DD statement under MVS or the NCILOG FILEDEF under VM).

A *level* operand can be set to determine the level at which the message is written to the log. This is compared with the LOGLEVEL statement, and if less than LOGLEVEL, the message will be written to the log. A *system-level* operand can be set to override the LOGLEVEL setting.

Format

LOG *text* [*level* [*system-level*]]

where

text	is any user definable message text.
level	is the level at which the text is to be written to the NCI/XF log. This must be numeric, between 0 and 9. The default is 9. The *level* operand is compared with the *system-level* operand, or if this is omitted, with the *level* operand on the LOGLEVEL statement. Text will be logged only if the *level* operand is less than or equal to the system level. If a *level* operand is not required, the *system-level* operand can only be entered if *level* is entered as '' or *&sysnull*.
system-level	will override the current system level (LOGLEVEL *level* operand). Text will be logged only if the *level* operand is less than or equal to the system level. This operand must be numeric, between 0 and 9. If omitted, text will only be written if *level* is less than or equal to the current LOGLEVEL *level* operand (*&sysllev*).

Note: If both *level* and *system-level* are omitted, the statement is always logged.

Related Statements

GETLOG

LOGLEVEL

Example

```
Loglevel 2                                /* set level 2
Set &*altlev = 7                          /* sys level 7
.
.
LOG 'terminal=&usrterm'                   /* logged
LOG 'jobname=&sysjobn' 3                  /* not logged
LOG 'panel=&usrpanl' 2                    /* logged
LOG 'time=&systime' 1                     /* logged
.
.
LOG 'return code=&usrretc' 7 &*altlev     /* logged
LOG 'date=&sysdate' 6 &*altlev            /* logged
LOG 'jobname=&sysjobn' 3  &*altlev        /* logged
LOG 'panel=&usrpanl' 2 &*altlev           /* logged
LOG 'time=&systime' 1 &*altlev            /* logged
LOG '*** END OF SUMMARY ***' 8 &*altlev /* not logged
.
.
Reset &*altlev
LOG '*** JOB ENDING ***' 1 &*altlev       /* logged
```

Notes

If the text of the message contains blanks (X'40'), the message must be enclosed in a pair of valid NCI/XF delimiter characters such as quotation marks ('), double quotation marks (") or slash (/).

NCI/XF variables may be included in the text of a message and will be resolved before transmission of the message.

A maximum of 111 bytes of user data will be displayed on a single line of the NCI/XF log; if the supplied text exceeds this, it will be truncated to 111 bytes.

LOGLevel

The LOGLEVEL statement sets the level at which log statements are to be written to the NCI/XF log. This is compared with the *level* operand on the LOG statement. If the LOG statement level is not coded or is less than or equal to the current LOGLEVEL setting, the message will be written to the NCI/XF log.

Format

LOGLEVEL *level*

where

level is the level at, or below which, the text is to be written to the NCI/XF log. If omitted, the default is 9.

Related Statements

LOG

Return codes

After execution of a LOGLEVEL statement, variable *&usrretc* is set to one of the following return codes:

0 Function completed successfully.
4 Invalid operand, must be numeric in the range 0 to 9.

Example

```
Loglevel 2                          /* set LOGLEVEL 2
.
.
LOG 'message 1'                     /* logged
LOG 'message 2' 3                   /* not logged
LOG 'message 3' 2                   /* logged
LOG 'message 4' 1                   /* logged
.
.
Loglevel 9                          /* set LOGLEVEL 9
.
.
LOG 'message 1'                     /* logged
LOG 'message 2' 3                   /* logged
LOG 'message 3' 2                   /* logged
LOG 'message 4' 1                   /* logged
```

LOGOFF

The LOGOFF statement allows a user to logoff from NCI/XF and return to VTAM.

Format

LOGOFF

Example

```
)proc
If &usraid eq pak1 then
   restart
If &1 = &sysnull then
   panel
If &1 = 1 then
   LOGOFF
If &1 = 2 then
   shutdown
```

will log the user off if the variable *&1* contains the value 1.

Notes

Variables relating to terminals are not reset as a result of the LOGOFF statement. You are recommended to issue RESET or GRESET statements if the terminal is to remain logged off for any length of time.

It is recommended that NCI/XF always controls all terminals via a VTAM LOGAPPL definition.

Message

The MESSAGE statement is used for sending messages between terminals. Messages are queued at the receiving terminal until they are referenced by the user variable *&usrmsg*. Once referenced, a message is automatically deleted by NCI/XF. The number of messages queued for a terminal is held in the user variable *&usrmsgn*. The maximum number of messages that can be queued for a terminal at one time is 5.

Format

MESSAGE *destination text*

where

destination can be either *luname* or OPERATOR:

luname is the VTAM NODENAME of the terminal to receive the message.

A message can be sent to multiple terminals in a single MESSAGE statement using mask characters as part of the *luname* operand. The two characters asterisk (*) and plus (+) are recognized as mask characters:

* specifies a generic group of terminals based on the characters before the asterisk. The message will be sent to all terminals where the characters before the asterisk match the corresponding character positions of the terminal name.

+ stipulates that this character position of the terminal name should be ignored when matching a terminal name against the *luname*.

OPERATOR indicates the message must be sent to the primary system operator console, unless MESSAGE is issued from a system console (or within a routine EXEC'd from a system console) using the extended modify support, in which case the message is returned to the issuing console.

text is the message text to be sent.

Return codes

After execution of a MESSAGE statement, *&usrretc* is set to one of the following return codes:

0 message sent to all specified terminals.
4 no target terminals connected to NCI/XF.
8 no target terminals known to NCI/XF.
12 message queue full at the single target terminal.
16 message not sent to at least one of the target group.

Notes

Text delimiters

If the text of the message contains blanks (X'40'), the message must be enclosed by a pair of valid NCI/XF delimiter characters such as quotation marks ('), double quotation marks (") or slash (/).

Variable text

NCI/XF variables may be included in the text of a message and will be resolved before transmission of the message.

Handling received messages

The MESSAGE statement will queue the message at the receiving terminal. It is the user's responsibility (at the receiving end) to ensure that received messages are taken from the queue and displayed. The number of messages queued at a terminal is available in variable *&usrmsgn*. This value is automatically incremented when a new message is received and added to the message queue. The text of received messages is available in variable *&usrmsg*. This variable will always contain the oldest message which has not been referenced by the receiving user. Whenever a message is referenced, by use of the variable *&usrmsg* in an NCI/XF language statement (for instance, SET &DISPMSG = &USRMSG) or by the display of a panel containing the variable *&usrmsg*, the message is automatically deleted from the queue and the count of pending messages in *&usrmsgn* is decremented. The next reference to *&usrmsg* will retrieve the next pending message.

Some of the techniques for handling received messages are:

1. Include the variable *&usrmsg* in every panel. Each message received will be displayed the next time any panel is displayed to the user.

2. Most panels will have an error message field in which to report user errors. This field may also be used to display received messages by including the following code at appropriate points in the)INIT and)UPDATE sections of the panels.

```
If &emsg = &sysnull   <--do not overwrite error msgs
 if &usrmsgn gt 0     <--any messages pending?
  set &emsg = &usrmsg <--display in error msg field
Panel                 <--show the panel
```

3. Include the variable *&usrmsgn* in every panel, with suitable descriptive text, to warn the user that messages are pending. Sound the terminal alarm if the value is non-zero. Provide a message viewing panel, which the user may invoke to see the messages.

```
If &usrmsgn gt 0
   beep
)body
terminal = &usrterm    pendingMESSAGES = &usrmsgn

<main body of panel>

Press PFK1 to see pending MESSAGES
)Proc
If &usraid = PF01
   panel viewmsg
```

4. Provide a separate message viewing panel, which may also invite the user to send a reply. At the terminal which sends the message, follow the MESSAGE statement with a TREFRESH statement to force the message viewing panel to be displayed at the receiving terminal.

```
M term001 '&systime from &usrterm-Hello'
Trefresh term001 viewmsg
```

Examples

The MESSAGE processor works using the terminal *luname*. Messages may be sent to a specific user or group of users by cross referring *userid* and *luname* with the QUEUE statement. A user should be added to the queue at logon time and removed at logoff time. When a message is to be sent, search the queue for the target userid to find the associated *luname* for use on the MESSAGE statement. The NCI/XF Demo System demonstrates this technique. (See *Demonstration menu (option 1)* on page 4.5 of the NCI/XF User Guide)

```
M local001 'hello terminal local001'
```

sends the message to terminal local001. Since the message contains blanks, it is enclosed in quotation marks.

```
M rmt007 'from &usrterm : Hello there'
```

sends the message to terminal rmt007. The sending terminal name is used to prefix the message. The variable *&usrterm* will be resolved before the message is sent.

```
M local003 /You need the 'REF' manual/
```

sends the message to terminal local003. As the message contains blanks and quotation marks, it is delimited by slashes (/).

```
M ++++++m2 'Message to model 2 screens'
```

sends the message to all terminals with the characters m2 in positions 7-8 of the *luname*.

```
M local* 'Message to local terminals'
```

sends the message to all terminals having an *luname* starting with the characters local.

```
Set &work = &syssuba(1:2)
Set &dest = '+++&work.*'
M &dest 'Message to LUs in NCIs subarea'
```

sends the message to all terminals where characters 4-5 of the *luname* match the VTAM subarea number in which NCI/XF is running.

```
)body
              MESSAGE SENDING PANEL

       Enter message text ==> &text

          and destination ==> &dest
)proc
M &dest '&systime From &usrterm :&text'
```

sends the message to the terminal specified by the user in variable *&dest*. The text of the message, typed by the user into variable *&text*, is prefixed by the time and the originating terminal name before being sent.

MODStats

The MODSTATS statement is used to extract statistics relating to the use of user written load modules invoked by the NCI/XF (F)CALL statement. The statistics are queued until they are referenced via the user variable *&usrmods*. Once referenced, the statistics are automatically removed from the queue by NCI/XF.

Format

MODSTATS *name* [*limit*]

where

name is the name of a load module or a variable set to the name of a load module. The statistics for the specified load module will be queued for later reference by use of the *&usrmods* variable. Mask characters can be specified to select a group of load modules whose statistics will be queued. The two characters asterisk (*) and plus (+) are recognized as mask characters:

- * specifies a generic group of load modules on the basis of the characters before the asterisk. Only those characters to the left of the asterisk are used as part of the search data.
- + specifies that this character position of the module name should be ignored when the module name is being matched with the mask.

limit specifies a numeric value or a variable containing the required value in the range 0 to 32767 and is used to limit the number of load module statistics queued as a result of group selection. If 0 is specified then the queue will be emptied. If omitted, there is no limit to the number of statistics queued.

Related Statements

CALL

FCALL

LOAD

UNLOAD

Return codes

After execution of a MODSTATS statement, variable *&usrretc* is set to one of the following return codes:

0 successfully extracted module statistics.
4 empty queue, no module statistics extracted.
8 syntax error.

Notes

Statistics queue

Whenever user variable *&usrmods* is referenced the statistics are automatically removed from the queue; if the queue is empty and *&usrmods* is referenced, then a null value will be returned.

The queued statistics from a MODSTATS statement are in order of module use. The first reference to *&usrmods* after a MODSTATS group selection will return the statistics of the module which was used earliest.

After a MODSTATS statement, if some of the queued statistics have not been referenced, some statistics will remain on the queue. If the remaining statistics are no longer required, the storage which they occupy can be freed by using a MODSTATS statement without operands, or by specifying a selection with a limit of 0.

Statistics available

When *&usrmods* is resolved, the following statistics will be returned separated by blanks:

- module name
- entry point address (set to zeros if module not loaded)
- number of calls requiring module to be loaded
- number of calls not requiring module to be loaded
- module load count
- number of CALL commands for this module currently active
- CPU time used by call routine (in milliseconds); this value is only accumulated when the call routine ends, and therefore does not include any calls to this module that are currently active.

  Note: This is only available under MVS/XA 2.2.0 and above; the value is zero for earlier versions.

Individual values can be extracted from *&usrmods* with the PARSE statement using blank as a delimiter.

Extending the statistics

Variable *&usrmods* may be extended in future releases of NCI/XF by adding new statistics. Any existing statistics will remain unchanged to maintain backward compatibility, and if PARSEd using blank as a delimiter, will maintain their relative position in the resultant variables *&usrpm01* to *&usrpmNN*.

Examples

```
MODS userprog
```

If module userprog has been used:

&usrretc = 0
&usrmods = userprog 00ABC860 0 3 1 0 10

indicating that userprog has been loaded and the entry point is at address 00ABC860, it has been called three times, was preloaded and has used ten milliseconds of CPU time.

If module userprog has not been used:

&usrretc = 4
&usrmods = a null value

```
MODS nc+log
```

assuming ncslog, ncqlog and ncvlog have been used:

&usrretc = 0
1st *&usrmods* = NCSLOG 00000000 1 0 0 0 4
2nd *&usrmods* = NCQLOG 81F223C0 2 3 2 0 9
3rd *&usrmods* = NCVLOG 81F31B58 0 10 1 0 39
4th *&usrmods* = a null value

NOALARM

The NOALARM statement disables a previous ALARM command.

Format

NOALARM

Related statements

ALARM

BEEP

NOBEEP

Example

```
NOALARM
If &1 = 'y' then
   goto yresp
Set &emsg = 'invalid option entered'
Alarm
Panel
Label yresp
```

NOBEEP

The NOBEEP statement turns off the terminal alarm for the next send only, after which the alarm setting reverts to the default as set by the ALARM or NOALARM statements.

Format

NOBEEP

Related statements

ALARM

NOALARM

BEEP

Example

```
Alarm
If &1 = 'y' then
   goto okresp
Set &emsg = 'invalid option entered'
Panel
*
Label okresp
NOBEEP
Panel &nxtpanl
```

OR

The OR statement enhances IF statement processing.

Format

IF *operand1* *function* *operand2*

OR *operand1* *function* *operand2* [*type*] *action-statements*

where

operand1	is the first of two operands to be compared. This can be a variable which will be resolved before the IF is performed.
function	can be EQ, NE, GE, GT, LE, LT or symbols =, >=, >, <=, <.
operand2	is the second of two operands to be compared. This can be a variable which will be resolved before the IF is performed.
type	is the type of comparison to be made between *operand1* and *operand2*. Refer to the IF statement for a complete description of the *type* operand.
action-statements	may be any valid NCI/XF language statements.

Related statement

IF

Notes

An OR statement must immediately follow, and start in the same column as, an IF statement or another OR statement.

Any OR statement which does not obey these rules will be treated as an IF statement.

For more details regarding operands, functions and comparator types, refer to the IF statement on page 1.62.

PADCenter

The PADCENTER statement will pad the specified variable to the requested length and center the data within the variable.

NCI/XF variables are of variable length, depending on the data set into them. Trailing blanks and nulls are truncated at SET time to conserve virtual storage. The PAD commands (PADC, PADL, PADR) allow a variable's length to be explicitly set by padding the existing data with a user specified pad character.

The PADC statement is particularly useful for centering error messages within a general purpose message area on the panel.

Where the padded length is shorter than the variable length, the variable will be truncated leaving only the central bytes.

Format

PADCENTER *&varname* *length* [*pad-char*]

where

&varname	is the name of the variable whose contents are to be padded and centered.
length	is the length to which the variable is to be padded.
pad-char	is the character to be used for padding the variable contents to the required length. If omitted, a default of space (X'40') is assumed.

Related statements

PADLEFT

PADRIGHT

Example

```
Set &emsg = 'invalid entry'
PADC &emsg 21 ' '
```

After execution of the above PADC statement, the variable *&emsg* is 21 characters long and the text 'invalid entry' is centered within the 21 bytes.

PADLeft

The PADLEFT statement will pad the specified variable to the requested length on the left side of the variable. The data is right-justified within the variable.

NCI/XF variables are variable length, depending on the data set into them. Trailing blanks and nulls are truncated at SET time to conserve virtual storage. The PAD commands (PADC, PADL, PADR) allow a variable's length to be explicitly set by padding the existing data with a user specified pad character.

The PADL statement is particularly useful for creating fixed length data for use as NCI/XF QUEUE elements.

Where the padded length is shorter than the variable length, the variable will be truncated to leave only the rightmost bytes.

Format

PADLEFT *&varname length* [*pad-char*]

where

&varname	is the name of the variable whose contents are to be padded and right justified.
length	is the length to which the variable is to be padded.
pad-char	is the character to be used for padding the variable contents to the required length. If omitted, a default of space (X'40') is assumed.

Related statements

PADCENTER

PADRIGHT

Example

```
Padr &userid 8
PADL &count 8 '0'
Queue add sysqueue first '&userid.&count'
```

After execution of the above PADL statement, the variable *&count* is 8 characters long and the data is right justified within the variable with leading zeros.

PADRight

The PADRIGHT statement will pad the specified variable to the requested length on the right side of the variable. The data is left-justified within the variable.

NCI/XF variables are variable length, depending on the data set into them. Trailing blanks and nulls are truncated at SET time to conserve virtual storage. The PAD commands (PADC, PADL, PADR) allow a variable's length to be explicitly set by padding the existing data with a user specified pad character.

The PADR statement is particularly useful for creating fixed length data for use as NCI/XF QUEUE elements.

Where the padded length is shorter than the variable length, the variable will be truncated leaving only the leftmost bytes.

Format

PADRIGHT *&varname length* [*pad-char*]

where

&varname	is the name of the variable whose contents are to be padded and left justified.
length	is the length to which the variable is to be padded.
pad-char	is the character with which to pad the variable's contents to the required length. If omitted, a default of space (X'40') is assumed.

Related statements

PADCENTER

PADLEFT

Example

```
PADR &userid 8
Padl &count 8 '0'
Queue add sysqueue first '&userid.&count'
```

After execution of the above PADR statement, the variable *&userid* is 8 characters long and the data is left justified within the variable with trailing spaces.

PANel

The PANEL statement is used to pass a user from one panel to another, or to redisplay the same panel.

Restrictions

PANEL is not allowed in

startup-panel	close-exec
initial-exec	final-exec
open-exec	MODIFY
update-exec	

Format

PANEL [*name* [CONT]]

where

name is the name of a panel or a variable set to the name of a panel. PANEL *name* causes the system to move from the current panel to panel *name*. If *name* is not specified, the current panel will be redisplayed, without re-executing the)INIT section. To force NCI/XF to process the INIT section when redisplaying the same panel, code *name* as *&usrpanl*.

CONT controls which panel is processed when PANEL statements have completed processing. The CONT option operates under three conditions as follows:

Example 1

PANEL *name1* CONT is coded. If a **RETURN** statement is encountered in panel *name1*, this is treated as a RESTART statement (returns a terminal to its highest-level panel)

Example 2

PANEL *name1* is coded. If a **RETURN CONT** statement is encountered in panel *name1*, this is treated as a RETURN statement (returns to the previous panel)

Example 3

PANEL *name1* CONT is coded. In panel *name1* PANEL *name2* is coded. If a RETURN CONT statement is encountered in panel *name2,* this causes the logic to resume at the statement following the PANEL *name1* CONT statement.

Examples of PANEL, RETURN and RESTART statements are provided on the following pages.

Related statements

RETURN

RESTART

Examples of PANEL, RETURN and RESTART statements

Statement	Current Panel	Current Panel becomes
RESTART	any	RESTART panel

The current panel becomes the RESTART panel.

Statement	Current Panel	Current Panel becomes	Previous panel	Previous Panel becomes
PANEL b	a	b		a
RETURN	b	a	a	
RETURN	a	RESTART panel		

RETURN at current panel b causes the system to revert to the first panel of the sequence, panel a, which has no previous panels. For storage and performance reasons NCI/XF retains a reference to one previous panel. If two RETURN statements are issued without an intervening PANEL statement, the effect is the same as RESTART.

Statement	Current Panel	Current Panel becomes	Previous panel	Previous Panel becomes
PANEL b	a	b		a
PANEL c CONT	b	c	a	
PANEL d	c	d		c
RETURN	d	c	c	
RETURN	c	RESTART panel		

In the first row the current panel moves from panel a to panel b and panel a becomes the previous panel. In the second row the current panel jumps from panel b to panel c; panel c has no predecessors because of PANEL c CONT and therefore any subsequent attempt to RETURN on that panel will result in a RESTART. In the third row the current panel moves from panel c to panel d and panel c becomes the previous panel. In the fourth row the current panel goes back to panel c from panel d and a null predecessor value is set for panel c by the PANEL CONT statement. In the fifth row a further attempt to RETURN at panel c, which has no predecessors and is therefore the initial panel of the sequence, forces a RESTART.

Statement	Current Panel	Current Panel becomes	Previous panel	Previous Panel becomes
PANEL b	a	b		a
PANEL c CONT	b	c	a	
PANEL d	c	d		c
RETURN CONT	d	b	a	

The first three rows are the same as in the previous table. The RETURN CONT on the final row returns the system to the statement after the last PANEL CONT statement, but causes the current and previous panels to be those applicable when the PANEL c CONT statement was issued, i.e. panel b and panel a.

Example

```
)proc
If &input = 1 then
   PAN ncimenu1
If &input = 2 then
   PAN ncimenu2
If &input = 3 then
   PAN ncimenu3
Set &emsg = “Invalid function selected”
PAN
```

Will pass the user to the specified panel depending on the contents of the variable *&input*.

Note

If PANEL *name* is not found in the PANEL or PBLIB libraries this will result in a runtime error and the NCI/XF ERROR-EXEC will be driven. This will, by default, cause panel NCERR1 to be displayed.

PANStats

The PANSTATS statement is used to extract statistics relating to the use of NCI/XF panels or subroutines invoked by the NCI/XF PANEL or EXEC statements. The statistics are queued until they are referenced via the user variable *&usrpans*. Once referenced, the statistics are automatically removed from the queue by NCI/XF.

Format

PANSTATS *name* [*limit*]

where

name is the name, or variable set to the name, of a panel or subroutine. The statistics for the specified panel or subroutine will be queued for later reference by use of the *&usrpans* variable. Mask characters can be specified to select a group of panels or subroutines whose statistics will be queued. The two characters asterisk (*) and plus (+) are recognized as mask characters:

- * specifies a generic group of panels or subroutines on the basis of the characters before the asterisk. Only those characters to the left of the asterisk are used as part of the search data.
- + specifies that this character position of the panel or subroutine name should be ignored when the panel or subroutine is being matched with the mask.

limit specifies a numeric value or a variable containing the required value in the range 0 to 32767 and is used to limit the number of statistics queued as a result of group selection. If 0 is specified then the queue will be emptied. If omitted, there is no limit to the number of statistics queued.

Related Statements

PANEL

EXEC

Return codes

After execution of a PANSTATS statement, variable *&usrretc* is set to one of the following return codes:

0 successfully extracted panel statistics.
4 empty queue, no panel statistics extracted.
8 syntax error.

Notes

Statistics queue

Whenever user variable *&usrpans* is referenced, the statistics are automatically removed from the queue; if the queue is empty and *&usrpans* is referenced, a null value will be returned.

The queued statistics from a PANSTATS statement are in alphabetical order.

After a PANSTATS statement, if some of the queued statistics have not been referenced, then some statistics will remain on the queue. If the remaining statistics are no longer required, the storage which they occupy can be freed by using a PANSTATS statement without operands, or by specifying a selection with a limit of 0.

Statistics available

When *&usrpans* is resolved the following statistics will be returned separated by blanks:

- panel/subroutine name
- total CPU time (in milliseconds) in the form T=*nnn*. Additionally for panels, the CPU time per panel section (milliseconds) in the form I=*n*, U=*nn* and P=*nn* for the INIT, UPDATE and PROC sections respectively, each separated by a comma (,). To summarize:

  for subroutines T=*nnn*

  for panels T=*nnn*,I=*n*,U=*nn*,P=*nn*

  Note: This is only available under MVS/XA 2.2.0 and above; the value is zero for earlier versions.

- number of executable statements in panel/subroutine

Individual values can be extracted from *&usrpans* with the PARSE statement using blank as a delimiter.

If the CPU time for a particular panel section is required, the individual values can be extracted (from the previously extracted CPU time field) with the PARSE statement using a comma (,) as a delimiter. The numeric values will then be in positions three onwards. This process is shown in the examples on the following pages.

Extending the statistics

Variable *&usrpans* may be extended in future releases of NCI/XF by adding new statistics to the end of the current statistics. Any existing statistics will remain unchanged to maintain backward compatibility and if PARSEd, using blank as a delimiter, will maintain their relative position in the resultant variables *&usrpm01* to *&usrpmnn*.

Examples

```
PANS testpan
```

If panel testpan has been used:

&usrretc = 0
&usrpans = testpan T=213,I=20,U=149,P=44 257

indicating that a total of 213 milliseconds of CPU time has been used by panel testpan, comprising 20 milliseconds in the INIT section, 149 milliseconds in the UPDATE section and 44 milliseconds in the PROC section. The statistics also show that there are 257 executable statements in the panel.

If panel testpan has not been used:

&usrretc = 4
&usrpans = null value

```
PANS testexec
```

If subroutine testexec has been used:

&usrretc = 0
&usrpans = testexec T=270 360

indicating that the subroutine has used a total of 270 milliseconds of CPU time and that there are 360 executable statements in the subroutine.

If subroutine testexec has not been used:

&usrretc = 4
&usrpans = null value

```
PANS ZZBRLOG*
```

Assuming the NCI/XF demonstration panel ZZBRLOG has been used to browse the NCI/XF log:

&usrretc = 0
1st *&usrpans* = ZZBRLOG T=1,I=1,U=0,P=0 45
2nd *&usrpans* = ZZBRLOGI T=594 103
3rd *&usrpans* = ZZBRLOGP T=1315 552
4th *&usrpans* = ZZBRLOG2 T=1,I=1,U=0,P=0 5
5th *&usrpans* = null value

Extracting CPU times for individual sections of a panel

```
PANS ZZBRLOG
if &usrretc EQ 0
  parse &usrpans using ‘ ‘
  set &name = &usrpm01
  set &stmts = &usrpm03
  parse &usrpm02 using ‘,’
  set &tot = &usrpm01(3:)
  if &usrpm00 GT 1
    set &init = &usrpm02(3:)
    set &upda = &usrpm03(3:)
    set &proc = &usrpm04(3:)
    set &type = PANEL
else
  set &type = EXEC
```

assuming the same values for ZZBRLOG as in the previous example (following the PANSTATS ZZBRLOG statement, *&usrpans* contains ZZBRLOG T=1,I=1,U=0,P=0 45) and after execution of this code, the following variables are set:

&name	=	ZZBRLOG
&stmts	=	45
&tot	=	1
&init	=	1
&upda	=	0
&proc	=	0
&type	=	PANEL

PARse

The PARSE statement allows a line of text to be split into separate tokens based on user defined separators.

Format

PARSE *&text* USING *&mask* [DELIMITERS *&dels*]

where

&text	is the text to be parsed.
USING	indicates that the following character or characters (*&mask*) are to be used as separators when parsing the text.
&mask	is the character or characters to be used as separator(s) when parsing the text. This may be a literal or a variable containing the mask characters. If a space is to be used as a separator, the entry must be enclosed in single quotes. If single quotes are to be used as separators the entry must be enclosed in double quotes.
DELIMETERS	indicates that the following two characters (*&dels*) are to be used to define the beginning and end of a part of the text which is not to be parsed.
&dels	are two characters used to define the beginning and end of a part of the text which is not to be parsed. If a space is to be included as one of the characters the entry must be enclosed in single quotes. If single quotes are to be used as beginning and end characters they must be enclosed in double quotes.

Examples

```
Set &input = ‘free format text’
PAR &input using ‘ ‘
```

will set

&usrpm00 = 3
&usrpm01 = free
&usrpm02 = format
&usrpm03 = text

Variables *&usrpm04* to *&usrpm99* will retain the values held before the PARSE was executed, therefore you may want to issue a GRESET *&usrpm** statement before the PARSE statement.

```
Set &input = ‘logon applid(tso) logmode(sna2)’
PAR &input using ‘ ()‘
```

will set:

&usrpm00 = 5
&usrpm01 = logon
&usrpm02 = applid
&usrpm03 = tso
&usrpm04 = logmode
&usrpm05 = sna2

Variables *&usrpm06* to *&usrpm99* retain the values held before the PARSE statement was executed.

```
Set &input = “find ‘Hello Fred’”
PAR &input using ‘ ‘ delimiters “‘’”
```

will set:

&usrpm00 = 2
&usrpm01 = find
&usrpm02 = ‘Hello Fred’

Variables *&usrpm03* to *&usrpm99* retain the values held before the PARSE statement was executed.

Notes

A PARSE statement will set the number of variables that have been produced in variable *&usrpm00* and the resulting tokens in variables *&usrpm01* to *&usrpm99*.

If a PARSE statement results in more than 99 individual tokens, variable *&usrpm00* is set to 100 and the first 99 tokens are placed into variables *&usrpm01* to *&usrpm99*. The remaining unparsed text is placed in variable *&usrpmrm*.

POST

The POST statement causes another terminal to continue processing at the instruction following its associated WAIT statement without having to display a screen and perform a TREFRESH or PREFRESH.

A terminal can be POSTed from any other terminal.

Format

POST *terminal-id*

where

terminal-id is the name of a terminal or a variable set to the name of a terminal.

Related statement

WAIT

PREFresh

The PREFRESH statement refreshes all terminals displaying the specified panel(s), with either the same or a different panel.

Format

PREFRESH *panel1* [*panel2* [*alarmopt*]]

where

panel1 is the name of a panel, or a variable set to the name of a panel. Every terminal currently displaying the specified panel will be refreshed with this panel; or if the *panel2* operand is supplied, refreshed with that panel. Mask characters can be specified to define a group of panels to be refreshed (see *Mask characters* on page 1.108).

panel2 is the name of the panel, or a variable set to the name of the panel, for display on all terminals currently displaying *panel1*. Mask characters can be specified to define the name of the panel (see *Mask characters* on page 1.108). If a *panel2* operand is not required, the *alarmopt* operand can only be entered if *panel2* is entered as a space between single quotes (' ') or as *&sysnull*.

alarmopt specify one of the following NCI/XF language statements to control the terminal alarm on the receiving terminal:

ALARM | BEEP | NOALARM | NOBEEP

These are described in this chapter under the corresponding language statements. If omitted, the default leaves the current alarm setting at the refreshed terminal.

Examples

Refer to the Notes for detailed explanations of the PREFRESH operands.

```
PREF menu1
```

All terminals displaying menu1 will be immediately refreshed with the same panel.

```
PREF menu1 menu1
```

All terminals displaying menu1 will be immediately refreshed. The)INIT section of panel menu1 will be redriven.

```
PREF menu1 news1
```

Panel news1 will be displayed on all terminals currently displaying menu1.

```
PREF menu*
```

All terminals displaying a panel with a name beginning menu will be immediately refreshed with that panel, for example, menu, menu5, menu051.

```
PREF ++++++m2
```

All terminals displaying a panel which has a name with m2 as its seventh and eighth characters will be immediately refreshed with that panel.

```
PREF logo1 news*
```

Panel news1 will be displayed on all terminals currently displaying panel logo1.

```
PREF ++++++m2 update*
```

All terminals displaying a panel which has a name with m2 as its seventh and eighth characters will be updated with panel updatem2.

```
PREF logo* news*
```

All terminals displaying a panel whose name starts with logo will be updated with a panel whose name begins news. The rest of the name will be built from the original name, for example:

LOGO123==>NEWS123

LOGOX==>NEWSX

```
PREF a+b+c* yz+++*
```

All terminals displaying a panel with a in position 1, b in position 3 and c in position 5 will be replaced with a panel starting yz, for example:

A1B2CNNN==>YZB2CNNN

A9B8C7==>YZB8C7

Notes

Mask characters

Two special characters (* and +) are recognized as mask characters when specifying operands on a PREFRESH statement. The meaning of the characters differs according to whether they are used as part of the first or second operand.

When used within the first operand (*panel1*)

- * specifies a generic group of panels based on the characters before the asterisk. When panels to refresh are being searched for, only those characters to the left of the asterisk are part of the search criteria.
- + specifies that this character position of the panel name should be ignored when matching the panel name with the mask.

When used within the second operand (*panel2*)

- * specifies that the remainder of the panel name, starting with the character position which contains the asterisk (*), should be built using the corresponding character positions from the name of the panel currently being displayed on the target terminal.
- + specifies that the character position containing the plus sign (+) should be replaced with the corresponding character position from the name of the panel currently being displayed on the target terminal.

Redisplaying the same panel

If only *panel1* is specified, the panel area of terminals currently displaying the requested panel will be refreshed (the same panel will be redisplayed). If any variables displayed on the panel have changed, the updated values will be displayed to the user. This is particularly useful if, for instance, the status of an application changes. The PREFRESH statement may be used to immediately refresh all terminals which display an application menu panel with the updated status information.

If you specify only the first operand, the)INIT section of the panel will not be redriven; it will be exactly as if a PANEL statement with no operands had been executed at the target terminal(s).

If you want to redisplay the same panel and redrive its)INIT section, code the same panel name as the first and second operands.

Displaying a different panel

If you specify *panel2*, all terminals currently displaying *panel1* will be immediately refreshed with *panel2*. The)INIT section of *panel2* will be driven, exactly as if a "PANEL *panel2*" statement had been executed at the target terminal(s).

Refreshing multiple panels

Several panels can be refreshed or replaced by using mask characters as part of the first operand of a single PREFRESH statement. The name of the replacement panel to be sent to the target terminals(s) may be specified generically by use of mask characters in the second operand.

Target terminal status

All terminals connected to NCI/XF have a refresh status which may be ON, OFF or DELAY. At initial connection time, all terminals have a refresh status of ON, indicating that REFRESH commands will be processed. Execution of a TURN REFRESH statement may subsequently alter the refresh status to OFF, to ignore REFRESH operations, or DELAY to queue them. Refer to the description of the TURN statement for further details.

Queueing of PREFRESH requests

PREFRESH updates the display area of all terminals displaying *panel1* when the PREFRESH command is issued. This is a synchronous operation and control is not returned to the issuing terminal until all required terminal SENDs have been queued to VTAM.

Since all NCI/XF mainline code, apart from the exceptions listed below, is itself serial and synchronous, there is no danger of the target terminals being interrupted in the middle of NCI/XF language processing.

Exceptions to this are terminals which have issued the following asynchronous requests:

- a CALL statement to invoke an asynchronous exit
- a STATUS request to obtain application status
- a CONNECT request, in which case the terminal may be connected to another application
- a VTAM SEND of a panel to a terminal

In all these cases, the PREFRESH request will be queued at the receiving terminal(s) and executed when NCI/XF regains control at the termination of the asynchronous request or the application session.

Users of previous releases

The PREFRESH command is particularly useful for advising all users of critical changes occurring within the network (such as application status changes, updates to hot news broadcast messages) or for sending messages to other terminal users.

Users of previous releases of NCI/XF, or customers upgrading from NCI, who use)UPDATE processing to handle the automatic updating of terminal displays are strongly recommended to consider PREFRESH as a more efficient alternative.

QUERYTErm

The QUERYTERM statement determines if a terminal is connected to NCI/XF. It also determines if a terminal has been passed from NCI/XF to an application using the CONNECT or CONNECTR statement. If the terminal has ever been connected to an application, the name of the application is placed in variable *&usrappl.*

Format

QUERYTERM *name*

where

name is the name of the terminal for which the connect information is required.

Return codes

After execution of a QUERYTERM statement, variable *&usrretc* is set to one of the following return codes:

0 terminal connected to NCI/XF.
4 terminal connected to application (previously connected to NCI/XF).
8 terminal is logged off.
20 unknown terminal (has never been connected to NCI/XF).

Example

```
QUERYTERM A01MS009
```

will return the following information if terminal A01MS009 has been connected to an application (APPL1) using a CONNECT or CONNECTR statement:

&usrappl = APPL1
&usrretc = 4

QUEue

Throughout the following sections, the term *queue* is used as a generic term to describe any collection of logically related data which is to be grouped and manipulated as a queue, a table, a chain or a push-down stack.

An NCI/XF queue is a free format collection of data. As with NCI/XF variables, no data definition of a queue is required. Using a queue name in an appropriate NCI/XF language statement automatically creates the queue. Whether the queue is viewed by the user as a queue, or as a chain, a table or a stack depends entirely on the data stored within it and the search method employed.

For an overview of NCI/XF queues refer to the NCI/XF User Guide, *Chapter 2 - How to code NCI/XF*.

The QUEUE statement may be used to build, maintain and search queues.

The QUEUE statement provides options to:

- ADD an element to a queue
- BUILD an index to an existing queue using BLDINDEX
- COPY an element or elements to a queue
- DELETE an element from a queue
- FIND a queue element by position, or by searching for a string or substring
- GPURGE a generic PURGE of one or more complete queues based on a queue mask
- PURGE a complete queue
- REPLACE existing data of a queue element with new supplied data (This is equivalent to QUEUE DELETE followed immediately by a QUEUE ADD.)

The various formats of the QUEUE statement are described on the following pages.

QUEue ADD

The QUEUE ADD statement is used to add an element to a queue. If the named queue does not exist, a new queue will be automatically created to contain the new element.

Format

QUEUE ADD *qname pos string*

where

qname is the name of the queue, or a variable containing it. If this queue does not exist, a new queue will be created. The queue name may be up to 255 bytes long.

pos identifies where in the queue the element is to be added and can be one of the following:

FIRST indicates that the element is to be added to the top of the queue.

LAST indicates that the element is to be added to the bottom of the queue.

n is the position in the queue which the new element is to become. The new element will effectively be inserted before the element which currently occupies position *n*. The new element will then become position *n*.

This may be an explicit numeric value, or a variable containing a numeric value.

CURR indicates that the element is to be added at the current position. The current position is the position of the element last accessed.

Following a successful add the current position is that of the element added.

PREV indicates that the element is to be added at the position immediately before the current position.

NEXT indicates that the element is to be added at the position immediately after the current position.

string is the data which is to be added at the new queue element. It may be specified as a literal or as a variable containing the required data.

Return codes

After execution of a QUEUE ADD statement, variable *&usrretc* is set to one of the following return codes:

0	element successfully added to the queue.
8	invalid position specified, for instance you have tried to add an element as position 8 and the queue only contains 6 entries.
12	broken queue (This should never occur. Report any occurrence to your local support office).
16	syntax error
20	QUEUE ADD cannot be performed on an index.

Examples

```
QUE ADD *qname first &data
```

adds an element to the top of a queue.

```
QUE ADD localq last user123
```

adds an element to the bottom of a queue.

```
QUE ADD myqueue 5 &data
```

adds an element at position 5 in the queue.

```
QUE ADD sysqueue first ‘user123 term005’
```

adds an element to the top of the queue. The data is specified as a literal and contains spaces, so the literal is enclosed in quotation marks.

```
que find *qname user1
Incr &usrqpos
QUE ADD *qname &usrqpos user2
```

adds an entry for user2 after the entry for user1.

```
que find *qname user1
QUE ADD *qname curr user2
```

establishes the current position as the element for user1 using QUEUE FIND; adds an entry for user2 before the entry for user1; the element for user2 becomes the current position.

```
que find *qname user1
QUE ADD *qname prev user2
```

establishes the current position as the element for user1 using QUEUE FIND; adds an entry for user2 before the element previous to the current position; the element for user2 becomes the current position.

```
que find *qname user1
QUE ADD *qname next user2
```

establishes the current position as the element for user1 using QUEUE FIND; adds an entry for user2 after the entry for user1; the element for user2 becomes the current position.

Notes

Defining a queue

As with NCI/XF variables, no explicit definition is required for an NCI/XF queue. If a queue of the specified *qname* does not exist when the QUEUE ADD request is issued, a new queue of that name will be automatically created to contain the added element.

If the specified *qname* begins with SYS or an asterisk (*), the queue is a system queue, and one queue of the specified name exists in the NCI/XF address space for access by any terminal.

If the specified *qname* begins with characters other than SYS or *, the queue is a local, or private, queue. A unique version of the queue will be maintained for each terminal accessing the specified *qname*.

Queue elements

A queue element may be any length up to the maximum allowed by the NCI/XF language. If a literal string is specified, the length is limited by the length of the line containing the QUEUE ADD statement. If a variable name is specified for the string, the length is limited to 32767 bytes, the maximum length of any NCI/XF variable.

Elements within the same queue may be of variable length; for instance, you may add an element which is 10 characters long, then add another which is 20 characters long. On a subsequent FIND operation, NCI/XF does not return a length value for the retrieved queue element. The LENGTH language statement may be used for this purpose. For details of techniques for handling fixed or variable length queue elements, refer to the following notes.

Elements can only be added to a queue by position. If you want to add an element relative to another element already in the queue, you must first find the position of the existing element using the QUEUE FIND statement. This will return the position of the existing element in variable *&usrqpos* which can then be incremented or decremented as required before issuing the QUEUE ADD request.

Fixed length element

Elements within a queue may be of fixed or variable length. Fixed length queue elements are often the most useful since they can be easily searched using the substring option of the QUEUE FIND statement. If each queue element conforms to the same fixed format, the column number for a substring search may be reliably predicted. (For further details on substring searches and the use of sub-queues, see *QUEue FIND* on page 1.126). Fixed length queue elements can be built by padding each component of the element, using one of the NCI/XF pad statements PADC, PADL, PADR, before building the element and adding it to the queue.

```
Set  &element = &userid
Padr &element 8
Set  &element = &element&usrterm
Padr &element 16
Set  &element = &element&group
QUE  ADD &sample last &element
```

By padding each component (userid and terminal name) to a fixed length, the terminal name always occurs in column 9 and the user group identifier is always in column 17. You may now reliably search the queue by userid, terminal name or user group, using the QUEUE FIND command because you can predict which columns will contain the search key required.

Variable length elements

Queues containing variable length elements are a more efficient means of storing data, when each element contains only one discrete piece of data, or where the queue will only ever be searched by position or by the major key starting in column 1. See the examples on the next page.

When accessing a queue with variable length elements, NCI/XF does not specifically inform the user of the length of the located element. The length is implied by the length of the variable *&usrqstr* which contains the element found by a QUEUE FIND operation. The LENGTH statement may be used to determine the length of the located element as follows

```
Que find sampleq &data
Length &datalen = &usrqstr
```

If a queue element is to contain more than a single discrete piece of data but the queue is only ever going to be searched by position (FIRST, LAST, CURR, PREV, NEXT) or by key starting in column 1, variable length queue elements may be used with suitable separator characters between the discrete parts of the data.

The PARSE statement can be used to break a retrieved element into its component parts as follows:

```
QUE ADD *sample last '&userid/&usrterm/&usrlogm'
Que find *sample last
Parse &usrqstr using '/'
Length &totllen = &usrqstr
Length &userlen = &usrpm01
Length &termlen = &usrpm02
Length &modelen = &usrpm03
```

If the element retrieved was for userid FRED on terminal LOCAL12 using logmode D4C32782, the above code would have caused the following values to be set:

&usrqstr	FRED/LOCAL12/D4C32782	=	located element
&totllen	21	=	element length
&usrpm01	FRED	=	userid
&userlen	4	=	userid length
&usrpm02	LOCAL12	=	terminal
&termlen	7	=	terminal length
&usrpm03	D4C32782	=	logmode
&modelen	8	=	logmode length

QUEue BLDINDEX

The QUEUE BLDINDEX statement is used to create an index on the basis of a key field in an existing queue. An element from the base queue can be retrieved by performing a QUEUE FIND on the index using a key.

Format

QUEUE BLDINDEX *qindex qbase* [*keypos, keylen, hashamt*]

where

qindex is the name of the index to be created, or a variable containing that name.

qbase is the name of the base queue containing the data to be indexed, or a variable containing that name.

keypos is the starting position of the key field in the base queue element. The number specified can be in the range 1 through 32767; if omitted the default is 1. If a *keypos* operand is not required, the *keylen* and *hashamt* operands can only be entered if *keypos* is entered as ' ' or *&sysnull*. Commas must separate *keypos*, *keylen* and *hash*.

keylen is the length of the key field in the base queue element. The number specified can be in the range 1 through 16; if omitted the default is 16. If a *keylen* operand is not required, the *hashamt* operand can only be entered if *keylen* is entered as ' ' or *&sysnull*. Commas must separate *keypos*, *keylen* and *hash*.

hashamt specifies the number of hash chains allocated for this index. The number specified can be in the range 2 through 32767; if omitted the default is 10. The optimum value for any index depends on the number of elements in the base queue. A larger value means more storage but less CPU.

Return codes

After execution of a QUEUE BLDINDEX statement, variable *&usrretc* is set to one of the following return codes:

0 index built successfully.
4 base queue not found or index name already exists.
12 broken queue (This should never occur. Report any occurrence to your local support office).
16 syntax error.

Example

```
QUE BLDINDEX indx base 10,8,2
```

builds an index indx based on the data elements in an existing queue base. The key data used as a basis for the index starts at character position 10 in the base queue elements and has a length of 8 bytes. Two hash chains will be used. QUEUE FIND against the index with a specified key will search the base queue for a matching key; this will result in variable *&usrqstr* being set to the complete contents of the matching base queue element.

Notes

Once built, the index is automatically maintained by NCI/XF; any updates to the base queue will be reflected in the index. This introduces an overhead to any updates performed on the base queue to an index. The order of existing base queue elements which have duplicate keys will be repeated in the index when it is initially built, but any subsequent maintenance to the base queue that results in the addition of another element with a duplicate key will cause that element to be added to the index as the last element amongst those with duplicate keys.

One base queue may have several indexes.

An index cannot be updated directly. QUEUE ADD, QUEUE COPY, QUEUE DELETE and QUEUE REPLACE are not permitted on an index and result in return code 20.

QUEUE PURGE can be used to purge an index without affecting the base queue, but if the base queue is purged then so is any associated index.

An index to a system queue must be a system index and an index to a user queue must be a user index; any attempt to mix base queue and index types will result in return code 16, which will also be produced if the base queue is itself an index and not a genuine base queue. An index cannot be indexed.

QUEue COPY

The QUEUE COPY statement is used to copy the contents of one queue into another queue. If elements already exist in the queue you are copying to, QUEUE COPY will insert the copied data into the existing queue; it will not overwrite. Element(s) may be copied to a queue on the basis of position in the queue.

Format

QUEUE COPY *qto pos qfrom* [FROMCURR/TOCURR]

where

qto is the name of the queue to copy into, or a variable containing the name.

pos indicates where in *qto* the elements(s) copied from *qfrom* is/are to be inserted. This can be one of the following:

FIRST indicates that the element or elements copied from *qfrom* are to be inserted at the top of *qto*.

LAST indicates that the element or elements copied from *qfrom* are to be inserted at the bottom of *qto*.

n is the position in *qto* which the copied element(s) will occupy. The copied element(s) will effectively be inserted before the element which currently occupies position *n*.

This may be an explicit numeric value, or a variable containing a numeric value.

CURR indicates that the element(s) copied from *qfrom* is/are to be inserted before the current position in *qto*.

The current position is the position of the element last accessed. The current position may be set using QUEUE FIND. The element returned will become the current position.

After a successful copy, the current position becomes that of the element copied. If multiple elements are copied, the current position becomes that of the last element copied.

An unsuccessful copy does not affect the current position.

PREV indicates that the element or elements copied from *qfrom* are to be inserted before the element which precedes the current position in *qto*.

NEXT indicates that the element or elements copied from *qfrom* are to be inserted after the current position in *qto*.

qfrom is the name of the queue to copy from, or a variable containing the name.

FROMCURR indicates that elements from and including the current position in *qfrom* to the last element in *qfrom* are to be inserted in *qto* at the position specified.

TOCURR indicates that the elements from the top of *qfrom* up to and including the current position in *qfrom* are to be inserted at the position specified.

Return codes

After execution of a QUEUE COPY statement, variable *&usrretc* is set to one of the following return codes:

0 element successfully copied to the queue.
4 FROM queue not found or element not found.
8 invalid position specified, for instance, you have tried to copy element 8 and the queue only contains 6 elements.
12 broken queue (This should never occur. Report any occurrence to your local support office).
16 syntax error.
20 QUEUE COPY cannot be performed using indexes.

Examples

```
QUE COPY *qname1 first *qname2
```

copies the contents of *qname2 to the top of *qname1; *qname2 remains unchanged.

Before		After	
qname1	qname2	qname1	qname2
user1 user10 user99	user5	**user5** user1 user10 user99	user5

.

```
QUE COPY *qname1 last *qname2
```

copies the contents of *qname2 to the bottom of *qname1; *qname2 remains unchanged.

Before		After	
qname1	qname2	qname1	qname2
user1 user10 user99	user5	user1 user10 user99 **user5**	user5

```
QUE COPY *qname1 4 *qname2
```

copies the contents of *qname2 to the fourth position in *qname1; *qname2 remains unchanged.

Before		After	
qname1	qname2	qname1	qname2
user1 user10 user99 user101	user5	user1 user10 user99 **user5** user101	user5

```
que find *qname1 user10
QUE COPY *qname1 curr *qname2
```

establishes the current position as that of the element for user10; copies the contents of *qname2 to the current position in *qname1; *qname2 remains unchanged.

Before		**After**	
qname1	qname2	qname1	qname2
user1 **user10** **(current)** user99 user101	user5	user1 **user5** user10 user99 user101	user5

```
QUE COPY localq1 prev localq2
```

copies the contents of localq2 to the last but one position before the current element in localq1; localq2 remains unchanged.

Before		**After**	
localq1	localq2	localq1	localq2
user1 user10 user99 **user101** **(current)**	user5	user1 user10 **user5** user99 user101	user5

```
QUE COPY localq1 next localq2
```

copies the contents of localq2 to the position immediately after the current position in localq1; localq2 remains unchanged.

Before		**After**	
localq1	localq2	localq1	localq2
user1 user10 **user99** **(current)** user101	user5	user1 user10 user99 **user5** user101	user5

```
QUE COPY localq1 next localq2 FROMCURR
```

copies all the elements from and including the current position up to the last element in localq2 and inserts them after the current position in localq1.

Before		**After**	
qname1	qname2	qname1	qname2
user1 user10 **user99** **(current)** user101	user5 **user6** **(current)** user7 user8	user1 user10 user99 user5 user6 user7 user8 user101	user5 user6 user7 user8

Notes

If the specified *qto* or *qfrom* begins with sys or an asterisk (*), the queue is a system queue, and one queue of the specified name exists in the NCI/XF address space for access by any terminal.

If the specified *qto* or *qfrom* begins with characters other than sys or an asterisk (*) the queue is a local, or private, queue. A unique version of the queue called *qto* or *qfrom* will be maintained for each terminal referencing the specified *qto* or *qfrom*.

You can copy to a queue by position (FIRST, LAST, *n*, CURR, PREV, NEXT). QUEUE FIND may be used to establish the current position when using the CURR, PREV or NEXT operands. When this technique is being used to copy elements to a queue, the COPY should immediately follow the FIND request in the same panel section. This will ensure that the queue is not modified by another terminal user, which might have altered the current position in the queue to which you are copying.

QUEue DELETE

The QUEUE DELETE statement is used to delete an element from a queue and free the storage occupied by the element.

Format

QUEUE DELETE *qname pos*

where

qname is the name of the queue, or a variable containing the name.

pos indicates where in the queue the element is to be removed from. This can be one of the following:

- FIRST indicates that the first (top) element in the queue is to be deleted.
- LAST indicates that the last (bottom) element in the queue is to be deleted.
- *n* is the position in the queue of the element to be deleted. This may be an explicit numeric value, or a variable containing a numeric value.
- CURR indicates that the element at the current position is to be deleted. The current position is the position of the element last accessed.
- PREV indicates that the element immediately before the current position is to be deleted.
- NEXT indicates that the element immediately after the current position is to be deleted.

Return codes

After execution of a QUEUE DELETE statement, variable *&usrretc* is set to one of the following return codes:

0 element successfully deleted from the queue.
4 queue not found or element not found.
8 invalid position specified, for instance you have tried to delete element 8 and the queue only contains 6 elements.
12 broken queue (This should never occur. Report any occurrence to your local support office).
16 syntax error.
20 QUEUE DELETE cannot be performed in an index.

Examples

```
QUE DELETE *qname first
```

removes the first (top) element from the queue.

```
QUE DELETE localq last
```

removes the last (bottom) element from the queue.

```
que find *qname user123
If &usrretc = 0
   QUE DELETE *qname &usrqpos
```

removes the entry for user123 from the queue.

```
QUE DELETE sysqueue 7
```

removes the seventh element from the queue.

```
que find *qname user1
QUE DELETE *qname curr
```

establishes the element for user1 as the current position using QUEUE FIND; removes the element at the current position from the queue.

```
que find *qname user2
QUE DELETE *qname prev
```

establishes the element for user2 as the current position using QUEUE FIND; removes the element immediately before the current position.

```
que find *qname user2
QUE DELETE *qname next
```

establishes the element for user2 as the current position using QUEUE FIND; removes the element immediately after the current position.

QUEue FIND

The QUEUE FIND statement searches the named queue and retrieves the element meeting the specified search criteria. An element may be retrieved on the basis of its position in the queue, its contents, or a substring of its contents. QUEUE FIND can be used after QUEUE BLDINDEX to search the index built by that command. See *Format of QUEUE FIND via an index* on page 1.134.

Format

QUEUE FIND *qname element* [*substring* [*subcol*]]

where

qname is the name of the queue, or a variable containing the name.

element identifies the element to be returned on the basis of its position in the queue or its contents, using one of the following:

FIRST If *substring* is not specified, the first (top) element on the queue will be returned. If *substring* is specified, the first element containing the specified *substring* starting in column *subcol* will be returned.

LAST If *substring* is not specified, the last (bottom) element on the queue will be returned. If *substring* is specified, the last element containing the specified *substring* starting in column *subcol* will be returned.

n If *substring* is not specified, the element in the specified position (*n*) in the queue will be returned. If *substring* is specified, the element returned will be the *n*th occurrence of an element containing the specified *substring* starting in the specified *subcol.* For instance, if *n* is 8, the eighth occurrence of an element containing the specified *substring* will be returned.

CURR will return the element at the current position. The current position is the position of the element last accessed.

PREV If *substring* is not specified, the element in the queue immediately before the current element will be returned. If *substring* is specified the first element containing the specified *substring* starting in the specified *subcol* before the current position in the queue will be returned.

NEXT If *substring* is not specified the element in the queue immediately after the current position will be returned. If *substring* is specified the first element containing the specified *substring* starting in the specified *subcol* after the current position in the queue will be returned.

string is a literal or variable to be matched against the elements in the queue.

If *substring* is not specified, the element returned will be the first element whose contents match *string*. The length used for the comparison is the length of each queue element. (If the length of *string* is different from the length of an element, that element will not be selected)

If substring is specified, matching proceeds as described above as if *substring* had not been specified, except that variable *&usrqgpo* will be set to indicate the position of the element within the subqueue defined by *substring*. (See *Subqueues* on page 1.136).

substring specifies that the search is to only match against those elements which contain the specified *substring* starting in the character position specified by *subcol*. This feature may be exploited to maintain subqueues within the main queue specified by *qname*. (See *Subqueues* on page 1.136).

subcol specifies the starting position of *substring*. If not coded, the default is 1.

Return codes

After execution of a QUEUE FIND statement, variable *&usrretc* is set to one of the following return codes:

0 element successfully found.
4 queue not found or no element found which matches the specified search criteria.
8 invalid position specified, for instance you have asked for element 9, and there are only 6 elements on the queue.
12 broken queue (This should never occur. Report any occurrence to your local support office).
16 syntax error.

Examples

The following examples show various QUEUE FIND statements and indicate the setting of the result variables with reference to the following sample table. (see *Result variables* on page 1.133). There are four pages of examples so the sample table is repeated on the top of each page. Each ▽ symbol represents a single space.

```
Element1==>    USER123▽▽▽LOCAL001▽▽OPER
Element2==>    USER1▽▽▽▽▽REMOT999▽▽USER
Element3==>    USER675▽▽▽LOCAL567▽▽TSG
Element4==>    USERTECH▽▽REMOT001▽▽TSG
Element5==>    USER444▽▽▽LOCAL987▽▽USER
```

```
QUE FIND sample first
```

retrieves the first element from the queue. This now becomes the current position, and variables are set as follows:

&usrretc = 0
&usrqstr = USER123▽▽▽LOCAL001▽▽OPER
&usrqpos = 1
&usrqgpo = 0

```
QUE FIND sample last
```

retrieves the last element from the queue. This now becomes the current position, and variables are set as follows:

&usrretc = 0
&usrqstr = USER444▽▽▽LOCAL987▽▽USER
&usrqpos = 5
&usrqgpo = 0

```
QUE FIND sample curr
```

retrieves the current element from the queue. This returns the same element as in the second example, since the pointer in the queue has not been moved. Variables are set as follows:

&usrretc = 0
&usrqstr = USER444▽▽▽LOCAL987▽▽USER
&usrqpos = 5
&usrqgpo = 0

```
Element1==>    USER123▽▽▽LOCAL001▽▽OPER
Element2==>    USER1▽▽▽▽▽REMOT999▽▽USER
Element3==>    USER675▽▽▽LOCAL567▽▽TSG
Element4==>    USERTECH▽▽REMOT001▽▽TSG
Element5==>    USER444▽▽▽LOCAL987▽▽USER
```

```
QUE FIND sample prev
```

retrieves the element immediately before the current element in the queue. The element retrieved becomes the current position, and variables are set as follows:

&usrretc = 0
&usrqstr = USERTECH▽▽REMOT001▽▽TSG
&usrqpos = 4
&usrqgpo = 0

```
QUE FIND sample next
```

retrieves the element immediately after the current element in the queue, and variables are set as follows:

&usrretc = 0
&usrqstr = USER444▽▽▽LOCAL987▽▽USER
&usrqpos = 5
&usrqgpo = 0

The length used when comparing the string with each element is the length of the element. If the string length and element length are different, the element will not be selected to retrieve the element containing USER675:

```
QUE FIND sample USER675
```

where

&usrretc = 4
&usrqstr not altered
&usrqpos not altered
&usrqgpo not altered

```
Element1==>    USER123▽▽▽LOCAL001▽▽OPER
Element2==>    USER1▽▽▽▽▽REMOT999▽▽USER
Element3==>    USER675▽▽▽LOCAL567▽▽TSG
Element4==>    USERTECH▽▽REMOT001▽▽TSG
Element5==>    USER444▽▽▽LOCAL987▽▽USER
```

```
QUE FIND sample first 'USER675▽' 1
```

finds the entry for a specific user, and sets variables as follows:

&usrretc = 0
&usrqstr = USER675▽▽▽LOCAL567▽▽TSG
&usrqpos = 3
&usrqgpo = 1

The first element has been found with the substring USER675 starting in position 1. It is the third element in the queue and the first occurrence to have USER675 in position 1, hence the setting of *&usrqgpo*.

```
QUE FIND sample first 'user675▽' 1
```

The element is not found in this example because the queue search is case sensitive. The following variable settings therefore apply.

&usrretc = 4
&usrqstr = not altered
&usrqpos = not altered
&usrqgpo = not altered

```
QUE FIND sample last TSG 21
```

&usrretc = 0
&usrqstr = USERTECH▽▽REMOT001▽▽TSG
&usrqpos = 4
&usrqgpo = 2

The last element has been retrieved with the substring TSG starting in position 21. It is the fourth element in the queue and the second element to have the substring TSG starting in position 21 (the second element in the subqueue of TSG users).

```
Element1==>    USER123▽▽▽LOCAL001▽▽OPER
Element2==>    USER1▽▽▽▽▽REMOT999▽▽USER
Element3==>    USER675▽▽▽LOCAL567▽▽TSG
Element4==>    USERTECH▽▽REMOT001▽▽TSG
Element5==>    USER444▽▽▽LOCAL987▽▽USER
```

```
QUE FIND sample 2
```

retrieves the second element from the queue, and sets variables as follows:

&usrretc = 0
&usrqstr = USER1▽▽▽▽▽REMOT999▽▽USER
&usrqpos = 2
&usrqgpo = 0

```
Set &num = 9
QUE FIND sample &num
```

&usrretc = 8
&usrqstr not altered
&usrqpos not altered
&usrqgpo not altered

Element 9 was requested; there are only 5 elements on the queue.

```
Set &num = 2
Set &substr = LOCAL
Set &subcol = 11
QUE FIND sample &num &substr &subcol
```

&usrretc = 0
&usrqstr = USER675▽▽▽LOCAL567▽▽TSG
&usrqpos = 3
&usrqgpo = 2

The 2nd occurrence of an element has been retrieved which has the substring LOCAL starting in column 11 (the 2nd element in the subqueue of local terminals).

```
Element1==>    USER123▽▽▽LOCAL001▽▽OPER
Element2==>    USER1▽▽▽▽▽REMOT999▽▽USER
Element3==>    USER675▽▽▽LOCAL567▽▽TSG
Element4==>    USERTECH▽▽REMOT001▽▽TSG
Element5==>    USER444▽▽▽LOCAL987▽▽USER
```

```
QUE FIND sample 'USER444▽▽▽LOCAL987▽▽USER'
```

&usrretc = 0
&usrqstr = USER444▽▽▽LOCAL987▽▽USER
&usrqpos = 5
&usrqgpo = 0

The element containing the specified character string has been retrieved. Since the character string contains spaces, it must be enclosed in quotes. Note that *&usrqgpo* has not been set and contrast this with the next example.

```
QUE FIND sample 'USER444▽▽▽LOCAL987▽▽USER' USER 21
```

&usrretc = 0
&usrqstr = USER444▽▽▽LOCAL987▽▽USER
&usrqpos = 5
&usrqgpo = 2

The same element has been retrieved as in the 'QUEUE FIND sample 2' example. However, because we specified a substring value, *&usrqgpo* has been set to 2 indicating that USER444 is the second element to contain USER in position 21 (the second element in the USER group subqueue).

Notes

If the optional QUEUE-MASK startup parameter was coded (see *QUEUE-MASK=x* on page 3.11), the character specified can be used within *string* or *substring* to denote character positions to be ignored when matching queue elements are being searched for. For example, assuming QUEUE-MASK=+ was specified,

QUEUE FIND LOCALQ1 'USER+++++++TSG'

will locate the first element in LOCALQ1 which is 14 characters long, and has USER in positions 1 to 4 and TSG in positions 12 to 14. Other character positions, designated by + characters, are not part of the search criteria.

Result variables

After a successful QUEUE FIND operation, indicated by a return code of zero in *&usrretc*, the following additional variables will be set:

&usrqstr the contents of the element located by the specified search criteria. If the string operand was specified, this variable will contain whatever was specified as *string*.

&usrqpos the position in the queue of the located element. If the *pos* operand was specified, this variable will contain whatever was specified as *pos*. If a QUEUE FIND via an index was performed before this FIND, then this variable will be set to zero for CURR, PREV or NEXT, until a FIRST, LAST, *n* or *string* operand is used.

&usrqgpo the position of the located element within the subqueue of elements which contain the characters defined in *substring* starting in position *subcol*, if the *substring* operand was specified. If *substring* was not specified, *&usrqgpo* will be set to zero. If the CURR, PREV or NEXT operands are specified this variable will be set to zero.

Execution of a QUEUE FIND statement does not affect the contents of the queue which is being searched. If the FIND is successful, a copy of the element is made available in variable *&usrqstr*. If you want to remove an element after retrieval by a QUEUE FIND statement, the QUEUE DELETE statement should be used. QUEUE FIND will set *&usrqpos* to point to the element. This value may be coded on the QUEUE DELETE request to remove the selected element. When using this technique to locate and remove an element from a system queue, code the QUEUE DELETE immediately after the QUEUE FIND. This will ensure that the queue is not updated by another user between the FIND and DELETE requests, possibly changing the position of the element to be deleted.

Format of QUEUE FIND via an index

QUEUE FIND *index-q-name* *element* *key* [*key-pos*]

where

index-q-name is the name of the index on which the find will be done.

element identifies the base queue element to be returned on the basis of its position in the queue or its contents, using one of the following:

- FIRST returns the first or only base queue element which has the key specified (duplicate keys are permitted).
- LAST returns the last or only base queue element which has the key specified.
- *n* returns the base queue element which has the key specified and is in the position specified by *n* amongst elements with the same key.
- CURR returns the element at the current position within the index. (See *Effects on current position when indexes are being used* on page 1.136)
- PREV returns that element, before the current position in the index, which has the same key as the current position.
- NEXT returns that element, first after the current position in the index, which has the same key as the current position.
- *string* returns the first element whose contents match *string* and whose key matches the key specified; *&usrqgpo* will be set to indicate the position of the element amongst those whose key matches that specified.

key is mandatory for FIRST, LAST, *n*, and *string*, but not for CURR, NEXT and PREV. It specifies the characters to be matched against the key field stipulated in QUEUE BLDINDEX. If the length of the QUEUE FIND is longer than specified on the BLDINDEX, return code 16 is returned; if shorter, it is padded out to the correct length with blanks. Mask characters cannot be used in the key.

key-pos specifies the start column of the key field within the base queue data; because the key start position was specified on the BLDINDEX it is not necessary to code it here, but it can be an aid to documentation; if you decide to code it, please note that it must match the key start position specified in the BLDINDEX, else return code 16 is issued.

Result variables after QUEUE FIND via an index queue

&usrqstr	is the content of the element from the base queue, located by the search criteria specified
&usrqpos	will be set to zero - the queue position in the index has no meaning
&usrqgpo	indicates the position of the located element among elements with a replicated key field. If the CURR, NEXT, PREV operands are specified, this variable will be set to zero.

Enclosing quotation marks

When coding a *string* or *substring* operand as a literal containing blanks (X'40'), the literal should be enclosed in quotation marks. For example:

```
QUE FIND sample 'USER444 LOCAL987 USER'
```

is equivalent to

```
Set &var = 'USER444 LOCAL987 USER'
QUE FIND sample &var
```

Enclosing brackets

Enclosing brackets must be used if the string to be searched for consists of any of the other operands which may be specified in place of *string* (FIRST, LAST, CURR, PREV, NEXT or a numeric value). For example:

```
QUE FIND sample first
```

will retrieve the first entry from the queue, and:

```
QUE FIND sample (first)
```

will search for an element containing the characters first.

Subqueues

The *substring* and *subcol* parameters in FIND statements can be used to partition a queue into subqueues based on the *substring* specified. The sample queue used in the examples on the previous pages illustrates this form of organization. The queue contains a list of userids, terminal names and a user group identifier for each user. The main queue can be keyed by userid on a first in first out basis by adding new elements to the bottom (using QUEUE ADD LAST) and retrieving entries from the top (using QUEUE FIND FIRST). By accessing the queue with QUEUE FIND statements which specify a substring starting in column 21 (the user group id), we effectively have a number of subqueues, one for each user group. For example, to retrieve the first entry from the subqueue of the TSG user group, we could use:

```
QUE FIND sample first TSG 21
```

Example of using subqueues

To illustrate this technique, imagine our sample queue is a list of users waiting to connect to an inactive application. The main list is in 'first in first out' order. Suppose also that we wish to always give preference to users in the TSG group. When a 'slot' becomes available, we use a QUEUE FIND statement to get the first TSG user and connect him to the application. If there are no TSG users on the queue (QUEUE FIND returns *&usrretc* = 4), we issue another QUEUE FIND statement to get another user from the top of the queue:

```
QUE FIND sample first TSG 21

If &usrretc eq 4
   QUE FIND sample first

Set &luname = &usrqstr(11:18)
Trefresh &luname connpanl
```

Effects on current position when indexes are being used

The current position within an index is maintained separately from the current position within the base queue, but they are closely related in that any updates made to the base queue will cause an automatic update of any index(es) and this will affect the current positions in both the base queue and any index(es). For example if a new element is ADDed to the base queue, this will cause a new element to be added to the index(es) and the current position will become that new element in both the base queue and the index(es).

QUEUE FIND is different in that a QUEUE FIND performed on the base queue will cause the current position within the base queue to be modified, but will have no effect on any index current positions. However, a QUEUE FIND on an index will cause both the current positions within the index and the base queue to be updated; this allows further processing to be performed on the base queue on the basis of the position established by locating an element through an index.

Notes

If the specified *qname* begins with sys or an asterisk (*), the queue is a system queue, and one queue of the specified name exists in the NCI/XF address space for access by any terminal.

If the specified *qname* begins with characters other than sys or an asterisk (*) the queue is a local, or private, queue; a unique version of the queue called *qname* will be maintained for each terminal referencing the specified *qname*.

Deleting elements

Elements can be deleted from a queue only by position. If you want to delete an element on the basis of the data contained in the element (for example delete the entry for user NCIUSER1), you should first find the position of the required element using the QUEUE FIND statement. This will return the position of the element in *&usrqpos* which can then be used as part of a QUEUE DELETE request.

When using this technique to delete an element from a system queue, the DELETE should immediately follow the FIND request in the same panel section. This will ensure that the queue is not modified by another terminal user, possibly altering the position of the element to be deleted.

QUEue GFIND

The QUEUE GFIND statement searches the named queue and retrieves the element meeting the specified search criteria. The retrieved element can include generic characters. An element can be retrieved on the basis of its position in the queue or a substring of its contents. QUEUE GFIND can be used after QUEUE BLDINDEX to search the index built by that command; it will perform in the same way as QUEUE FIND.

QUEUE GFIND is most useful when substring is specified with FIRST, LAST, *n*, PREV or NEXT.

Format

QUEUE GFIND *qname element* [*substring* [*subcol*]]

where

qname is the name of the queue, or a variable containing the name.

element identifies the element to be returned on the basis of its position in the queue using one of the following:

FIRST If *substring* is not specified, the first (top) element on the queue will be returned. If *substring* is specified, the first element containing the specified *substring* starting in column *subcol* will be returned.

LAST If *substring* is not specified, the last (bottom) element on the queue will be returned. If *substring* is specified, the last element containing the specified *substring* starting in column *subcol* will be returned.

n If *substring* is not specified, the element in the specified position (*n*) in the queue will be returned. If *substring* is specified, the element returned will be the *n*th occurrence of an element containing the specified *substring* starting in the specified *subcol.* For instance, if *n* is 8, the eighth occurrence of an element containing the specified *substring* will be returned.

CURR will return the element at the current position. The current position is the position of the element last accessed.

PREV If *substring* is not specified, the element in the queue immediately before the current element will be returned. If *substring* is specified the first element containing the specified *substring* starting in the specified *subcol* before the current position in the queue will be returned.

NEXT If *substring* is not specified the element in the queue immediately after the current position will be returned. If *substring* is specified the first element containing the specified *substring* starting in the specified *subcol* after the current position in the queue will be returned.

string is a literal or variable to be matched against the elements in the queue.

If *substring* is not specified, the element returned will be the first element whose contents match *string*.

If *substring* is specified, matching proceeds as described above as if *substring* had not been specified, except that variable *&usrqgpo* will be set to indicate the position of the element within the subqueue defined by *substring*. (See *Subqueues* on page 1.136).

substring specifies that the search is to only match against those elements which contain the specified *substring* starting in the character position specified by *subcol*.

The element can contain mask characters as defined by the QUEUE-MASK startup option (if coded, otherwise the default is +) and/or a generic asterisk (*) character. A mask character gives a match in that position in the queue element data. An asterisk (*) gives a match in that and subsequent positions in the queue element data.

If *substring* is longer than an element which contains an asterisk (*) but the characters preceding the asterisk (if any) match the same positional characters in *substring*, then a match will occur. The length of substring is limited to 255 characters.

The normal mask function is not available for QUEUE GFIND - ie mask characters in the substring parameter will be considered as actual characters to match.

subcol specifies the starting position of *substring*. If not coded, the default is 1.

Return codes

After execution of a QUEUE GFIND statement, variable *&usrretc* is set to one of the following return codes:

0 element successfully found.
4 queue not found or no element found which matches the specified search criteria.
8 invalid position specified, for instance you have asked for element 9, and there are only 6 elements on the queue.
12 broken queue (This should never occur. Report any occurrence to your local support office).
16 syntax error.

Example

Consider the following queue, where the QUEUE-MASK character is +.

Element 1 - ABC++
Element 2 - ABCD+
Element 3 - DEF*
Element 4 - A+MEF
Element 5 - ABCDEF
Element 6 - F++D
Element 7 - +B*
Element 8 - A*
Element 9 - *

Then if the following QUEUE GFIND statements are issued in succession:

```
QUEUE GFIND sample FIRST ABCD
```

returns element 1 to variable *&usrqstr*.

```
QUEUE GFIND sample NEXT ABCDE
```

returns element 2 to variable *&usrqstr*.

```
QUEUE GFIND sample NEXT MM 2
```

returns element 4 to variable *&usrqstr*.

```
QUEUE GFIND sample NEXT ABCD
```

returns element 5 to variable *&usrqstr*.

```
QUEUE GFIND sample NEXT ABCD
```

returns element 7 to variable *&usrqstr*.

Example queue, as shown on page 1.140:

Element 1 - ABC++
Element 2 - ABCD+
Element 3 - DEF*
Element 4 - A+MEF
Element 5 - ABCDEF
Element 6 - F++D
Element 7 - +B*
Element 8 - A*
Element 9 - *

```
QUEUE GFIND sample NEXT ABCD
```

returns element 8 to variable *&usrqstr*.

```
QUEUE GFIND sample NEXT ABCD
```

returns element 9 to variable *&usrqstr*.

```
QUEUE GFIND sample PREV FFF
```

returns element 6 to variable *&usrqstr*.

```
QUEUE GFIND sample 3 ABC
```

returns element 5 to variable *&usrqstr*.

```
QUEUE GFIND sample LAST ABCD
```

returns element 9 to variable *&usrqstr*.

Notes

After a successful QUEUE GFIND operation, indicated by a return code of zero in *&usrretc*, the following additional variables will be set:

&usrqstr the contents of the element located by the specified search criteria. If the string operand was specified, this variable will contain whatever was specified as *string*.

&usrqpos the position in the queue of the located element. If the *n* operand was specified, this variable will contain whatever was specified as *n*. If a QUEUE GFIND via an index was performed before this GFIND, then this variable will be set to zero for CURR, PREV or NEXT, until a FIRST, LAST, *n* or *string* operand is used.

&usrqgpo the position of the located element within the subqueue of elements which contain the characters defined in *substring* starting in position *subcol*, if the *substring* operand was specified. If *substring* was not specified, *&usrqgpo* will be set to zero. If the CURR, PREV or NEXT operands are specified this variable will be set to zero.

Execution of a QUEUE GFIND statement does not affect the contents of the queue which is being searched. If the QUEUE GFIND is successful, a copy of the element is made available in variable *&usrqstr*. If you want to remove an element after retrieval by a QUEUE GFIND statement, the QUEUE DELETE statement should be used. QUEUE GFIND will set *&usrqpos* to point to the element. This value may be coded on the QUEUE DELETE request to remove the selected element. When using this technique to locate and remove an element from a system queue, code the QUEUE DELETE immediately after the QUEUE GFIND. This will ensure that the queue is not updated by another user between the QUEUE GFIND and QUEUE DELETE requests, possibly changing the position of the element to be deleted.

QUEue GPURGE

The QUEUE GPURGE statement deletes all entries from a set of system or user queues via a *generic-queue-name* mask. In addition, this statement frees all storage associated with the specified queues and removes all references to them from the NCI/XF environment. There are three types of queue, all of which may be purged using this statement:

- system queues beginning SYS
- system queues beginning with an asterisk (*)
- user queues which can begin with any character other than SYS or asterisk (*).

Format

QUEUE GPURGE *generic-queue-name*

where

generic-queue-name is the name, or a variable containing the name, of the system or user queue. Two special characters, the asterisk (*) and the plus sign (+) can be used when defining the *generic-queue-name*. Refer to the notes and examples on the following pages for rules relating to the use of these special characters.

Return codes

After execution of a QUEUE GPURGE statement, variable *&usrretc* is set to one of the following return codes:

0 at least one queue was purged successfully.
4 no matching queues were found.
12 a broken queue was found. No further attempts at matching were attempted, but prior matches will have been purged. (This should not occur. Report any occurrence to your local support office).
16 syntax error.

Notes

Two special characters, the asterisk (*) and the plus sign (+) can be used when defining the *generic-queue-name*. If they are used as the first character of the *generic-queue-name* they have the following meaning:

*	indicates a system queue starting with an asterisk (*), unless the *generic-queue-name* consists of a single asterisk (*), in which case it purges all user queues
+	indicates a user queue

In any other column, these two special characters are recognized as mask characters.

Mask Character	Meaning
*	During a search of queues to be purged only those characters to the left of the asterisk (*) are used as part of the criteria. Those to the right are ignored.
+	Stipulates that this character may be matched by any non-blank character in the queue name. There is no limit to the number of plus signs (+) that may be used in this way.

As plus (+) and asterisk (*) symbols are acceptable characters in queue names, these cannot be used in generic purges (the generic characters are dominant).

Example

```
QUE GPURGE +*
```

will clear all user queues.

```
QUE GPURGE +YS*
```

will clear all user queues with the characters YS in the second and third position.

```
QUE GPURGE **
```

will clear all system queues starting with an asterisk (*).
Note: This should be used with extreme caution.

```
QUE GPURGE *X*
```

will clear all '*' type system queues with the character X in the second position.

```
QUE GPURGE S*
```

will clear user queues starting with S. This does not apply to SYS queues.

```
QUE GPURGE SY*
```

will clear user queues starting with SY. This does not apply to SYS queues.

```
QUE GPURGE +Y*
```

will clear user queues with a second character of Y. This does not apply to SYS queues.

```
QUE GPURGE +
```

will clear all user queues that contain one character only. This does not apply to SYS queues.

```
QUE GPURGE SYS*
```

will clear all system queues starting SYS, but not those starting * or user queues.
Note: This should be used with extreme caution.

```
QUE GPURGE SYSX
```

will clear the SYS system queue named SYSX. The result is the same as issuing QUEUE PURGE SYSX.

```
QUE GPURGE A+B+C
```

will clear all user queues whose names are five characters in length which have A as the first character, B as the third character and C as the fifth character.

```
QUE GPURGE SYS++
```

will clear all SYS system queues whose names are five characters long.

```
QUE GPURGE *
```

will clear all user queues, not * system queues. This is an exception to the general rule that an asterisk (*) in the first position indicates an * system queue.

```
QUE GPURGE ABC
```

will clear the user queue named ABC. This is equivalent to issuing the QUEUE PURGE ABC command.

QUEue PURGE

The QUEUE PURGE statement deletes all entries from a queue, frees all storage associated with the queue and removes all reference to the queue from the NCI/XF environment.

Format

QUEUE PURGE *qname*

where

qname is the name of the queue or index, or a variable containing the name.

Return codes

After execution of a QUEUE PURGE statement, variable *&usrretc* is set to one of the following return codes:

0 queue successfully purged.
4 queue not found.
12 broken queue (This should not occur. Report any occurrence to your local support office).
16 syntax error.

Example

```
QUE PURGE *usertab
QUE PURGE localtab
```

Notes

If the specified *qname* begins with sys or an asterisk (*), the queue is a system queue. Purging a system queue makes the queue unavailable to any terminal.

If the specified *qname* begins with characters other than sys or an asterisk (*) the queue is a local, or private, queue. Purging a local queue will only purge the version of the queue belonging to the terminal issuing the PURGE request. Other local queues of the same name, owned by other terminals, will be unaffected.

QUEue REPLACE

The QUEUE REPLACE statement allows data referenced by any queue element to be replaced, thus avoiding the double reference of QUEUE DELETE followed by QUEUE ADD.

Format

QUEUE REPLACE *qname element data-string*

where

qname is the name of the queue, or a variable containing the name.

element identifies the element to be replaced on the basis of its position in the queue or its contents, using one of the following:

- FIRST indicates that the first (top) element in the queue is to be replaced.
- LAST indicates that the last (bottom) element in the queue is to be replaced.
- *n* is the position in the queue of the element to be replaced. This may be an explicit numeric value, or a variable containing a numeric value.
- CURR indicates that the element at the current position is to be replaced. The current position is the position of the element last accessed.
- PREV indicates that the element immediately before the current position is to be replaced.
- NEXT indicates that the element immediately after the current position is to be replaced.

data-string is the data which will replace the current queue element. It may be specified as a literal or a variable containing the required data.

Return codes

After execution of a QUEUE REPLACE statement, variable *&usrretc* is set to one of the following return codes:

0	element successfully replaced in the queue.
4	queue not found or element not found.
8	invalid position specified (For instance you have tried to delete element 8 and the queue only contains 6).
12	broken queue (This should never occur. Report any occurrence to your local support office).
16	syntax error.
20	QUEUE REPLACE can not be performed on an index.

Example

```
QUE REPLACE sample first &data
```

The first (top) element in queue sample is replaced with the contents of the variable *&data*.

QUEUE SORT

The QUEUE SORT statement allows NCI based queues to be sorted.

Format

QUEUE SORT *qname startcol forcols* [A/D]

where:

qname	is the name of the queue to sort.
startcol	is the position within each queue element to start sorting.
forcols	is the number of bytes to sort in the range 1 - 255.
A/D	is whether to sort in Ascending or Descending order. The default is Ascending.

Examples

```
QUEUE SORT localq 2001 250 A
```

The queue with name localq is sorted starting at byte position 2001 and ending at byte position 2250.

```
QUEUE SORT myqueue 204 89
```

The queue with name myqueue is sorted starting at byte position 204 and ending at byte position 292.

```
QUEUE SORT sysqueue 987 255 D
```

The queue with name sysqueue is sorted in descending order starting at byte position 987 and ending at byte position 732.

Notes

If a queue element is shorter than startcol, the element will not be sorted and will maintain its relative position within the queue.

If there is a request to sort past the end of a queue element, the element is sorted as if it was padded with nulls.

The variables *&usrqpos*, *&usrqstr* and *&usrqgpo* remain unchanged after using QUEUE SORT.

RESET

The RESET statement is used to delete a variable or variables. All storage occupied by a variable is freed. After a RESET, a variable is considered to have a value equal to *&sysnull*.

If RESET is specified with no operands, all user-defined non-system variables for the terminal will be reset.

Format

RESET [*&var* .....]

where

&var ..... are the variable names to reset. Up to 16 variables can be reset with one statement.

Related statement

GRESET

Examples

```
RESET &userid &passwd &group &usrmenu
```

deletes the named variables.

```
RESET
```

deletes all private variables owned by the terminal executing the statement.

RESETAttr

The RESETATTR statement is used to release storage associated with ATTR statements processed in a)PANEL section. This is particularly valuable in reducing the storage overhead of panel attributes that may not be required after a CONNECT statement.

Format

RESETATTR

Example

```
)panel
ATTR @ high protected
ATTR + normal protected
ATTR ! normal unprotected cursor
ATTR % nod protect numeric
)body
.
.
.
.
.
)proc
.
.
.
.
.
RESETATTR
CONNECT
```

RESETExec

The RESETEXEC statement is used to remove the return path from the calling EXEC such that if an EXIT statement is encountered, RESTART processing is carried out instead.

EXIT statement processing within EXECS called from the EXEC containing RESETEXEC remains unaffected. Language statements such as RETURN, RETURN CONT and PANEL are also unaffected.

Restrictions

If RESETEXEC is issued within the PROC, UPDATE or INIT SECTIONS it has no effect.

Format

RESETEXEC

Example

```
)PROC
EXEC A
LOG 'return from exec chain'

)ROUTINE A
EXEC B
EXIT

)ROUTINE B
EXEC C

)ROUTINE C

RESETEXEC
CONNECT TSO
EXIT
```

If EXEC C did not contain the RESETEXEC statement, control would eventually return to the LOG statement in the)PROC section when the EXIT statement was issued. Introducing the RESETEXEC statement causes the RESTART panel to be driven instead.

Note

RESETEXEC can be used to reduce virtual storage overheads in a large EXEC chain where a long running statement is executed (such as CONNECT) and a PANEL statement is likely to be executed afterwards.

All storage for the chain of EXECs will be released while the user is in TSO.

RESTART

The RESTART statement is used to return a terminal to its highest level panel which is the primary panel for the terminal as defined in its TDT entry.

Restrictions

RESTART is not allowed in:

startup-panel

initial-exec

open-exec

update-exec

close-exec

final-exec

MODIFY

Format

RESTART

Related statements

PANEL

RETURN

Example

```
)proc
If &usraid eq pak1 then
    RESTART
```

will return the terminal to the primary panel as defined in its TDT entry if the user presses the <PA1> key.

Notes

Variables relating to terminals are not reset as a result of the RESTART statement. It is recommended that RESET or GRESET be considered following RESTART. (Refer also to the LOGOFF statement)

If the TDT is dynamically updated while NCI/XF is active (when using the DEFINE statement) and the terminal's initial panel is changed, a RESTART will display the newly updated initial panel.

RETURN

The RETURN statement is used to return to the previous panel. Control is transferred to the)INIT section of the previous panel.

Restrictions

RETURN is not allowed in:

startup-panel

initial-exec

open-exec

update-exec

close-exec

final-exec

MODIFY

Format

RETURN [CONT]

where

CONT returns control to the NCI/XF statement which follows a previous PANEL *name* CONT statement (see *Notes* below).

Related statements

PANEL

RESTART

Example

```
)proc
If &usraid eq pak1 then
    restart
If &usraid eq pf03 then
    RETURN
```

will return the terminal to the previous panel if the user presses <F3>.

Notes

For storage and performance reasons NCI/XF retains a reference to one previous panel. If two RETURN statements are issued without an intervening PANEL statement, the effect is the same as RESTART.

For further information See *PANel* on page 1.95

REVerse

The REVERSE statement allows the order in which characters appear within a variable to be reversed.

Format

REVERSE *&var1* = *&var2*

where

&var1 is the variable into which the reversed string will be placed.

&var2 is the literal or variable containing the string to be reversed.

Examples

```
REV &foobar = 'abcde'
```

sets variable *&foobar* to edcba.

```
REV &foobar = &sysopid
```

sets *&foobar* to svm, when *&sysopid* equals mvs.

ROWs

The ROWS statement is used to establish the height of the PANEL to be displayed.

Restrictions

ROWS is valid in the)PANEL section only.

Format

ROWS [*nnn*]

where

nnn is a numeric value specifying the PANEL size to be displayed. This should normally be set to the screen size of the terminal displaying this panel. Panels smaller than the real screen size will be displayed in the uppermost portion of the screen.

Possible values are 12, 24, 27, 32 or 43.

If not specified, the default is 24.

Related statement

COLUMNS

Example

```
)panel
 Interval 1
 ROW      12
 Columns  80
 Attr     @ high protected
```

sets the height of the panel to be displayed to 12 rows.

SCAN

The SCAN statement enables a line of text to be searched for a character string as specified by the user.

Format

SCAN *&text* FOR *&string* [BACKWARD]

where

&text	is the text to be scanned.
&string	is the character (or characters) for which the text is to be scanned.
BACKWARD	specifies that the text is to be scanned in reverse (from the last text character to the first).

Return codes

After execution of a SCAN statement, variable *&usrretc* will contain the start position of the first occurrence of the string within the data (or the last occurrence if backward was specified). *&usrretc* will contain 0 if the string is not found.

Examples

```
Set &¢data = cdeabab
Set &¢str = ab
SCAN &¢data for &¢str
```

sets *&usrretc* to 4, indicating that the first occurrence of string ab starts in position 4.

```
Set &¢data = cdeabab
Set &¢str = ab
SCAN &¢data for &¢str backward
```

sets *&usrretc* to 6 indicating that the last occurrence of string ab starts in position 6.

SET

The SET statement allows the setting of variables within a panel.

Format

SET *&var1* = *&var2* [PAD*d* *n*]

where:

&var1 is the variable to be set (the target).

&var2 is the value or variable to be resolved before performing the SET statement.

PAD*d* specifies that padding is required in the set variable; this can be either PADR(ight), PADL(eft) or PADC(enter).

n is the total length to which *&var1* will be padded.

If *n* is less than the length of *&var2*, *&var1* will be truncated to the specified length. Truncation will occur from the right if PADR is specified or from the left if PADL is specified. If PADC is specified, the variable will be truncated from the right and left leaving only the central bytes.

Examples

```
SET &¢msg = 'Invalid function specified'
```

The literal is moved into variable *&¢msg*.

```
SET &¢msg = 'Connect to &appl failed'
```

Variables within *operand2* are resolved before the SET.

```
SET &varc = &vara&varb
```

The contents of *&vara* and *&varb* are concatenated and the result placed in *&varc*.

```
SET &var1 = &sysdate(4:5)&sysdate(1:2)
```

Substrings of variables can be extracted and SET into another variable. In this example, bytes 4 to 5 of *&sysdate* and bytes 1 to 2 of *&sysdate* are concatenated and SET into *&var1*; so the value 3112 would be stored in *&var4* on 31st December when the DATE=MM/DD/YY option was used.

```
SET &count = 2
SET &index = 3
SET &&line&count = &&table&index
```

Variable variables are supported on both sides of a SET operation.

```
SET &n1 = 2
SET &n2 = 3
SET &count = 4
SET &&var&count = 'Answer is &&var1&var4.(&n1.:&n2.)'
```

Variable variables, concatenations, substrings and literals can be combined in a single SET statement. Periods must be inserted after each variable name within the substring notation, since a colon does not delimit a variable name.

Examples using the PAD operand

The following examples assume that *&var2* contains ABC123.

```
SET &var1 = &var2 PADR 16
```

sets *&var1* to 'ABC123 ' (ABC123 followed by 10 blanks).

```
SET &var1 = &var2 PADL 12
```

sets *&var1* to ' ABC123' (ABC123 preceded by 6 blanks).

```
SET &var1 = &var2 PADC 21
```

sets *&var1* to ' ABC123 ' (ABC123 preceded by 7 blanks and followed by 8 blanks).

SHUTDown

The SHUTDOWN statement is used to terminate NCI/XF.

Format

SHUTDOWN [*code*]

where

code is a specified condition code. The job will end with this condition code which may then be examined by the environment used to start NCI/XF.

Example

```
)proc
If &usraid eq pak1 then
   restart
If &1 = &sysnull then
   panel
If &1 = 1 then
   logoff
If &1 = 2 then
   SHUTD 999
```

will terminate NCI/XF with a condition code of 999 if variable *&1* contains 2.

STATus

The STATUS statement is used to establish the availability of on-line applications.

Format

STATUS *nodename* [*result-var*]

where

nodename is the VTAM NODENAME of an application or a variable set to the required value.

result-var is the name of a system variable where the result of the STATUS request will be stored. Specify the actual variable name without a leading &, or the name of a variable (with a leading &) containing the required variable name.

If this parameter is not specified, the result will be stored in *&usrretc*. See also the descriptions of the asynchronous operation of STATUS later in this section. The result variable must be a system variable (name starting with sys or asterisk (*)).

Return codes

After execution of a STATUS statement, variable *&usrretc* (or the variable named in *result-var*) is set to one of the following return codes:

0 application active.
4 application inactive.
8 application never accepts logons.
12 application has temporarily disabled logons.
16 application is shutting down.
20 application not known to VTAM.
24 application status unknown (The specified resource is known to VTAM but is either not an application, or NCI/XF could not obtain the status of the application, for instance a cross-domain node is not accepting status requests).
28 unknown error.

Examples

The status of the application is verified before attempting to connect to it.

```
)proc
STAT IMS
If &usrretc eq 0
   connect IMS
   panel
Set &errmsg = 'IMS not active, try later'
panel
```

will check the status of IMS and depending on the result, the terminal will either be connected to IMS or an error message displayed.

```
Set &appl = A01TSOA
STAT &appl *stso
```

The status of the application is tested and the resulting return code is placed in system variable *&*stso*.

```
Set &appl = A01CICS
Set &¢retvar = syscics
STAT &appl &¢retvar
```

The status of the application is tested and the resulting return code is put in system variable *&syscics*.

Notes

The most common use of STATUS is to display a meaningful status message for each application as part of an application selection menu.

Execution of a STATUS command will cause NCI/XF to issue an INQUIRE APPSTAT request to VTAM.

It is definitely *not* recommended that the)INIT section of an application selection menu issue STATUS requests for all applications appearing on the menu. This would flood VTAM with INQUIRE APPSTAT requests every time the menu panel is displayed or refreshed and cause extended response times for users who request the panel while the STATUS requests are processed (a major consideration with cross-domain applications).

The recommended method is to code STATUS requests in the UPDATE-EXEC (or the)UPDATE section of the NCI/XF startup panel). This way the STATUS requests will be issued regularly (say every 30 seconds or a minute) depending on the UPDATE-INTERVAL in use. The required status information having been obtained, the UPDATE-EXEC (or the)UPDATE section of the NCI/XF startup panel) should put an appropriate acronym in a set of system variables, one for each application. Including the system variable names within the)BODY of each application selection menu panel means that users who request the panel will automatically obtain the latest status.

If the UPDATE-EXEC finds that an application status has changed, the PREFRESH statement can be used to update all terminals which currently display a panel which shows the old status information. Because the status is in a system variable, the new value will be picked up as the relevant terminals are refreshed.

It is always good practice to issue a STATUS request immediately before a CONNECT request, to check that the status of the requested application has not changed within the last few seconds while the user was typing.

Sample panels are shown on the following pages.

Sample UPDATE-EXEC

```
)ROUTINE
*          Test status every update cycle
Set &saveit = &*stso
Set &*stso = 'INACTIVE'
STATUS TSO
If &usrretc eq 0
   set &*stso = 'ACTIVE'
If &saveit ne &*stso
   log 'Status of TSO changed from &saveit to &*stso'
   set &change = 'Y'
*
*... repeat for other applications ...
*
If &change = 'Y'
   reset &change
   prefresh menu*
```

Sample startup panel

```
)comment
Issue status every update interval
)init
*         Get initial status of TSO
Set &*stso = 'INACTIVE'
STATUS TSO
If &usrretc eq 0
   set &*stso = 'ACTIVE'
*
*... repeat code for other applications ...
*
)UPDATE
*         Test status every update cycle
Set &saveit = &*stso
Set &*stso = 'INACTIVE'
STATUS TSO
If &usrretc eq 0
   set &*stso = 'ACTIVE'
If &saveit ne &*stso
   log 'Status of TSO changed from &saveit to &*stso'
   set &change = 'Y'
*
*... repeat for other applications ...
*
If &change = 'Y'
   reset &change
   prefresh menu
)body
)proc
)end
```

Sample menu panel

```
)comment
This menu displays status set in the startup panel
)panel
)init
)body
     ... heading lines ...

     PFK1    TSO    &*stso    Program Development
Environment

     ... other applications ...
)proc
If &usraid eq PFK1
   if &*stso eq 'ACTIVE'
      connect TSO &usrlogm &cinit
      panel
   set &¢emsg = 'TSO is not currently available'
   panel
```

Asynchronous operation of STATUS

A STATUS statement specified without *result-var* is executed asynchronously from mainline code and synchronously from the issuing panel. Once the INQUIRE APPSTAT has been issued, execution of the panel containing the STATUS verb is suspended until VTAM completes the request. Meanwhile, NCI/XF is free to process requests on behalf of other users. When VTAM returns the requested application status information, processing of the panel continues with the statement after the STATUS verb, and would normally contain code to put a suitable acronym into a system variable, on the basis of the value returned in *&usrretc.*

The interval between NCI/XF issuing the INQUIRE APPSTAT and VTAM returning the application status is normally very short. However this may not always be the case, particularly when inquiring on cross-domain applications; an intermediate NCP may be in slowdown, or a remote system may be under stress.

When the *result-var* operand is specified, the STATUS request is executed asynchronously from the issuing panel. An INQUIRE APPSTAT is issued to VTAM and execution of the panel continues immediately. You may choose to issue other STATUS requests or perform any other processing before waiting for an interval within which you might reasonably expect the STATUS requests to finish (this may vary for different domains). If any request has not completed within that time, assume the application is not responding and set the status on the menus accordingly.

Note that the INQUIRE APPSTAT request will remain outstanding until VTAM eventually posts a result. If you issue STATUS requests in the UPDATE-EXEC, you can check during each update cycle to determine if any STATUS requests issued but not satisfied in the previous cycle have now completed.

A sample *startup panel* exploiting the asynchronous version of STATUS is shown on the next page.

Sample asynchronous startup panel

```
)comment
Sample startup panel using asynchronous STATUS
)init
*               Issue STATUS for all appls.
Set &*tcode = 99
Set &*ccode = 99
Set &*icode = 99
STATUS TSO  *tcode
STATUS CICS *ccode
STATUS IMS  *icode
*          Wait for INQUIRE APPSTATS completion
Call ncwait 2
*                 Check results have come back
Set &*tstat = 'Not responding'
If &*tcode ne 99
   set &*tstat eq 'INACTIVE'
   if &*tcode = 0
      set &*tstat = 'ACTIVE'
   reset &*tcode
*
*... repeat above code for other applications ...
*
)update
*       Check requests outstanding from last time
Set &savet = &*tstat
If &*tcode ne 99
   if &*tcode ne &sysnull
      set &*tstat = 'INACTIVE'
      if &*tcode = 0
         set &*tstat = 'ACTIVE'
   set &*tcode = 99
   STAT TSO *tcode
*
*... repeat above code for other applications ...
*
Call ncwait 2
*                   Check results have come back
Set &*tstat = 'Not responding'
If &*tcode ne 99
   set &*tstat eq 'INACTIVE'
   if &*tcode = 0
      set &*tstat = 'ACTIVE'
   reset &*tcode
If &savet ne &*tstat
   log 'Status of TSO changed from &savet to &*tstat'
   set &change = 'Y'
   reset &savet
*
*... repeat above code for other applications ...
*
If &change = 'Y'
   prefresh menu*
   reset &change
)body
)proc
)end
```

STRING

The STRING statement allows you to join up to 99 variables together, each one being prefixed by a 2 byte hex length indicator. The length indicator contains the length of the variable and excludes the length indicator itself. Each variable to be joined in this way must have a common stem, eg *&inp0*, *&inp1*, *&inp2* etc. The number of stem variables to be strung together is determined by the value in the zero suffixed stem variable eg *&inp0*; this cannot exceed 99.

Format

STRING *&var1* = *stem*

where

&var1	is the variable to contain the string of data created.
stem	is the variable stem.

Related statement

UNSTRING

Example

The following example assumes that:

&inp0 = 3

&inp1 = "12345"

&inp2 = "" (a null entry)

&inp3 = "ABC"

Therefore:

```
STRING &total = inp
```

results in *&total* being set to:

"*aa*12345*bbcc*ABC"

where

aa	is x'0005', the length of "12345" (the contents of *&inp1*)
bb	is x'0000', the length of the null field (*&inp2*)
cc	is x'0003', the length of "ABC" (the contents of *&inp3*)

If a variable in the series is omitted, it is treated as a null value.

Return codes

After execution of a STRING statement, variable *&usrretc* is set to one of the following return codes:

0	completed successfully.
4	parameter error, or the length of the data exceeds 32K, or the number of stems specified exceeds 99.

STRIPCent

The STRIPCENT statement allows characters to be stripped from both sides of a variable.

Format

STRIPCENT *&variable* [*&chars*]

where

&variable	is the name of variable to be stripped.
&chars	is a string of one or more characters to be stripped from the variable. If the string begins or ends with a space, the entry must be in single quotes. If this parameter is omitted, blanks are stripped by default.

Related statements

STRIPRIGHT

STRIPLEFT

STRIPEMBED

Examples

```
STRIPC &name ‘ ‘
```

will strip leading and trailing blanks from *&name*.

```
Set &user = ‘**C*K*S**’
STRIPC &user *
```

will result in *&user* containing C*K*S.

```
Set &user = ‘ABCDE’
STRIPC &user ‘a’
```

will leave *&user* unchanged (STRIPCENT is case sensitive).

STRIPEmbed

The STRIPEMBED statement allows all specified characters to be stripped from a variable.

Format

STRIPEMBED *&variable* [*&chars*]

where

&variable	is the name of variable to be stripped.
&chars	is a string of one or more characters to be stripped from the variable. If the string begins or ends with a space, the entry must be in single quotes. If this parameter is omitted, blanks are stripped by default.

Related statements

STRIPRIGHT

STRIPLEFT

STRIPCENT

Examples

```
STRIPE &name 'y '
```

will strip embedded blanks and y's from *&name.*

```
Set &user = '**C*K*S**'
STRIPE &user *
```

will result in *&user* containing CKS.

```
Set &user = 'ABCDE'
STRIPE &user 'a'
```

will leave *&user* unchanged (STRIPEMBED is case sensitive).

STRIPLeft

The STRIPLEFT statement allows characters to be stripped from the left of a variable.

Format

STRIPLEFT *&variable* [*&chars*]

where

&variable	is the name of the variable to be stripped.
&chars	is the string of one or more characters to be stripped from the variable. If the string begins or ends with a space, the entry must be in single quotes. If this parameter is omitted, blanks are stripped by default.

Related statements

STRIPRIGHT

STRIPCENT

STRIPEMBED

Examples

```
STRIPL &name '* '
```

will strip leading blanks and asterisks from the variable *&name*.

```
Set &user = '**C*K*S**'
STRIPL &user *
```

will result in *&user* containing C*K*S**.

```
Set &user = 'ABCDE'
STRIPL &user 'a'
```

will leave *&user* unchanged (STRIPLEFT is case sensitive).

STRIPRight

The STRIPRIGHT statement allows characters to be stripped from the right of a variable.

Format

STRIPRIGHT *&variable* [*&chars*]

where

&variable	is the name of the variable to be stripped
&chars	is a string of one or more characters to be stripped from the variable. If the string begins or ends with a space, the entry must be in single quotes. If this parameter is omitted, blanks are stripped by default.

Related statements

STRIPLEFT

STRIPCENT

STRIPEMBED

Examples

```
STRIPR &name ‘ ‘
```

will strip trailing blanks from the variable *&name*.

```
Set &user = ‘**C*K*S**’
STRIPR &user *
```

will result in *&user* containing **C*K*S.

```
Set &user = ‘ABCDE’
STRIPR &user ‘e’
```

will leave *&user* unchanged (STRIPLEFT is case sensitive).

SWITCHlog

The SWITCHLOG statement requests NCI/XF to switch from the current log to the alternative log.

Format

SWITCHLOG

Related statements

LOG

GETLOG

Example

```
If &systime GE 12:00:00 then SWITCH
```

Note

Refer to *LOG switching between NCILOG1 and NCILOG2* on page 4.3 of the NCI/XF Installation Manual.

TERMUPdate

The TERMUPDATE statement dynamically modifies the terminal update interval originally defined by the TERMINAL-UPDATE startup option. See *TERMINAL-UPDATE=nnnnnn* on page 3.13 of the NCI/XF Installation Manual.

Format

TERMUPDATE *num*

where

num defines the number of NCI/XF timer intervals, defined by the UPDATE-INTERVAL startup option or a subsequent UPDATE statement, which make up the terminal update interval.

Specify a numeric value in the range 0 to 999999, or a variable containing the required value.

If *num* is 0, terminal updating is quiesced.

Example

```
TERMUP 20
```

The time interval for terminal update processing is set to twenty times the NCI/XF basic timer interval.

Note

NCI/XF automatically maintains variable *&systupd* to indicate the current setting of the terminal update multiple.

When NCI/XF is started, *&systupd* is set to the value specified or defaulted by the TERMINAL-UPDATE startup option.

After execution of a TERMUPDATE statement, *&systupd* is updated to reflect the value specified.

TRACE

The TRACE statement is used to invoke the VTAM tracing routines within NCI/XF. The output produced is sent to the NCI/XF log.

Format

TRACE *options* .....

where

options .... are any of the following options, specified in the order required:

MODULE	traces NCI/XF module invocation
VTAM	traces all VTAM requests
RPL	traces VTAM requests plus RPLs
DATA	as for RPL, plus any associated data
INTERNAL	traces NCI/XF internal control blocks
PROG0	traces main logic flow within panels
PROG1	traces all executed statements
PROG2	traces all executed statements Variables within a traced statement with PROG2 will be logged in one of two ways; 1. where an NCI/XF statement changes the value of a variable regardless of its contents, as in the following three statements which overwrite *&var1*: SET &var1=&var2 TRANSLATE &var1=&var2 UPPERCASE &var1=&var2 the name of the variable will be resolved down to a single ampersand variable name. 2. where a value is assigned dependent on the variable's original contents, as with *&var1* in the following two statements: STRIPRIGHT &var1 INCREMENT &var1 the variable is resolved to give the initial value (before the operation is performed).
ALL	all the above options
OFF	disables the tracing

Related statements

TRACEEXEC

TRACETERM

Example

```
If &input = 1
   TRACE OFF
If &input = 2
   TRACE VTAM RPL
If &input = 3
   TRACE ALL
Reset &input
Panel
```

will disable the trace if variable *&input* is set to 1. If *&input* is set to 2 all VTAM and RPL requests will be traced, and if *&input* is set to 3 all options will be traced.

Examples of variables logged when using the PROG2 option

In all these PROG2 examples, variables are set as follows; &A=FRED, &B=1, &C=2, &FRED1=TED, &TED=BERT, &TED2=ALF.

When NCI/XF language statements assign values to variables, the statements fall into four categories. There are four sets of examples, one for each category, as follows:

Category1

In this category the first variable in an expression is assigned a new value regardless of its current contents. The first variable will be resolved down to a single ampersand variable name. The following language statements are in this category:

CALCULATE	DECRYPTION	ENCRYPTION	FCALCULATE
LENGTH	REVERSE	SET	TRANSLATE
TYPE	UPPERCASE		

When the PROG2 option is used, variables in this type of language statement are logged as shown in the following table:

Statement executed as:	Shown in the TRACE log as:
SET &C=1	SET &C='1'
LENGTH &C= &FRED1	LENGTH &C='TED'
UPPERCASE &&A&B=12	UPPERCASE &A1='12'
TYPE &A=&&&FRED1	TYPE &A='BERT'
REVERSE &&A&B=&&TED&C	REVERSE &A1='ALF'
ENCRYPTION &&&FRED1=&A	ENCRYPTION &TED='FRED'
FCALCULATE &&A&B=12+(&c.*&b.)	FCALCULATE &A1='12+(2*1)'

Category 2

In this category all variables are resolved to give the initial value. The following language statements are in this category:

ACQUIRE	ALARM	ATTRIBUTE	AUTOINPUT	BEEP
CALL	CHECKFIX	CHECKPANEL	CONNECT	CONNECTR
CURSOR	DECREMENT	DEFINE	DELETE	DEQUEUE
DIRECT	DISABLE	ELSE	ENABLE	ENQUEUE
EXECUTE	EXIT	FCALL	GETLOG	GLOBALV
GOTO	IF	INCREMENT	INTERVAL	LABEL
LOAD	LOG	LOGLEVEL	LOGOFF	MESSAGE
MODSTATS	NOALARM	NOBEEP	OR	PADCENTER
PADLEFT	PADRIGHT	PANEL	PANSTATS	PARSE
POST	PREFRESH	QUERYTERM	QUEUE	RESETATTR
RESETEXEC	RESTART	RETURN	SCAN	SHUTDOWN
STATUS	STRIPCENT	SRTIPEMBED	STRIPLEFT	STRIPRIGHT
SWITCHLOG	TERMUPDATE	TRACE	TRACEEXEC	TRACETERM
TRACEVAR	TRACEVERB	TREFRESH	TRMSTATS	TRUNCCENT
TRUNCLEFT	TRUNCRIGHT	TURN	UNLOAD	UPDATE
UVBL	VALIDATE	VARNAMEL	VERIFY	VEXECUTE
WAIT	WTO			

When the PROG2 option is used, variables in this type of language statement are logged as shown in the following table:

Statement executed as:	Shown in the TRACE log as:
PANSTATS &FRED1	PANSTATS 'TED'
IF &&FRED&B=&&TED&C	IF 'TED'='ALF'
IF &&&FRED1=&A	IF 'BERT'='FRED'
LOAD &&&FRED1	LOAD 'BERT'
GOTO FRED&B	GOTO 'FRED1'
SCAN &&TED&C FOR &A	SCAN 'ALF' FOR 'FRED'

Category 3

In this category all variables in an expression are assigned a new value regardless of their current contents. All variables are resolved down to a single ampersand variable name. The following language statements are in this category:

RESET GRESET ASSIGN

When the PROG2 option is used, the variables in this type of language statement are logged as shown in the following table:

Statement executed as:	Shown in the TRACE log as:
RESET &C &B	RESET &C &B
GRESET &&A&B	GRESET &A1

Category 4

In this category, statements are coded in the)PANEL section. No variable resolution is performed for tracing. The following language statements are in this category:

VARIABLE COLUMNS ROWS ALIAS

Notes

If the TRACE-TERM startup option was specified, or a TRACETERM statement has been executed, tracing will be restricted to the terminal or terminals specified. Otherwise the specified level of trace will be applied to ALL terminals connected to NCI/XF.

If the TRACE-EXEC startup option was specified, or a TRACEEXEC statement has been executed, tracing will be restricted to the EXEC or EXECs specified. Otherwise the specified level of trace will be applied to ALL EXECs.

Variables which have been fully resolved within a TRACE PROG2 statement are enclosed in single quotes so that the length of the variable can be visually determined.

Data that has been encrypted using the ASSIGN statement will be suppressed.

TRACEExec

The TRACEEXEC statement can be used to restrict the range of NCI/XF EXEC (or PANEL) routines that are to be traced.

Format

TRACEEXEC *execname*

where

execname is the name, or a variable set to the name, of an NCI/XF EXEC (or panel) routine to be traced.

A mask value can be entered, or a variable set to a mask value, which identifies a range of NCI/XF EXEC (or PANEL) routines to be traced. The specified mask value will be compared with the name of each EXEC (or PANEL) routine when the trace is active, and trace records will be written for each one whose name matches the mask. Two special characters are recognized when specifying a mask:

- * an asterisk specifies a generic group based on the characters before the asterisk (this means that only characters to the left of the asterisk are used as part of the search data).
- + a plus sign specifies that this character position should be ignored when matching the name with the mask.

Related statements

TRACE

TRACETERM

Notes

Trace output is produced only for the specified EXEC(s). If the EXEC invokes further EXECs they will not be traced unless they are included in the mask.

After execution of a TRACEEXEC statement, the value specified will be stored in the variable *&systrex*. The current setting of the trace EXEC mask may be determined at any time by reference to this variable.

The TRACEEXEC statement does not cause the NCI/XF trace routines to be activated. It identifies which EXECs will be traced when the NCI/XF trace variables are activated by a TRACE request.

TRACEEXEC works in conjunction with the TRACETERM statement, which can be used to restrict tracing to a single terminal or group of terminals.

TRACETerm

The TRACETERM statement can be used to restrict the scope of the NCI/XF trace routines to record the activity of a single terminal or a group of terminals.

Format

TRACETERM *tracing*

where

tracing indicates the type of tracing to be performed. This can be one of the following:

termid is the name, or a variable set to the name, of a terminal whose activity is to be traced.

mask is a mask value, or a variable set to a mask value, which identifies a group of terminals to be traced. The specified mask value will be compared with the name of each terminal connected to NCI/XF when the trace is active, and trace records will be written for each terminal whose name matches the mask.

Two special characters are recognized when specifying a terminal name mask:

* an asterisk specifies a generic group based on the characters before the asterisk (this means that only characters to the left of the asterisk are used as part of the search data).

+ a plus sign specifies that this character position should be ignored when matching the terminal name with the mask.

UPDATE specifies that NCI/XF's internal update scan is to be traced. The output will indicate the reasons why a terminal was or was not selected for)UPDATE processing.

STARTUP selects tracing of either the background execs or NCI/XF startup panel.

NCI selects tracing of NCI/XF internal mainline code.

MODIFY selects tracing of NCI/XF extended modify processing.

Related statements

TRACE

TRACEEXEC

Examples

```
TRACET ny1*
```

restricts tracing to terminals with names starting with the characters ny1.

```
TRACET &usrterm
```

restricts tracing to the terminal which issued the TRACETERM request.

```
TRACET term001
```

restricts tracing to terminal term001.

```
TRACET ++abc++d
```

restricts tracing to terminal whose name has *abc* in positions 2-5 and a d in position eight.

```
TRACET +x*
```

restricts tracing to terminals with names having the character x in position 2.

Notes

After execution of a TRACETERM statement, the tracing value specified will be stored in variable *&systrtr*. The current setting of the trace terminal mask can be determined at any time by reference to this variable.

The TRACETERM statement does not cause the NCI/XF trace routines to be activated. It identifies which terminals will be traced when the NCI/XF trace routines are activated by a TRACE request.

TRACETERM works in conjunction with the TRACEEXEC statement which can be used to restrict tracing to a single EXEC or group of EXECs.

TRACEVar

The TRACEVAR statement is used to monitor references to variables. When a variable entered as a TRACEVAR operand is referenced (either SET, RESOLVEd or RESET), a message is output to the NCI/XF log showing the EXEC that referenced it, the variable name and the variable contents. Each time a TRACEVAR statement is processed, the number of variables currently being traced is placed in system variable *&systv0* and the names of the variables being traced placed in system variables *&systv1* through *&systvNN* (up to 99).

Format

TRACEVAR [*operands* .....]

where

operands..... are variable names without the leading ampersand (&) (or variables that contain a variable name without the leading ampersand) which are to be monitored. The number of variables is limited to 16.

If no operands are entered all current TRACEVAR masks are reset and monitoring of variable names is disabled.

Related statements

TRACEEXEC

TRACETERM

Notes

Subsequent TRACEVAR statements will add variables to the variables already entered by TRACEVAR statements.

The number of terminals and EXECs which are subject to the TRACEVAR statement can be limited by the TRACETERM and TRACEEXEC commands.

The variable name, when logged, has a maximum of 21 printed characters including the ampersand (&). A plus sign (+) as the 21st character indicates that the variable name has been truncated for logging purposes.

The variable contents, when logged, are restricted in length to the remainder of the log line. A plus sign (+) in the final character position of the line indicates that the variable contents have been truncated for logging purposes.

Encrypted variables (using the ASSIGN statement) will show as N/A.

TRACEVErb

The TRACEVERB statement is used to trace all invocations of an NCI/XF language statement, or group of NCI/XF language statements. The output produced is sent to the NCI/XF log.

Format

TRACEVERB [*verbs* .....]

where

verbs ..... are valid NCI/XF language statements. The abbreviated form of each language statement can be used. If no parameter is entered, all TRACEVERB masks will be reset. If an invalid operand is specified, it will be ignored and a warning message written to the NCI/XF log. A maximum of 16 language statements can be entered.

Related statements

TRACEEXEC

TRACETERM

Return codes

After execution of a TRACEVERB statement, one of the following values will be returned:

0 function completed successfully.

4 one or more of the operands was not a valid NCI/XF verb.

Related variable

&systrvb

Examples

```
TRACEVE update termupdate
```

will trace the update and termupdate verbs.

```
TRACEVE
```

will disable TRACEVERB tracing.

Notes

The terminals and EXEC routines which are subject to the TRACEVERB statement can be limited by TRACETERM and TRACEEXEC commands.

TRANslate

The TRANSLATE statement is used to change all non-printable characters within a string to dots, or to translate one character in a string to another.

Format

TRANSLATE *&var1* = *&var2* [*fromlist tolist*]

where

&var1	is the name of the variable where the result of the translation will be stored.
&var2	is a literal string to be translated, or a variable containing a string. Those characters which match the characters in *fromlist* will be translated to the corresponding characters in *tolist*.
fromlist	is a list of characters to be translated to corresponding characters in *tolist*.
tolist	is the list of characters, corresponding to the character positions in *fromlist*, to be used in translation.

Examples

```
TRAN &print = &unprint
```

All unprintable characters within *&unprint* are translated into dots and the complete string with replacement dots is placed in *&print*.

```
Set &foobar = 'abcdef'
TRAN &result = &foobar ac yz
```

results in *&result* being set to ybzdef.

```
Set &idate = '89/06/17'
TRAN &foobar = 12/34/56 56/34/12 &idate
```

sets *&foobar* equal to 17/06/89.

TREFresh

The TREFRESH statement refreshes the specified terminals with either the panel currently being displayed or a different panel.

Format

TREFRESH *termid* [*panel* [*alarmopt*]]

where

termid is the name of a terminal or a variable set to the name of a terminal. The specified terminal will be refreshed with the current panel; or if the *panel* operand is supplied, refreshed with that panel. Mask characters can be specified to define a group of terminals to be refreshed (see *Mask characters* on page 1.188).

panel is the name of the panel, or a variable set to the name of the panel, for display on the terminal(s) specified by *termid*. Mask characters can be specified to define the name of the panel (see *Mask characters* on page 1.188).

If this operand is omitted, the panel currently displayed in *termid* will be refreshed

If a *panel* operand is not required, the *alarmopt* operand can only be entered if *panel* is entered as a space between single quotes or as *&sysnull*.

alarmopt specify one of the following NCI/XF language statements to control the terminal alarm on the receiving terminal:

ALARM | BEEP | NOALARM | NOBEEP

These are described in this chapter under the corresponding language statements. If omitted, the default leaves the current alarm setting at the refreshed terminal.

Examples

See Notes on page 1.188 for detailed explanations of the TREFRESH operands.

```
TREF lu01
```

Terminal lu01 will be immediately refreshed. The panel currently displayed will be redisplayed. Its)INIT section will not be driven.

```
TREF lu01 *
```

Terminal lu01 will be immediately refreshed. The)INIT section of the panel currently displayed on lu01 will be driven.

```
TREF lu02 menu1
```

Panel menu1 will be immediately displayed on terminal lu02. The)INIT section of panel menu1 will be driven.

```
TREF local*
```

All terminal names starting with the characters local will be immediately refreshed.

```
TREF ++++++M2
```

All terminal names with characters m2 in positions 7 to 8 will be immediately refreshed.

```
TREF term1 news*
```

Terminal term1 will be updated with a panel starting news. The name of the new panel will depend on the panel currently displayed on term1.

MENUMOD2 ==> NEWSMOD2
LOGOMOD3 ==> NEWSMOD3

```
TREF local* news*
```

All terminal names starting with local will be updated with a panel name beginning with news. The rest of the panel name will be built from the name of the panel originally displayed. For example:

LOGO123 ==> NEWS123
MENUX ==> NEWSX

```
TREF term* *
```

All terminal names starting with term will redisplay the same panel. The)INIT section of the panels will be redriven.

```
TREF a+b+c* ++++wed
```

All terminal names with an a in position 1, b in position 3 and c in position 5 will be replaced with a panel which has wed in positions 5 through 7 of its name. The rest of the panel name will be built from the name of the panel currently displayed on each terminal

MENU01 ==> MENUWED
LOGOTUES ==> LOGOWED

Notes

Mask characters

Two special characters (* and +) are recognized as mask characters when specifying operands on a TREFRESH statement. The meaning of the characters depends on whether they are used as part of the first or second operand.

When used within the first operand (*termid*)

- * specifies a generic group of terminals based on the characters before the asterisk. During a search for terminals to refresh, only those characters to the left of the asterisk are used as part of the criteria.
- + stipulates that this character position of the terminal name be ignored when the panel name is being matched with the mask.

When used within the second operand (*panel*)

- * stipulates that the remainder of the panel name, starting with the character position which contains the asterisk (*), be built with the corresponding positions from the name of the panel currently being displayed on the target terminal.
- + stipulates that the character position containing the plus sign (+) be replaced with the corresponding position from the name of the panel currently being displayed on the target terminal.

Redisplaying the same panel

If only *termid* is specified, the display area of matching terminals will be refreshed, that is the panel currently displayed will be redisplayed. If any variables on the panel have changed, the updated values will be displayed. This is particularly useful if, for instance, the hot news broadcast messages are updated. The TREFRESH statement may be used to cause all terminals to be immediately refreshed with the updated broadcast messages.

If only the first operand is specified, the)INIT section of the panel will not be redriven. Action will be taken as if a PANEL statement with no operands had been executed at the target terminal(s).

If the same panel is to be redisplayed and its)INIT section redriven, code an asterisk (*) mask character as the second operand.

Displaying a different panel

If the second operand (*panel*) is specified, all terminals matched by *termid* will be immediately refreshed with the new panel. The)INIT section of *panel* will be driven, as if a PANEL *panel* statement had been executed at the target terminal(s).

Refreshing multiple terminals

Multiple terminals may be refreshed with a single TREFRESH statement by the use of mask characters as part of *termid*. The name of the replacement panel to be sent to the target terminals(s) may be based on the name of the panel currently displayed by means of mask characters included in the *panel* operand.

Target terminal status

All terminals connected to NCI/XF have a refresh status which may be ON, OFF or DELAY. At initial connection time, all terminals have a refresh status of ON, indicating that REFRESH commands will be processed. Execution of a TURN REFRESH statement may subsequently alter the refresh status to OFF, causing REFRESH operations to be ignored, or DELAY causing REFRESH operations to be queued. Refer to the description of the TURN statement for further details.

Queueing of TREFRESH requests

TREFRESH updates the display area of all terminals matched by *termid* when the TREFRESH command is issued. This is a synchronous operation and control is not returned to the issuing terminal until all required terminal SENDs have been queued to VTAM.

Since all NCI/XF mainline code, apart from the exceptions listed below, is itself serial and synchronous, there is no danger of the target terminals being interrupted in the middle of NCI/XF language processing.

Exceptions to this are terminals which have issued the following asynchronous requests:

- a CALL statement to invoke an asynchronous exit
- a STATUS request to obtain application status
- a CONNECT request, in which case the terminal may be connected to another application
- a VTAM SEND of a panel to a terminal

In all these cases, the TREFRESH request will be queued at the receiving terminals and executed when NCI/XF regains control at the end of the asynchronous request or application session.

Users of previous releases

The TREFRESH command is particularly useful for immediately advising all users of critical changes occurring within the network, such as application status changes, updates to hot news broadcast messages or for sending messages to other terminal users. Users of previous releases of NCI/XF, or customers upgrading from NCI, who use)UPDATE processing to handle the automatic updating of terminal displays are strongly recommended to consider the use of TREFRESH as a more efficient alternative.

TRMStats

The TRMSTATS statement is used to extract statistics relating to terminals. The statistics are queued until they are referenced via the user variable *&usrtrms*. Once referenced, the statistics are automatically removed from the queue by NCI/XF.

Format

TRMSTATS *name* [*limit*]

where

name is the name of a terminal or a variable set to the name of a terminal. The statistics for the specified terminal will be queued for later reference by the use of the variable *&usrtrms*. Mask characters may be specified to select a group of terminals whose statistics will be queued. Two special characters asterisk (*) and (+) are recognized as mask characters:

- * specifies a generic group of terminals on the basis of the characters before the asterisk (*). Only those characters left of the asterisk (*) are used as part of the search data.
- + specifies that this character position of the terminal name should be ignored when the terminal name is being matched with the mask.

limit specifies a numeric value, or a variable containing the required value, in the range 0 through 32767 and is used to limit the number of terminal statistics queued as a result of group selection. If 0 is specified then the queue will be emptied. If omitted, there is no limit to the number of statistics queued.

Return codes

After execution of a TRMSTATS statement, variable *&usrretc* is set to one of the following return codes:

0 successfully extracted terminal statistics.
4 empty queue, no terminal statistics extracted.
8 syntax error.

Examples

```
TRMS userterm
```

If terminal userterm has been used:

&usrretc = 0
&usrtrms = userterm 1225

indicating that terminal userterm has used 1225 milliseconds of CPU time since LOGON.

If terminal userterm has not been used:

&usrretc = 4
&usrtrms = a null value

```
TRMS A01++++8
```

Assuming A01MS248, A01MS258 and A01XX258 have been used

&usrretc = 0
1st *&usrtrms* = A01MS248 3855
2nd *&usrtrms* = A01MS258 21
3rd *&usrtrms* = A01XX258 2733
4th *&usrtrms* = a null value

Notes

Statistics queue

Whenever user variable *&usrtrms* is referenced the statistics are automatically removed from the queue; if the queue is empty and *&usrtrms* is referenced, a null value will be returned.

The queued statistics from a TRMSTATS statement are in alphabetical order.

After a TRMSTATS statement, if some of the queued statistics have not been referenced, they will remain on the queue. If the remaining statistics are no longer required, the storage they occupy can be freed by using a TRMSTATS statement without operands, or by specifying a selection with a limit of 0.

Statistics available

When variable *&usrtrms* is resolved, the following statistics will be returned separated by blanks:

- terminal name
- CPU time since LOGON (in milliseconds)

Individual values can be extracted from *&usrtrms* with the PARSE statement using blank as a delimiter.

Special terminal names

TRMSTATS can also be used to extract statistics for two of NCI/XF's internal functions. A terminal name of STARTUP identifies the NCI/XF startup panel or background EXECS. A terminal name of PANSTORE identifies the background task that frees panel storage after a DELETE statement has marked a panel for deletion.

Extending the statistics

Variable *&usrtrms* may be extended in future releases of NCI/XF by adding new statistics to the end of the current statistics. Any existing statistics will therefore remain unchanged to maintain backward compatibility. If PARSEd, using blank as a delimiter, they will maintain their relative positions in the resultant variables *&usrpm01* to *&usrpmNN*.

TRUNCCent

The TRUNCCENT statement is used to truncate a string of characters to a specific length, by removing characters from both ends of the string.

Format

TRUNCCENT *variable n*

where

variable is the name of the variable in which the string to be truncated is held.

n is the length to which the string is to be truncated.

Related statements

TRUNCLEFT

TRUNCRIGHT

Examples

```
Set &foobar = 'abcdefgh'
TRUNCC &foobar 4
```

would result in *&foobar* being equal to cdef.

```
Set &foobar = 'abcdefgh'
TRUNCC &foobar 5
```

would result in *&foobar* being equal to cdefg.

TRUNCLeft

The TRUNCLEFT statement is used to truncate a string of characters to a specific length, by removing characters from the left of the string.

Format

TRUNCLEFT *variable n*

where

variable	is the name of the variable in which the string to be truncated is held.
n	is the length to which the string is to be truncated.

Related statements

TRUNCRIGHT

TRUNCCENT

Examples

```
Set &foobar = 'abcdefgh'
TRUNCL &foobar 4
```

would result in *&foobar* being equal to efgh.

```
Set &foobar = 'abcdefgh'
TRUNCL &foobar 5
```

would result in *&foobar* being equal to defgh.

TRUNCRight

The TRUNCRIGHT statement is used to truncate a string of characters to a specific length, by removing characters from the right of the string.

Format

TRUNCRIGHT *variable* *n*

where

variable	is the name of the variable in which the string to be truncated is held.
n	is the length to which the string is to be truncated.

Related statements

TRUNCLEFT

TRUNCCENT

Examples

```
Set &foobar = 'abcdefgh'
TRUNCR &foobar 4
```

would result in *&foobar* being equal to abcd.

```
Set &foobar = 'abcdefgh'
TRUNCR &foobar 5
```

would result in *&foobar* being equal to abcde.

TURN

The TURN statement is used to specify terminal status options.

Format

TURN *option1* *option2*

where

option1 is one of the following:

MAXEXEC When *option2* is specified with a value *n*, this value will override the MAXEXEC startup option for the current terminal. The MAXEXEC startup option determines the system wide default for the maximum number of EXEC statements in a panel section. This value must be an integer between 0 and 32767. A value of 0 will turn off checking.

If *option2* is specified as SYSTEM, the system wide default will be used.

See associated user variable *&usrmaxe* on page 3.4.

MAXGOTO When *option2* is specified with a value *n*, this value will override the MAXGOTO startup option for the current terminal. The MAXGOTO startup option determines the system wide default for the maximum number of GOTO statements in a panel section. This value must be an integer between 0 and 32767.

If *option2* is specified as SYSTEM, the system wide default will be used.

See associated user variable *&usrmaxg* on page 3.4.

NOLOG Refer to the *GROUP statement* on page 3.22 of the NCI/XF Installation Manual.

See associated user variable *&usrnlog* on page 3.4.

The NOLOG option requires *option2* to be specified as either ON or OFF.

NORES Refer to the *GROUP statement* on page 3.22 of the NCI/XF Installation Manual.

See associated user variable *&usrnres* on page 3.5.

The NORES option requires *option 2* to be specified as either ON or OFF.

NOSIM Refer to the *GROUP statement* on page 3.22 of the NCI/XF Installation Manual,.

See associated user variable *&usrnsim* on page 3.5.

The NOSIM option requires *option2* to be specified as either ON or OFF.

PRODM (VM only) causes a GCS ACCESS command to be issued before any panel or exec routine is loaded from a minidisk into storage.

See associated system variable *&sysprod* on page 2.5.

The PRODM option requires *option 2* to be specified as either ON or OFF.

REFRESH When *option2* is specified as ON, this will allow the terminal to receive REFRESH commands.

When *option2* is specified as OFF, REFRESH commands will be ignored.

If *option2* is specified as DELAY, REFRESH commands will be queued until *option2* is returned to ON.

See associated user variable *&usrreff* on page 3.5.

SEND when *option2* is specified as PROT, this will send only protected fields back to the terminal until the command is cancelled (by a different TURN command) or a new panel is displayed.

when *option2* is specified as ALL, this will send all fields back to the terminal, as normally performed. This is the default setting used when a terminal logs on or displays a different screen.

TIMESTAMP When *option2* is specified as HIGH, this will cause the timestamp on the NCI/XF log to be displayed in the format:SS.NNNN, where SS represents seconds and NNNN ten thousandths of a second.

When *option2* is specified as LOW, this will cause it to revert to the normal setting of HH:MM:SS.

If TURN TIMESTAMP *n* is specified, where *n* is a single digit in the range zero to six, this specifies the number of decimal places to be displayed on the timestamp.

Only columns one to eight of the log are available for the time and hence only the eight low order digits are displayed.

UPPER When *option2* is specified as ON, this will cause all data transmitted to the terminal to be translated to uppercase. When OFF is specified, transmitted data will be received as sent.

See associated user variable *&usruppr* on page 3.6.

option2 is dependent upon the *option1* specified. The following *option2* settings are available:

ON	turns on NOLOG, NORES, NOSIM, PRODM, REFRESH and UPPER.
OFF	turns off NOLOG, NORES, NOSIM, PRODM, REFRESH and UPPER.
DELAY	refer to the REFRESH option.
PROT	refer to the SEND option.
ALL	refer to the SEND option.
HIGH	refer to the TIMESTAMP option.
LOW	refer to the TIMESTAMP option.
SYSTEM	specifies that the system-wide default for the MAXEXEC or MAXGOTO option will be used as defined in the MAXEXEC and MAXGOTO startup options.
n	is an integer between 0 and 32767 which specifies a new value for the MAXEXEC or MAXGOTO options.

TYPe

The TYPE statement is used to establish whether the type of data supplied for testing is alphabetic or numeric.

Format

TYPE *&var1* = *&var2*

where

&var1	is the variable to be set to the type of data supplied. After execution of TYPE, this variable will contain either CHAR or NUMB for character and numeric data respectively. If numeric data contains one or more alphabetic characters, the type will be defined as CHAR.
&var2	is the data to be tested.

Notes

The TYPE statement also supports the use of variable variables.

For further information on the use of variable variables, see page 2.18 of the NCI/XF User Guide.

UNLOAD

The UNLOAD statement allows a user to delete a copy of a re-entrant module previously loaded into core with the LOAD statement.

Format

UNLOAD *modules* .....

where

modules are the module names to process. Up to 16 module names can be provided in one instruction.

Related Statements

CALL

FCALL

LOAD

MODSTATS

Return codes

If all modules are successfully unloaded, variable *&usrretc* is set to return code 0, otherwise the return code is set to the position in the list of the first module that failed to unload.

Notes

Processing is halted when any module fails to unload.

A module load count is maintained for each loaded module. The count is incremented each time a LOAD statement is executed, and decremented when an UNLOAD statement is executed. Only when this count is decremented to zero will the module be deleted from core.

Statistics at NCI/XF shutdown pertaining to the number of times a module has been called and loaded can be produced by coding the startup option CALL-STATS=YES

Each module which has been called is then displayed with a count of the number of times it was called, and the number of times it required loading from disk. The same type of module-related statistics can be obtained during NCI/XF execution by either the MODSTATS verb or the @UTSTATS exec routine.

UNSTRING

The UNSTRING statement allows you to split a string of data, created by a STRING statement or some similar means, into its individual components. These components are placed in a stem variable set. The zero suffixed stem entry contains the number of components.

Format

UNSTRING *stem = &var1*

where

stem is the variable stem.

&var1 is the variable containing the string of data to be split.

Example

The following example assumes that

&alldata = "*aa*12345*bb*ABCD"

where

aa is x'0005', the length of 12345

bb is x'0004', the length of ABCD

Therefore:

```
UNSTRING part = &alldata
```

results in the following variables being set:

&part0 = "2"', the number of variables set

&part1 = "12345"

&part2 = "ABCD"

Return codes

On completion of the UNSTRING command, one of the following return codes will be stored in *&usrretc*:

0 completed successfully.

4 parameter error or invalid data.

UPDate

The UPDATE statement allows the basic timer interval, as defined by the UPDATE-INTERVAL startup parameter, to be dynamically modified whilst NCI/XF is running. For further details see *Timer services* on page 3.2 of the NCI/XF User Guide. The UPDATE statement will be ignored if the startup option UPDATE-INTERVAL=NO was specified.

Format

UPDATE [*n*]

where

n specifies the new timer interval in seconds. A timer interval must be numeric in the range 0 - 999999.

A entry of 0 stops all timer update processing until restarted by a subsequent UPDATE command. This facility is useful for temporarily quiescing update processing, at night or over a weekend.

Notes

Specifying UPDATE without operands

This will cause an immediate update cycle to be invoked, bypassing TERMINAL-UPDATE.

Terminals which display a panel that has INTERVAL 1 specified or defaulted in the)PANEL section, and have performed no terminal activity since the previous update cycle will have the)UPDATE section of the panel immediately driven, so bypassing TERMINAL-UPDATE.

Terminals which display a panel that specifies INTERVAL greater than 1, and have performed no terminal activity since the previous update cycle will have their internal interval count incremented. If this causes the internal count to reach the value laid down in the INTERVAL statement, the)UPDATE section of the panel will be driven, otherwise the internal counter remains incremented and the terminal is bypassed.

The terminal which invokes an immediate update cycle is exempt from update processing because it is considered active during that update cycle.

Immediate update processing does not affect the current update cycle. For instance, if the update interval is 45 seconds and the last update cycle 'popped' 25 seconds before the UPDATE statement was processed, the next update cycle will 'pop' 20 seconds later.

Specifying *n*

Specifying a numeric operand other than zero will change the update interval to the specified number of seconds. The current update cycle is halted and a new cycle is started using the new value.

Variable &systint

NCI/XF automatically maintains variable *&systint* to indicate the current update timer interval.

When NCI/XF is started, *&systint* is set to the default timer interval of 60 seconds.

If the UPDATE-INTERVAL startup parameter has been coded, *&systint* will be updated to reflect the value specified.

If UPDATE-INTERVAL=NO was specified, variable *&systint* will be set to DISABLED.

Whenever an UPDATE statement is executed, other than a statement with no operands, *&systint* will be updated to reflect the new update interval.

UPPERcase

The UPPERCASE statement is used to set the contents of a variable into uppercase.

Format

UPPERCASE *&var1* = *&var2*

where

&var1	is the variable into which the uppercased value will be placed.
&var2	is the variable to be uppercased.

Examples

```
UPPER &user = &userid
```

The value stored in *&userid* is uppercased and placed in variable *&user*.

```
UPPER &a = &a
```

The value stored in *&a* is uppercased and put in *&a*.

Notes

The UPPERCASE statement supports the use of variable variables.

For further information see *Variable variables* on page 2.18 of the NCI/XF User Guide.

UVBL

The UVBL statement is used to change the size of a User Variable Block (UVB) for a specific user. The value takes effect for any new storage allocations; it does not change any existing allocations for that user. The value specified by the UVBL statement overrides the size of the UVB specified by the UVBL pre-startup option (see page 3.5 of the NCI/XF Installation Manual).

Format

UVBL *n*

where

n is the size in bytes of the next UVB to be allocated. Valid sizes are 128 through 8192.

Return codes

After execution of a UVBL statement, variable *&usrretc* is set to one of the following return codes:

0 UVB size successfully changed.
4 UVB size requested is outside the valid range.

VALidate

The VALIDATE statement can be used to verify a userid and password against a security database. When this statement is processed, control is passed to user exit NCS*xxxx*, where *xxxx* is the system name as coded in the *system* operand (see below). The user exit informs NCI/XF via register 15 whether the userid/password combination is valid and the value in register 15 is returned in *&usrretc*. The user exit cannot use any of the NCDIALOG macro services for resolving and setting variables. To use these services you should use the CALL or FCALL statement. VALIDATE provides invocation of a limited range of program names. The CALL or FCALL commands should be used rather than VALIDATE which is maintained by NCI/XF for backwards compatibility only.

Format

VALIDATE *system luname userid passwd* [*newpass*]

where

system defines the security system to be invoked to do the validation. It may be specified as:

- ACF2 - ACF2 by Computer Associates
- RACF - RACF by IBM
- TOPS - TOPSECRET by CA
- SAC - ALERT/SAC by GOAL Systems
- NCI - NCI/XF's own Directory of Users

luname is a terminal name, or a variable containing a terminal name. This operand is not used in the supplied sample exit (but can be used by the exit if extended security tests are required). It is recommended that *luname* is coded as *&sysnull* or a space between single quotes.

userid is the userid, or a variable containing the userid, to be verified.

passwd is the password, or a variable containing the password, to be verified.

newpass specifies a new password.

Non ACF2 return codes

After execution of a VALIDATE statement on a non ACF2 system, variable *&usrretc* is set to one of the following return codes:

0 Userid and password validation successful.
4 Userid not found.
8 Invalid password specified.
12 Password expired.
16 Invalid new password specified.
20 Unknown error.
24 RACROUTE REQUEST = VERIFY failed by installation exit routine (RACF only).
28 Userid revoked.
32 Command not supported in DOS/VSE.

ACF2 return codes

The return codes from VALIDATE ACF2 are derived from the ACF2 message number produced, using the following rules:

0	Userid and password correct.
0*xx*	ACF2 error message - ACF010*xx*.
1*xx*	ACF2 warning message - ACF011*xx*.
1000	ACF2 not active.
2000	Parameter list error.
3000	Unknown error.

More common values for *xx*, the ACF2 message id, are:

4	Logon ID not found.
11	Logon ID suspended.
12	Password not matched.
13	Logon ID suspended due to password violations.
14	Logon ID expired.
15	Invalid syntax for password.
17	Password expired.
18	Invalid syntax for new password.
19	Password less than minimum length.
20	New password less than minimum length.
23	Old and new passwords same.
128	Invalid syntax for new password warning.
129	Password successfully altered notification.
130	New password less than minimum length.
131	Old and new passwords same warning.
134	Password expiration soon warning.

Example

```
VAL RACF &usrterm user1 pass1
If &usrretc eq 8
   set &msg = 'Invalid password specified - Restart'
```

will validate the given criteria and set the variable *&msg* to the above text if an incorrect password was entered.

Notes

Mainline NCI/XF code does very little processing in response to a VALIDATE request. A parameter list is built containing the specified VALIDATE operands and control is passed to the VALIDATE exit routine appropriate to the security system specified on the VALIDATE statement. The security system exit routines are as follows:

RACF	-	NCSRACF
ACF2	-	NCSACF2
TOPS	-	NCSTOPS
SAC	-	NCSSAC
NCI	-	NCSNCI

Refer to *Security services* on page 3.20 of the NCI/XF User Guide.

Modifying VALIDATE processing

The assembler source of all VALIDATE routines except NCSNCI is supplied in the SAMPLE library of the NCI/XF distribution tape. The source routines can be amended as required. NCSNCI is supplied in load module format only and cannot be amended. It provides a userid and password verification against NCI/XF's Directory of Users. Users can extend the scope of VALIDATE processing and write separate user exit routines, invoked with the CALL statement, to provide additional specialized access to their security database.

VARiable

The VARIABLE statement allows the specification of a character other than the ampersand (&) to be used to represent variables within a panel, thus enabling the coding of the ampersand (&) character as a literal in the)BODY section.

Restrictions

VARIABLE is valid in the)PANEL section only. A variable that has had its prefix changed with the VARIABLE statement can be used only in the)BODY section of a panel. Always code an ampersand (&) when you use a variable name in the)INIT,)UPDATE and)PROC sections.

Format

VARIABLE *char*

where

char is any alphanumeric character.

Example.

```
)panel
 Interval 1
 Rows     24
 Columns  80
 Attr     @ high protected
 VAR      +
 )body
 @date:+sysdate
 @time:+systime

        WELCOME TO A & B ONLINE SYSTEMS
 )proc
```

will set the variable name prefix to the plus sign (+).

VARNAMEL

The VARNAMEL statement is used in a)PANEL section to set the maximum length of variable names in an NCI/XF panel or routine.

Restrictions

VARNAMEL is valid in the)PANEL section only.

Format

VARNAMEL *nnn*

where

nnn is the default maximum length of variable names in the range 8 to 255.

Notes

VARNAMEL will apply for the duration of processing for the current panel or routine only.

Executed routines will support a)PANEL section coded before the)ROUTINE section to enable use of the VARNAMEL option.

If the VARNAMEL option is not coded, the setting will default to the VARNAMEL startup option. If the startup option is not coded, the default is 8.

VERify

The VERIFY statement is used to verify that any data entered at the terminal is in the correct character format.

Format

VERIFY *&data* = *mask*

where

&data	is the variable containing data to be verified.
mask	is the mask used for the verification. This can be a literal or a variable containing the mask data.

Return codes

After execution of a VERIFY statement, variable *&usrretc* is set to one of the following return codes:

0 data matches mask.
4 data does not match mask (*&usrvpos* is set to the position in the input string of the character which failed to match the mask).
8 the mask was invalid.

Examples

The following examples assume use of the supplied default mask table:

```
Set &partnum = 1234b/65
VER &partnum = NNNNA/NN
```

VERIFY checks for the following format:

characters 1-4	must be numeric.
character 5	must be alphabetic uppercase.
character 6	must be a slash (/) character.
characters 7-8	must be numeric.

In this case, the data does not meet the mask specification because character 5 is lower case.

&usrretc will be set to 4 (VERIFY failed) and *&usrvpos* will be set to 5.

```
Set &mask = $****.00
Set &amount = $3792.00
VER &amount = &mask
```

VERIFY checks for the following format:

character 1	must be a dollar symbol.
characters 2-5	can be any character.
character 6	must be a decimal point.
characters 7-8	must be zeros.

In this case, the data conforms to the mask specification. *&usrretc* will be set to 0 (success) and *&usrvpos* will be set to 0.

Notes

Verification proceeds left to right one character at a time. Each data character is matched with the corresponding mask character. If the mask character is defined in the NCI/XF mask table, the data character must conform to the appropriate mask table definition (such as numeric). If the mask character does not appear in the NCI/XF mask table, the data character must exactly match its corresponding mask character.

Default mask table

A mask table is supplied with the following defaults:

A	alphabetic uppercase
C	alphabetic upper and lowercase
9	numerics 0 to 9 and + - / * ( )
N	numerics 0 to 9
H	hexadecimal (0-9, A-F)
*	any character

The mask characters A, C, N and H must be coded in uppercase.

Modifying the mask table

The NCI/XF default mask table is contained in module NCVERT of the NCI/XF load library. The source of NCVERT is supplied in the SAMPLE library. The default table may be modified by updating and reassembling this member.

VEXECute

The VEXECUTE statement executes a variable routine. Each operand of VEXECUTE is resolved and forms a line of an internally generated EXEC routine which is then executed and deleted.

Format

VEXECUTE *line* .....

where

line ..... are the individual lines of the EXEC routine. Up to 16 operands can be specified for each VEXECUTE statement.

Related statement

EXECUTE

Example

```
SET &¢line1 = 'SET &num1 = 1'
SET &¢line2 = 'SET &num2 = 1'
SET &¢line3 = 'CALC &tot = &num1.+&num2.'
VEXEC &¢line1 &¢line2 &¢line3 'reset &num1 &num2'
```

sets *&tot* to the sum of *&num1* and *&num2* (in this case 2).

Note

If an error is encountered while an internal version of the EXEC routine is being generated then the error panel NCERR2 will be displayed.

WAIT

The WAIT statement causes panel processing to be suspended until a POST *terminal-id* command is issued.

Format

WAIT

Related statement

POST

WTO

The WTO statement writes a specified data line to the Operator Console using console routing codes 2 and 11.

Format

WTO *&line* [NON-ROLL]

where

&line	contains the line to be printed to the console.
NON-ROLL	allows non-rollable WTOs to be issued unless the startup parameter WTO-ROLL=YES is specified.

Example

```
WTO 'Please mount tape TY123'
```

will write the specified text to the screen.

Note

This statement is functionally equivalent to the message operator except that the job name is not prefixed to the line.

X2C

The X2C statement converts hexadecimal data to character format.

Format

X2C *operand 1* = *operand2*

where:

operand1 is the variable to receive the converted data.

operand2 is the value or variable to be converted. The data to be converted must consist of an even number of hexadecimal digits. The same variable as *operand1* may be specified.

Related statements

A2E

C2D

C2X

D2C

D2P

E2A

Example

```
X2C &char = C1C2C3
```

will result in variable *&char* being set to ABC.

Chapter 2 - NCI/XF defined system variables

NCI/XF reserved system variables 2.2
System variables provided by the User Support Environment (USE) 2.10
System variables provided by NC-MHO 2.11
System variables provided by NC-XMS 2.12

Note: Variables marked with an asterisk (*) in the left hand margin are not set when NCI/XF is running MODE=STANDARD.

Variables starting with *&sys* are reserved.

&*#USE If not set, the system does not have MODE=ENHANCED.

If NO, MODE=ENHANCED is set but USE is not.

If YES, both MODE=ENHANCED and USE are set.

&*#LOGO name entered on LOGO-EXEC= startup option.

&SYSACB the name of the ACB under which NCI/XF is currently running.

&SYSAMOD defines the addressing mode as 24BIT or 31BIT.

&SYSAMP contains the ampersand (&) character. Used to include an ampersand (&) character in a data string and therefore prevent NCI/XF treating the data as a variable name, for instance:

'A&SYSAMP.B ' means 'A&B'

'A&B' means A followed by the contents of &B.

&SYSAUTH indicates which version of NCI/XF is authorized by the CODE= startup option. Possible versions are:

NCI BASE	the basic version of NCI/XF
NCI/XF	the extended function version of NCI/XF

This has a NULL value if neither is authorized.

&SYSBLNK is set to a field of 255 blanks. Useful for setting user variables to blanks, for instance SET &a = &SYSBLNK(1:22).

&SYSCONM the number of unprinted lines in the print buffer.

&SYSCPU the CPUID as follows:

MVS	-	the CPUID of machine running NCI/XF.
VM	-	if NCI/XF is running authorized, this variable contains the CPUID of the virtual processor running NCI/XF during startup processing. If NCI/XF is not running authorized, this variable is set to *&sysnull*.

&SYSDATE contains the current date in the format specified by the DATE= startup option (default DD/MM/YY).

&SYSDATF the date format (the literal string specified for the DATE= startup option).

&SYSDAY a three digit value representing the Julian day of the year, for instance 030 for the 30th January.

&SYSDAYN a numerical value set to the day number of the week, where the value 1 indicates MONDAY, 2 indicates TUESDAY, 3 indicates WEDNESDAY, etc.

&SYSDIRN the dataset name pointed to by the DIRECT DD statement under MVS or the DIRECT FILEDEF command under VM.

&SYSECPU the CPU time as follows:

MVS - the total CPU time since startup (in milliseconds).

VM - not available; always set to *&sysnull*.

&SYSEREX the name of the ERROR-EXEC.

&SYSFAIL the total number of failed PANEL scavenges.

&SYSFFIX the number of the first fix for this release.

&SYSFIX0 through &SYSFIX9 contains a table indicating which fixes have been applied to this version of NCI/XF. Each variable is 100 bytes in length and indicates the status of all fixes from 0 to 999:

&sysfix0	fixes - 0 to 99
&sysfix1	fixes - 100 to 199
&sysfix2	fixes - 200 to 299, etc.

Each character position within the table may contain:

0	fix not applied
1	problem
2	customization

&SYSFMID contains the product FMID identifier for the control program, indicating the version of MVS on which NCI/XF is executing.

&SYSGETP reserved.

&SYSGOTO the value specified in the MAXGOTO= startup option.

&SYSHIGH high water panel storage in bytes.

&SYSHACQ the maximum number of asynchronous ACQUIRE commands which have been processed concurrently.

&SYSHINQ the maximum number of asynchronous STATUS commands which have been processed concurrently.

&SYSHQCR the maximum number of CALL routines which have been queued concurrently; queuing occurs when the number of active CALL routines exceeds the MAXIMUM-SUBTASKS value specified as a Startup option.

&SYSJOBN Under MVS, this contains the JCL job name that executed NCI/XF.

Under VM this contains the virtual machine ID in which NCI/XF is currently running.

&SYSLDDN the current LOG ddname.

&SYSLLEV The level at, or below, which text is to be written to the NCI/XF log. This variable is set by the LOGLEVEL statement. When a LOG statement is invoked, this level is compared with the *level* operand on the LOG statement to determine which level will apply.

&SYSLLPR the number of the last printed record.

&SYSLMAX the value specified in the MAX-LOG-SIZE= startup option.

&SYSLNEW the number of the newest record in the LOG buffer.

&SYSLOLD the number of the oldest record in the LOG buffer.

&SYSLOST contains the name of the panel to be processed when a session is unexpectedly lost (startup option LOSTERM). Note that when MODE=ENHANCED is running this has a fixed value of *@nclostt*.

&SYSLSIZ the number of records printed to the current log.

&SYSLUST contains the name of the panel to be driven when a power-off LUSTAT is received, specified in the LUSTAT startup option. Note that when MODE=ENHANCED is running this variable has a fixed value of *@nclust*.

&SYSMAXP the maximum panel storage in bytes.

&SYSMLEV contains the current PUT level.

&SYSMODF the maximum number of MODIFY commands which have been processed concurrently.

&SYSNACR the current number of active CALL routines.

&SYSNDEL the number of panels deleted.

&SYSNEXE the current number of EXEC routines built and held in core.

&SYSNLUS the total number of terminals which have connected to NCI/XF since it was started.

&SYSNMOD the number of MODIFY commands received.

&SYSNODE NCI/XF's VTAM node name (unavailable in the INITIAL-EXEC).

&SYSNPAN the current number of panels built and held in core.

&SYSNQCR the current number of queued CALL routines. Queuing occurs when the number of active CALL routines exceeds the MAXIMUM-SUBTASKS value specified as a Startup option.

&SYSNQUE the current number of system queues in use by NCI/XF.

&SYSNULL contains blank or binary zero (which must be distinguished from character zero); it is used during comparisons to check that one comparate contains nothing at all.

&SYSNVAR the number of system variables currently defined.

&SYSOPID refers to the operating system under which NCI/XF is running. Possible values are MVS, VSE and GCS.

&SYSP0 the total number of &sysp*nn* variables set (MODE=ENHANCED only).

&SYSP1 through &SYSP*nn* Every variable contains unrecognized startup option (MODE= ENHANCED only).

&SYSPANN contains the name of the partitioned dataset pointed to by the PANEL DD statement used under MVS or the PANEL FILEDEF command under VM. If concatenated libraries were specified, this variable only names the first.

&SYSPRIV Under MVS, this contains either YES or NO to determine if NCI/XF is running APF authorized.
Under VM, this contains either YES or NO to determine if NCI/XF is running GCS authorized.

&SYSPROC Under MVS, this contains the name of the JCL step which executed the PROC which executed NCI/XF.

Under VM, this is not available; always set to *&sysnull*.

&SYSPROD indicates whether NCI/XF is currently running PROD-MODE=YES or NO.

&SYSPRPA contains the PROD-MODE status of the //PANEL dataset which can be YES, NO or DISABLED. When NCI/XF has disabled PROD-MODE operation, the message:

PROD-MODE DISABLED FOR 'PANEL'

will be output to the system console and written to the NCI/XF log dataset.

&SYSPRPB contains the PROD-MODE status of the //PBLIB dataset if a //PBLIB DD has been coded. The status can be YES, NO or DISABLED. When NCI/XF has disabled PROD-MODE operation, the message:

PROD-MODE DISABLED FOR 'PBLIB'

will be output to the system console and written to the NCI/XF log dataset.

&SYSQMSK contains the mask character used when searching NCI/XF queues using the QUEUE FIND statement, specified by the QUEUE-MASK startup option.

&SYSQST contains the current amount of storage, in bytes, in use by NCI/XF for system queues below the 16 mb line. See related variable *&sysxqst*.

&SYSRAND contains a three digit random number.

&SYSREGN contains the region size, in kilobytes, of the address space in which NCI/XF is currently running.

&SYSRLOS name entered on REMOTE-LOSTERM-EXEC= startup option.

&SYSRQPN The RELREQ panel as specified in the RELEASE= startup option. Note that when running MODE=ENHANCED this has a fixed value of *@ncrelrq*.

&SYSRUEC is the total number of VTAM RPL USER FIELD ERRORS detected by NCI/XF. This figure should always be zero; if not, contact your local support centre.

&SYSSCAV the total number of successful scavenges.

&SYSSCPU

MVS	SRB CPU time since startup (in milliseconds).
VM	not available; always set to *&sysnull*.

&SYSSHUT This is either YES or NO depending on whether the system is shutting down or not.

&SYSSINQ the maximum storage in bytes used for asynchronous STATUS commands which have been processed concurrently.

&SYSSMF Under MVS, this contains the SMFID of the system in which NCI/XF is currently running.

Under VM, this contains the virtual machine ID in which NCI/XF is currently running.

&SYSSPL the list of authorized NCI/XF subproducts.

* **&SYSSTDT** the date the system started up.

&SYSSTEP Under MVS, this contains the name of the JCL step that executed NCI/XF.

Under VM, this is not available; always set to *&sysnull*.

&SYSSTOM contains the amount of storage used by NCI/XF modules loaded below the 16 mb line. This does not include storage used to load call routines. This value is not included in the *&sysstor* value. See related variable *&sysxstm*.

&SYSSTOP contains the current panel storage in bytes below the 16 mb line. See related variable *&sysxstp*.

&SYSSTOR The amount of GETMAINed storage, in bytes, that NCI/XF is currently using below the 16 mb line. This value does not include the storage required to load the actual NCI/XF modules (see *&sysstom* and *&sysxstm*), or storage obtained by user call routines for their own use. This value will include ALL (below the 16 mb line) storage that NCI/XF owns. For example it will include Print Buffer (if BUFF=*nnnn* used), screen images for active VTAM sends, and the following NCI/XF control blocks if NCI/XF is running XA-MODE=NO: terminal control blocks (*&sysstot*) and panel storage (*&sysstop*).

&SYSSTOT contains the amount of GETMAINed storage in bytes that NCI/XF has currently allocated for terminal control blocks below the 16 mb line. Terminal control blocks are allocated above the 16 mb line if XA-MODE=YES is specified and VTAM supports 31 bit addressing. See related variable *&sysxstt*.

&SYSSTOW the total storage in bytes saved/wasted (Sn/Wn) by VSM processing.

&SYSSTPN the name of the startup panel.

&SYSSTSA the amount of GETMAINed storage, in bytes, that NCI/XF is currently using for save area storage below the 16 mb line. This amount is included in the *&sysstor* figure.

* **&SYSSTTI** time system started up, specified in HH:MM format.

&SYSSUBA the number of the network subarea in which NCI/XF is currently running.

&SYSSVCI is the total number of GETMAIN/FREEMAIN SVCs issued by VSM (Virtual Storage Manager).

&SYSSVCS is the total number of GETMAIN/FREEMAIN SVCs saved by VSM (Virtual Storage Manager).

&SYSTASK reserved.

&SYSTCPU Under MVS, this contains the TCB CPU time since startup (in milliseconds).

Under VM, this is not available; always set to *&sysnull*.

&SYSTDTN the dataset name pointed to by the TDT DD statement under MVS or the TDT FILEDEF command under VM.

&SYSTIME the current time of day, specified in HH:MM:SS format.

&SYSTINT contains the value, in seconds, of the current update interval, as set by the UPDATE-INTERVAL startup option or the execution of an UPDATE statement. If update processing has been disabled (UPDATE-INTERVAL=NO), this variable will be set to DISABLED. This is forced to 30 if USE is used.

&SYSTQST contains the total storage currently in use by NCI/XF for user queues below the 16 mb line. See related variable *&sysxtqs*.

&SYSTQUE the total number of user queues currently in use by NCI/XF.

&SYSTRAC contains the status of each possible TRACE option. The variable is always set, each character representing the status of MODULE (M), VTAM (V), RPL (R), DATA (D), INTERNAL (I), PROG0 (P), PROG1 (P) and PROG2 (P) options respectively, for example:

XXXXXXXX	all options OFF.
MVRDIPPP	all options ACTIVE.
XVRXXXXX	VTAM and RPL options active.

CKS may extend this variable in the future as more trace options are introduced. Existing options will, however, remain in the same positions.

&SYSTREX contains the current TRACEEXEC mask, used to limit NCI/XF tracing to a subset of exec routines. The value reported will have been set by the TRACE-EXEC startup option or the last TRACEEXEC statement executed.

&SYSTRTR contains the current TRACETERM mask, used to limit NCI/XF tracing to a subset of terminals. The value reported will have been set by the TRACE-TERM startup option or the last TRACETERM statement executed.

&SYSTRVB contains the language statements which are currently being traced by TRACEVERB.

&SYSTSOA Under MVS, if NCI/XF is running authorized, this variable contains the current number of TSO users running in this domain. If NCI/XF is not running authorized, this variable is set to *&sysnull*.

Under VM, this is not available; always set to *&sysnull*.

&SYSTSOM Under MVS, if NCI/XF is running authorized, this variable contains the current maxuser value of the TSO running in this domain. If NCI/XF is not running authorized, this variable is set to *&sysnull*.

Under VM, this is not available; always set to *&sysnull*.

&SYSTUPD the current TERMINAL-UPDATE value.

&SYSTV0 the number of variables currently being traced as a result of the TRACEVAR statement.

&SYSTV1 through &SYSTV99 the names of the variables currently being traced as a result of the TRACEVAR statement.

&SYSTVAR the total number of user variables currently defined to all NCI/XF users.

&SYSUIC is the current UPDATE-INTERVAL count, that is the number of basic timer pops until the next terminal timer pop.

&SYSVERS indicates which release level of NCI/XF is running.

&SYSVSM1 through &SYSVSMU information concerning Virtual Storage Manager (VSM) queues for storage below 16 mb. (Note - it is recommended to use variables &SYSVSX1 - &SYSVSXU which provide more accurate figures). The format is as follows:

0 - 4	size of element
5 - 9	number of elements acquired
10 - 14	number of elements released
15 - 15	S=Saved, W=Wasted
16 - 26	Number of bytes saved or wasted

&SYSVSX1 through &SYSVSXU information concerning Virtual Storage Manager (VSM) queues. These variables contain larger values that may be returned in variables &SYSVSM1 through &SYSVSMU; the format is as follows:

0 - 6	size of element
7 - 13	number of elements acquired
14 - 20	number of elements released

&SYSVTID contains the VTAM version identifier, for example:

VE341	-	VTAM ESA	3.4.1
VE34	-	VTAM ESA	3.4
VE33	-	VTAM ESA	3.3
VX33	-	VTAM XA	3.3
VT33	-	VTAM	3.3
VT32	-	VTAM	3.2
VV33	-	VM VTAM	3.3

&SYSXC00 through &SYSXCFF is a string of 255 bytes with the hex values equivalent to the last two characters of the variable name. For example, SYSXC1C(1) would be X'1C'.

* **&SYSXFCL** name entered on CLOSE-EXEC= startup option.

* **&SYSXFFI** name entered on FINAL-EXEC= startup option.

* **&SYSXFIN** name entered on INITIAL-EXEC= startup option.

* **&SYSXFLG** name entered on LOGON-EXEC= startup option.

* **&SYSXFLO** name entered on LOSTERM-EXEC= startup option.

* **&SYSXFLU** name entered on LUSTAT-EXEC= startup option.

* **&SYSXFOP** name entered on OPEN-EXEC= startup option.

* **&SYSXFPO** name entered on LOSTERM= startup option.

* **&SYSXFPQ** name entered on RELEASE= startup option.

* **&SYSXFPU** name entered on LUSTAT= startup option.

* **&SYSXFRQ** name entered on RELEASE-EXEC= startup option.

* **&SYSXFT1** name entered on TIMER-EXEC= startup option.

* **&SYSXFT2** interval entered on TIMER-EXEC.

* **&SYSXFUP** name entered on UPDATE-EXEC= startup option.

&SYSXQST contains the current amount of storage, in bytes, in use by NCI/XF for system queues above the 16 mb line. See related variable *&sysqst*.

&SYSXSM1 through &SYSXSMU information concerning Virtual Storage Manager (VSM) queues for storage above 16 mb. (Note - it is recommended to use variables &SYSXSX1 - &SYSXSXU which provide more accurate figures). The format is as follows:

0 - 4	size of element
5 - 9	number of elements acquired
10 - 14	number of elements released
15 - 15	S=Saved, W=Wasted
16 - 26	Number of bytes saved or wasted

&SYSXSTA the total storage available for saving above the 16 mb line.

&SYSXSTM contains the amount of storage used by NCI/XF modules loaded above the 16 mb line. This does not include storage used to load call routines. This value is not included in the *&sysstor* value. See related variable *& sysstom*.

&SYSXSTP contains the current panel storage in bytes above the 16 mb line. See the related variable *&sysstop*.

&SYSXSTR The amount of GETMAINed storage, in bytes, that NCI/XF is currently using above the 16 mb line. This value does not include the storage required to load the actual NCI/XF modules (see *&sysstom* and *&sysxstm*), or storage obtained by user call routines for their own use. This value will include ALL (above the 16 mb line) storage that NCI/XF owns. For example it will include Print Buffer (if BUFF=Xnnnn used), and the following NCI/XF control blocks, if allocated above the 16 mb line: terminal control blocks (*&sysxstt*) and panel storage (*&sysxstp*).

&SYSXSTT contains the amount of GETMAINed storage in bytes that NCI/XF has currently allocated for terminal control blocks above the 16 mb line. Terminal control blocks are allocated above the 16 mb line if XA-MODE=YES is specified and VTAM supports 31 bit addressing. See related variable *&sysstot*.

&SYSXSTW is the total storage in bytes saved/wasted (Sn/Wn) by VSM processing above the 16 mb line.

&SYSXSX1 through &SYSXSXU information concerning Virtual Storage Manager (VSM) queues for storage above 16 mb. The format is as follows:

0 - 6	size of element
7 - 13	number of elements acquired
14 - 20	number of elements released

&SYSXTQS contains the total storage currently in use by NCI/XF for user queues above the 16 mb line. See related variable *&systqst*.

&SYSYSM1 through &SYSYSMU contains information concerning Virtual Storage Manager (VSM) queues for storage above the line, relating to larger elements than those catered for by &SYSXSM1 - &SYSXSMU. (Note - it is recommended to use variables &SYSYSX1 - &SYSYSXU which provide more accurate figures).

&SYSYSX1 through &SYSYSXU information concerning Virtual Storage Manager (VSM) queues for storage above 16 mb. The format is as follows:

0 - 6	size of element
7 - 13	number of elements acquired
14 - 20	number of elements released

&SYS0800 count of VTAM buffer shortages found by NCI/XF.

System variables provided by the User Support Environment (USE)

The additional system variables shown below are provided by USE:

*	**&*#SDATE**	system date in format chosen in administration panels, as described by *&*#sdatef*
*	**&*#SDATE1**	system date in date format 1
*	**&*#SDATE2**	system date in date format 2
*	**&*#SDATE3**	system date in date format 3
*	**&*#SDATE4**	system date in date format 3
*	**&*#STIME**	time in HH:MM format
*	**&*#SDATEF**	date format: 1 - MM/DD/YY, 2 - YY/MM/DD, 3 - DD/MM/YY, 4 - DDMMMYY **Note:** the separator character (/) is set in *&*#dsep*
*	**&*#DSEP**	date separator character, replacing (/ in 1, 2 and 3 above
*	**&*#STOUT**	system panel timeout value in minutes **Note:** *&systupd* gives the same value.
*	**&*#PRIME**	primary system - if variable is null then this indicates several applications are being supported by USE
*	**&*#LGOFF**	logoff to VTAM allowed (Y) or not allowed (N)
*	**&*#ICLK**	inhibit clock flag, 0=use clock, 1=inhibit clock
*	**&*#MAXU**	maximum users allowed on system
*	**&*#TLOG**	terminal logon allowed (Y) or not allowed (N)
*	**&*#EXIT07**	if YES, the startup and shutdown exit WEXIT07 is enabled
*	**&*#EXIT09**	if YES, the LOGON post USE logon exit WEXIT09 is enabled
*	**&*#EXIT10**	if YES, the terminal connect exit WEXIT10 is enabled

System variables provided by NC-MHO

The additional system variables shown below are provided by NC-MHO:

&*MH_ACB set to the ACB name for this NC-MHO

&*MH_PAS set to the ACB password (optional)

&*MH_TRA set to TRACE to enable tracing facilities, set to NOTRACE to disable tracing facilities

System variables provided by NC-XMS

The additional system variables shown below are provided by NC-XMS:

&*%XMX01 defines EXEC routine name executed before SET function

&*%XMX02 defines EXEC routine name executed before EXEC function

&*%XMX03 defines EXEC routine name executed before RESOLVE function

&*%XMX04 defines EXEC routine name executed before LOG function

&*%XMSYS contains the name of the last system started

&%XMAUTH set to null if the caller of an authorization exit or XMDIALOG EXEC is not APF authorized; or set to non-null (for instance APF) if a caller is authorized

&%XMSYS contains the name of the current system for this user

&*XMLGL trace level for XMS EXECs

&*XMLGP trace level for program

&*XMLGD *&sysnull* if message level is not being displayed

&*XMLGU *&sysnull* if messages are being displayed in upper case

&*%XMCOR default maximum number of cross memory service work elements

Chapter 3 - NCI/XF defined user variables

NCI/XF reserved user variables 3.2
User variables provided by the USE environment after logon 3.7

Note: The variables marked with an asterisk (*) in the left hand margin of the following pages are set if an (F)CALL routine fails.

Variables starting with *&usr* are reserved.

NCI/XF reserved user variables

NCI/XF automatically maintains these user variables. A unique copy of each variable exists for every terminal connected to NCI/XF; the copy may be denoted but not updated in user written panels.

* **&¢ABENDD** displacement (from start of the failing program) of the failing instruction.

* **&¢ABENDN** name of the failing program.

* **&¢ABENDS** system abend code.

* **&¢ABENDU** user abend code.

&USRAID is the attention identifier for the key last used at the keyboard to enter data. Possible values are

AUTO	AUTOINPUT statement executed
ENTR	enter key
PAK*n*	program attention key
PF*nn*	program function key
SELP	selector light pen or the cursor select key
TCLR	clear key
LU01	means input received from an *lutype1* device (SCS printer, 3767 or NTO device). The received data indicates which key was pressed to initiate the inbound send operation.

&USRAIPC is the number of alternate implicit partition columns. The number of columns will be zero if the terminal is non-queriable or RPQ is pending.

&USRAIPR is the number of alternate implicit partition rows. The number of rows will be zero if the terminal is non-queriable or RPQ is pending.

&USRAPPL the name of the application that a terminal has been connected to using the CONNECT or CONNECTR statement. The application name is placed in variable *&usrappl* on successful conclusion of the QUERYTERM statement (if the terminal has connected to an application).

&USRASCC is the alternate number of columns the screen will support. The value will be zero if USRPSSC=03 and RPQ is pending.

&USRASCR is the alternate number of rows the screen will support. The value will be zero if USRPSSC=03 and RPQ is pending.

&USRAUAC is the number of alternate usable area columns. The number of columns will be zero if the terminal is non-queriable or if RPQ is pending.

&USRAUAR is the number of alternate usable area rows. The number of rows will be zero if the terminal is non-queriable or if RPQ is pending.

&USRCOLR indicates whether the terminal supports highlighting and color screen attributes (note that although the device may claim to support these features, it may not use them). The value is NO if RPQ is pending.

&USRCURC is the column number containing the cursor.

&USRCURN is the name of the variable where the cursor is positioned.

&USRCURR is the row number containing the cursor.

&USRCURS is the data contained in the variable where the cursor was positioned at the time terminal data was sent to NCI/XF.

&USRDATA is the CINIT data that was passed to NCI/XF at connection time. This data is automatically deleted if a CONNECT statement is processed.

&USRDISP is the name of the panel being processed. Useful in a subroutine to determine which panel called the routine.

&USRECNT is the error count from a panel build.

&USRECPU the total CPU time used by a terminal since logon (milliseconds).

&USREX0 the number of parameters entered into an EXEC routine specified by parm1 to parm*n* on an EXEC statement.

&USREX1 through &USREX*n* the contents of the parameters entered into an EXEC routine specified by the parameters parm1 to parm*n* on an EXEC statement.

&USRF0 the number of fixes applied to an NCI PBLIB panel as determined by the CHECKFIX statement.

&USRF1 through &USRF999 the names of the fixes applied to an NCI PBLIB panel as determined by the CHECKFIX statement

&USRGRP the group name associated with a terminal defined in the TDT, (Terminal Definition Table).

&USRINME is the first variable name in a queue of names of variables that were modified on a screen image. The variable name is automatically deleted from the queue when accessed, allowing the next queued name to be accessed.

&USRIVAR the number of input variables modified by a user on a panel before pressing an aid key.

&USRLEXC contains the name of the exec routine that executed the current routine.

For example, if exec routine GENREC has the statement EXEC READIT, then within routine READIT variable *&usrlexc* will have a value of GENREC.

Note: Use of the PANEL statement resets this variable.

&USRLOGL the line from the NCI/XF log buffer, returned from a GETLOG operation.

&USRLOGM is the terminal logmode name that NCI/XF is using for the session. If a terminal tries to connect to NCI/XF with an invalid logmode, NCI/XF will reject the bind and attempt to acquire the terminal with a valid set of bind parameters. In this case *&usrlogm* will be set to:

SNA2	for an SNA terminal (LUTYPE2)
BSC2	for an BSC terminal (LUTYPE0)
SCS2	for an SCS printer (LUTYPE1)
NTO2	for an NTO device
3767	for a 3767 or compatible device.

&USRLOGT The LOGMODE, if any, specified for the terminal in the TDT.

&USRLOGU is the original (user presented) logmode name that was specified as part of the logon to NCI/XF. This will usually be the same as *&usrlogm*. However, in cases where NCI/XF found that either a TDT logmode override should be used or the original logmode was not acceptable and used its own set of bind parameters to acquire the terminal (see *&usrlogm* above), this variable will contain the original logmode name.

&USRLTYP indicates the *lutype* of the terminal, where:

0	=	non-SNA screen
1P	=	SCS printer
2	=	SNA screen
1N	=	NTO device
1T	=	3767 teletype device

&USRMAXE is the current value of MAXEXEC as set by the TURN MAXEXEC statement for the maximum number of EXEC statements that may be processed in a panel section. Set to null if the system-wide default is being used.

&USRMAXG is the current value of MAXGOTO as set by the TURN MAXGOTO statement for the maximum number of GOTO statements that may be processed in a panel section. Set to null if the system-wide default is being used.

&USRMODS The first in a queue of load module statistics built by a MODSTATS statement. The first module statistics in the queue are automatically deleted when referenced, allowing the next module's statistics to be accessed. See *MODStats* on page 1.86.

&USRMSG is the first message in a queue of messages for a terminal. The message is automatically deleted when accessed, allowing the next queued message to be accessed.

&USRMSGN is the number of messages queued for the terminal.

&USRNET identifier of the network where the terminal is defined

Note: only available for terminals which present control vector X'0E' in CINIT data on logon.

&USRNLOG indicates whether the terminal is marked NOLOG in the TDT or whether a TURN statement has been used to alter this (possible values are YES or NO). See *GROUP statement* on page 3.22 of the NCI/XF Installation Manual.

&USRNQUE the number of user queues this user is currently using.

&USRNRES indicates whether the terminal is marked NORES in the TDT or whether a TURN statement has been used to alter this (possible values are YES or NO). See *GROUP statement* on page 3.22 of the Installation Manual.

&USRNSIM indicates whether the terminal is marked NOSIM in the TDT or whether a TURN statement has been used to alter this (possible values are YES or NO). See *GROUP statement* on page 3.22 of the Installation Manual.

&USRNVAR the number of local variables currently defined by this user.

&USRPANL is the name of the panel or routine currently being processed. Within a routine, variable *&usrdisp* gives the name of the PANEL which called the routine.

&USRPANS The first in a queue of panel statistics built by a PANSTATS statement. The first panel statistics in the queue are automatically deleted when referenced, allowing the next panel's statistics to be accessed. See *PANStats* on page 1.98.

&USRPDR panel diagnostics record

&USRPIPC is the number of columns in the primary implicit partition. The number of columns will be zero if the terminal is non-queriable or if RPQ is pending.

&USRPIPR is the number of rows in the primary implicit partition. This will be zero if the terminal is non-queriable or if RPQ is pending.

&USRPM0 the number of *&usrpmn* variables containing tokens derived by a PARSE operation.

&USRPM1 through &USRPM99 the tokens derived by a PARSE operation.

&USRPSCC the PRIMARY number of COLUMNS which the screen will support.

&USRPSCR the PRIMARY number of ROWS which the screen will support.

&USRPSSC is the PSERVIC size code on bind. The possible values are 00,01,02,03,7E or 7F.

&USRQGPO is set by execution of a QUEUE FIND command which specifies a substring search. Indicates the position of the found queue element within the subqueue which comprises all queue elements that contain the specified substring value.

&USRQPOS is set by execution of a QUEUE FIND command to indicate the position in the queue of the located element.

&USRQST contains the total amount of storage currently used by this user for NCI/XF user queues below the 16 mb line. See related variable *&usrxqst*.

&USRQSTR After execution of a QUEUE FIND command, this variable contains a copy of the contents of the located element.

&USRQUER is set to the value YES or NO, indicating whether or not the QUERY bit was set in the logmode used to bind the terminal. This does not necessarily indicate that the terminal supports extended datastreams (see *&usrcolr*).

&USRREFF the REFRESH status entered in a TURN statement. Possible values are YES, NO or DELAY.

&USRREST the name of the panel which will be displayed by a RESTART operation, defined by the PANEL statement in the TDT entry for the terminal.

&USRRET the name of the panel that was displayed before the current one (the panel which will be displayed if a RETURN statement is issued in the current panel).

&USRRETC the return code from the last command to set a return code.

&USRRPQP This is either YES or NO depending on whether a Read Partition Query (RPQ) is pending or not. An RPQ will be pending for a queriable device only after Power On or Logon to NCI/XF, and before any screen is sent. If data from the RPQ is needed before the first screen can be sent, the RPQ can be forced by sending a blank screen using AUTOINPUT.

&USRRTCD the value of the VTAM RPLRTNCD field associated with the last CLSDST PASS function, for instance, set to 144B if an invalid logmode was specified.

&USRSECT indicates which panel section is currently being processed: INIT, UPDATE, or PROC, which is useful in EXEC routines.

&USRSENS the SENSE Code associated with the last CLSDST PASS, for instance, set to 0801, if the terminal is not authorized to use the application.

&USRSUBA the subarea number of a logical unit logging on to NCI/XF. This is a four byte hexadecimal field. (For example, if VTAM passes the value X'0000001C in the Control Initiate Request Unit, the variable will be set to 28). If the value cannot be determined it will be set to zero.

&USRTASK reserved.

&USRTDT reserved.

&USRTERM the *luname* for the terminal.

&USRTRMS The first in a queue of terminal statistics built by a TRMSTATS statement. The first terminal statistics in the queue are automatically deleted when referenced, allowing the next terminal's statistics to be accessed. See *TRMStats* on page 1.190.

&USRUAC is the number of columns in the usable area. The number of columns will be zero if the terminal is non-queriable or if RPQ is pending.

&USRUAR is the number of rows in the usable area. The number of rows will be zero if the terminal is non-queriable or if RPQ is pending.

&USRUPPR the status of the UPPER operand specified in the TURN statement or on a TDT group statement. Possible values are ON or OFF.

&USRVPOS This is set by the VERIFY verb to give the position of the first invalid character.

&USRXQST contains the total amount of storage currently used by this user for NCI/XF user queues above the 16 mb line. See related variable *&usrqst*.

User variables provided by the USE environment after logon

&#USERID	*userid* padded with blanks to eight characters long
&#USER	*userid* not padded
&#LUNAME	terminal *luname*, normally the same as *&usrterm* at logon
&#LANG	the preferred language of the user, E=English, F=French, D=German
&#TTYPE	the terminal model, 1 through 5
&#PRIME	the user's primary product
&#UDATA	the user data field used by exits
&#SECUR	password validation, N, I, R, A, S, T
&#PASSWD	user's password
&#CAAUTH	user's authority type
&#UGRP	user's authority group
&#LGOFF	1=logoff to logo, 2=logoff to VTAM, subject to the setting of *&*#lgoff*
&#CAFLG	Y=user is authorized for admin, authority type is A or O

This page intentionally left blank

Chapter 4 - NC-VSAM routines

VSLOGON 4.2
Format 4.2
IOOPEN 4.3
KSDS files 4.3
Format 4.3
Variables 4.3
Return codes 4.4
IOREAD 4.5
Format 4.5
Selecting records 4.6
Returning data 4.11
Data conversion 4.12
Segmented reads 4.13
Variables 4.14
Return codes 4.16
IOADD 4.17
Format 4.17
Variables 4.18
Return codes 4.20
IOUPDATE 4.21
Format 4.22
Variables 4.22
Return codes 4.23
IODELETE 4.24
Format 4.24
ESDS deletion of records 4.24
Variables 4.25
Return codes 4.26
IOFIXES 4.27
Format 4.27
Return Codes 4.28
IOCLOSE 4.29
Format 4.29
Variables 4.29
Return codes 4.29
IOSTATS 4.30
Format 4.30
Variables 4.30
Return codes 4.31
IORESET 4.32
Format 4.32

This chapter should be read in conjunction with the NCI/XF User Guide, *Chapter 5 - A guide to NC-VSAM*.

VSLOGON

The VSLOGON routine provides NC-VSAM administration facilities. It opens the NC-VSAM control file (if it is not already opened) sets up the messages used by NC-VSAM and provides the NC-VSAM MAINTENANCE MENU.

Format

EXEC VSLOGON *parm1 parm2*

where

parm1 is an optional parameter to define the userid to be used throughout NC-VSAM. If this parameter exceeds eight characters, the remaining characters will be ignored. Specify *&sysnull* if this parameter is not required when the following parameter (*parm2*) is specified.

parm2 is an optional parameter to override the system defaults used for printing NC-VSAM records; it consists of *destination id*, *printer type*, *lines per page*, *copies*, *line length*, *class* and *form*.

For a description of each of these fields see *NC-VSAM print option* on page 5.17. The printer options are kept permanently in routine VSPRTDF and may be amended as required.

IOOPEN

The IOOPEN routine, opens a file for processing and:

- retrieves the file definition from the NC-VSAM control file
- checks if the file is already open. If already open for the same type of processing (input/output or input only), IOOPEN returns control to the caller with a return code of zero; if already open for a different type of processing the return code will be 97.
- if the file definition specifies dynamic allocation, IOOPEN dynamically allocates the file using the dataset name specified in the file definition
- opens the file for either input/output or input only, depending on the entry variable *&?input*.

Before executing IOOPEN, you must:

- define the file to NC-VSAM using the Define a physical file option on the NC-VSAM MAINTENANCE menu. This process involves assigning an internal file name, up to thirty characters in length, by which all NC-VSAM functions will refer to the file. This makes all NC-VSAM I/O requests independent of operating system file names.
- set variable *&¢file* to contain the NC-VSAM name of the file to be opened

KSDS files

NC-VSAM requires that a KSDS be preloaded (that is, it should not be empty). You should IDCAMS REPRO at least one record into the KSDS file before issuing an NC-VSAM OPEN.

We would recommend REPROing a record with binary zeros in the VSAM key to reduce the likelihood of a CI split.

Issuing IOREAD for an empty KSDS gives:

&¢retc=36

&?reg15=8

&?fdbk=116

Format

```
SET     &¢file = nc-vsam-file-name
EXEC    IOOPEN
```

Variables

Input and exit variables are provided with IOOPEN as described below.

Entry variables

Before executing IOOPEN, the following variable must be set:

&¢file the NC-VSAM internal name for the physical file

Optional entry variables

In addition to the mandatory variable(s), the following optional variables can be set prior to executing IOOPEN:

&?input specifying a value of Y will result in the file being opened for input processing only. If *&?input* is not set or is set to a value other than Y the file is opened for I/O processing.

WARNING: Be sure to reset *&?input* before using IOOPEN to open the next file that requires output processing.

&?reuse specifying a value of Y will result in the file being reset at open time (the high used RBA is set to zero).

This option is only valid for ESDS files, which must have been defined to VSAM with the REUSE option.

WARNING: Be sure to reset *&?reuse* before using IOOPEN to open the next file that does not require this option. Failure to do so may result in the next dataset opened being reused.

Exit variables

On completion of IOOPEN, the following variable will be returned

&¢retc return code, as described below

Return codes

On completion of IOOPEN, one of the following return codes may be set in variable *&¢retc*:

0 File has been successfully opened or the file was already open.

4 The NC-VSAM control file is not available, or reserved variable *&syscntl* has been modified or reset by the user.

12 NC-VSAM has abnormally terminated. The termination code is available in variable *&¢abendc*.

97 File already open with different input/output specification. For example, an attempt has been made to open a file for output that is already open for input only (or vice versa).

98 No file name was supplied in *&¢file* or the NC-VSAM control file does not contain a file definition for the named file.

99 NC-VSAM could not open the file. The reason is contained within a message written to the NCI/XF log. The message may include, if applicable, the VSAM open return or error code which is described in the appropriate IBM documentation. The possible reasons for an open failure include

- No DD statement or DLBL command which corresponds to the DD name specified in the file definition was coded.
- I/O contention or hardware reserve on the dataset or device (NC-VSAM will wait up to 3 minutes for the OPEN request to succeed. If it still can not get to the dataset, NC-VSAM will abandon the OPEN request and detach the IOVSAM subtask.)

IOREAD

The IOREAD routine allows you to read from 1 through 9999 records from a VSAM KSDS or ESDS in a single execution.

IOREAD

- retrieves the record definition from the NC-VSAM control file, if it is not already in storage.
- selects from 1 through 9999 records based on your specified selection criteria. Note that NC-VSAM can select based on any field(s) within a record, whether or not they occur within the VSAM key.
- returns the values of selected fields from each record into a set of NCI/XF variables.

After reading records via IOREAD, examine *&?rcdsrd* to determine if the records were successfully read. If it equals zero, then either you are at EOF or some other error occurred; examine *&¢retc*, *&?reg15* or *&?fdbk*.

Format

SET	&¢file = *filename*	
SET	&¢record = *recordname*	
SET	&¢selct1 = *record selection criteria*	(optional)
SET	&¢fields = *fields to return*	(optional)
SET	&¢resume = *start key*	(optional)
SET	&¢num = *number*	(optional)
EXEC	IOREAD	

Sequential reading through a file can use the *&¢resume* variable. On an IOREAD set *&¢resume* to 'KEY,GT,' and suffixed by the VSAM key returned on the previous read. More simply, this can be replaced by IOREAD NEXT which uses NC-VSAM's internal pointers and returns the next sequential record irrespective of key (unless this is the first read of the file where these variables will be used). Compare the following

without IOREAD NEXT	***with*** IOREAD NEXT
`Label READRCD`	`Label READRCD`
`Set &¢selct1 = 'USERID,EQ,*'`	`Exec IOREAD NEXT`
`Exec IOREAD`	`If &?rcdsrd ne 0`
`If &?rcdsrd ne 0`	`  goto READRCD`
`  set &¢resume = 'KEY,GT,&?endkey'`	
`  goto READRCD`	

Selecting records

NC-VSAM allows you to select records on the basis of any data within them. *&¢selctn* is optional, and if it is omitted, all records will be selected. You can select records for reading on the basis of:

- a field or fields within the VSAM key
- data in a field or fields not in the VSAM key
- the partial contents of a field or fields
- generic selection within a field or fields
- any combination of the above

Selection criteria can be specified on a single read operation to obtain exactly the records you require, without the overhead of maintaining multiple alternate indexes.

Specifying *&¢selctn*

Your selection criteria are passed to NC-VSAM by setting one or more of the variables *&¢selct1* through *&¢selct5* before executing the IOREAD routine.

SET &¢selctn = '*fldname*,*oper*,*data*,*fldname*,*oper*,*data*.....'

where

fldname is the field name as defined in the NC-VSAM record format definition.

oper is the operator used to compare the data with the record contents. Supported values are

EQ	field contents equal data
NE	field contents not equal to data
GT	field contents greater than data
GE	field contents greater than or equal to data
LT	field contents less than data
LE	field contents less than or equal to data

If you want to select a field within the VSAM key, please note that a field cannot have the operator LT, LE or NE when it is first denoted in the search string. For instance, if all userids less than W are required and the userid occurs in the VSAM key, then for *&¢selctn* to successfully execute IOREAD you should code:

USERID,EQ,*,USERID,LT,W

because the IBM VSAM access method only performs direct key searches on an inclusive basis. VSAM must be told initially what records are required, for example, USERID,EQ,*. The exclusion mechanism (USERID,LT,W) is an NC-VSAM technique to filter out unwanted records.

data is the comparison data or an NCI/XF variable set to the comparison data. Generic and masked selection data are supported to allow record selection based on partial contents of a character field, field type c. Specify a stub with a trailing asterisk (*) to select only on characters before the asterisk. A plus (+) character causes NC-VSAM to ignore a character position when selecting records.

Note: If you code NCI/XF variables in the *&¢selctn* variable when reading records, ensure each variable is delimited by a period (.). A comma (,) is **not** a valid NCI/XF variable delimiter. For example:

SET &¢selct1 = 'INITIAL , EQ , &initl . , SURNAME , EQ , &surname'

The period after *&initl* is necessary to delimit the variable.

The period is not necessary if you code non-varying select data as follows:

SET &¢selct1 = 'INITIAL , EQ , J , SURNAME , EQ , SMITH'

Selection criteria

The following examples show simple selection based on a single criterion. Descriptions and examples of selection criteria are provided on subsequent pages.

Single criterion examples

Read the first or only record where the CUSTOMER field contains the characters ABCD.

```
SET &¢selct1 = 'CUSTCODE,EQ,ABCD'
SET &¢num = 1
```

Note: If the field CUSTCODE is defined as being more than four characters in length, NC-VSAM will pad the selection data with nulls, to the field length, before starting the selection process. If the file records have trailing spaces, the selection data should be right padded with blanks before executing IOREAD, as illustrated in the next example.

Read the first or only record where the customer field matches the customer code typed via an NCI/XF panel into variable *&cust*. The field CUSTCODE is eight characters long and the data is stored with trailing blanks.

```
PADR &cust 8
SET &?selct1 = 'CUSTCODE,EQ,&cust'
SET &?num = 1
```

Read up to twenty records where the customer field matches the customer code typed via an NCI/XF panel into variable *&cust*. The field CUSTCODE is eight characters long and the data is stored with trailing blanks.

```
PADR &cust 8
SET &¢selct1 = 'CUSTCODE,EQ,&cust'
SET &?num = 20
```

Read up to twenty records where the first three characters of the userid field are XYZ.

```
SET &¢selct1 = 'USERID,EQ,XYZ*'
SET &?num = 20
```

Read up to twelve records where the first character of the USERID field is a and the third character is C.

```
SET &?selct1 = 'USERID,EQ,A+C*'
SET &¢num = 12
```

Read the first or only record having flag bit field FLAG1 set on.

```
SET &¢selct1 = 'FLAG1,EQ,1'
SET &¢num = 1
```

Read up to twelve records where the amount owed exceeds ¢1000.

```
SET &?selct1 = 'AMOUNT,GT,1000'
SET &?num = 12
```

AND and OR: selection criteria

NC-VSAM allows the specification of more than one selection criterion to select records using AND and OR conditions. As with all NC-VSAM record selection, this selection can be applied to key or non-key fields.

Selection criteria can be specified within a single *&¢selctn* variable to select records where only records meeting all criteria will be selected; that is, the AND condition evaluates to true for those criteria. For instance, select all invoice records for customer X where the amount outstanding exceeds Y. The examples below illustrate this technique.

Any number of selection criteria can be specified provided the total length of the *&¢selctn* variable does not exceed 32766 bytes. If the total length exceeds that which can be typed within the line length of an NCI/XF language statement, the variable can be concatenated with itself on a succeeding statement.

Examples of AND

Read the first or only record where the customer code is ABCD and the amount outstanding is greater than or equal to $50.

```
SET &¢selct1 = 'CUSTCODE,EQ,ABCD,AMOUNT,GE,50'
SET &?num = 1
```

Read the first or only record where the customer name is WIDGET CORPORATION and the amount outstanding is greater than or equal to $1,000,000.

```
SET &¢cust = 'WIDGET CORPORATION'
PADR &¢cust 30
SET &?selct1 = 'CUSTOMER,EQ,&¢cust,'
SET &?selct1 = '&?selct1.AMOUNT,GE,1000000'
SET &¢num = 1
```

Read up to twenty four records for customers with credit rating of A who owe more than $5000.

```
SET &¢selct1 = 'CREDIT,EQ,A,AMOUNT,GT,5000'
SET &¢num = 24
```

Read up to twenty four records for any customer who owes more than $5000.

```
SET &?selct1 = 'AMOUNT,GT,5000'
SET &¢num = 24
```

Read up to ten records where USERID starts with ABC, application is IMS and the LUNAME ends with 2.

```
SET &¢selct1 = 'USERID,EQ,ABC*,APPL,EQ,IMS,'
SET &¢selct1 = '&¢selct1.LUNAME,EQ,+++++++2'
SET &¢num = 10
```

OR

Several *&¢selctn* variables may be used to specify alternate criteria for record selection. In this case a record will be selected and returned to you if any - possibly but not necessarily all - of the individual *&¢selctn* strings indicate that the record should be selected. The OR condition applies here.

Any or all of the *&¢selctn* variables can themselves contain selection criteria as described on the previous pages. When searching for records to select, NC-VSAM applies each *&¢selctn* string in turn against each record. If all conditions in any *&¢selctn* string indicate that the record is to be selected, then the record is returned to you.

NC-VSAM supports up to five sets of OR conditions, using variables *&¢selct1* to *&¢selct5*. Each of these five OR conditions can themselves be up to 32766 bytes long, specifying multiple AND type selection criteria.

Examples of OR

Read up to 20 records for all customers with a credit rating of A or B.

```
SET &¢selct1 = 'RATING,EQ,A'
SET &¢selct2 = 'RATING,EQ,B'
SET &¢num = 20
```

Read the first record

where

application is IMST and date is before 1st Jan 94,

or

application is IMSP and date is after 1st Jan 94

```
SET &¢date = '940101'
SET &?selct1 = 'APPL,EQ,IMST,DATE,LT,&¢date'
SET &¢selct2 = 'APPL,EQ,IMSP,DATE,GE,&¢date'
SET &?num = 1
```

Read up to twenty records with a USERID, a field which resides in the VSAM key, that does not start with W.

```
SET &¢selct1 = 'USERID,EQ,*,USERID,NE,W*'
SET &?num = 20
```

Returning data

Having located and read one or more records which match your selection criteria, as described on the previous pages, NC-VSAM allows you to specify, on a field by field basis, the data from the selected records to be returned to your NCI/XF panel. Before executing the IOREAD, you should set the variable *&¢fields* to indicate which data fields you wish to be extracted from the selected records and the names of the NCI/XF variables into which the data is to be put. To reduce the length of the *&¢fields* variable you should consider grouping TYPE=C (CHR) fields together and reading a group of fields as one field into one variable. Specification of *&¢fields* is optional. You may, for instance, wish merely to check for the presence of a record, in which case *&¢fields* need not be specified. The record will be read, *&?rcdsrd* will be set and the record will then be automatically discarded by NC-VSAM.

Format

SET &¢fields = '*fldname*,*varname*,*fldname*,*varname*,...'

where

fldname is the field name as defined in the NC-VSAM record format definition.

varname is the name of the NCI/XF variable name where the contents of *fldname* will be stored. The variable must be specified without the leading ampersand (&).

When reading multiple records in a single execution of IOREAD, *varname* should be suffixed with sufficient asterisks (*) (c'*',X'5C') to contain the highest possible record number. NC-VSAM will replace the asterisks with a numeric suffix, right justified with leading zeros, indicating the relative position of the records read (1 for the first record selected, 2 for the second etc). The fields returned from a multiple read may therefore be regarded as returned in tables of variables indexed by a numeric suffix. So:

```
Set &?fields = 'ADDRESS,addr**'
```

reads up to 99 records which return the values in variables *&addr01* to *&addr99*.

You must specify enough asterisks (*) to contain the highest possible record number. For example, if *&¢num* = 5, *varname* must contain at least one trailing asterisk; if *&¢num* = 100, *varname* must contain at least three asterisks. If this is not specified correctly the results are indeterminate.

On a multiple read, the numeric suffix will always be right justified with leading zeros up to the number of asterisks.

You may specify as many *fldname*,*varname* pairs as are required to determine the fields you wish returned. The variable *&¢fields* must be an NCI/XF variable and cannot exceed 32766 characters.

Data conversion

NC-VSAM allows data fields within records to be held as binary numbers, packed decimal numbers or flag bits. When data is read from such fields, it is converted into a character format suitable for processing NCI/XF panels.

- Binary fields, field type B, are converted to character decimal with leading zeros removed.
- Packed fields, field type P, are converted to character decimal with leading zeros removed. A leading minus sign is added when appropriate.
- Flag fields, bit fields, field type F, are converted to character decimal. 0 indicates the flag is off, 1 indicates the flag is on.

Examples

Read the customer master file record for customer WIDGET CORP and return the customer contact name, the contact phone number and the customer's credit limit.

```
SET &¢cust = 'WIDGET CORP'
SET &?selct1 = 'CUSTNAME,EQ,&¢cust'
SET &?fields = 'NAME,name,PHONE,num,CRLIM,lim'
SET &?num = 1
EXEC IOREAD
```

Assuming the record exists (*&?rcdsrd* = 1)

&name	contains the contact name (from field NAME).
&num	contains the phone number (from field PHONE).
&lim	contains the credit limit (from field CRLIM).

If the record could not be found (*&?rcdsrd* = 0), the variables remain unchanged.

Select the first ten customers whose outstanding balance exceeds $1,000 and return company name, credit limit and the balance outstanding.

```
SET &¢selct1 = 'AMOUNT,GT,1000'
SET &¢fields = 'CUSTNAME,cust**,AMOUNT,amt**,'
SET &¢fields = '&¢fields.CRLIM,limit**'
SET &¢num = 10
EXEC IOREAD
```

After execution, *&?rcdsrd* contains the number of records read (from 0 to 10) which matched the selection criteria. The variables are set as follows:

&cust01	contains the first customer name.
&amt01	contains the first customer's outstanding balance (for instance -847).
&limit01	contains the first customer's credit limit.
&cust02	contains the second customer name.
&amt02	contains the second customer's outstanding balance.
&limit02	contains the second customer's credit limit.

If, for example, only three matching records were located, variables suffixed 04 to 10 would remain unchanged. If no matching records were located, all variables (01 to 10) would remain unchanged.

Segmented reads

Segmented reads enable you to read a number of records under particular selection criteria, process the results and read further records with the same criteria, but starting where you left off. A frequent application of this technique in screen based applications is the processing of lists. Enough records are read to fill the first page or screen. If the user then requests further records a further read is done to fill the next screen page. This avoids holding large numbers of records in storage. *&?endkey* and *&?1stkey* are passed to IOREAD at a subsequent execution to reposition NC-VSAM within the file before starting the next read/select operation.

After each successful execution of IOREAD, variable *&?endkey* contains either the full VSAM key of the last record returned for a KSDS file, or the RBA of the last record returned for an ESDS file.

After each successful execution of IOREAD, variable *&?1stkey* contains the RBA of the first record returned if an ESDS file is being processed.

Format

SET &¢resume = KEY,operator,&?endkey

where

KEY must be coded exactly as shown.

operator must be one of the following:

EQ	The last record is to be reread as part of the current read operation (only valid for ESDS files).
GE	The last record is to be reread as part of the current read operation (only valid for KSDS files).
GT	Reading starts with the record after the last record.

&?endkey must be coded exactly as shown.

Example

Read and process all customer records in groups of ten.

```
RESET &¢resume
SET   &¢selct1 = 'CUSTNAME,EQ,*'
SET   &¢fields = 'NAME,name**,PHONE,num**,CRLIM,lim**'
SET   &¢num = 10
LABEL READLOOP
EXEC  IOREAD

..... logic to process up to 10 records .....

SET   &¢resume = 'KEY,GT,&?endkey'
GOTO  READLOOP
```

Variables

Input and exit variables are provided with IOREAD as described below.

Entry variables

Before executing IOREAD, the following two variables must be set

&¢file the NC-VSAM internal name for the physical file

&¢record the NC-VSAM internal name for the record type

If either variable is omitted, NC-VSAM assumes they are the same.

Optional entry variables

In addition to the two mandatory variables, the following optional variables may be set before executing IOREAD

&¢selct1 contains a selection criterion defining which record(s) are to be read. Up to five selection criteria may be specified in variables *&¢selct1* to *&¢selct5*.

&¢num is the maximum number of records to be returned from this execution of IOREAD. If this variable is not set on entry, a default of 1 is assumed. Maximum value is 9999.

&¢fields specifies the fields from the selected records to be returned and the names of the NCI/XF variables where the data is to be stored.

&¢resume specifies the VSAM key of the record from which reading is to restart, for KSDS segmented reads.

For ESDS segmented reads, it specifies the relative byte address (RBA) of the record from which reading is to start.

Exit variables

On completion of IOREAD, the following values may be returned

&¢retc is a return code. The meaning of each return code is described in the subsection entitled Return codes after this list of variables.

&?rcdsrd is the number of records returned from the read operation. This may be equal to or less than the number actually requested in *&¢num*.

&?endkey After a successful read operation on a KSDS, this variable contains the full VSAM key of the last (or only) record read. After a successful read operation on a ESDS, this variable contains the RBA of the last, or only, record read. This is particularly useful when doing segmented reads, for instance during list processing.

&?1stkey After a successful read operation on an ESDS, this variable contains the RBA of the first record read. If only one record is read this will be the same as the *&?endkey* variable.

&?reg15 If the I/O operation failed because of a VSAM error (*&¢retc*=36), this variable contains the contents of general register 15 returned from the appropriate VSAM macro. These codes are documented in the appropriate IBM documentation.

&?fdbk If the I/O operation failed because of a VSAM error (*&¢retc*=36), this variable contains the feedback code returned by VSAM. These codes are documented in the appropriate IBM documentation.

&?offset If the I/O operation failed because of a syntax or validation error in the *&selctn* or *&¢fields* parameters, (*&¢retc*=20 to 33), this variable indicates where the error occurred in the string. Depending on the actual error, *&?offset* contains a number which indicates either the offset into the variable name or a field name.

In general, numbers indicate invalid field names (for instance fields not present in record definition); variable names indicate that the data associated with the field name is invalid (for instance specifying nine bytes of data as part of the selection criteria when the field is defined as only eight bytes long); and field names may also indicate invalid data extracted from the VSAM record.

&¢abendc If the I/O operation terminated because of an abend in an NC-VSAM module (*&¢retc*=12), this variable contains the abend code.

&¢ioemsg is an error message, suitable for display on a terminal, that contains the error information from the above variables.

&¢rln A series of variables, where *n* is the number of the record returned, containing the length of the record. These variables are only returned when reading an ESDS file.

Return codes

On completion of IOREAD, one of the following return codes may be set in variable *&¢retc*

0 read operation completed without errors.

Note: Failure to read the requested number of records is not an error and *&¢retc* will be set to 0. To determine the success of your read you are advised to test *&?rcdsrd*, which indicates the number of records found which matched the search criteria.

4 the NC-VSAM control file is not available, or reserved variable *&syscntl* has been modified or reset by the user.

5 the record name passed in *&¢record* cannot be found in storage or in the NC-VSAM control file.

6 indicates an internal error. Inform your local support office.

7 indicates an internal error. Inform your local support office.

8 the number of records to read, passed in *&¢num*, is invalid. Specify only a numeric value from 1 through 9999, or ensure *&¢num* is reset before executing IOREAD.

9 *&¢fields* is invalid.

10 *&¢resume*, the restart point for a segmented read, is invalid.

12 NC-VSAM has abnormally terminated. The termination code is available in variable *&¢abendc*.

20 indicates invalid field name in select string. *&?offset* points at the operand in error.

24 indicates invalid operator in select string. *&?offset* points at the operand in error.

28 signifies invalid comparate in select string. *&?offset* points at the operand in error.

31 signifies that a VSAM packed data field (type=P) did not contain valid packed decimal data. *&?offset* is set to the appropriate VSAM field name.

32 signifies invalid *&¢fields* parameter. *&?offset* points at the operand in error.

33 *&¢fields* parameter refers to a field defined as an automatic fixed field. *&?offset* points at the operand in error.

36 a VSAM access method I/O error occurred. *&?reg15* and *&?fdbk* indicate the VSAM return and feedback codes.

The most common errors on a read request are:

- *&?reg15* = 8 and *&?fdbk* = 4, indicating that the end of the file has been reached.
- *&?reg15* = 8 and *&?fdbk* = 16, indicating that the record cannot be found.

37 signifies that a VSAM packed decimal field (type=P) did not contain valid packed decimal data on an IOREADR function.

WARNING: Variable *&?offset* is not set.

76 the file specified in *&¢file* is not open.

84 the *&¢fields* variable exceeds 32766 bytes in length.

IOADD

The IOADD routine allows you to add a record to a VSAM KSDS or ESDS. Data is supplied on a field by field basis. Any fields for which data is not supplied, with the exception of mandatory add fields, are written as nulls, binary zeros, X'00'. Records are added with a length set to the end of the field which has the highest end position as defined in the NC-VSAM record definition.

If you want to store copies of records - before or after the add request - in journal or log files, note that record contents may be returned in variables *&?bircd* and *&?aircd* (before and after images respectively).

IOADD

- checks that data has been supplied for all fields designated as mandatory add.
- checks all supplied data for correct data type.
- checks all supplied data for correct length. Data longer than the defined field length will be rejected and data shorter than the defined field length is padded on the right with nulls, binary zeros, X'00'. If you require alternative padding, use the NCI/XF PAD language statements before executing IOADD.
- adds the record to the file. The length of the record written is determined from the NC-VSAM record definition. The offset plus the length of each defined field is examined. The highest value obtained is the implied record length.

Format

SET &¢file = *filename*

SET &¢record = *recordname*

SET &¢fields = '*fname*,*varname*,*fname*,*varname*,...'

EXEC IOADD

Example

Ask the user for customer details and add a record to the file.

In the interests of clarity, attribute character specification, validation and error checking have been omitted.

```
)INIT
Reset &cust &name &phone &crlim
)BODY
    CREATE CUSTOMER MASTER RECORD
        Customer name => &cust
       Contact person => &name
     Telephone number => &phone
         Credit limit => &crlim

     Press <Enter> to add record

&¢emsg
)PROC
Padr &cust 30
Padr &name 30
Padr &phone 20
Padl &crlim 6 ‘0’
Set &?file       = ‘CUSTOMER.ORDER FILE’
Set &?record     = ‘CUSTOMER.MASTER.RECORD’
Set &¢fields = ‘CUSTNAME,cust,CONTACT,name,PHONE,phone,’
Set &¢fields = ‘&¢fields.CRLIM,crlim’
Exec IOADD
Set &¢emsg = ‘Record successfully added for &cust’
If &¢retc ne 0
 set &¢emsg = &¢ioemsg
Panel &usrdisp
)END
```

Variables

Input and exit variables are provided with IOADD as described below.

Entry variables

Before executing IOADD, the following variables must be set

&¢file the NC-VSAM internal name for the physical file

&¢record the NC-VSAM internal name for the record type

&¢fields the name of the field (*fname*), as defined in the NC-VSAM record definition, into which the data is to be put and the NCI/XF variable which contains the data to be added (*varname*)

Exit variables

&¢retc return code. (The meaning of each return code is described in the section entitled *Return codes* on the next page.)

&?reg15 If the I/O operation failed because of a VSAM error (*&¢retc*=36), this variable contains the contents of general register 15 returned from the appropriate VSAM macro. These codes are documented in the appropriate IBM documentation.

&?fdbk If the I/O operation failed because of a VSAM error (*&¢retc*=36), this variable contains the feedback code returned by VSAM. These codes are documented in the appropriate IBM documentation.

The most common error occurring on an ADD operation is:

&?reg15 = 8 and *&?fdbk* = 8

indicating that a record already exists with the specified VSAM key.

&?bircd is set to the contents of the record added, if *&?bimage* is y or Y.

&?aircd contains after image of record (so contains the same information as *&?bircd*) if *&?aimage* is y or Y.

&?offset If the I/O operation failed because of a syntax or validation error in the *&¢fields* parameter (*&¢retc* = 20 to 33), this variable indicates where the error occurred in the string. Depending on the actual error, *&?offset* may contain a number which indicates the offset into the variable or a field name. In general, numbers indicate invalid field names (for instance: field not present in record definition) and field names may signify that the data associated with the field name is invalid (for instance: specifying 9 bytes of data when the field is defined only 8 bytes long).

&¢abendc If the I/O operation terminated because of an abend in an NC-VSAM module (*&¢retc* = 12), this variable contains the abend code.

&¢ioemsg an error message (suitable for display on a terminal) containing the error information from the above variables.

&?lwdlf the inserted record contains an LWD field (field type=L), the contents of the 5 bytes of data written to this are returned in variable *&?lwd*.

&?1strba ESDS file only, the RBA of the record written to the file.

Return codes

0 add operation completed without errors.

4 The NC-VSAM control file is not available, or reserved variable *&syscntl* has been modified or reset by the user.

5 The record name passed in *&¢record* cannot be found in storage or in the NC-VSAM control file.

6 indicates internal error. Inform your local support office.

7 indicates internal error. Inform your local support office.

9 *&¢fields* is invalid or contains *&sysnull*.

12 NC-VSAM has abnormally terminated. The termination code is available in variable *&¢abendc*.

32 Invalid *&¢fields* parameter. *&?offset* points at the operand in error.

33 *&¢fields* parameter refers to a field defined as an automatic fixed field. *&?offset* points at the operand in error.

36 A VSAM I/O error occurred. *&?reg15* and *&?fdbk* indicate the VSAM return and feedback codes.

52 The file is open for input only; an add operation is not permitted.

64 No data has been supplied for a field defined as mandatory add. *&?offset* contains the name of the field in error.

76 The file specified in *&¢file* is not open.

84 The *&¢fields* variable exceeds 32766 bytes in length.

IOUPDATE

The IOUPDATE procedure updates a record with data supplied in variables named in the *&¢fields* parameter. Updating is handled on a field by field basis. To update a record, it is necessary to supply data for fields which are to be changed. For a KSDS file it is only necessary to specify sufficient fields in the VSAM key to uniquely identify the record. Only fields specified in *&¢fields* will be updated. The updated data fields are merged into the existing record before the record is rewritten. Fields with FLDTYPE=CHR which are too short will be padded right with trailing binary zeros. Owing to VSAM record format restrictions it is not possible to use the normal update procedure to change data in the VSAM key. If you want to change a key field, the record must be deleted using IODELETE and a new record added with IOADD.

For an ESDS file the Relative Byte Address (RBA) of the record must be supplied via the variable *&?updrba*. The length of the record can not be changed. Only files specified in *&¢fields* will be updated. The updated data fields are merged into the existing record before the record is rewritten. Fields with FLDTYPE=CHR which are too short will be padded right with trailing binary zeros.

It is possible to redefine or overlay fields with record format definitions. For example a 2 byte binary field could be returned by NC-VSAM as a variable in either 2 byte binary or EBCDIC. The following record formats would apply

```
FLDA    FLDTYPE=CHR , FLDLEN=2 , FLDOFF=4

FLDB    FLDTYPE=BIN , FLDLEN=2 , FLDOFF=4
```

The fields variable would be set

```
SET &¢fields = 'FLDA , a , FLDB , b'
```

NC-VSAM would return

```
&a=X '0100'

&b=C '256'
```

If you want to store copies of records - before or after the update request - in journal or log files, note that record contents may be returned in variables *&?bircd* and *&?aircd* (before and after images respectively).

IOUPDATE

- validates all supplied data for correct data type
- validates all supplied data for correct length
- updates the record on the file by merging fields supplied in the *&¢fields* parameter with the existing data

In addition

- data longer than the defined field length will be rejected
- data shorter than the defined field length is padded on the right with nulls, binary zeros, X'00'. If you require alternative padding, use the NCI/XF PAD language statements before executing IOADD to pad the data as required.
- if update write integrity was specified by including an LWD field (type L) in the record definition, NC-VSAM reads the LWD field from the record and compares it with the value passed in *&¢fields*. If a mismatch occurs, the record has been updated by another user since you read it and the update operation is terminated with return code 72.

Format

```
SET &¢file = filename
SET &¢record = recordname
SET &¢fields = 'fname,varname,fname,varname,...'
EXEC  IOUPDATE
```

Variables

Input and exit variables are provided with IOUPDATE as described below.

Entry variables

&¢file the NC-VSAM internal name for the physical file

&¢record the NC-VSAM internal name for the record type

&¢fields specifies the fields to be updated (*fname*) and the name of the NCI/XF variables containing the data to be stored in the fields (*varname*). If a type L, update write integrity field is present in the record definition, *&¢fields* must specify this field. The associated variable must contain the value obtained from a previous read of the same record.

&?updrba ESDS only - the RBA of the record to be updated

Exit variables

&¢retc return code (The meaning of each return code is described in the section entitled *Return codes* on the next page.)

&?reg15 If the I/O operation failed because of a VSAM error (*&¢retc*=36), this variable contains the contents of general register 15 returned from the appropriate VSAM macro. These codes are documented in the appropriate IBM documentation.

&?fdbk If the I/O operation failed because of a VSAM error (*&¢retc*=36), this variable contains the feedback code returned by VSAM. These codes are documented in the appropriate IBM documentation.

&?bircd contains record before update if *&?bimage* is y or Y.

&?aircd contains record after update if *&?aimage* is y or Y.

&?offset If the I/O operation failed because of a syntax or validation error in the *&¢fields* parameter (*&¢retc*=20 to 33), this variable indicates where the error occurred in the string. Depending on the actual error, *&?offset* may contain a number which indicates the offset into the variable or a field name. In general, numbers signify invalid field names (for instance: field not present in record definition) and field names may indicate that the data associated with the field name is invalid (for instance: specifying 9 bytes of data when the field is defined as only eight bytes long).

&¢abendc If the I/O operation terminated because of an abend in an NC-VSAM module (*&¢retc*=12), this variable contains the abend code.

&¢ioemsg an error message (suitable for display on a terminal) containing the error information from the above variables.

Return codes

0 update operation completed without errors.

4 the NC-VSAM control file is not available, or reserved variable *&syscntl* has been modified or reset by the user.

5 the record name passed in *&¢record* cannot be found in storage or in the NC-VSAM control file.

6 indicates internal error. Inform your local support office.

7 indicates internal error. Inform your local support office.

9 *&¢fields* is invalid or contains *&sysnull*.

12 NC-VSAM has abnormally terminated. The termination code is available in variable *&¢abendc*.

32 invalid *&¢fields* parameter. *&?offset* points at the operand in error.

33 *&¢fields* parameter refers to a field defined as an automatic fixed field. *&?offset* points at the operand in error.

36 a VSAM access method I/O error occurred. *&?reg15* and *&?fdbk* indicate the VSAM return and feedback codes. The most common error on an UPDATE request is *&?reg15*=8 and *&?fdbk*=16 (record not found).

52 the file is open for input only, an update operation is not permitted.

64 the record being updated contains an LWD (type L) field. *&¢fields* does not contain a reference to this field. *&?offset* contains the name of the omitted LWD field.

72 the record being updated contains an LWD (type L) field. The value passed in *&¢fields* does not match the value currently stored in the record, indicating that the record has been updated since you last read it.

76 the file specified in *&¢file* is not open.

84 the *&¢fields* variable exceeds 32766 bytes in length.

IODELETE

The IODELETE procedure deletes a record from a VSAM KSDS cluster.

IODELETE is not valid for a VSAM ESDS file. For a means of processing deletes from a VSAM ESDS cluster, please refer to *ESDS deletion of records* (below).

The *&¢fields* parameter must supply sufficient information in the VSAM key to uniquely identify the record to be deleted. Fields occurring within the VSAM key area that are not passed in *&¢fields* are assumed to be nulls, binary zeros, X'00'.

If you want to store copies of records - before or after the delete request - in journal or log files, note that record contents may be returned in variables *&?bircd* and *&?aircd* (before and after images respectively). *&?aircd* is set to contents of record deleted.

IODELETE

- validates all supplied data for correct data type.
- validates all supplied data for correct length. Data longer than the defined field length will be rejected and data shorter than the defined field length is padded on the right with nulls, binary zeros, X'00'. If you require alternative padding, use the NCI/XF PAD language statements before executing IODELETE.
- deletes the record from the file.

If update write integrity was specified by including an LWD field (type L) in the record definition, NC-VSAM reads the LWD field from the record and compares it with the value passed in *&¢fields*. If a mismatch occurs, the record has been updated by another user since it you read it and the delete operation is terminated with return code 72.

Format

SET &¢file = *filename*
SET &¢record = *recordname*
SET &¢fields = '*fname*,*varname*,*fname*,*varname*,...'
EXEC IODELETE

ESDS deletion of records

VSAM does not allow the deletion of records from a VSAM ESDS cluster. The suggested method of dealing with this problem is to set a field in the record to a unique value to indicate that the record is deleted. Any read or update is then responsible for checking the record to determine whether it has been flagged as deleted. The value should be unique so it may be necessary to define a field specifically as a delete flag.

Variables

Input and exit variables are provided with IODELETE as described below.

Entry variables

&¢file the NC-VSAM internal name for the physical file.

&¢record the NC-VSAM internal name for the record type.

&¢fields fields within the VSAM key area sufficient to uniquely identify the record to be deleted.

If a type L, update write integrity field, is present in the record definition; *&¢fields* must specify this field. The associated variable must contain the value obtained from a previous read of the same record.

Exit variables

&¢retc return code (The meaning of each return code is described in *Return codes* on page 4.26.)

&?reg15 If the I/O operation failed because of a VSAM error (*&¢retc*=36), this variable contains the contents of general register 15 returned from the appropriate VSAM macro. These codes are documented in the appropriate IBM documentation.

&?fdbk If the I/O operation failed because of a VSAM error (*&¢retc*=36), this variable contains the feedback code returned by VSAM. These codes are documented in the appropriate IBM documentation.

The most common error on a DELETE request is:

&?reg15 = 8

and

&?fdbk = 16

indicating that the record cannot be found.

&?bircd contains record before deletion if *&?bimage* is set to y or Y.

&?aircd contains deleted record - so it contains the same record as *&?bircd*. It is set only if *&?aimage* is set to y or Y.

&?offset If the I/O operation failed because of a syntax or validation error in the *&¢fields* parameter (*&¢retc*=20 to 33), this variable indicates where the error occurred in the string. Depending on the actual error, *&?offset* may contain a number which indicates the offset into the variable or a field name. In general, numbers signify invalid field names (for instance: field not present in record definition) and field names may indicate that the data associated with the field name is invalid (for instance: specifying 9 bytes of data when the field is defined as only 8 bytes long).

&¢abendc If the I/O operation terminated because of an abend in an NC-VSAM module (*&¢retc*=12), this variable contains the abend code.

&¢ioemsg an error message (suitable for display on a terminal) containing the error information from the above variables.

Return codes

0 Delete operation completed without errors.

4 The NC-VSAM control file is not available, or reserved variable *&syscntl* has been modified or reset by the user.

5 The record name passed in *&¢record* cannot be found in storage or in the NC-VSAM control file.

6 indicates internal error. Inform your local support office.

7 KSDS - Internal error. Inform your local support office.

ESDS - A delete was attempted against an ESDS; delete is not permitted on an ESDS file.

9 *&¢fields* is invalid or contains *&sysnull*.

12 NC-VSAM has abnormally terminated. The termination code is available in variable *&¢abendc*.

32 Invalid *&¢fields* parameter. *&?offset* points at the operand in error.

33 *&¢fields* parameter refers to a field defined as an automatic fixed field. *&?offset* points at the operand in error.

36 A VSAM access method I/O error occurred. *&?reg15* and *&?fdbk* indicate the VSAM return and feedback codes.

52 The file is open for input only, a delete operation is not permitted.

64 The record being updated contains an LWD (type L) field. *&¢fields* does not contain a reference to this field. *&?offset* contains the name of the omitted LWD field.

72 The record being updated contains an LWD (type L) field. The value passed in *&¢fields* does not match the value currently stored in the record, indicating that the record has been updated since you last read it.

76 The file specified in *&¢file* is not open.

84 The *&¢fields* variable exceeds 32766 bytes in length.

IOFIXES

The IOFIXES routine can be used to generate and display the NC-VSAM fix grid indicating which fixes, if any, are applied to the NC-VSAM I/O subsystem. The EXEC allows SETTING, LOGGING or RESETTING of the variables that contain the fix grid.

An example of the fix grid being logged to the NCI/XF log is shown below:

```
11:59:26 IOFIXES - NC-VSAM VERSION 1.7.0 RELEASED 04 APR 93 FOR MVS
11:59:26 IOFIXES - THE FOLLOWING FIXES HAVE BEEN APPLIED
11:59:26 IOFIXES -       0 1 2 3 4 5 6 7 8 9          0 1 2 3 4 5 6 7 8 9
11:59:26 IOFIXES -      --------------------         --------------------
11:59:26 IOFIXES - 0 | . . . . . . . . . .        1 | . . . . . . . . . .
11:59:26 IOFIXES - 2 | . . . . . . . . . .        3 | . . . . . . . . . .
11:59:26 IOFIXES - 4 | . . . . . . . . . .        5 | . . . . . . . . . .
11:59:26 IOFIXES - 6 | . . . . . . . . . .        7 | . . . . . . . . . .
11:59:26 IOFIXES - 8 | . . . . . . . . . .        9 | . . . . . . . . . .
11:59:26 IOFIXES -    --------------------         --------------------
```

Format

EXEC IOFIXES *option1 option2.....option15*

where

option1 provides the prefix (up to eight bytes in length) to be displayed on the log, if the LOG option is specified. Although only required for the LOG function, this is a positional parameter and is therefore required before specifying *option2* to *option15*. The variable *&sysnull* may be used to allow it to default to a value of IOFIXES. For example:

EXEC IOFIXES &SYSNULL SET

will simply set the variables.

option 2 through *option 15* can consist of any of the following values, if specified:

SET	to set the variables
LOG	to log the variables
RESET	to reset the variables

If SET, LOG and RESET are specified, they will override the default settings and will be executed in the order encountered. For example:

EXEC IOFIXES "MY-NAME" RESET LOG SET

will reset the variables, log nothing and then set the variables, which may not produce the required result.

If the panel is executed without options (EXEC IOFIXES), it will default to:

EXEC IOFIXES 'IOFIXES' 'SET' 'LOG' 'RESET'

This will set the variables *&iovfixh*, *&iovfixn* and *&iovfix1-9*, depending on the value set in *&iovfixn*. Having been set, the variables will be logged with the first option (set to IOFIXES) prefixing each line (as above). Finally the variables will be reset and hence unavailable for use.

Exit Variables

These exit variables are dependent on the options specified leaving the variables set. The following variables are available:

&iovfixh	provides the version and release date information at the start of the log.
&iovfixn	provides the number of lines of the display.
&iovfix1 through *&iovfixn*	provide the fix information displayed in these lines. In the example on the previous page no fixes are installed, hence this area contains only dots (.)

When a fix is applied the dot in the position which corresponds to the fix applied will be replaced by one of the following characters:

P represents a fix applied in response to a problem

C represents a fix applied for customization purposes

All other values are set by the panel during the LOG function.

Return Codes

0	Options requested worked successfully
1-16	The option with this number was in error
60	The IOVERS module could not be loaded.

IOCLOSE

The IOCLOSE routine closes a file and stops the NC-VSAM subtask, which was attached at open time to manage all I/O to the physical file, leaving it available for further use.

If the file was automatically allocated by IOOPEN, the IOCLOSE routine will automatically deallocate the file.

If the file is not open, the close request is ignored.

The IOCLOSE routine will check whether the subtask has completed before returning control. The length of time allowed for completion may be specified, if not, the routine waits for a maximum of one minute.

Format

SET &¢file = *filename*

EXEC IOCLOSE [*time* [*repeat*]]

where

time is the frequency in one tenths of a second at which the routine checks whether the file has been closed and the subtask stopped. If unspecified, defaults to 1.

repeat is the maximum number of times the routine checks whether the file is closed and the subtask stopped. If unspecified, defaults to 600.

Variables

Input and exit variables are provided with IOCLOSE as described below.

Entry variables

Before executing IOCLOSE, the following variables must be set

&¢file the NC-VSAM internal name for the physical file

Exit variables

&¢retc return code (The meaning of each return code is described below.)

Return codes

0 file is closed and NC-VSAM subtask detached

99 NC-VSAM subtask has not been detached within the specified or default wait time.

IOSTATS

The IOSTATS routine extracts VSAM statistics for a specified dataset.

Format

SET &¢file = *filename*

SET &¢record = *recordname*

EXEC IOSTATS

Variables

Input and exit variables are provided with IOSTATS as described below.

Entry variables

Before executing IOSTATS, the following variables must be set

&¢file	the NC-VSAM internal name for the physical file
&¢record	the NC-VSAM internal name for the record type

Exit variables

&¢retc	return code (The meaning of each return code is described in the section entitled *Return codes* on the next page.)
&?#drcds	number of data records
&?#ddel	number of deleted data records
&?#dins	number of inserted data records
&?#dupd	number of updated data records
&?#dread	number of retrieved data records
&?dcispl	number of times a data control interval was split
&?dcaspl	number of times a data control area was split
&?dexcps	number of EXCPS issued for data
&?#dxtnt	number of data extents
&?dcisiz	size of data CI
&?dfrees	data percentage free space 'CA CI'
&?dalloc	data space allocation 'TYPE(PRI SEC)'
&?#ircds	number of index records
&?#idel	number of deleted index records
&?#iins	number of inserted index records
&?#iupd	number of updated index records
&?#iread	number of retrieved index records
&?icispl	number of times an index control interval was split
&?icaspl	number of times an index control area was split
&?iexcps	number of EXCPS issued for index
&?#ixtnt	number of index extents
&?icisiz	size of index CI

&?ifrees	index percentage free space 'CA CI'
&?ialloc	index space allocation 'TYPE(PRI SEC)'
&?#ilvls	number of index levels
&?iofdel	number of IODELETEs against a file
&?iofrea	number of IOREADs against a file
&?iofupd	number of IOUPDATEs against a file
&?iofadd	number of IOADDs against a file
&?iordel	number of times IODELETE routine called for the given record
&?iorrea	number of times IOREAD routine called for the given record
&?iorupd	number of times IOUPDATE routine called for the given record
&?ioradd	number of times IOADD routine called for the given record

* Maintenance of these counts is switchable. They are enabled by setting variable *&*?iocts* to Y.

Return codes

0 function completed without errors

76 the file specified in *&¢file* is not open

IORESET

The IORESET routine resets all variables used by NC-VSAM. This routine should be used with care as any subsequent NC-VSAM request will be interpreted as the first request to the product.

Format

EXEC IORESET

Chapter 5 - NC-SAM routines

IOSOPEN 5.2
Format 2
Notes 3
Return codes 3
IOSREAD 4
Changing stem variables 4
Format 4
Note 5
Return codes 5
IOSWRITE 6
Notes 6
Return codes 7
IOSCLOSE 8
Format 8
Note 8
Return codes 8
IOSRESET 9
Format 9
Note 9
Return codes 9
Example 10
Technical notes 11
Return codes (in both &usrretc and &?sretc) 11
Abends 13

IOSOPEN

The IOSOPEN routine allows you to specify, via a series of operands, the file that you want to open, and the processing mode you want to use (input, output, etc). As with NC-VSAM, you can specify the file either by DDNAME, for a preallocated file, or by DSNAME with any other necessary JCL type parameter variables (for example DISP, SPACE, etc.) in which case the file will be dynamically allocated. It is possible to read the complete directory by supplying an unqualified PDS name.

Upon successful return, a unique value will be returned in an NCI/XF variable. This value must be used, in subsequent calls, to identify the file to be read, written or closed.

IOSOPEN will, as the primary entry point, be supplied with as many parameters as necessary to check and open the required file. The main requirements for IOSOPEN will be:

- the DDNAME or DSNAME (non-dynamic selection; or dynamic selection).
- the file disposition (DISP) may be NEW, OLD, MOD, or SHR. The default for non-dynamic files is defined by the JCL.
- the member name (this may also be supplied in the fully qualified dynamic DSNAME). If omitted for a PDS, a member list is returned.
- the DCB information for dynamic allocation of new files.
- the space information for dynamic allocation of new files.
- the file access mode. This can be INPUT or OUTPUT.

Format

```
SET    variables
EXEC   IOSOPEN
```

where

variables are the following:

&?smode	mode as required, INPUT or OUTPUT (the default is INPUT).
&?sddnam	DDNAME, only if pre-allocated (but never with *&?sdsnam*).
&?sdsnam	DSNAME only if not pre-allocated (but never with *&?sddnam*).
&?smembe	member name (optional).
&?sndisp	normal DISP (NEW, SHR, etc). Default is SHR.
&?sdisp2	second DISP (CATLG, etc). Default is KEEP.
&?sunit	UNIT (SYSDA, 3380, etc). No default.
&?svolse	VOLSER (PACK01, etc). No default.
&?sdsorg	DSORG (PS, PO, DA (PS)). No default.
&?srecfm	RECFM (FBA, VB, etc). Default is F.
&?slrecl	LRECL (1-32760). Default is 80.
&?sblksi	BLKSIZE (19-32760). Default is 80.

&?sspcal	CYL, TRK, or 19-32760. No default.
&?sspcpr	primary space allocation (1 - 99999). No default.
&?sspcse	secondary space allocation (1 - 99999). No default.
&?sspcdi	number of directory blocks (1 - 99999). No default.
&?spack	"P" if **new** output file or member is to be in standard SPF packed format. (Note that SPF packed input files are unpacked automatically during IOSREAD operations and existing packed output files are repacked during IOSWRITE operations. In both cases *&?spack* need not be specified.) No default.

All JCL parameters are checked internally by the dynamic allocation routines. If any are in error, the return code is set to 32 and variables *&?sirc* and *&?serc* are set to the values documented in the IBM manual GC28-1465, *'Authorized Assembler Programming Guide'*, sections 23.4.1 - 23.4.2.

Once a file has been opened the variable *&?s_id* will contain a value which must be maintained for all subsequent calls to the file as a pointer to all the information that IOSAM requires to perform work against the file.

Notes

1. If an existing PDS is specified, but without a member name, the subsequent IOSREADs will reference the file's directory entries, not the file itself. (See IOSREAD for details). When processing has finished, IOSCLOSE must be issued to release storage.

2. In all cases, when an IOSOPEN is done, the variables passed to it must be regarded as a discrete set; that is to say all variables not required for that invocation must be reset or set to null. This is to prevent possible allocation failure.

   If an IOSOPEN is successful, variable *&s-id* will be returned. This means that if you have more than one file open at a time, *&s-id* must be saved after IOSOPEN and restored before each subsequent IOSREAD, IOSWRITE or IOSCLOSE for that file.

3. If *&?sdisp2* is set to "DELETE" on an IOSOPEN execution, the file will be **deleted immediately**; you will not be able to read it, write to it or close it.

Return codes

See *Return codes (in both &usrretc and &?sretc)* on page 5.11.

IOSREAD

The IOSREAD routine allows you to read one, several or all records from a file. You can also read a file starting from an offset. The records are returned as one or more entries in stem variables. The default settings for the stem is *?sr*. Variables *&?sr1* to *&?srnnnn* will contain the records and *?sr0* will contain the number of items returned (*nnnn*).

Note: The default stem name can be changed by means of the *&?sstem* variable (for example *&?sstem* = "out"). The stem must be no more than three characters.

Changing stem variables

Record variables *&?sr0* through *&?srnnnn* can be substituted (both in IOSREAD and IOSWRITE) by using the variable *&?sstem*. If for example variable *&?sstem* has been set to 'six', variables *&six0* through *&sixnnnn* will be referenced in their place.

Note: As the maximum length of a variable name including the ampersand (&) is eight, the maximum length of the contents of *&?sstem* is three characters, and the maximum number of records returned will be 9999 (length of *&?sstem* = 4 is invalid, whereas 1 is valid).

Format

SET *variables*
EXEC IOSREAD [*nn*]

where

nn is the number of records to be read.

variables are the following:

&?s_id	IOS address (mandatory).
&?sfirst	first record to read/update.
&?sfor	count of records to read.

Variable *&?sfirst* may be set to show where to start reading. If not set, the initial (or next) is assumed.

Variable *&?sfor* may be set to show how many records are to be read. To read all records, specify ALL on the execution of IOREAD (EXEC IOREAD ALL). If there is neither a value specified in *&?sfor* nor ALL specified on the EXEC, then the next (or first) record will be read (the word NEXT can be specified on the EXEC statement to specify the default).

On return from a read, variable *&?sr0* will contain the count of records returned. It can, for example, be the total number of records in the file or possibly a number less than the number requested, if the file does not contain that number. Variables *&?sr1* to *&?srnnnn* will contain the retrieved records (where *nnnn* is a number, with no leading zeros, equal to the value in *&?sr0*).

The word NEXT may be replaced by any numeric value, in which case variable *&?sfor* will be set to that value automatically.

If a subsequent read is done, any value in *&?sfirst* will still be honored, and reading will start after a further *&?sfirst* records have been read.

When reading records sequentially, NEXT is defined as being the current record pointer (the position following the last record read). When it is the initial read, the first record will be read.

When the variables to be returned are directory entries of a PDS, they are returned in *&?sm1* through *&?smnnnn* with *&?sm0* set to the value of *nnnn*. The stem cannot be reset with the *&?sstem* variable.

Note

1. If the file is a variable file (RECFM=VB) and is opened for input, the Record Descriptor Word (RDW) is NOT returned in the first four bytes of the returned record variables *&?sr1* to *&?srnnnn.* The length of the record can be extracted via the NCI/XF LENGTH language statement.

   Packed files are handled normally whether they are fixed or variable.

2. If the IOSOPEN referred to a PDS, but no member name was specified, the read operator will refer to directory entries. These will be returned in variables *&?sm1* through *&?smnnnn* with *&?sm0* set to the value of *nnnn*. The stem **cannot** be reset with the *&?sstem* variable.

Return codes

See *Return codes (in both &usrretc and &?sretc)* on page 5.11.

IOSWRITE

The IOSWRITE routine allows you to write a number of records to a file or PDS member. As in IOSREAD, records are extracted from stem variables. Refer to *IOSREAD* on page 5.4.

Format

SET *variables*
EXEC IOSWRITE [*nn*]

where

nn is the number of records to be read.

variables are the following:

&?s_id IOS address (mandatory).

&?sr0

&?sr1
through
&?srnnnn

Record variables *&?sr1* to *&?srnnnn* should be set prior to a write. If the value that IOSWRITE determines is the number of records to write (either by *&?sr0* being set to a value, or by a value being specified on the invocation of IOSWRITE) is greater than the number of variables set, those extra variables will result in blank (or null for variable files) records being written. The same will apply if an intermediate variable is missing.

Variable *&?sr0* may be set to show how many records are to be written. If it is not set or contains zeros, and EXEC IOSWRITE NEXT is specified, one record will be written. In this situation, EXEC IOSWRITE is the same as EXEC IOSWRITE NEXT.

The word NEXT may be replaced by any numeric value, in which case variable *&?sr0* will be set to that value automatically.

When writing records sequentially, NEXT is defined as being the current record pointer (the position following the last record written). When it is the initial write, the disposition of the file will be taken into account.

Notes

If the file is a variable file (RECFM=VB) and is opened for output, the Record Descriptor Word (RDW) must not prefix the records. The record variables *&?sr1* to *&?srnnnn* must be set to the exact length required and the implied length will be used by NC-SAM to build the RDW.

If *&?sndisp* = MOD, writing will start after the current end of file marker. **MOD is invalid for PDS members**.

Synopsis

- *&?sr0* not set, EXEC IOSWRITE, will result in 1 record being written.
- *&?sr0* = 12, EXEC IOSWRITE 5, will result in 5 records being written.
- *&?sr0* = 5, EXEC IOSWRITE 77, will result in 77 records being written.
- *&?sr0* = 6, *&?sr1* = A, *&?sr4* = B, null values for *&?sr2*, *&?sr3*, *&?sr5*, &?sr6, EXEC IOSWRITE, will result in 6 records being written, but 2, 3, 5 and 6 will be blank (or null for variable files).

Writing to output files that are packed is supported, but results with variable length files may be unpredictable.

An attempt to write a record longer than the LRECL of the file results in return code 11, and no records are written.

Return codes

See *Return codes (in both &usrretc and &?sretc)* on page 5.11.

IOSCLOSE

The IOSCLOSE routine closes a file. If the output file is a PDS member, the standard SPF date/time/id fields are updated.

Format

SET *variables*
EXEC IOSCLOSE

where

variables are the following variables:

&?s_id IOS address (mandatory).

Note

When a file is closed, subsequent calls using *&?s_id* set to the value in an IOCLOSE invocation could result in an abend. Although the abend is trapped, it will result in response degradation. It could also lead to the corruption of another file; so great care must be taken to ensure that the *&?s_id* returned by IOSOPEN is stored and passed to successive IOSREADs or IOSWRITEs, and to the IOSCLOSE. When the IOSCLOSE has ended, the storage variable should be deliberately RESET.

Return codes

See *Return codes (in both &usrretc and &?sretc)* on page 5.11

IOSRESET

The IOSRESET routine is designed to reset ALL the variables needed by IOSOPEN. This is an easy way to ensure that all variables are reset prior to setting some, or all, of them before IOSOPEN, and its subsequent IOSREAD (or IOSWRITE) and IOSCLOSE operations.

Format

EXEC IOSRESET

Note

IOSRESET should always be used after IOSCLOSE to free all the variables (including the record variables, *&?sr0* to *&?sr*nnn).

Return codes

Always zero.

Example

Read a sequential file, extract certain records and write these to a new PDS member:

```
Exec IOSRESET                      /* Reset all NC-SAM variables               */
Set &?sddnam  = &sysnull           /* This is not a predefined file (via JCL)  */
Set &?sdsnam = "GEN.INFO.FILE"     /* Dynamically allocate input PS file.      */
Set &?sndisp = "SHR"               /* Set read disposition to SHR              */
Set &?smode = "INPUT"              /* The file will be opened as "INPUT"       */
Exec IOSOPEN                       /* Open the file                            */
Set &userin  =  &?s_id             /* Save IOSAM's Internal DCB number         */

Exec IOSRESET                      /* Reset all NC-SAM variables               */
Set &?sddnam  = &sysnull           /* This is not a predefined file (via JCL)  */
Set &?sdsnam = "OUTPUT.PDS.FILE"   /* Dynamically allocate output PDS          */
Set &?smembe = "AARDVRK1"          /*       with this member name.             */
Set &?sndisp = "OLD"               /* Set read disposition to OLD              */
Set &?smode = "OUTPUT"             /* Create or overwrite member               */
Exec IOSOPEN                       /* Open the file                            */
Set &userout  =  &?s_id            /* Save IOSAM's Internal DCB number         */
Set &sam0 = 0                      /* Initialize counter                       */
Set &?s_id = &userin               /* Say we're going to read input file.      */

LABEL ILOOP
Exec IOSREAD next                  /* Read the next (or first) - one assumed)  */
If  &?sr0 <> 1                     /* Assume EOF on error                      */
    goto OUTPUT                    /*    so go write output.                   */
If &?sr1(1:8) <> "AARDVARK"        /* Only use records with that in 1-8        */
   goto ILOOP                      /*    else loop to next...                  */
Incr &sam0                         /* Increment output counter                 */
Set &&sam&sam0 = &?sr1             /* Output record = input record.            */
Goto ILOOP                         /* Loop back for next record                */

LABEL OUTPUT
Set &?s_id  =  &userout            /* Say it's the output file, and give the   */
Set &?sstem = "sam"                /*  new output stem name (&sam0 is new      */
Exec IOSWRITE                      /*  count) and write them out...            */

Set &?s_id  =  &userout            /* Specify the output file                  */
Exec IOSCLOSE                      /*  Now close the output file               */
Set &?s_id  =  &userin             /*    and then the input                    */
Exec IOSCLOSE                      /*    file...                               */
Reset &?sstem                      /* Reset the stem name.                     */
```

Technical notes

If an NCI/XF EXEC or PANEL does not call IOSCLOSE explicitly, it will be posted once for each open file by the NCI/XF SHUTDOWN task to close all open files. This method may waste areas of storage unnecessarily and is not recommended.

The enqueuing of dataset names will adhere to IBM standards. PS files will be enqueued as EXClusive if the DISP is NEW, OLD or MOD; and as SHR if the DISP is SHR. PO files will be treated in the same way as PS files if no member name is specified (if a directory read is being done). PO files with member names specified will obey the same rules as SPF. They will be enqueued as EXC under the SPFEDIT major name, or minor name of dsname followed by member name, if opened for output. In all cases the IBM standard of major name SYSDSN, and minor name of dataset name will apply by default.

Return codes (in both *&usrretc* and *&?sretc*)

In most cases, variable *&?semsg* will contain an explanation of the following error codes; in many cases *&?sreg0* and *&?sreg15* may contain further information:

- 0 Request successfully processed.
- 4 EOD encountered on INPUT file.
- 5 Internal error.
- 6 Invalid mode specified (not Input, Output, Update), or invalid request (for example write to input file).
- 7 Invalid request (not Open, Close, etc), or read/write before Open.
- 8 NC-SAM task is not active.
- 9 *&?s-id* is probably invalid or the file has already been closed.
- 10 READ has been issued past EOD (end of file marker).
- 11 Attempt to write an over-length record. Variable *&?sreg15* contains the variable stem number.
- 16 Trying to do I/O (or CLOSE) to CLOSEd file. I/O type is set in *&?reg15* as follows:
  - 4 OPEN
  - 8 READ
  - 12 WRITE
  - 16 CLOSE
  - 20 UPDATE
  - 24 DELETE
- 28 IOSINIT failed - see *&?sloadm,* which contains the module name.
- 32 DDNAME not allocated by dynamic allocation. Variables *&?serc* and *&?sirc* may contain the DAIR codes which may be found in the IBM manual GC28-1465, *'Authorized Assembler Programming Guide'*, sections 23.4.1 - 23.4.2. Variable *&?sreg15* may be set as follows:
  - 56 DSNAME not supplied (*&?sdsnam*).
  - 58 One of the DISP variables is invalid (*&?sndisp, &?sdsp2*).
  - 60 DSORG variable invalid (*&?sdsorg*).
  - 64 One of the SPACE variables is invalid (*&?sspcal*, *&?sspcpr*, *&?sspcse*, *&?sspcdi*).
  - 68 RECFM variable invalid (*&?srecfm*).
  - 76 UNIT variable invalid (*&?sunit*).
  - 78 LRECL variable invalid (*&?slrecl*).
  - 80 BLKSIZE variable invalid (&?sblksi).

99 NC-SAM has been shut down (see your administrator).

100 NC-SAM has not been started (see your administrator).

128 IOSTERM failed - see *&?sloadm*, which contains the load module name.

132 DDNAME **not** freed by dynamic allocation. Variables *&?serc* and *&?sirc* contain the DAIR codes which may be found in the IBM manual GC28-1465, *'Authorized Assembler Programming Guide'*, sections 23.4.1 - 23.4.2.

256 Internal error.

330 Member name requested for PS file.

332 DSORG invalid (not PO/PS).

333 DISP is inconsistent with the request (for instance, member list for new file).

337 RECFM of file is U. This is invalid for PDS with member name specified.

349 Member of PDS not found on input.

360 IOSOPEN - *&?sddname* and *&?sdsname* both specified. These are mutually exclusive.

361 IOSOPEN - neither *&?sddname* nor *&?sdsname* are specified.

362 IOSOPEN - *&?sndisp* = MOD and member name specified. You cannot MOD a PDS member.

370 IOSREAD - the value in *&?sfirst* is invalid.

371 IOSREAD - the value in *&?sfor* is invalid.

372 IOSREAD - the value on the invocation of IOSREAD is invalid (for example EXEC IOSREAD ABC).

373 IOSREAD - the length of the stem set in *&?sstem* (if set) is greater than 3 (maximum), for example *&?sstem* = abcd.

380 IOSWRITE - value in *&?sfirst* is invalid.

382 IOSWRITE - value on invocation of IOSWRITE is invalid (for example EXEC IOSWRITE 2X5).

383 IOSWRITE - the length of the stem set in *&?sstem* (if set) is greater than 3 (maximum), for example *&?sstem* = abcd.

384 IOSWRITE - final value in *&?sr0* is invalid.

512 I/O error. I/O type is set in *&?sreg15* as follows:

- 1 Member not found.
- 4 OPEN (return code from I/O operation is stored in *&?sreg0*):
  - 32 ENQUEUE of member to be opened for output has failed, probably already open elsewhere.
  - 34 Input member of PDS not found.
  - 36 Unable to ENQ dataset; file, or member, is probably enqueued elsewhere in EXC (exclusive) state.
- 8 READ (return code from I/O operation is stored in *&?sreg0*; if *&?sreg0* is zero, then no records have been specified to read, or records read are not those expected).
- 12 WRITE (return code from I/O operation is stored in *&?sreg0*; if *&?sreg0* is zero, then no records have been specified to write, or records written are not those expected).
- 16 CLOSE (return code from I/O operation is stored in *&?sreg0*):
  - 12 Directory full. Member could not be added. Redefine file with more directory blocks.
  - 40 DEQUEUE of member failed.

1000 Unable to start NC-SAM.
1001 NC-SAM cannot initialize.
1024 Internal chaining error. Check *&?sreg15* and call support.
4096 An abend has occurred; *&?sreg15* contains the abend code. (See the following section on abends).

Abends

If an abend occurs that is directly related to an I/O operation (for example 213, B37, 001, etc.) the file (or member) will be automatically closed by the NC-SAM abend routines. There will be no need to issue a further IOSCLOSE.

If a serious internal error should occur that entails storage failure (0C1, 0C4, etc.) NC-SAM will terminate and all files will be automatically closed. The next execution of IOSOPEN will automatically restart NC-SAM.

This page intentionally left blank

Chapter 6 - NC-MHO routines

Introduction 6.2
Pre-install definitions 6.3
- LU0 protocol 6.3
- Sockets protocol 6.3
- APPC protocol 6.3

Controlling the NC-MHO environment 6.9
- MHOCLOSE 6.10
- MHODLINK 6.11
- MHOLINK 6.12
- MHOOPEN 6.14
- MHOSIDE 6.16
- MHOSTART 6.17
- MHSTATUS 6.20
- MHOSTOP 6.21
- MHOTRACE 6.22

Using the NC-MHO environment 6.23
- MHOEXEC 6.24
- MHOLOG 6.27
- MHOREPLY 6.29
- MHORESLV 6.31
- MHOSET 6.33

Customization (EXEC and exit points) 6.35
- User authorization exit points 6.35
- EXEC exit points 6.36
- Logon authorization EXEC 6.37
- Set authorization EXEC 6.38
- EXEC authorization EXEC 6.39
- Resolve authorization EXEC 6.40
- Log authorization EXEC 6.41
- Session established EXEC 6.42
- Lost session EXEC 6.43
- No session established EXEC 6.44
- Protocol end EXEC 6.45
- Send completion EXEC 6.46
- Receive completion EXEC 6.47
- Subsystem end EXEC 6.48
- Encrypt/decrypt exit 6.49
- Available variables 6.50

Using the NC-MHO low level APPC environment 6.57
- The MHOAPPC EXEC 6.57
- APPCCMD completion exit 6.66
- Session limits, FMH5 and PIP data 6.68

Using the NC-MHO low level Sockets environment 6.73
- MHOSOCK EXEC 6.73
- Format 6.73
- Sockets completion EXEC 6.79
- Connect data 6.80

Data format 6.83
RPC routines 6.86
- MHRPCSRV 6.86
- MHRPCLNT 6.88

Introduction

The routines described in this chapter should be consulted in conjunction with the NCI/XF User Guide, *Chapter 7 - A guide to NC-MHO*.

This chapter is divided into the following sections:

- Pre-install definitions (VTAM).
- Controlling the NC-MHO environment.
  This section describes the EXEC routines that are provided to define and start the environment, start and stop a protocol handler and to establish and take down a link.
- Using the NC-MHO environment.
  This describes the EXEC routines available to perform remote procedural calls (RPCs) on a remote NC-MHO
- Customization (EXEC and exit points).
- Using the NC-MHO low level APPC environment.
  This describes the EXEC routines available to develop your own Sockets or LU6.2 applications.
- Using the NC-MHO low level Sockets environment.
- Data format.
- RPC routines.

Pre-install definitions

The following VTAM definitions must be set prior to starting the NC-MHO subsystem.

LU0 protocol

For LU0, applications to be used must be defined to VTAM as follows:

applname APPL ACBNAME=*acbname*

where

applname represents the application name concerned
acbname represents the ACB name concerned.

Note: PARSESS=YES must **not** be coded nor implied by the use of APPC=YES. MODETAB and DLOGMOD may be omitted if the default IBM table has not been modified.

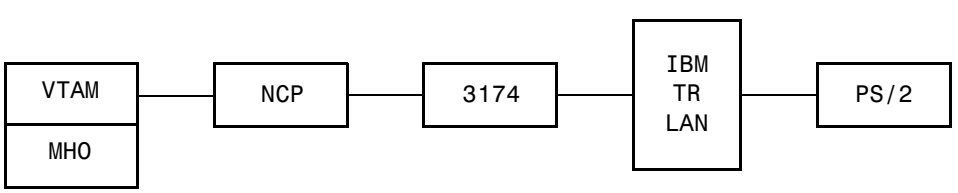
Sockets protocol

For applications using Sockets, the TCP/IP port number and internet addresses must be defined before using the MHO environment. These are defined using the MHOSIDE EXEC, typically in the NCI/XF startup EXEC.

APPC protocol

For applications using APPC, appropriate definitions must be made for both the MVS and non-MVS platforms. These definitions are dependant on given hardware and software configurations.

The following example defines a POSSIBLE configuration that could be used to connect a non-NCI platform to an MVS address space using NCI/XF and MHO. This configuration has a PC running CM/2 that is LAN attached to a 3174 with a LAN adaptor; the LAN adaptor is link-attached to a Front End Processor (FEP) that is channel attached to a mainframe running MVS and VTAM. The devices are DEPENDENT LUs; APPN is not being used. The statements assume VTAM 3.4 and NCP 5.4 (check the manual appropriate to your release of software).



The implementation of this configuration comprises the following steps. The times shown in parentheses assume the steps are done by someone familiar with VTAM/NCP systems programming, 3174 customization, and OS/2 customization:

1. NETID/SSCPNAME (5min)

   Determine network and SSCP name values from VTAM start parameters (this assumes you already have VTAM running).

2. CDRMNAME (5min)

   Determine local domain manager name from the local CDRM definition (this assumes you already have VTAM running).

3. USS/MODE TABLES (15min)

   Assemble and link-edit the sample USS and mode tables. If you have suitable tables then this step is optional.

4. MHO APPLICATION DEFINITION (10min)

   Add MHO application definition to VTAMLST library and activate it. If you already have a suitable definition then this step is optional.

5. 3174 CUSTOMISATION (30min)

   Customize 3174 to match NCP definitions. If you already have a gateway 3174 then this step may be omitted.

6. NCP DEFINITION (15min+gen time+load time)

   Add definitions for gateway 3174 and downstream PU(s). If you have suitable definitions then this step should be omitted. If necessary reload FEP.

7. CM/2 DEFINITIONS (15min)

   Add definitions to CM/2 to define partner LU and symbolic destination. Add link and PU definitions if you have not used CM/2 to access host applications previously.

Coding for this example

Lowercase fields should be replaced by your own values. If these fields appear more than once in different definitions, they **must** match.

- Virtual Telecommunications Access Method (VTAM) startup parameters must include the following statements. If any of these parameters are changed then VTAM must be restarted. See VTAM Resource Definition manual for more details.

```
NETID=netid,
HOSTSA=nn,
SSCPNAME=sscpname,
```

- Virtual Telecommunications Access Method (VTAM) domain manager must include the following statements. If any of these parameters are changed then the major node must be inactivated and reactivated. If the SSCPNAME and the domain manager name are different then use the domain manager name. See VTAM Resource Definition manual for more details.

```
labelc   VBUILD TYPE=CDRM
labeln   NETWORK NETID=netid
sscpname CDRM  SUBAREA=nn,CDRDYN=YES,CDRSC=OPT
```

- Virtual Telecommunications Access Method (VTAM) application definition must include the following statements. If any of these parameters are changed then the major node must be inactivated and reactivated; applname is what you specify on the *&mh_acbn* statement in the NCI address space prior to issuing MHOOPEN. See *VTAM Resource Definition* manual for more details.

```
labela   VBUILD TYPE=APPL
applname APPL  AUTH=(ACQ,PASS),PARSESS=YES,MODETAB=modetabl,
ACBNAME=applname,APPC=YES
```

- Network Control Program should include the following statements. Note that punameu and any associated lunameu(s) are for devices attached to the 3174 directly; puname is the DOWNSTREAM PU that is used by the PC attached via the LAN adapter. Check the NCP Resource Definition manual for which parameters may be changed and introduced by recycling the NCP, and those that require an NCP generation process. The device punameu MUST be active prior to attempting any connection from a downstream PU. The downstream puname must be in a state PCTD2 prior to attempting connection.

```
linename LINE ADDRESS=(aaa,yyyy),
                MODETAB=modetabl,
                DLOGMOD=mode_ent,
                USSTAB=uss_tabl
punameu  PU ADDR=bb,
              ANS=CONT,
              PUTYPE=2,
              MAXOUT=7
lunameu1 LU LOCADDR=2
"        "  "
lunameun LU LOCADDR=n
puname   PU ADDR=cc
              ANS=CONT,
              PUTYPE=2
luname1  LU LOCADDR=2,
              PACING=1,
              VPACING=1
luname2  LU LOCADDR=3,
              PACING=1,
              VPACING=1
```

- Sample mode table definition. This table must be assembled and link-edited into a library that is allocated to the VTAM address space VTAMLIB DDname. The module may be changed, reassembled, and link-edited while VTAM is active but the F NET,TABLE command must be used to notify VTAM of the changes. See the VTAM Operation manual for more details.

```
modetabl MODETAB
mode_ent MODEENT LOGMODE=mode_ent,
              FMPROF=X'13',
              TSPROF=X'07',
              PRIPROT=X'B0',
              SECPROT=X'B0',
              COMPROT=X'D0B1',
              RUSIZES=X'8585',
              PSERVIC=X'060200000000000000000100'
         MODEEND
     END
```

- Sample USS table definition. This table must be assembled and link-edited into a library that is allocated to the VTAM address space VTAMLIB DDname. The module may be changed, reassembled, and link-edited while VTAM is active but the F NET,TABLE command must be used to notify VTAM of the changes. See the VTAM Operation manual for more details.

```
uss_tabl USSTAB
         USSMSG MSG=00,TEXT=
         USSMSG MSG=01,TEXT=
         USSMSG MSG=02,TEXT=
         USSMSG MSG=03,TEXT=
         USSMSG MSG=04,TEXT=
         USSMSG MSG=05,TEXT=
         USSMSG MSG=06,TEXT=
         USSMSG MSG=07,TEXT=
         USSMSG MSG=08,TEXT=
         USSMSG MSG=09,TEXT=
         USSMSG MSG=10,TEXT=
         USSMSG MSG=11,TEXT=
         USSMSG MSG=12,TEXT=
         USSMSG MSG=13,TEXT=
         USSMSG MSG=14,TEXT=
         USSEND
          END
```

- Replies to 3174 configuration questions must include the following answers. See *3174 Customizing Guide* manual for more details.

  900 - Token-Ring Network Address for the Gateway must match the value specified in the CM/2 NDF file DESTINATION_ADDRESS gateway_addr

  908 - Link subsystem name should match the 3174 punameu.

  940 - Ring address assignment should look something like:

  S Ring-address SAP
  cc xxxx xxxx xxxx 04

  941 - Ring transmission definition should look something like:

  S Ring-address SAP F W
  cc xxxx xxxx xxxx 04 0 2

  - note that the cc field must match the NCP downstream PU ADDR field.

- Sample CM/2 Network Definition File

```
DEFINE_LOCAL_CP  FQ_CP_NAME(netid.puname     )
                 CP_ALIAS(puname )
                 NAU_ADDRESS(2)
                 NODE_TYPE(EN)
                 NODE_ID(X'05D00000')
                 NW_FP_SUPPORT(NONE)
                 HOST_FP_SUPPORT(YES)
                 HOST_FP_LINK_NAME(HOST0001)
                 MAX_COMP_LEVEL(NONE)
                 MAX_COMP_TOKENS(0);
```

```
DEFINE_LOGICAL_LINK  LINK_NAME(HOST0001)
                     FQ_ADJACENT_CP_NAME(netid.sscpname   )
                     ADJACENT_NODE_TYPE(LEN)
                     DLC_NAME(IBMTRNET)
                     ADAPTER_NUMBER(0)
                     DESTINATION_ADDRESS(X'gateway_addr')
                     ETHERNET_FORMAT(NO)
                     CP_CP_SESSION_SUPPORT(NO)
                     ACTIVATE_AT_STARTUP(YES)
                     LIMITED_RESOURCE(USE_ADAPTER_DEFINITION)
                     LINK_STATION_ROLE(USE_ADAPTER_DEFINITION)
                     SOLICIT_SSCP_SESSION(YES)
                    MAX_ACTIVATION_ATTEMPTS(USE_ADAPTER_DEFINITION)
                     USE_PUNAME_AS_CPNAME(NO)
                     EFFECTIVE_CAPACITY(USE_ADAPTER_DEFINITION)
                     COST_PER_CONNECT_TIME(USE_ADAPTER_DEFINITION)
                     COST_PER_BYTE(USE_ADAPTER_DEFINITION)
                     SECURITY(USE_ADAPTER_DEFINITION)
                     PROPAGATION_DELAY(USE_ADAPTER_DEFINITION)
                     USER_DEFINED_1(USE_ADAPTER_DEFINITION)
                     USER_DEFINED_2(USE_ADAPTER_DEFINITION)
                     USER_DEFINED_3(USE_ADAPTER_DEFINITION);

DEFINE_LOCAL_LU  LU_NAME(luname1)
                 LU_ALIAS(luname1)
                 HOST_LINK_NAME(HOST0001)
                 NAU_ADDRESS(4);

DEFINE_LOCAL_LU  LU_NAME(luname2)
                 LU_ALIAS(luname2)
                 HOST_LINK_NAME(HOST0001)
                 NAU_ADDRESS(3);

DEFINE_PARTNER_LU  FQ_PARTNER_LU_NAME(netid.applname )
                   PARTNER_LU_ALIAS(applname)
                   PARTNER_LU_UNINTERPRETED_NAME(applname)
                   MAX_MC_LL_SEND_SIZE(32767)
                   CONV_SECURITY_VERIFICATION(NO)
                   PARALLEL_SESSION_SUPPORT(NO);

DEFINE_PARTNER_LU_LOCATION FQ_PARTNER_LU_NAME(netid.applname )
                           WILDCARD_ENTRY(NO)
                           FQ_OWNING_CP_NAME(netid.sscpname   )
                           LOCAL_NODE_NN_SERVER(NO);

DEFINE_MODE  MODE_NAME(mode_ent)
             COS_NAME(#CONNECT)
             DEFAULT_RU_SIZE(YES)
             RECEIVE_PACING_WINDOW(4)
             MAX_NEGOTIABLE_SESSION_LIMIT(32767)
             PLU_MODE_SESSION_LIMIT(8)
             MIN_CONWINNERS_SOURCE(0)
             COMPRESSION_NEED(PROHIBITED)
             PLU_SLU_COMPRESSION(NONE)
             SLU_PLU_COMPRESSION(NONE);
```

```
DEFINE_DEFAULTS  IMPLICIT_INBOUND_PLU_SUPPORT(YES)
                 DEFAULT_MODE_NAME(BLANK)
                 MAX_MC_LL_SEND_SIZE(32767)
                 DIRECTORY_FOR_INBOUND_ATTACHES(*)
                 DEFAULT_TP_OPERATION(NONQUEUED_AM_STARTED)
                 DEFAULT_TP_PROGRAM_TYPE(BACKGROUND)
                 DEFAULT_TP_CONV_SECURITY_RQD(NO)
                 MAX_HELD_ALERTS(10);

DEFINE_CPIC_SIDE_INFO  SYMBOLIC_DESTINATION_NAME(a_sym_name )
                       FQ_PARTNER_LU_NAME(netid.applname )
                       MODE_NAME(mode_ent)
                       TP_NAME(a_tp_name);
START_ATTACH_MANAGER;
```

Controlling the NC-MHO environment

This section describes the EXEC routines needed to control the NC-MHO environment:

MHOCLOSE	closes (stops) a protocol handler
MHODLINK	takes down a link to a remote partner
MHOLINK	establishes a link to a remote partner
MHOOPEN	opens (starts) a protocol handler that will run under an NC-MHO environment
MHOSIDE	defines a synonym for a TCP/IP internet address/port number combination.
MHOSTART	defines an NC-MHO subsystem environment
MHOSTATUS	examines the status of a protocol handler or link
MHOSTOP	terminates an NC-MHO subsystem environment
MHOTRACE	traces the logic and data flow of a protocol handler and/or a link

MHOCLOSE

The MHOCLOSE EXEC terminates a protocol handler and is executed as follows:

EXEC MHOCLOSE *locacb*

where

locacb is the node application name of the local protocol handler that is to be deactivated. If *locacb* is not defined, the local application name must be defined in variable *&mh_acbn*. For VTAM based protocols this is the VTAM application name.

Note that variable *&mh_id* (returned by the MHOSTART EXEC) must be set prior to issuing this EXEC.

Return Codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOCLOSE:

0 Successful MHOCLOSE.

4 Required parameter not passed (either *&mh_id* or *&mh_acbn*).

8 Timeout occurred during CLOSE function. See *Timeout of synchronous events* on page 7.12 of the NCI/XF User Guide.

12 No protocol handler found.

Example

If ANAPPL describes a VTAM application, then

EXEC MHOCLOSE ANAPPL

closes the VTAM application ANAPPL protocol handler.

MHODLINK

The MHODLINK EXEC terminates a link from the NC-MHO environment to a remote partner in an orderly fashion. See *Technical notes* below for more details on how the link will be broken. The general format of the MHODLINK EXEC is:

EXEC MHODLINK *remnode*

where

remnode is the name of the remote partner to be deactivated.

Note that the variable *&mh_id* (returned by the MHOSTART EXEC) must be set prior to issuing this EXEC.

Return codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOLINK:

0 link is now inactive.

4 an invalid parameter was passed or a required parameter was omitted. Reissue the command with the correct syntax.

8 the link is terminating but no indication has been received before the timeout period. The link may subsequently close normally. See the NCI/XF User Guide, *Chapter 7 - A guide to NC-MHO*.

12 no link exists or *&mh_id* is set incorrectly.

16 unexpected error. See NCI/XF log for more details.

Example

EXEC MHODLINK OTHRAPPL

will terminate a link for NC-MHO to application OTHRAPPL.

Technical notes

For LU0 the EXEC of an MHODLINK will result in a VTAM TERMSESS macro being issued and the related lost session EXEC being driven.

For LU6.2 in high level mode it will result in several VTAM APPCCMD MACROs being issued (ALLOC-ALLOC and REJECT-CONV) and the lost session EXEC being driven.

For LU6.2 in low level mode the user is responsible for terminating the link.

For sockets in high level mode it will result in a SHUTDOWN request being issued.

For Sockets in low level mode the user is responsible for terminating the link.

MHOLINK

The MHOLINK EXEC connects an MHO environment to a session partner. If successful it will result in one or more sessions being established depending on the communications protocol in use and the mode in which it is being used. See *Technical notes* on the next page for more details.

The syntax of the MHOLINK command depends upon the protocol in use and whether it is invoked in HLAPI or LLAPI.

EXEC MHOLINK *remnode*	(for LU0, LU6.2 or Sockets)
EXEC MHOLINK *remnode* [*tpn*] [*mode*]	(for LU6.2 environment only)
EXEC MHOLINK *remnode* *locacb* [*mode*]	(for LU6.2 low level environment only)

where

remnode	is an eight character remote application name (use alphanumeric/national characters). For the Sockets protocol, *remnode* must match an entry defined by the MHOSIDE EXEC.
tpn	is an optional eight character parameter specifying the transaction program name to be used by the remote partner. If the partner is an NC-MHO this field must be set to NC-ISC62 (the default), if specified.
mode	is an optional eight character name specifying a mode to be used in an LU6.2 environment. It has no meaning for other protocol partners. The default is RECORD. (Use alphanumeric/national characters).
locacb	is a required parameter for links in a low level APPC environment. It is the local application name to be used to establish a connection.

Note that the *&mh_id* (returned by the MHOSTART EXEC) must be set prior to issuing this EXEC.

Return codes

0 link is already started or has successfully started.

4 an invalid parameter was passed or a required parameter was omitted. Reissue the command with the correct syntax.

8 unexpected link status encountered, check the NCI/XF log for error messages. Some possible common reasons for the MHOLINK to fail are:

- an MHOOPEN is still to be executed for the remote application, therefore no link is possible.
- an attempt has been made to establish an LU0 to LU0 link, but at least one of the VTAM APPL definitions has APPC=YES coded.
- an attempt has been made to establish an LU6.2 to LU6.2 link, but at least one of the VTAM APPL definitions does not have APPC=YES coded.

12 MHOSTART has not been issued or *&mh_id* is set incorrectly.

20 link is in the process of stopping. MHODLINK has been issued.

24 NCI/XF SIGNAL command failed; contact your local CKS office.

28 unable to update **mh_links* or **mh_tasks* queue; contact your local CKS office.

Technical notes

For LU0 the EXEC of an MHOLINK will result in a VTAM REQSESS macro being issued and the related LOGON and SCIP exits being driven.

For LU6.2 it will result in several VTAM APPCCMD macros being issued (OPRCNTL-CNOS, ALLOC-ALLOCD and DEALLOC-CONFIRM). This in turn will produce two sessions if the partners are parallel session capable.

For LU6.2 in low level mode it will result in a VTAM APPCCMD OPRCNTL-CNOS being issued.

For Sockets in high level mode it will result in several Sockets calls (Socket, getsockname, connect).

For Sockets in low level mode it is the user's responsibility to start a link.

The precise commands issued depend on the environments previously established using the MHOSTART and MHOOPEN EXECs. For LU0, Sockets and LU6.2 environment combinations the MHOLINK user specifies only the partner application name and NC-MHO scans the available tasks and their related protocols for the best fit. This scanning process is controlled by the specification of the *&mh_ordr* variable in the SOE. For example, if *&mh_ordr*= LU62SOCKLU0 and the remote application can only support LU0, the first session establishment would fail using LU6.2 (the second would also fail using Sockets). The LU0 protocol would then be tried successfully.

Examples

EXEC MHOLINK OTHRAPPL

allows connection to a remote NC-MHO protocol handler which uses application OTHRAPPL in the specified environment.

EXEC MHOLINK OTHRAPPL NC-ISC62 MHOMODE

allows connection to a remote NC-MHO protocol handler which uses application OTHRAPPL in an LU6.2 environment, using a locally defined mode name of MHOMODE.

EXEC MHOLINK OTHRAPPL ANAPPL

allows connection to a remote application, known to VTAM as OTHRAPPL in a low level environment from an NC-MHO protocol handler using application ANAPPL.

EXEC MHOSIDE OTHRSOCK 1024 9.42.67.143

EXEC MHOLINK OTHRSOCK

allows a connection to an MHO Sockets application running on host internet address 9.42.67.143 using port 1024.

MHOOPEN

The MHOOPEN EXEC opens/starts a protocol handler and is executed as follows:

EXEC MHOOPEN *nodename*

where

nodename is the application name to open. If not present the variable *&mh_acbn* must be set. For Sockets protocol, *nodename* must match an entry defined by the MHOSIDE EXEC.

Note: The following variables must be set prior to executing this routine:

&mh_id returned by the MHOSTART exec.

&mh_prot the type of protocol that this protocol handler is to support. The current values are LU0, LU62 or SOCK.

&mh_pass the optional application password (not applicable to Sockets).

&mh_jobn if your TCP/IP jobname is not TCPIP, specify jobname using this variable (Sockets only).

Return Codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOOPEN:

0 protocol handler opened/started successfully.

4 this protocol handler is already open.

8 unable to create an environment. No environment had been started for *&mh_id*. NC-MHO has tried but failed to automatically create one.

16 failure to start subsystem, see NCI/XF log for further details. A summary of reasons for failure is listed on the next page.

20 timeout doing open function. See *Timeout of synchronous events* on page 7.12 of the NCI/XF User Guide.

24 required parameter missing (*acbname* or *&mh_id*). For Sockets protocol, this may mean that the MHOSIDE EXEC has not been issued.

28 a protocol handler already exists for this protocol type.

Examples

```
SET  &mh_acbn = ANAPPL
SET  &mh_prot = LU62
EXEC  MHOOPEN
```

opens an LU6.2 protocol handler using VTAM application ANAPPL.

```
EXEC MHOSIDE ANAPPL 1024
EXEC MHOOPEN ANAPPL
```

opens a Sockets protocol handler using the local internet address and port 1024.

Subsystem initialization failure codes

If MHOOPEN completes and sets *&usrretc* to 16 a reason code for the subsystem initialization failure is displayed on the NCI/XF log with a message as listed below:

4 FINISH EXEC NOT SPECIFIED
No finish EXEC has been defined to NC-MHO.

5 APPLICATION NAME NOT SPECIFIED
No application name as been defined to NC-MHO.

6 INVALID TRACE VALUE
An invalid trace option has been detected by NC-MHO; the permitted values are TRACE and NOTRACE.

7 UNABLE TO LOAD MODULE
An error was detected while LOADING a module required for NC-MHO execution. The NC-MHO log will contain the name of the offending module. Please review the STEPLIB/LOADLIB library definition in the NCI/XF startup deck.

8 UNABLE TO OPEN APPLICATION
NC-MHO has tried to open the application control block (ACB) and has detected a failure; see the NC-MHO log for more details.

9 STOPPED WHILE WAITING FOR ACB TO BECOME ACTIVE.
The subsystem has been stopped while the application was being activated; this may be due to operator intervention.

10 ERROR DURING INITIALIZATION
NC-MHO has detected a failure while issuing a VTAM SETLOGON macro. Contact your local support office if the problem persists.

11 UNKNOWN PROTOCOL
An unsupported protocol has been defined to MHOOPEN in variable *&mh_prot*.

12 VTAM DOES NOT SUPPORT LU6.2
MHOOPEN has been requested to start a subsystem using LU6.2 protocols, but NC-MHO has detected that VTAM does not support LU62.

13 MH_AUTO NOT SET TO YES/NO.
The variable *&mh_auto* has not been set to YES or NO.

14 ATTACH OF SERVERTASK FAILED
NC-MHO has detected a failure during the ATTACH of the server task. If the problem persists please contact your local support office.

15 TCP/IP FOR MVS IUCV INITIALIZATION FAILED
This reason code can result in various messages being issued to the NCI/XF log. If the messages persist contact your local support office.

16 SOCKET SERVER INITIALIZATION FAILED
This reason code can result in various messages being issued to the NCI/XF log. If the messages persist contact your local support office.

Other reasons for subsystem initialization failure that may be reported in the log are:

- A timeout has occurred while starting the task. If this problem persists please contact your local support office.
- No start exec defined. Please note @MHSTASK is supplied and should not be deleted from the PBLIB library.

Note: An MHOOPEN command for a nonexistent or inactive application will wait until the application becomes available. If the MHOOPEN is for a Sockets node it will wait for TCP/IP address space to activate.

MHOSIDE

The MHOSIDE EXEC maintains a list of synonyms for TCP/IP partners and is executed as follows:

EXEC MHOSIDE *node* *port* [*internet-address*]

where

node	is 1 to 8 characters specifying the name given to an NC-MHO using the Sockets interface. This name is used in place of the application name in all other EXECs.
port	is a numeric field specifying the port number.
internet-address	is the internet address of the host where NC-MHO resides in dotted decimal format (for example 1.2.3.4). If omitted the local internet address is used.

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOSIDE.

0 successful execution.

4 invalid parameters.

Example

```
EXEC  MHOSIDE  MYHOST  1024
EXEC  MHOSIDE  OTHRHOST 1024  1.2.3.4
SET &mh_prot = SOCK
EXEC MHOOPEN MYHOST
EXEC MHOLINK OTHRHOST
```

Defines the name of the local Sockets application and the remote application on a remote host 1.2.3.4. The local protocol handler is opened, and used to connect to the remote partner.

MHOSTART

Before the NC-MHO communication facility can be used, a subsystem environment must be created using MHOSTART. The MHOSTART routine is executed as follows:

EXEC MHOSTART [*parm1*]

where

parm1 is optional and is the name of the Startup Options Exec (SOE). If not coded the value in variable *&mh_star* will be used. If no SOE is available (no *parm1* and variable *&mh_star* not set), the function of the SOE (the setting of environment related values), must be established prior to executing MHOSTART or the environment will be started with default values.

Variables

The following variables are used by MHOSTART to define the environment:

&mh_appc the name of the APPCMD completion EXEC. This is only applicable for LU6.2 protocols running in low level APPC mode. The default is no EXEC.

&mh_auto determines whether the subsystem is to run in Sockets, APPC, high or low level mode.

Y = HLAPI (default)

N = LLAPI

this parameter is only applicable where *&mh_prot* = LU62 or SOCK.

&mh_comp indicates that compression is required as default for the subsystem.

Y = compression required

N = compression not required (default)

&mh_ex01 the name of the encrypt/decrypt exit. It can be up to eight characters in length. The routine is written in Assembler and is given control before data is sent (for encryption) and after data is received (for decryption). The default is no encrypt/decrypt exit will be invoked.

&mh_init the name of the EXEC routine to be run when a link has been successfully created. The specified name can be up to eight characters long. The default is no EXEC.

&mh_lost the name of the EXEC routine to be run when a link has ended. The specified name can be up to eight characters in length. The default is no EXEC.

&mh_nse the name of the EXEC routine to be run when a link has failed to establish (the No Session Established (NSE) EXEC). The specified name can be up to eight characters long. The default is no EXEC.

&mh_ordr contains a list of protocol types that determine the order in which attempts will be made to establish a connection to a remote partner. Each protocol type can be up to four characters long. The default value is LU0 LU62SOCK (an LU0 link will be tried first and if that fails an LU62 link will be tried, followed by Sockets). Current valid protocols are LU0, LU62 and SOCK.

&mh_recv the name of the EXEC routine to be run when unformatted data is received by NC-MHO. The specified name can be up to eight characters long. The default is @MHUNFR. If the partner is an NC-MHO 1.1 this will default to @MHREC.

&mh_sndc is the name of the EXEC routine to be run after NC-MHO has completed a send to the remote partner.

&mh_sock the name of the EXEC routine to be run when a Sockets call completes in low level API mode. The default is no EXEC.

&mh_trac defines the default trace value for the subsystem.

Y = tracing on

N = tracing off (default)

MHOTRACE is a related function.

&mh_tpe the name of the EXEC routine to be run when a protocol has ended unexpectedly (for instance VTAM's TPEND has been driven because VTAM has stopped). The default is no EXEC.

&mh_x00 the name of the EXEC routine to be run to authorize a remote partner logging on to this NC-MHO. The specified name can be up to eight characters long. The default is no EXEC.

&mh_x01 the name of the EXEC routine to be run to authorize a SET request. The specified name can be up to eight characters long. The default is no EXEC.

&mh_x02 the name of the EXEC routine to be run to authorize an EXEC request. The specified name can be up to eight characters long. The default is no EXEC.

&mh_x03 the name of the EXEC routine to be run to authorize a RESOLVE request. The specified name can be up to eight characters long. The default is no EXEC.

&mh_x04 the name of the EXEC routine to be run to authorize a LOG request. The specified name can be up to eight characters long. The default is no EXEC.

Note: All these variables will be reset on completion of the EXEC.

Return codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOSTART:

0 successful start.

4 invalid or missing value; see log for more details.

8 the specified SOE does not exist.

12 the maximum number of environment subsystems (9999) has been exceeded.

16 MHOOPEN attempted but failed; see NCI/XF log for further details.

Following a successful execution of MHOSTART a four byte value will be returned in the variable *&mh_id*. This is the identity of the subsystem and must be set in variable *&mh_id* for all subsequent function calls.

Example

EXEC MHOSTART STARTOPT

The Startup Options EXEC STARTOPT will define the required parameters to establish a subsystem environment for protocol handlers.

MHSTATUS

The MHSTATUS EXEC queries the status of a protocol handler or a link to a remote application. It is executed as follows:

EXEC MHSTATUS *locnode* [*remnode*]

where

locnode	is the name of the local application whose subsystem status is required.
remnode	is the name of a remote application, which in combination with *locnode* defines the LINK whose status is being queried.

Variable *&mh_id* (returned by the MHOSTART EXEC) must be set prior to issuing this EXEC.

Variable *&¢mhprot* is returned containing the type of protocol being used for the subsystem link.

Note: if one parameter is set the status of a protocol handler is queried, if two parameters are set the status of LINK is queried.

Return Codes

The following return codes are set in *&usrretc* when querying the status of a protocol handler:

0	protocol handler running and ready.
4	protocol handler is waiting (not stopped but ended).
8	protocol handler is stopping but not ended.
12	protocol handler is stopped and ended.
16	no protocol handler found.
20	invalid task id.
28	missing or invalid mandatory parameter.
32	protocol handler is pending activation (started but not yet ready).

The following return codes are set when querying the status of a link:

0	link active.
1004	no link exists.
1008	link is pending activation.
1012	link is stopping.
1016	link is inactive.
1020	link is in error (this is a transient condition).
1024	link is in use (LU6.2 only).
1028	unknown link status.

Example

EXEC MHSTATUS ANAPPL OTHRAPPL

if the protocol handler using application ANAPPL is connected to remote application OTHRAPPL and is in use, using an LU6.2 protocol, a return code of 1024 is set in variable *&usrretc*.

MHOSTOP

The MHOSTOP EXEC terminates all functions relating to a subsystem environment and is executed as follows:

EXEC MHOSTOP

If there are any active protocol handlers or links using the specified environment they will also be terminated.

Variable *&mh_id* (returned by the MHOSTART EXEC) must be set prior to issuing this EXEC.

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOSTOP:

0 environment now stopped.

4 warning - all tasks already stopped.

8 timeout (waiting for MHOSTOP function to complete). See *Timeout of synchronous events* on page 7.12 of the NCI/XF User Guide.

12 required parameter *&mh_id* not set.

Example

```
SET &mh_id = 0001
EXEC MHOSTOP
```

terminates the environment identified by *&mh_id* = 0001.

MHOTRACE

The MHOTRACE EXEC traces transmission requests and is executed as follows:

EXEC MHOTRACE *parm1* [*parm2* [*parm3*]]

where

parm1 is one of the following:

TRACE	turns on NC-MHO tracing
NOTRACE	turns off NC-MHO tracing
MODULE	traces NCI/XF module invocation
VTAM	traces all NCI/XF VTAM requests
RPL	traces NCI/XF VTAM requests plus RPLs
DATA	as for RPL, plus any associated data
INTERNAL	traces NCI/XF internal control blocks
OFF	disables NCI/XF tracing
ALL	all NCI/XF trace options
PROG0	traces the main logic flow within NCI/XF panels
PROG1	traces all executed statements

parm2 applies only when *parm1* is set to TRACE or NOTRACE and is the local application name to which TRACE/NOTRACE applies. If omitted all local applications are TRACEd/NOTRACEd.

parm3 can only be entered if *parm 2* is specified. It is the remote application name to which TRACE/NOTRACE applies. If omitted all remote applications are TRACEd/NOTRACEd.

NOTRACE does not prevent some trace records relating to errors or records that cannot be associated with a particular session, being issued.

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOTRACE.

0 successful execution.

4 invalid parameters (probably invalid trace option).

8 NC-MHO not started.

16 trace request failed; see log for detail.

Note: MHOTRACE will alter the value of the NCI/XF traceterm mask.

Example

EXEC MHOTRACE ANAPPL

traces all NC-MHO activity between local protocol handlers using application ANAPPL

Using the NC-MHO environment

This section describes the EXEC routines available to perform remote procedure calls:

MHOEXEC	executes an NCI/XF EXEC routine on the remote NC-MHO.
MHOLOG	writes a log message to the remote NCI/XF log.
MHOREPLY	obtains a reply from a previous MHOEXEC, MHOLOG, MHORESLV or MHOSET EXEC that was preformed asynchronously.
MHORESLV	resolves an NCI/XF variable from the remote NC-MHO.
MHOSET	sets an NCI/XF variable on the remote NC-MHO.

MHOEXEC

An EXEC routine may be run on a remote NC-MHO by executing MHOEXEC as follows:

EXEC MHOEXEC *remote execname* [*execparm* [*replyid* [*replyexec*]]]

where

remote is the application name of the remote NC-MHO.

execname is the name of the EXEC routine to be run on the remote NCI/XF.

execparm is an optional parameter to pass to the EXEC when run via the variable *&usrex1*. If no parameter is required and the following fields are to be specified, enter *&sysnull* or a double quotes sign (”).

replyid is an optional parameter and can be one of the following:

ASY an eight character TOKEN will be returned in the variable *&mh_tokn* which may be used as the *replyid* on subsequent use of MHOREPLY when the success or failure of the MHOEXEC function is being checked.

SYN the MHOEXEC function will be performed synchronously. Control will be returned only after the function has completed or the timeout limit has been exceeded. See the NCI/XF User Guide, *Chapter 7 - A guide to NC-MHO*.

aa...a where *aa...a* represents up to eight characters which uniquely identify this MHOEXEC.

Note: the value in the variable *&usrterm* is reserved.

Specifying this causes notification to be passed to NC-MHO on completion of the MHOEXEC function. These characters or *replyid* can be used with MHOREPLY to check the success or failure of the MHOEXEC function.

If *replyid* is omitted the MHOEXEC function will be performed asynchronously and no indication of the success or failure of the operation can be obtained.

replyexec is an optional parameter that can only be entered if *replyid* is specified. It is the name of an EXEC to be executed automatically when NC-MHO is notified that the EXEC is completed. The variable *&mhusfld*, set by the *execname* will be returned. The return code from *execname* will be returned in the variable *&mhusret*.

Return codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOEXEC.

0 successful.

4 invalid parameters.

8 the same *replyid* was specified for a previous EXEC and has not been cleared with an EXEC of MHOREPLY.

16 NC-MHO not started.

20 error transmitting data; see NCI/XF log. If a linkage failure is reported see the section on MHOLINK return code 8 for more details.

24 timeout during synchronous EXEC.

28 remote EXEC authorization EXEC denied use of EXEC.

32 *&mh_id* not set.

36 MHOEXEC cannot be executed in low level mode.

40 EXEC *execname* does not exist on the remote NCI/XF.

Note: The EXEC routine specified by *execname* can include coding to assign variable *&mhusfld*. The data entered into *&mhusfld* can reflect the result of the *execname* routine. For instance, if the routine is required to validate certain criteria, the value YES or NO may be entered in *&mhusfld* to indicate a positive or negative result.

The variable will be examined by MHOREPLY. Its contents are entered into variable *&mhusfld* on the originating NCI/XF.

Example

The remote protocol handler is known as OTHRAPPL and its associated NCI/XF has the following EXEC:

```
)ROUTINE DEMOEXEC
IF &usrex1=ZERO
   SET &¢rc=0
   GOTO EXIT
ELSE
   SET &¢rc=4
LABEL EXIT
   SET &mhusfld=&usrex1
EXIT &¢rc
```

Following the above:

EXEC MHOEXEC OTHRAPPL DEMOEXEC ZERO SYN

results in the following variables being set on completion:

&mhusfld = ZERO

&mhusret = 0000

Also:

EXEC MHOEXEC OTHRAPPL DEMOEXEC XXXX SYN

results in the following variables being set on completion:

&mhusfld = XXXX

&mhusret = 0004

MHOLOG

A log record may be written to a remote NCI/XF log by executing MHOLOG as follows:

EXEC MHOLOG *remote* *log-message* [*replyid* [*replyexec*]]

where

remote is the application name of the remote NC-MHO.

log-message is the text of the message to be sent, to a maximum 32000 characters.

replyid is an optional parameter and can be one of the following:

ASY an eight character TOKEN will be returned in the variable *&mh_tokn* which may be used as the *replyid* on subsequent use of MHOREPLY when the success or failure of the LOG function is being checked.

SYN the MHOLOG function will be performed synchronously. Control will be returned only after the function has completed or the timeout limit has been exceeded. See the NCI/XF User Guide, *Chapter 7 - A guide to NC-MHO*.

aa...a where *aa...a* represents up to eight characters which uniquely identify this EXEC.

Note: The value in the variable *&usrterm* is reserved.

Specifying this causes notification to be passed to NC-MHO on completion of the MHOLOG function. These characters or *replyid* can be used with MHOREPLY to check the success or failure of the MHOLOG function.

If *replyid* is omitted the MHOLOG function will be performed asynchronously and no indication of the success or failure of the operation can be obtained.

replyexec is an optional parameter that can only be entered if *replyid* is specified. It is the name of a routine to be executed automatically when NC-MHO is notified that the MHOLOG function is complete.

Note: *&mh_id* (returned by the MHOSTART EXEC) must be set prior to issuing this EXEC.

Return codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOLOG:

0 successful function.

4 invalid parameters.

8 the *replyid* was not ASY or SYN, was used by a previous EXEC and has not been cleared with an EXEC of MHOREPLY.

16 NC-MHO has not been started using MHOSTART.

20 a problem has been detected contacting a remote NC-MHO; see the NCI/XF log for details. If a linkage problem was reported refer to MHOLINK error code 8 for more details.

24 a timeout has occurred doing a synchronous log; the LOG function may still be complete.

28 the remote NC-MHO has denied the LOG function in the LOG authorization EXEC.

32 *&mh_i*d has not been set.

36 MHOLOG cannot be issued in low level mode.

Example

EXEC MHOLOG OTHRAPPL 'THIS IS A TEST MESSAGE'

logs the text 'THIS IS A TEST MESSAGE' on the remote NCI/XF log. The remote protocol handler is known as OTHRAPPL.

MHOREPLY

The success or failure of an asynchronous EXEC such as MHOEXEC or MHOSET can be checked by calling the MHOREPLY routine as follows:

EXEC MHOREPLY *replyid* [CANCEL]

where

replyid	is the eight character code given on the asynchronous EXEC which identifies the transmission on the token returned in variable *&mh_tokn* if the ASY mode was used.
CANCEL	is an optional keyword which may be entered to cancel the specified *replyid* and associated parameters.

Return codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOREPLY:

0 reply received.

4 invalid MHOREPLY parameters.

8 EXEC *execname* not found for MHOEXEC or variable *varname* not found for MHORESLV.

16 *replyid* not found or variable *&mh_id* not set.

20 no reply received; LU to LU session ended.

28 remote denied function in an authorization EXEC.

99 reply is still outstanding.

Notes

If a *replyid* has been specified for an asynchronous EXEC, the same *replyid* can only be used again if MHOREPLY is executed using the *replyid* and the reply has completed, or the CANCEL option has been used.

If the completion code is 0 and the operation being checked was initiated through MHOEXEC, the variable *&mhusfld* as set by the EXEC on the remote NCI/XF will be examined by MHOREPLY and its contents written to variable *&mhusfld* on the originating NCI/XF. The return code set by the remote exec EXIT statement will be set in variable *&mhusret*.

If MHOREPLY is invoked from the *replyexec* option of an MHOSET or MHOEXEC routine, the value of *&mhusfld* will only then be available within that *replyexec*. If it is required outside the *replyexec*, it should be stored in a system variable or queue.

The data written to variable *&mhusfld* may reflect the result of the *execname* routine identified by *replyid*. For example, if the MHOEXEC *execname* was required to validate certain criteria, and was coded to set variable *&mhusfld*, the value YES or NO could be entered in *&mhusfld* to indicate a positive or negative result.

See *Transmission modes* on page 7.10 of the NCI/XF User Guide.

Example

EXEC MHOREPLY OTHRAPPL DEMOEXEC &sysnull AREPLYID

runs EXEC DEMOEXEC asynchronously where the remote protocol handler is OTHRAPPL.

The status of the operation can be interrogated by issuing the following command:

EXEC MHOREPLY AREPLYID

If the EXEC DEMOEXEC has not run then *&usrretc* will be set to 99, if it subsequently completes *&usrretc* will be set to 0.

MHORESLV

An NCI/XF variable at a remote NCI/XF can be resolved by executing MHORESLV as follows:

EXEC MHORESLV *remote varname replyid* [*replyexec*]

where

remote is the application name of the remote NC-MHO.

varname is the name of the variable on the remote NC-MHO to be resolved. Only system variables starting with an asterisk (*) should be used. The maximum length allowed is 250 characters.

replyid is one of the following:

ASY an eight character TOKEN will be returned in the variable *&mh_tokn* which may be used as the *replyid* on subsequent use of MHOREPLY when the success or failure of the MHORESLV function is being checked.

SYN the MHORESLV function will be performed synchronously. Control will be returned only after the function has completed or the timeout limit has been exceeded. See the NCI/XF User Guide, *Chapter 7 - A guide to NC-MHO*.

aa...a where *aa...a* represents up to eight characters which uniquely identify a remote NC-MHO. Specifying this causes notification to be passed to NC-MHO on completion of the MHORESLV function. These characters or *replyid* can be used with MHOREPLY to check the success or failure of the MHOREPLY function.

Note: The value in the variable *&usrterm* is reserved.

replyexec is an optional parameter that can only be entered if *replyid* is specified. It is the name of an EXEC to be executed automatically when NC-MHO is notified that the MHORESLV function is complete.

Note: Variable *&mh_id* (returned by the MHOSTART EXEC) must be set prior to issuing this EXEC.

If the MHORESLV function has been performed synchronously the variable *&mhusfld* will contain the value of the resolved variable. It will also contain the resolved value after MHOREPLY has been successfully completed for an MHORESLV.

Return codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHORESLV:

- 0 successful function.
- 4 invalid parameters.
- 8 the *replyid* was not ASY or SYN, was used by a previous EXEC and has not been cleared with an EXEC of MHOREPLY.
- 16 NC-MHO has not been started using MHOSTART.
- 20 a problem has been detected contacting the remote NC-MHO; see NCI/XF log for details. If a linkage failure was reported refer to MHOLINK return code 8 for more details.
- 24 a timeout has occurred doing a synchronous resolve; the resolve function may still be complete.
- 28 the remote has denied the RESOLVE function in the RESOLVE authorization EXEC.
- 32 error detected on remote during resolve function. The variable does not exist or its contents exceed 31,596 characters in length.
- 36 *&mh_id* has not been set or is invalid.
- 40 MHORESLV cannot be executed in low level mode.

Examples

EXEC MHOSET OTHRAPPL *DEMO 'HELLO' SYN

sets an NCI/XF variable &*DEMO where OTHRAPPL is the remote protocol handler.

EXEC MHORESLV OTHRAPPL *DEMO SYN

sets the variable *&mhusfld* to HELLO when it completes.

MHOSET

A variable on a remote NC-MHO can be set by executing MHOSET as follows:

EXEC MHOSET *remote varname varvalue* [*replyid* [*replyexec*]]

where

remote is the application name of the remote NC-MHO.

varname is the name of the variable to be set. Only system variables (starting with asterisk (*)) must be set to ensure that user access to such variables is not restricted. The maximum length allowed is 250 characters.

Note: the name of the variable should be specified with the leading ampersand (&) omitted.

varvalue is the value to set in the variable. If omitted the variable will be reset. The maximum length is 32000 characters.

replyid is an optional parameter and can be one of the following:

ASY an eight character TOKEN will be returned in the variable *&mh_tokn* which may be used as the *replyid* on subsequent use of MHOREPLY when the success or failure of the MHOSET function is being checked.

SYN the MHOSET function will be performed synchronously. Control will be returned only after the function has completed or the timeout limit has been exceeded. See the NCI/XF User Guide, *Chapter 7 - A guide to NC-MHO*.

aa...a where *aa...a* represents up to eight characters which uniquely identify this SET. Specifying this causes notification to be passed to NC-MHO on completion of the MHOSET function. These characters or *replyid* can be used with MHOREPLY to check the success or failure of the MHOSET function.

Note: The value in the variable *&usrterm* is reserved.

If *replyid* is omitted the MHOSET function will be performed asynchronously and no indication of the success or failure of the operation can be obtained.

replyexec is an optional parameter that can only be entered if *replyid* is specified. It is the name of a routine to be executed automatically when NC-MHO is notified of the completion of the SET variable operation.

Return codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHOSET:

0 successful.

4 invalid parameters.

8 the same *replyid* was specified for a previous MHOEXEC and has not been cleared with an EXEC of MHOREPLY.

16 NC-MHO not started.

20 no link available to remote NC-MHO, or problems sending data. See NCI/XF log for more details. If a linkage failure was reported refer to MHOLINK return code 8 for more details.

24 timeout value exceeded during synchronous set.

28 remote SET authorization EXEC denied use of SET.

32 *&mh_id* not set or invalid.

36 MHOSET not permitted in low level mode.

Example

EXEC MHOSET OTHRAPPL *test TESTING-MHOSET SYN

sets variable *&*test* to the value TESTING-MHOSET on the remote NCI/XF where the remote protocol handler is OTHRAPPL. The effect is equivalent to specifying SET &*test = TESTING-MHOSET on the remote NCI/XF.

As the SYN option has been specified, control will be returned only after confirmation that the SET function has been actioned.

Customization (EXEC and exit points)

This section describes the optional user EXIT points to customize the NC-MHO system.

User authorization exit points

The following user authorization exit points enable users to allow or disallow various MHO functions. The names indicated are those of the supplied sample user authorization exits but any name can be used provided the appropriate variable is set in the user SOE. If the variable for a particular exit is not set, the associated exit point will not be taken.

```
&MH_X00 = MHZAXSES   /*SESSION AUTH EXEC NAME
&MH_X01 = MHZAXSET   /*SET AUTH EXEC NAME
&MH_X02 = MHZAXEXE   /*EXEC AUTH EXEC NAME
&MH_X03 = MHZAXRES   /*RESOLVE AUTH EXEC NAME
&MH_X04 = MHZAXLOG   /*LOG AUTH EXEC NAME
```

The supplied exit points, detailed in this section may be subdivided as follows:

- EXEC exit points
- Authorization EXECs
  - Logon authorization
  - Set authorization
  - Exec authorization
  - Resolve authorization
  - Log authorization
- Environment EXECs
  - Session established
  - Lost session exec
  - No session established
  - Protocol end exec
  - Send completion
  - Receive completion
  - Subsystem end
- Assembler EXIT points
  - Encrypt/decrypt exit

EXEC exit points

All environments have exit points at which locally developed EXECs may be executed. The following variables in the SOE define the EXEC names to be run for the given events:

&mh_appc run when an APPCCMD invocation has completed; this is only invoked in a low level APPC environment.

&mh_fini run at NC-MHO subsystem shutdown, resulting from an MHOSTOP.

&mh_init run when a session setup is complete, resulting from an MHOLINK.

&mh_lost run when a session is terminated due to error or any other reason.

&mh_nse run when a session setup fails.

&mh_recv run when a receive operation has completed.

&mh_sndc run when a send operation has completed.

&mh_sock run when a Sockets call has completed; this is only invoked in low level Sockets environments.

&mh_tpe run when a protocol handler environment has terminated, for example VTAM has ended.

Logon authorization EXEC

This EXEC is invoked as part of session establishment between two applications and can be used, for example, to reject a session between two particular applications.

Input variables

&mh_acb current application name.

&mh_comp compression available (Y or N).

&mh_func function code set to I.

&mh_id environment id

&mh_prot protocol (LU62, LU0 or SOCK).

&mh_racb remote application name.

&mh_rcmp is Y if the session has been established using compression or N if the session has been established without compression.

Further input variables are available depending on the protocol being used to establish the session as described below:

Variables only available with LU6.2 protocol (indicated by the variable *&mh_prot* having the value LU62):

&mh_auto auto mode will be set to Y.

&mh_mode mode name.

&mh_sslu single session will be set to Y if the session partner is a single session application.

&mh_udat PIP data.

Variables only available with LU0 protocol (indicated by the variable *&mh_prot* having the value LU0):

&iscuser user data.

Variables only available with Sockets protocol (indicated by the variable *&mh_prot* having the value SOCK):

&mh_adrl local internet address.

&mh_adrr remote partner internet address.

&mh_prtl local port number.

&mh_prtr remote partner port number.

For details of the variables available to this EXEC refer to the *Available variables* on page 6.50. The following notes apply to the variables listed there.

Return codes

0 session is authorized (accept session).

4 session is not authorized (reject session).

Set authorization EXEC

This EXEC is invoked as part of the MHOSET function prior to the variable being set and may be used, for example, to disallow the setting of a particular variable.

Input variables

&usrex1	function code. The value is S for set exit.
&usrex2	name of the variable to be set, up to 250 characters without a leading ampersand (&).
&usrex3	data to which the variable will be set.
&usrex4	NC-MHO subsystem identifier.
&usrex5	application name of remote user requesting the set.

Return codes

0	authorized, set the variable.
non-zero	not authorized, do not set the variable.

EXEC authorization EXEC

This EXEC is invoked as part of the MHOEXEC function prior to the running of the EXEC and may be used, for example, to disallow the execution of a variable.

Input variables

&usrex1	function code. The value is E for EXEC exit.
&usrex2	name of the EXEC to be executed, up to eight characters.
&usrex3	data to be passed to the routine.
&usrex4	NC-MHO subsystem identifier.
&usrex5	application name of the remote user requesting the EXEC.

Return codes

0	authorized, execute the routine.
non-zero	not authorized, do not execute the routine.

Resolve authorization EXEC

This EXEC is invoked as part of the MHORESLV function prior to the variable being set resolved, and may be used, for example, to disallow the resolution of a variable.

Input variables

&usrex1	function code. The value is R for resolve exit.
&usrex2	name of the variable to be resolved, up to 250 characters.
&usrex3	null operand for compatibility with other exits.
&usrex4	NC-MHO subsystem identifier.
&usrex5	application name of the remote user requesting the resolve.

Return codes

0	authorized, resolve the variable
non-zero	not authorized, do not resolve the variable.

Log authorization EXEC

This EXEC is invoked as part of the MHOLOG function prior to the data being written to the NCI/XF log and may be used, for example, to disallow the writing of sensitive information (such as passwords), to the NCI/XF log.

Input variables

&usrex1	function code. The value is L for log exit.
&usrex2	data to be written to the NCI/XF log.
&usrex3	null operand for compatibility with other exits.
&usrex4	NC-MHO subsystem identifier.
&usrex5	application name of the remote user requesting write to log.

Return codes

0	authorized, write data to NCI/XF log
non-zero	not authorized, do not write data to NCI/XF log

Session established EXEC

This EXEC is driven when session connection setup has been successfully completed as a result of previous commands. It is defined by setting the variable *&mh_init* prior to starting the NC-MHO task.

For details of the variables available to this exit refer to *Available variables* on page 6.50. The following notes apply to the variables listed there.

Protocol (*&mh_prot*)
This field will be set to LU0, LU62 or SOCK.

Compression (*&mh_comp*)
This field will be set to Y if compression is locally supported, and N if not. The variable *&mh_rcmp* contains a Y or N to indicate whether the session partner supports compression or not.

Low level mode (*&mh_auto*)
This field will be set to N if the conversation is being run in low level mode, and Y if not.

Single session (*&mh_sslu*)
This field will be set to Y if the conversation is not using parallel sessions in LU6.2 mode.

Lost session EXEC

This EXEC is driven when an active session connection ends. This should be distinguished from the NSE exit which is driven when session connection setup fails.

The exit is defined by setting the variable *&mh_lost* prior to issuing the MHOSTART EXEC.

The exit is driven under the following circumstances:

- when the protocol being used is LU0 and the VTAM NSEXIT is driven for cleanup, or when the VTAM LOSTERM exit is driven.
- when the protocol is LU6.2 and the VTAM ATTN exit is driven for loss.
- when the protocol is Sockets and the outstanding receive completes with an indication of error or Socket closed.

For details of the variables available to this exit refer to *Available variables* on page 6.50. The following notes apply to the variables available to this EXEC.

Return code (*&mh_retc*) If *&mh_prot* = LU62 or SOCK
This field contains the session deactivation code set in the RPL extension.
If *&mh_prot*=LU0 then *&mh_retc* is not set.

Protocol (*&mh_prot*)
This field takes the value LU0, LU62 or SOCK for LU0, LU6.2 or SOCK tasks respectively.

Sense code (*&mh_sens*)
MHO sets the following sense codes when running in APPC mode to indicate invalid parameters during initial conversation setup:

08050000	session limit > 1.
080F0000	access denied by partner logon authorization EXEC.
08350000	PIP data invalid.
0835000A	TPN invalid.
08350018	compression requested but not supported.
08350019	APPC/low level APPC mode mismatch.

No session established EXEC

The No Sessions Established (NSE) EXEC is driven when a session setup or connection fails and should be distinguished from the exit driven when an active session ends due to a LOSTERM (LU0) or ATTN-LOSS (LU6.2) exit.

The exit is defined by setting the variable *&mh_nse* prior to issuing the MHOSTART EXEC.

Specifically the exit is driven under the following circumstances:

- when the protocol being used is LU0 and a REQSESS MACRO fails.
- when the protocol being used is LU6.2 and an OPRCNTL-CNOS, ALLOC-ALLOCD, or DEALLOC-CONFIRM MACRO fails during the initial transaction invoked by the MHOLINK EXEC.
- when the protocol being used is Sockets and a Socket or Connect call fails.

For details of the variables available to this EXEC refer to *Available variables* on page 6.50.

Protocol end EXEC

This EXEC is driven when:

8. The VTAM TPEND EXIT is driven which is usually as a result of a VTAM crash or an NC-MHO application being inactivated by an operator command.
9. A sever interrupt is received.

The exit is defined by setting the variable *&mh_tpe* in the Startup Options Exec (SOE) or prior to issuing the MHOSTART EXEC.

For details of the variables available to this exit refer to *Available variables* on page 6.50. The following notes apply to the variables available to this EXEC.

Protocol (*&mh_prot*)
This field takes the value LU0, LU62 or SOCK

Return code (*&mh_retc*)
This field contains the VTAM TPEND code if running with VTAM based protocols, or the first 8 bytes of the sever interrupt text if running with TCP/IP protocols.

Send completion EXEC

This EXEC is driven when a send operation completes.

The suggested purpose of this EXEC is to override the standard action which is to shutdown the session/connection if an error is detected during a send operation. If the EXEC sets a non-zero return code NC-MHO will take no further action; it becomes the EXEC's responsibility to take any necessary recovery action or issue an MHODLINK EXEC to terminate the session. A zero return code will allow the operation to continue.

The exit is defined by setting the variable *&mh_sndc* in the Startup Options EXEC (SOE) or prior to issuing MHOSTART.

For details of the variables available to this exit refer to *Available variables* on page 6.50.

Receive completion EXEC

This EXEC is driven when a receive operation from an NC-MHO partner completes. It is defined by setting variable *&mh_recv* prior to executing MHOSTART.

For details of the variables available to this EXEC see *Available variables* on page 6.50.

If no receive EXEC is defined a value of @MHUNFR is assumed. Care should be taken to ensure that no EXEC of this name exists if you are using NC-MHO in APPC mode.

Subsystem end EXEC

This EXEC is driven when an NC-MHO subsystem ends (when MHOSTOP is run). It is defined by setting variable *&mh_fini* before executing MHOSTART.

Variable *&mh_id* is set to the identity of the subsystem that is ending.

Encrypt/decrypt exit

An exit point is available to NC-MHO to allow encryption/decryption of data. The exit is enabled by setting the name of a module which contains the encryption/decryption routine into the *&mh_ex01* variable in the Startup Options Exec (SOE) defined on the MHOSTART command.

A sample Assembler user exit is provided called SCENCDEC (in the sample source library) that can be called to encrypt/decrypt data. It adds 1 to every byte to encrypt data before it is sent and subtracts 1 from every byte to decrypt it after receipt. This is just one example of its use; if the data length is not altered the exit can be modified to suit any requirement.

On entry to this exit, R1 points to a parameter list. The parameter list contains information such as:

- whether the exit is being called for ENCRYPTION or DECRYPTION
- from where the data originated
- to whom the data has been sent
- the data and its length

The parameter list is mapped by the EXIT macro found in the distributed macro library.

If new exits are written, modules must be written in Assembler and link-edited RMODE(24), AMODE(24), reentrant, reusable AC(1).

Available variables

The following list describes the variables available to the exits with a brief description of their contents. It should be used in conjunction with the cross reference tables which follow this subsection.

&iscacbn Up to eight bytes containing the local application name.

&iscdata Contains user data, sent to or received from the remote partner.

&ischost Up to eight bytes containing the remote application name.

&iscn800 Contains the number of times a RTNCD/FDB2 of 0800 (VTAM short of storage) was detected.

&iscnlus Contains the number of links that were established.

&iscnrcv Contains the number of data transfers from the remote application.
For the Lost Session EXEC it is the number for that session.
For the Protocol End EXEC it is the number for that protocol handler.

&iscnsnd Contains the number of data transfers to the remote application.
For the Lost Session EXEC it is the number for that session.
For the Protocol End EXEC it is the number for that protocol handler.

&iscreca Contains the recovery code from the requested function (generally the contents of register 0).

&iscretc Contains the general completion code of the requested function (generally the contents of register 15).
For the TPEND function it contains the reason for shutdown.
For the Lost Session EXEC it contains the deactivation type code if the protocol in use is LU6.2.

&iscuser Contains user CINIT data.

&mh_acb Up to eight bytes containing the local application name.

&mh_adrl (Sockets only) Contains the local internet address in dotted decimal format.

&mh_adrr (Sockets only) Contains the remote internet address in dotted decimal format.

&mh_auto For LU6.2 and Sockets only - set to Y for HLAPI and N for LLAPI.

&mh_ccst For LU6.2 and VTAM 3.3 onwards only - contains the current conversation state (RPL6CCST) and contains one of the following values:

END-CON
P-DEALLOC
P-RECLOG
P-SEND
RECEIVE
REC-CON
REC-COND
REC-CONS
RESET
SEND

&mh_comp Set to Y if compression is available.

&mh_data Contains user data sent to or received from the remote partner.

&mh_func Contains the function code for the logon authorization EXEC and contains the character I.

&mh_id Contains the subsystem ID.

&mh_mode For LU6.2 contains the mode name.

&mh_msgn (Sockets only) Contains the IUCV message number.

&mh_n070 Contains the number of times an RCPRI/RCSEC of 0070000 (VTAM short of storage) was detected.

&mh_nrcv Contains the number of data transfers from the remote application.
For the Lost Session EXEC it is the number for that session.
For the Protocol End EXEC it is the number for that protocol handler.

&mh_nsnd Contains the number of data transfers to the remote application.
For the Lost Session EXEC it is the number for that session.
For the Protocol End EXEC it is the number for that protocol handler.

&mh_prim For LU62 only, contains the primary return code of the related VTAM APPCCMD (RPL6RCPR).

&mh_prm (Sockets only) Contains the IUCV PRMMSG field.

&mh_prot Contains the protocol to use or in use. (Up to four bytes in length). The current values are LU0, LU62 and SOCK.

&mh_prtl (Sockets only) Contains the local port number.

&mh_prtr (Sockets only) Contains the remote port number.

&mh_qual For LU6.2 only, contains the QUALIFY= field of the VTAM APPCCMD and will be one of the following values:

ABNDPROG	DATACON
ABNDSERV	DATAFLU
ABNDTIME	DEFINE
ABNDUSER	DISPLAY
ACTSESS	ERROR
ALLOCD	FLUSH
ANY	IMMED
CNOS	RESUME
CONFIRM	RQSEND
CONFIRMD	SPEC
CONV	SESSION
CONWIN	SUSPEND
DACTSESS	UNDEFND
DATA	

&mh_racb Up to eight bytes containing the remote application name.

&mh_rcmp Contains Y if the link has been established using compression and N if not.

&mh_rcv1 For LU6.2 only, contains the CONTROL= field of the VTAM APPCCMD. It contains a string of nine Y or N flags indicating various bit settings in the RPL extension WHAT-RECEIVED flag (RPL6RCV1), in the following order:

DATA
DATA-COMPLETE
DATA-INCOMPLETE
SEND
CONFIRM
DEALLOC
LOG-DATA
PS-HEADER (not VTAM 3.2)
PARTIAL PS-HEADER (not VTAM 3.2)

&mh_reca Contains the recovery code from the requested function (generally the contents of register 0).

&mh_rep (Sockets only) Contains the IUCV reply buffer.

&mh_retc Contains the general completion code of the requested function.
For the TPEND function it contains the reason for shutdown.
For the Lost Session EXEC under LU6.2 protocol it contains the deactivation type code.

&mh_req For LU6.2 only, contains the CONTROL= field of the VTAM APPCCMD and will be one of the following values:

ALLOC
DEALLOC
OPRCNTL
PREPRCV
RCVFMH5
RECEIVE
REJECT
RESETRCV
SEND
SETSESS

&mh_rtcd Contains the RPLRTNCD/RPLFDB2 value for the requested function.

&mh_rtun For LU6.2 only, contains the returned indicator (RPL6RTUN) from the related VTAM APPCCMD.

&mh_sckt (Sockets only) Contains the Socket number.

&mh_sec For LU6.2 only, contains the secondary return code of the related VTAM APPCCMD (RPL6RCSC).

&mh_sens contains the sense code (RPLFDBK2) for the requested function.

&mh_sig For LU6.2 only, contains any signal data returned to the VTAM APPCCMD. (RPL6SIG1).

&mh_sslu For LU6.2 only, contains Y if the session is a single session application.

&mh_trg (Sockets only) Contains the IUCV TRGCLS field.

&mh_udat For LU6.2, contains PIP data. For Sockets, contains the user data in the signon dataflow.

&mh_ufld Contains the RPLUSFLD of the requested function.

Note: More information on the meaning of variables representing VTAM error conditions may be found in the VTAM programming manual chapters entitled *Handling errors* and *Special conditions*.

Variables available to each exit

The following tables document the variables available to each exit.

The first character represents the maximum length of the variable:

1-9	number of characters
*	up to 32000 bytes

The second character represents the format of the variable:

C	alphanumeric/national
X	hexadecimal

NC-MHO 1.1 variables - these variables are retained for compatibility:

EXEC	VARIABLE					
	&ISCACBN current appl name	&ISCDATA user data	&ISCHOST remote appl name	&ISCN800 LU0 short on storage	&ISCNLUS number of links	&ISCNRCV number of receives
Session established	8C		8C			
Logon authorization	8C		8C			
Lost session	8C		8C			
No session established	8C		8C			
Protocol end	8C					
Send completion	8C	*C	8C			
APPCCMD completion	8C	*C	8C			
Receive completion	8C	*C	8C			
Subsystem end	8C		8C	8X	8X	8X

EXEC	VARIABLE			
	&ISCNSND number of sends	&ISCRECA recovery code	&ISCRETC return code	&ISCUSER user data
Session established				*C
Logon authorization				*C
Lost session			8X	
No session established		8X	8X	
Protocol end			8X	
Send completion			8X	*C
APPCCMD completion		8X	8X	
Receive completion				*C
Subsystem end	8X	8X	8X	*C

NC-MHO 1.2 variables

EXEC	VARIABLE										
	&MH_ACB current appl name	&MH_ADRL local internet addr	&MH_ADRR remote internet addr	&MH_AUTO low level mode	&MH_CCST conversation state	&MH_COMP compression	&MH_DATA user data	&MH_FUNC function code	&MH_ID environment	&MH_MODE mode name	&MH_MSGN IUCV message number
Session established	8C	*C	*C	1C		1C			4C	8C	
Logon authorization	8C	*C	*C	1C		1C		1C	4C	8C	
Lost session	8C	*C	*C						4C	8C	
No session established	8C	*C	*C		8C				4C		
Protocol end	8C	*C							4C		
Send completion	8C	*C	*C		8C		*C		4C		
APPCCMD completion	8C			1C	8C	1X	*C		4C	8C	
Receive completion	8C	*C	*C				*C		4C		
Subsystem end									4C		
Sockets completion	8C	*C	*C				*C	1C	4C		*C

EXEC	VARIABLE										
	&MH_N070	&MH_NRCV in data count	&MH_NSND out data count	&MH_PRIM primary retcode	&MH_PRM IUCV PRMMSG field	&MH_PROT protocol	&MH_PRTL local port number	&MH_PRTR remote port number	&MH_QUAL qualify value	&MH_RACB remote appl name	&MH_RCMP compression remote
Session established						4C	*C	*C		8C	
Logon authorization						4C	*C	*C		8C	1C
Lost session		8C	8C			4C	*C	*C		8C	
No session established				2X		4C	*C	*C	2X	8C	
Protocol end		8C	8C			4C	*C				
Send completion				2X		4C	*C	*C	2X	8C	
APPCCMD completion				2X		4C			2X	8C	1C
Receive completion						4C	*C	*C		8C	
Subsystem end	8X										
Sockets completion					*C	4C	*C	*C		8C	

NC-MHO 1.2 variables

EXEC	VARIABLE									
	&MH_RCV1 what received	&MH_RECA recovery code	&MH_REP IUCV reply buffer	&MH_RETC return code	&MH_REQ control value	&MH_RTCD RTNCD FDBK2	&MH_RTUN returned indicator	&MH_SCKT socket number	&MH_SEC secondary retcode	&MH_SENS sense code
Session established								*C		
Logon authorization								*C		
Lost session				8X				*C		8X
No session established	9C	8X		8X	2X	4X	8C	*C	2X	8X
Protocol end				8X						
Send completion	9C	8X		8X	2X	4X	8C	*C	2X	8X
APPCCMD completion	9C	8X		8X	2X	4X	8C		2X	8X
Receive completion		8X				4X		*C		8X
Subsystem end										
Sockets completion			*C	8X		4X		*C		

EXEC	VARIABLE					
	&MH_SIG signal data	&MH_SSLU single session	&MH_TRG IUCV TRGCLS field	&MH_UDAT user data	&MH_UFLD RPL user field	&MH_TRGC IUCV function
Session established		1C		*C		
Logon authorization		1C		*C		
Lost session						
No session established	8X				8X	
Protocol end						
Send completion	8X			*C	8X	
APPCCMD completion	8X	1X		*C	8X	
Receive completion				*C	8X	
Subsystem end						
Sockets completion			*C			8C

Using the NC-MHO low level APPC environment

The MHOAPPC EXEC

The MHOAPPC EXEC is used when NC-MHO is run in low level APPC mode for LU6.2 connections. (When *&mh_auto* is set to N in the Startup Option Exec (SOE) passed to the MHOSTART EXEC). Some knowledge of LU6.2 protocols is required to use this interface. The IBM manual Programming for LU6.2 contains general and coding details for LU6.2 including the APPCCMD macro on which the MHOAPPC EXEC is based. It is also necessary to understand the MHOAPPC completion EXEC which is driven on completion of all MHOAPPC executions (see heading *APPCCMD completion exit* on page 6.66).

Format

EXEC MHOAPPC *remacb control qualify* [*options*.....]

where

remacb is the one to eight character alphanumeric/national name of the remote application involved in the interaction.

control is the IBM APPCCMD MACRO CONTROL keyword. The *control* keyword is used in conjunction with the *qualify* keyword and associated *options*. Specific *qualify* keywords apply to each *control* keyword. Keyword combinations that are NOT supported are listed on the next page. Supported *control* keywords are ALLOC, DEALLOC, OPRCNTL, PREPRCV, RECEIVE, REJECT and SEND; each of these are described, with their associated *qualify* keywords, on subsequent pages.

qualify is the IBM APPCCMD MACRO QUALIFY keyword. The *qualify* keyword is used in conjunction with the *control* keyword and associated *options*. Specific *qualify* keywords apply to each *control* keyword. Keyword combinations that are NOT supported are listed on the next page. Descriptions for each of the supported *control* keywords and associated *qualify* keywords are provided on subsequent pages.

options..... are optional keywords, specific to each *control/qualify* combination. Descriptions for each of the supported options are provided on subsequent pages under the appropriate *control* keyword description.

Return Codes

On completion, one of the following codes will be set in variable *&usrretc* to indicate the success or failure of MHOAPPC:

0 command completed successfully.

4 invalid parameter(s).

8 no link exists. If NC-MHO is starting a link with an LU6.2 capable partner, MHOLINK must be issued first.

Non-supported keywords

The *control* and *qualify* keyword combinations shown below are NOT supported by the MHOAPPC EXEC (however some of these are issued in the course of MHOAPPC EXEC processing as indicated):

Non-supported *control* and *qualify* keywords		
control	***qualify***	**notes**
CHECK	-	issued on completion of all MHOAPPC operations.
OPRCNTL	ACTSESS	always issued by NC-MHO to accept session establishment requests. There is no user supported environment in which this operation can be meaningfully issued.
OPRCNTL	DACTSESS	
OPRCNTL	DEFINE	
OPRCNTL	DISPLAY	
RCVFMH5	RCVFMH5	always issued when the ATTN exit is driven for an FMH5 event. There is no user supported environment in which this operation can be meaningfully issued.
RECEIVE	ANY	
REJECT	SESSION	
RESETRCV	-	
SETSESS	RESUME	
SETSESS		

Supported keywords

Supported *control* keyword entries are described on the following pages. Details of the appropriate *qualify* keywords are provided in each case.

For information on *control/qualify* combinations, refer to the chapter entitled *LU6.2 macro instruction syntax and operands* in the VTAM Programming for LU6.2 manual.

ALLOC

The *control* keyword ALLOC is used to allocate a conversation. It is typically used after an MHOLINK EXEC has been successfully issued (this issues an OPRCNTL-CNOS verb to negotiate session limits with the local VTAM). The issuer is in SEND state when this verb completes.

The ALLOC keyword is used in the MHOAPPC EXEC as follows:

EXEC MHOAPPC *remacb* ALLOC *qualify* [*tpn*] [*mode*]

where

remacb is the one to eight character alphanumeric/national name of the remote application involved in the interaction.

qualify is one of the following:

ALLOCD if a session is not available, VTAM will attempt to create one for this new conversation subject to the constraints imposed by session limits; if no sessions are available VTAM queues the request.

CONWIN similar to ALLOC except VTAM considers only contention winner sessions.

IMMED attempts to use a contention winner session; if none are available the ALLOC fails.

tpn is the Transaction Program Name to be executed by the recipient of the ALLOC command. This is particularly useful for communicating with non NC-MHO partners (for instance CICS). The default name is NC-ISC62. The name can consist of one to eight alphanumeric/national characters.

mode is the mode name to be used with this conversation. The default is RECORD. A *tpn* must be specified before this keyword. The name can consist of one to eight alphanumeric/ national characters.

DEALLOC

The *control* keyword DEALLOC terminates a conversation and is typically used to signal the end of a unit of work or transaction. The issuer is in END CONVERSATION state when this verb completes.

The DEALLOC keyword is used in the MHOAPPC EXEC as follows:

EXEC MHOAPPC *remacb* DEALLOC *qualify* [*sense*]

where

remacb is the one to eight character alphanumeric/national name of the remote application involved in the interaction.

qualify is one of the following:

ABNDPROG indicates a transaction program error.

ABNDSERV indicates a service program error.

ABNDTIME indicates a timeout condition.

ABNDUSER indicates a user defined error; the sense code field may be used to indicate a specific error. The first byte of the sense code should be 00 as these are reserved for user errors; setting other values may conflict with existing IBM sense codes.

CONFIRM indicates that the DEALLOC will wait until a SEND-CONFIRMED is issued by the conversation partner and so can be used to ensure all data has been delivered before terminating a conversation.

DATACON is similar to CONFIRM but the *&mh_data* variable contains data sent with this verb.

DATAFLU terminates the transaction after sending any data in the *&mh_data* variable. All data is FLUSHed to the session partner.

FLUSH FLUSHes any data still locally accumulated and terminates the conversation.

sense can only be used with ABNDUSER. This is eight hexadecimal digits describing a sense code which will be set in variable *&mh_sens* in the APPCCMD completion EXEC and sent to the conversation partner. The issuer is in END CONVERSATION state when this verb completes. The default is 00000000.

OPRCNTL

The *control* keyword OPRCNTL is used to negotiate session limits for subsequent conversations. This MHOAPPC command is issued internally by the MHOLINK EXEC to establish the NCI/XF environment for those conversations and is not usually issued by the application developer. The issuer is in RESET status following this verb.

The OPRCNTL keyword is used in the MHOAPPC EXEC as follows:

EXEC MHOAPPC *remacb* OPRCNTL *qualify* [*tpn*] [*mode*] [*limits*]

where

remacb is the one to eight character alphanumeric/national name of the remote application involved in the interaction.

qualify is the keyword CNOS, which indicates control and negotiation of sessions.

tpn is the Transaction Program Name to be executed by the recipient of the OPRCNTL command. This is particularly useful for communicating with non NC-MHO partners (such as CICS). The default name is NC-ISC62. The name can consist of one to eight alphanumeric/national characters.

mode is the mode name to be used with this conversation. The default is RECORD. A *tpn* must be specified before this keyword. The name can consist of one to eight alphanumeric/ national characters.

limits is 14 hexadecimal digits specifying the session link block. See the chapter entitled *CNOS session limits control block* in the VTAM Programming manual for more details.

PREPRCV

The *control* keyword PREPRCV sends/flushes any data locally accumulated and places the issuer in a RECEIVE state. The issuer must be in a SEND state prior to the verb. Typically the verb would be followed by a RECEIVE SPEC to accept data/signals from the conversation partner.

The PREPRCV keyword is used in the MHOAPPC EXEC as follows:

EXEC MHOAPPC *remacb* PREPRCV *qualify*

where

remacb is the one to eight character alphanumeric/national name of the remote application involved in the interaction.

qualify is one of the following:

CONFIRM indicates that the PREPRCV will wait until the remote partner issues a SEND_CONFIRMED.

DATACON is similar to CONFIRM but the *&mh_data* variable contains data sent with this verb.

DATAFLU transfers data and flushes all stacked data.

FLUSH flushes all stacked data.

RECEIVE

The *control* keyword RECEIVE receives any data queued to that conversation or waits for data/signals. The issuer must be in RECEIVE state.

The RECEIVE keyword is used in the MHOAPPC EXEC as follows:

EXEC MHOAPPC *remacb* RECEIVE *qualify*

where

remacb	is the one to eight character alphanumeric/national name of the remote application involved in the interaction.
qualify	is the keyword SPEC which receives data/signals for the current conversation.

REJECT

The *control* keyword REJECT terminates a conversation and can be issued in almost any conversation state and is typically used as a last resort during error recovery. It causes the lost session EXEC to be driven for both session partners (or the equivalent of the VTAM ATTN exit for the LOSS function). The qualify value is as follows:

The REJECT keyword is used in the MHOAPPC EXEC as follows:

EXEC MHOAPPC *remacb* REJECT *qualify* [*sense*]

where

remacb is the one to eight character alphanumeric/national name of the remote application involved in the interaction.

qualify is the keyword CONV which terminates current conversation.

sense is eight hexadecimal digits describing a sense code which will be set in variable *&mh_sens* in the APPCCMD completion EXEC and sent to the conversation partner. The issuer is in END CONVERSATION state when this verb completes. The default is 00000000.

SEND

The *control* keyword SEND is used to send data to a conversation partner and can only be issued in *send* state except SEND-ERROR, SEND-RQSEND and SEND-CONFIRMD.

The SEND keyword is used in the MHOAPPC EXEC as follows:

EXEC MHOAPPC *remacb* SEND *qualify* [*sense*]

where

remacb is the one to eight character alphanumeric/national name of the remote application involved in the interaction.

qualify is one of the following:

- CONFIRM — issued to request the remote partner to respond with SEND-CONFIRMED.
- CONFIRMD — issued in RECEIVE state to confirm receipt of data in response to a partner issuing commands such as DEALLOC-CONFIRM, PREPRCV-DATACON etc.
- DATA — queues data to the local VTAM and may flush the data if the local buffers are full. The *&mh_data* variable should contain the data to send.
- DATACON — flushes data and waits for a SEND-CONFIRMED for the session partner.
- DATAFLU — flushes data.
- ERROR — issued in most conversation states to indicate an error. The error indication will not flow to the conversation partner until a subsequent send of data containing a valid FMH7 is sent. This should be built in variable *&mh_data* and sent using SEND-DATAFLU.
- FLUSH — flushes data.
- RQSEND — notifies the partner LU that the local application is requesting to enter SEND state. The conversation will be changed to SEND state when the local application subsequently receives SEND indication from the remote application.

sense can only be used with ERROR. This is eight hexadecimal digits describing a sense code which will be set in variable *&mh_sens* in the APPCCMD completion EXEC and sent to the conversation partner. The issuer is in END CONVERSATION state when this verb completes. The default is 00000000.

APPCCMD completion exit

This exit is driven when an APPCCMD MACRO completes as a result of an MHOAPPC EXEC or an internally issued VTAM APPCCMD MACRO. In APPC mode (*&mh_auto*=Y) the exit is suppressed. In low level APPC mode (*&mh_auto*=N) the exit is driven for all APPCCMD MACRO completions. The exit is defined by setting the variable *&mh_appc* prior to issuing the MHOSTART EXEC.

For details of the variables available to this exit refer to *Available variables* on page 6.50. The following notes apply to variables available to this exit.

Protocol

This field will be set to LU62.

Control value

This field takes one of the following values:

ALLOC
DEALLOC
OPRCNTL
PREPRCV
RCVFMH5
RECEIVE
REJECT
RESETRCV
SEND
SETSESS

Qualify value

This field takes one of the following values:

ABNDPROG	DATACON
ABNDSERV	DATAFLU
ABNDTIME	DEFINE
ABNDUSER	DISPLAY
ACTSESS	ERROR
ALLOCD	FLUSH
ANY	IMMED
CNOS	RQSEND
CONFIRM	SESSION
CONFIRMD	SPEC
CONV	SUSPEND
CONWIN	RESUME
DACTSESS	UNDEFND
DATA	

What received (&MH_RCV1)

This field contains a string of nine Y or N flags indicating the various bit settings of the RPL extension what received flags in the following order:

CONFIRM

DATA

DATA-COMPLETE

DATA-INCOMPLETE

DEALLOC

LOG-DATA

PS-HEADER (not VTAM 3.2)

PARTIAL-PS-HEADER (not VTAM 3.2)

SEND

Returned indicator (&MH_RTUN)

This field takes one of the following values:

FMH5RCV

LOGRCV

SIGRCV

Conversation state (not VTAM 3.2) (&MH_CCST)

This field takes one of the following values:

END-CON

P-DEALOC

P-ENDCON

P-RECLOG

P-SEND

REC-CON

REC-CONS

REC-COND

RECEIVE

RESET

SEND

Session limits, FMH5 and PIP data

This section documents the format of the session limits, the Function Management Header type 5 (FMH5), and the Program Initialization Parameter (PIP) data required by NC-MHO 1.3 from a non NC-MHO partner (such as CICS or a PC).

Note: These dataflows may be used to communicate with either a high or low level NC-MH0 partner.

The session limits are enforced by NC-MHO for APPC partners to ensure transactions are serialized correctly. When NC-MHO receives a CNOS notification that the session limit is set to 1, validation continues with the FMH5 as described below. When the session limit is not set to 1, the conversation is rejected with a sense of 08050000.

The FMH5 and associated PIP data is used by NC-MHO to ensure that session partners are using the appropriate protocols. When NC-MHO receives an FMH5, the Transaction Program Name (TPN) and the presence of PIP data are checked. The TPN must be NC-ISC62 when running in HLAPI mode (it is not validated when running in low level APPC mode). If validation fails, a sense code of 0835000A is set.

If the PIP data is not supplied no further checking is done. This means the ensuing conversation(s) will take default characteristics for such attributes as compression as set by the Startup Options Environment (SOE) EXEC in variable *&mh_comp*. It is the responsibility of the conversation partners to ensure that the conversation characteristics match.

Example

Communication with NC-MHO is in LLAPI mode. If PIP data is present it should be in the following format:

```
        CKS NC-ISC 1.1.ONNNNNNNN  ANYTHING
XX1FYY1ECDE4DC6CEC4F4F4FDDDDDDDDZZCDEECCDC
XX25YY22322053092301B1B055555555ZZ15838957
```

where

XX = length of PIP data
YY = length of first PIP parameter
ZZ = length of second PIP parameter

The string CKS NC-ISC 1.1.0 is validated first, and if successful the following 8 bytes of flags are processed. Note that the first PIP parameter must start with the data as shown but any user data may be appended to it. The second PIP parameter may contain anything you like.

Flag 1 Compression (Y/N)

Flag 2 low level APPC mode (N)/high level APPC mode (Y)

Flag 3 Reserved

Flag 4 Reserved

Flag 5 Reserved

Flag 6 Reserved

Flag 7 Reserved

Flag 8 Reserved

If validation fails the conversation is rejected (with a sense code of 08350000). In low level APPC mode, processing continues with default characteristics for compression etc. Further subfields appropriate to the user application may be supplied. The second PIP parameter is passed to the user as *&mh_udat* in the logon authorization EXEC.

If an NC-MHO partner requests compression where it is not supported, the conversation setup is rejected with a sense of 08350018. For NC-MHO to NC-MHO sessions this will result in a second attempt to establish communication, this time with compression not requested.

If a low level APPC mode NC-MHO partner requests a session to an environment running in HLAPI mode, the session setup is rejected with a sense of 08350019.

When NC-MHO is issuing an ALLOCATE the FMH5 may include a TPN of up to eight bytes left justified, and padded with blanks. In APPC mode (*&mh_auto* = Y) the TPN is set to the value supplied by the partner that initiated the unit of work. PIP data will be present, synchronization will be none and basic conversation mode will be indicated.

PIP data will always be sent and will take the format described above with the flags set to indicate the capabilities of the issuing NC-MHO.

APPC low level data flows

The following data flow diagrams define the required handshaking between LU6.2 partners for session establishment/termination and the ASY, SYN and NOREP functions for transactions. The communication protocol must be obeyed when using the six logical functions described.

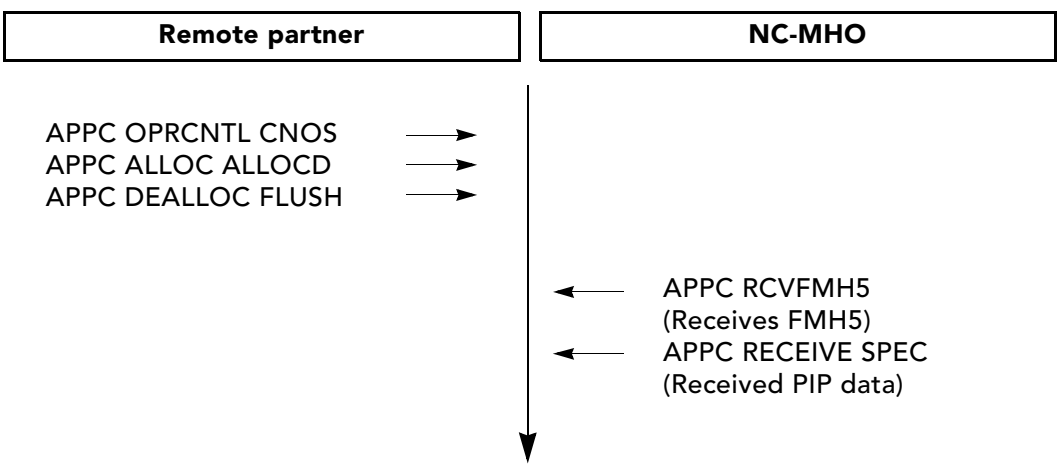
Note: These dataflows may be used to communicate with either a high or low level NC-MH0 partner.

SYNCLEVEL (CONFIRM) (definite confirmation mode) is an assumed prerequisite for all data flows.

The handshaking protocol adopted is based on a de facto standard used by typical mainframe APPC users (for instance CICS). The adopted dataflow for a given transaction results in allocation of a conversation, data flow and then conversation deallocation.

Session Establishment

MHOLINK will deliver and expect the following data flows (in sequence from top to bottom):

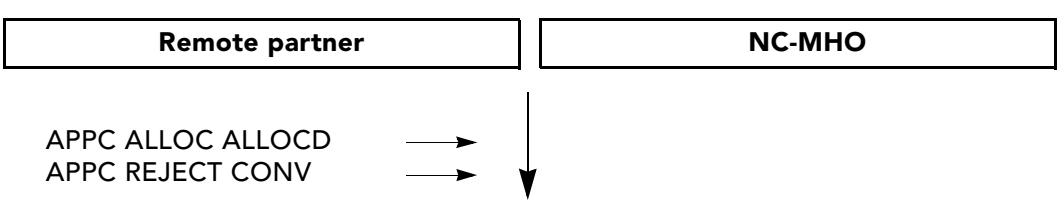


PIP data contains rules for APPC partners for subsequent transactions across the session (for example whether data is compressed).

Session termination

The APPC session remains established. However, a REJECT CONVERSATION flows in order for the remote partner to be aware that the host has broken the link and to inform any remote applications that the link is terminated.

The link can be viewed as deactivated (inactive) by any application, however the underlying physical session remains active.

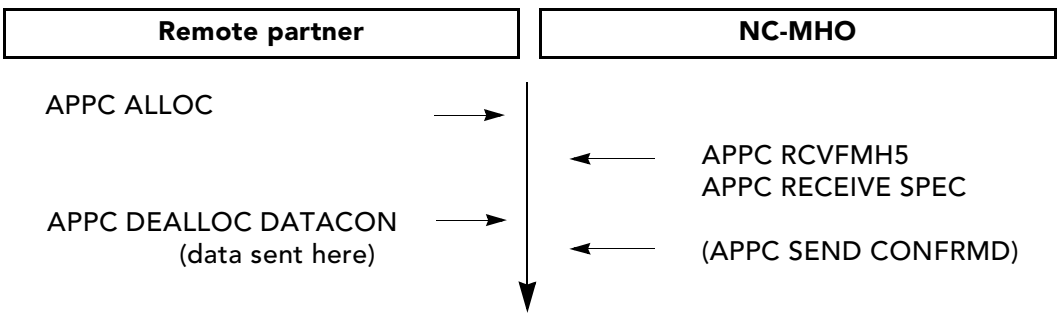


LU6.2 protocol termination

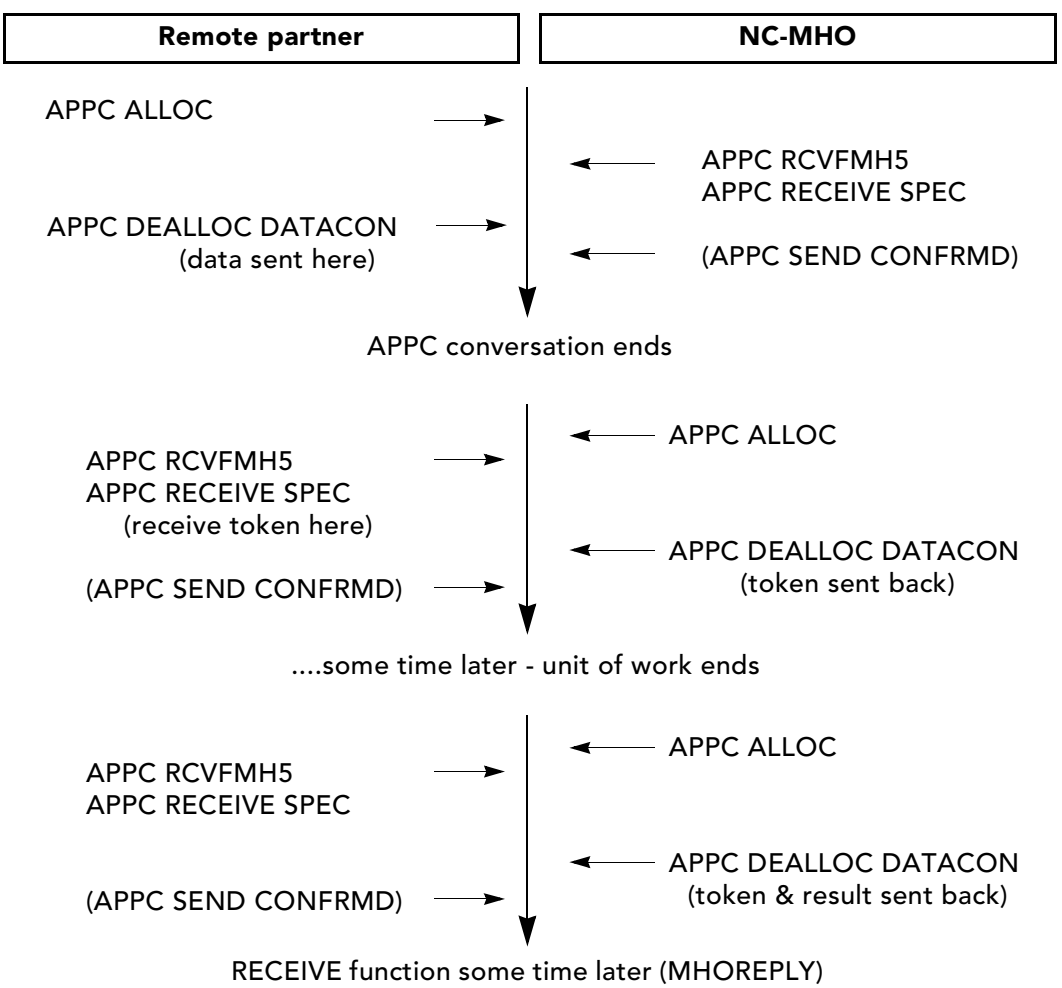
Closing the local application via MHOCLOSE will result in VTAM driving the ATTN exit or its equivalent, if any, on other platforms with the LOSS function.

Transaction requests

ASY-NOREP

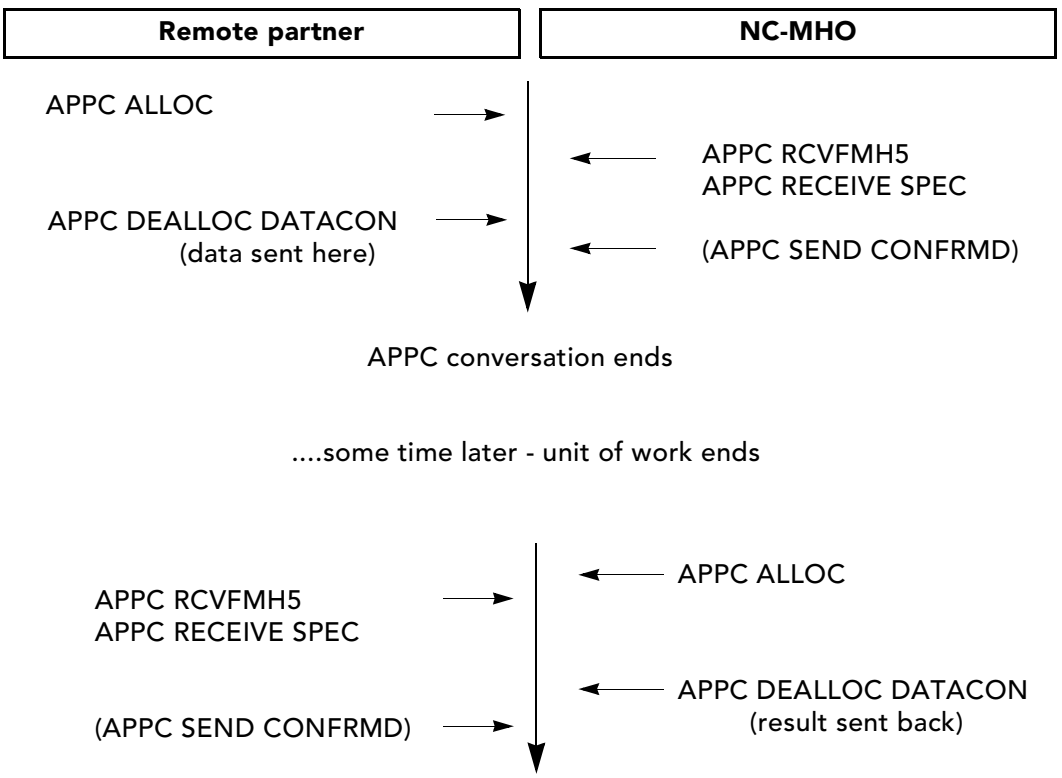


ASY-REP



Note: As the system that has performed the unit of work has informed the initiator, without being solicited, the initiator only needs to examine its local queues to determine if the remote asynchronous event is complete.

SYN



Using the NC-MHO low level Sockets environment

MHOSOCK EXEC

The MHOSOCK EXEC is used when NC-MHO is run in low level sockets mode for TCP/IP connections (&*mh_auto*=N in the SOE). Some knowledge of Sockets protocol is required to use this interface. The IBM manual 'TCP/IP for MVS Programmers Reference' contains general and coding details for Sockets including the IUCV interface on which the MHOSOCK EXEC is based. It is also necessary to understand the MHOSOCK completion EXEC (see *Sockets completion EXEC* on page 6.79).

Note: For compression and encryption, variables *&mh_comp* and *&mh_ex01* have no effect in the Socket LLAPI environment. These features require knowledge of the underlying record format.

Format

EXEC MHOSOCK *operation sockn* [*options*.....]

where

operation is the Sockets operation, which can be one of the following:

ACCEpt	LISTen
BIND	RECEive
CANCel	SEND
CLOSe	SHUTdown
CONNect	SND2
GETSocknam	SOCKet
IOCTI	

These *operation* keywords are described on the following pages. Each *operation* keyword is shown with the appropriate *sockn* and *options* fields and provides a description of each of the available options.

sockn is the Socket number to use.

options are optional keywords specific to a Sockets operation. Descriptions for each of the supported options are provided on subsequent pages under the appropriate Sockets operation description.

Return Codes

On completion, one of the following codes will be set in variable *&usrretc* to indicate the success or failure of MHOSOCK:

0	command completed successfully.
4	invalid parameter(s).
8	task entry not found.
12	Socket already connected

ACCEpt

The *operation* keyword ACCEpt is used to accept an incoming CONNECT operation.

EXEC MHOSOCK ACCEpt *sockn newsockn*

where

sockn is the previously allocated socket number in decimal format

newsockn is the socket number, in decimal format, to be used for the incoming connection

Note: the completion EXEC will contain the value for *newsockn* in the variable *&mh_sckt*

BIND

The *operation* keyword BIND is used to bind a socket to a port number and internet address.

EXEC MHOSOCK BIND *sockn portn* [*addr*]

where

sockn is the previously allocated socket number in decimal format

portn is the port number, in decimal format, to which the socket is to be bound

addr is the internet address to which the Socket is to be bound; if omitted the default local host address in dotted decimal format is used

CANCel

The *operation* keyword CANCel is used to terminate an outstanding sockets call.

EXEC MHOSOCK CANCel *sockn function msgnum*

where

sockn is the previously allocated socket number in decimal format

function is the IUCV function code for a previously issued MHOSOCK. This is returned in the first four bytes of the MHOSOCK completion panel variable *&mh_trg*.

msgnum is the IUCV message number for a previously issued MHOSOCK. This is returned in the MHOSOCK completion panel variable *&mh_msgn*.

CLOSe

The *operation* keyword CLOSE is used to close a socket.

EXEC MHOSOCK CLOSE *sockn*

where

sockn is the previously allocated socket number in decimal format

CONNect

The *operation* keyword CONNect is used to connect a socket to a remote port/ internet address.

EXEC MHOSOCK CONNect *sockn portn* [*addr*]

where

sockn is the previously allocated socket number in decimal format

portn is the port to which this socket is to be connected, in decimal format

addr is an optional internet address where the remote port resides. If omitted the default is the local internet host in dotted decimal format.

GETSockname

The *operation* keyword GETSockname is used to obtain the port and internet address associated with the specified socket.

EXEC MHOSOCK GETSockname *sockn*

where

sockn is the previously allocated socket number in decimal format

IOCTl

The *operation* keyword IOCTl is used to set the mode in which sockets calls will operate.

EXEC MHOSOCK IOCTl *sockn socal*

where

sockn is the previously allocated socket number in decimal format

socal can be one of the following:

BLOCK indicates that sockets calls are to wait for completion.

NOBLOCK indicates that sockets calls are to return immediately if no data will satisfy them.

LISTen

The *operation* keyword LISTen is used to convert the socket open from active to passive.

EXEC MHOSOCK LISTen *sockn*

where

sockn is the previously allocated socket number in decimal format

RECEive

The *operation* keyword RECEIVE is used to receive data.

EXEC MHOSOCK RECEIVE *sockn* [*len*]

where

sockn is the previously allocated socket number in decimal format

len is the length of the receive buffer in decimal format; if omitted the default is 4096

SEND

The *operation* keyword SEND is used to send data using a TCP layer socket.

EXEC MHOSOCK SEND *sockn data*

where

sockn	is the previously allocated socket number in decimal format
data	is the data to be sent

SHUTdown

The *operation* keyword SHUTdown is used to terminate a connection.

EXEC MHOSOCK SHUTdown *sockn*

where

sockn	is the previously allocated socket number in decimal format

SND2

The *operation* keyword SND2 is used to send data using a UDP layer socket.

EXEC MHOSOCK SND2 *sockn portn addr data*

where

sockn	is the previously allocated socket number in decimal format
portn	is the port number, in decimal format, to which the data is to be sent
addr	is the internet address to which the data is to be sent; if 0.0.0.0 is specified, the default local host address is used
data	is the data to be sent

SOCKet

The *operation* keyword SOCKET is used to allocate a socket.

EXEC MHOSOCK SOCKET *sockn* [*layer*]

where

sockn is the socket number, in the range 1 to 49, to be allocated.

layer is an optional keyword which can be one of the following:

UDP indicates that a UDP transport layer socket is required.

TCP indicates that a TCP transport layer socket is required (this is the default).

Sockets completion EXEC

This panel is driven whenever an IUCV socket call is scheduled and when it completes. In HLAPI mode this exit is suppressed. For LLAPI operation set *&mh_auto* to N and *&mh_sock* to the panel name required before issuing the MHOSTART command.

For details of the variables available to this exit see *Available variables* on page 6.50.

When the Sockets completion exit is driven, these variables will contain the following information:

&mh_adrl the local internet address in dotted decimal format.

&mh_adrr the remote internet address in dotted decimal format where applicable.

&mh_func when a sockets operation is scheduled this will be set to S; when it completes it will be set to C.

&mh_msgn the contents of the IUCV parameter list field IPMSGID in EBCDIC (for example X'01020304' will be returned as 01020304).

&mh_prm the contents of the IUCV macro PRMMSG field in EBCDIC (for example X'0000000200000000' will be returned as 0000000200000000).

&mh_prot will be set to SOCK.

&mh_prtl the local port number where applicable.

&mh_prtr the remote port number where applicable.

&mh_rep the contents of the IUCV macro reply buffer in EBCDIC (for example X'FFFFFFFF00000009' will be returned as FFFFFFFF00000009).

&mh_sckt the socket number in use.

&mh_trg the contents of the IUCV macro TRGCLS field in EBCDIC (for example X'00100001' will be returned as 00100000).

&mh_trgc the Sockets operation in EBCDIC. For example, if an MHOSOCK CONNECT is issued, *&mh_trgc* will contain CONNECT.

Connect data

For LU6.2 connections, information is exchanged at signon in the FMH5 data. For Sockets applications a special string of data must be the first data sent to an NC-MHO partner; this data must be in the format shown below and must be preceded by a binary fullword containing its length + 4:

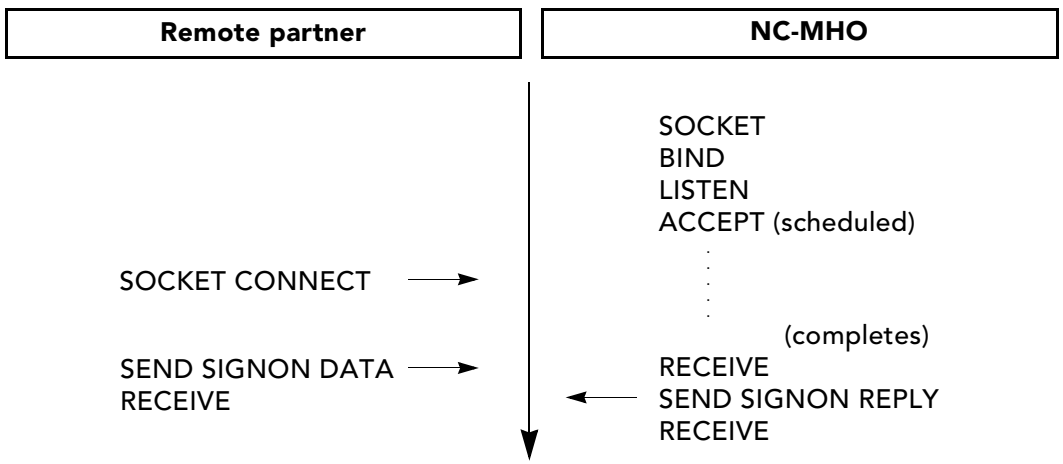
Offset	Length	Comment
+0	6	NC-MHO as an EBCDIC constant
+6	8	The name of the application being accessed. This must match the name used in the MHOOPEN EXEC.
+14	8	A string of eight flags containing Y or N as follows: 1 - Compression (Y or N) 2 - HLAPI (Y or N) 3 - reserved 4 - reserved 5 - reserved 6 - reserved 7 - reserved 8 - reserved
+22	8	Your application name, an arbitrary character string identifying you
+30	1	Length of following field in binary
+31	*	Any user data; this will appear in the *&mh_udat* variable in the appropriate panel exits

If validation of this data is failed by NC-MHO, the senders receive will complete with a 'closed connection' indication.

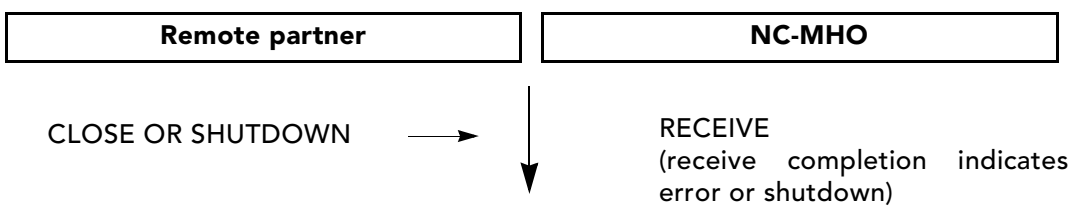
Sockets data flows

The following data flow diagrams define the required handshaking between Sockets High Level API partners for session establishment/termination and the ASY, SYN and NOREP functions for transactions.

Session Establishment



Session termination

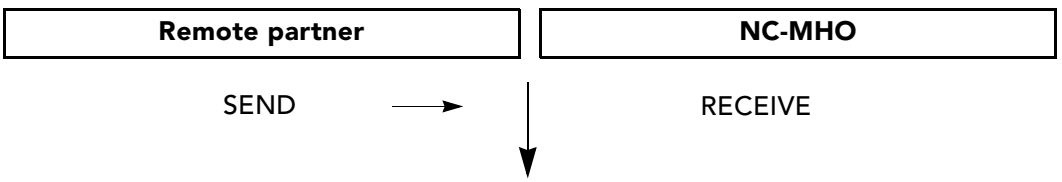


Sockets protocol termination

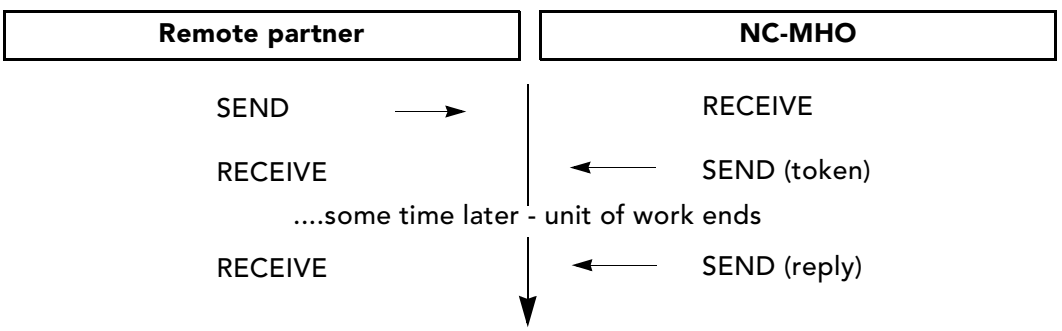
Closing the local application via MHOCLOSE will cause NC-MHO to unregister from the IUCV interface and active connections will drive the loss exit.

Transaction requests

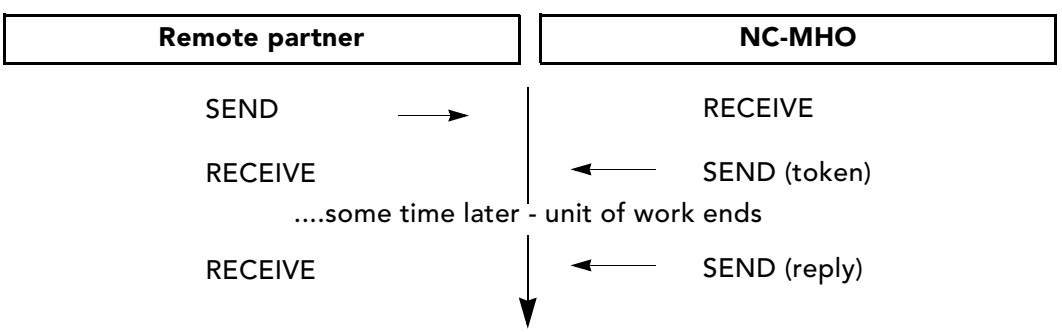
ASY-NOREP



ASY-REP



SYN



Data format

This section defines the format of the data that must be sent to NC-MHO to request NC-MHO to do one of the following functions:

- run an NCI/XF EXEC routine
- log a message to the NCI/XF log
- resolve a variable from the NCI/XF address space
- set a variable on the NCI/XF address space

The format of the data sent to NC-MHO to request an operation is:

POS	LEN	CONTENTS
1	1	Must be zero
2	1	Mode: N Asynchronous R Asynchronous-with-reply S Synchronous T Asynchronous-with-reply and request-for-token U Unformatted (used to send data that will NOT be interpreted by NCI/XF but will be passed to the receive EXEC; if this mode is used the following fields are not applicable and may contain user data)
3	1	Operation: E Run an NCI/XF EXEC L Write a message to the NCI/XF log R Resolve an NCI/XF variable S Set an NCI/XF variable
4	8	Transaction Program Name (TPN): The name of the transaction to be set in the FMH5 by MHO when initiating a results transaction (LU62 only).
12	8	Token: Used in asynchronous-with-reply mode to uniquely identify the operation.

For operation E:

20	8	name of the EXEC to be run
28	*	parameter to pass to EXEC

For operation L:

20	*	LOG message

For operations R and S:

20	3	length of variable name. This must be an integer in the range 1 through 250 with leading zeros.
23	*	variable name up to 250 bytes in length. If S (SET) followed immediately by contents to set variable to a maximum of 32000 bytes. If R (RESOLVE) not applicable.

Notes

1. When NC-MHO replies to a transmission from a remote partner, it returns user data in the same encrypted/unencrypted format in which the request was received. When NC-MHO initiates the transmission, it will encrypt the data by default. If you do not want the data to be encrypted, set variable *&mh_enc* to N prior to issuing the required command (using MHOSET).

2. When data is sent using the LU62 protocol, it must be preceded by a binary halfword containing its length, including the length of the field itself.

3. When data is sent using the Sockets protocol, it must be preceded by a binary fullword containing its length, including the length of the field itself.

The format of the data returned by NC-MHO is:

POS	LEN	CONTENTS
1	1	Reserved.
2	1	set to A to indicate an answer
3	1	type of answer: C reply contains a return code T reply contains a token R reply contains a return code and data
4	8	token
12	4	MHO environment identifier (*&mh_id*)
16	8	MHO application name (*&mh_acb*) SENDING the answer
24	4	return code: for SET 0000 SET function successful 0008 SET function denied by SET authorization EXEC. for LOG 0000 LOG function successful 0008 LOG function denied by LOG authorization EXEC. for RESOLVE 0000 RESOLVE function successful 0004 Variable contains more than 31596 bytes or is a null variable. 0008 RESOLVE function denied by RESOLVE authorization EXEC. for EXEC 0000 EXEC function successful 0004 EXEC routine not found, function not done 0008 EXEC function denied by EXEC authorization EXEC.
28	*	data depending on the operation: E the return code set by the EXEC followed by a semi colon (;) followed by the contents of the *&mh_usfld* variable if set. L not applicable R the value of the resolved variable; this field may be up to 31596 bytes in length S not applicable.

RPC routines

This section describes the two RPC routines MHRPCSRV and MHRPCLNT which control the Server and Client functions respectively.

MHRPCSRV

The server function is controlled by invoking the MHRPCSRV routine as follows:

EXEC MHRPCSRV *function pgmn* [*option1*] [*option2*]

where

function can be one of the following:

- START starts the server.
- STOP stops the server.
- TRACE sets the internal trace facility using *option1* to set ON or OFF.
- STATUS determines the server status

pgmn is the remote procedure call program number.

*option*1 when the START function is specified, this is the name of the TCP/IP address space. If *option1* is omitted the default is TCPIP. If *option2* is specified, an entry for *option1* must be included

when the TRACE function is specified, *option1* can be either ON or OFF to start and stop tracing.

option2 only applies to the START function. This is the PORTMAP well known port number. The default is 111.

Return codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHRPCSRV:

- 0 server started successfully.
- 4 invalid parameters.
- 8 server already started (START) or no server started (STOP/TRACE/STATUS).
- 12 timeout during START/STOP.
- 16 error during START; see NCI/XF log for more details.
- 20 unable to initialize IUCV environment.

Example

To start a server as program 150000:

EXEC MHRPCSRV START 150000

Server API information

The RPC server registers with PORTMAP using the program number specified. The version is 1. UDP and TCP transport layers are supported. The following procedures are supported:

0 RPCINFO

1 IBM's GENESEND sample

2 write message to NCI/XF log

3 set an NCI/XF variable

4 execute an NCI/XF EXEC

5 resolve an NCI/XF variable

The data passed/returned to the above procedures takes the following format:

Procedure	In	Out
0		
1	integer	integer
2	character string containing message text	integer return code
3	character string containing variable name character string containing variable contents	integer return code
4	character string containing panel name character string containing panel execution parameter available to the remote EXEC in variable *&mhrpcpm*	integer return code character string containing *&usrretc* variable (set by remote EXEC) character string containing *&mhrpcpm* variable (set by remote EXEC)
5	character string containing variable name	integer return code character string containing resolved variable data

The credential and verification fields must be set to zero.

Note: Any panel executed by procedure 4 may set the variable *&mhrpcpm* which will be returned to the caller.

MHRPCLNT

The client function is controlled by invoking the MHRPCLNT routine as follows:

EXEC MHRPCLNT *ia pn pv pp pr j iv ov pm*

where

ia	is the internet address where the server resides, in dotted decimal format. An asterisk (*) or zero (0) can be used to signify the local host.
pn	is the server program number (no default).
pv	is the server program version. An asterisk (*) will invoke a default of 1.
pp	is the server program procedure. An asterisk (*) will invoke a default of 1.
pr	is the transport layer to use, TCP or UDP. An asterisk will invoke a default of TCP.
j	is the name of the TCP/IP address space. An asterisk (*) will invoke a default of TCPIP.
iv	is the input data stub variable name; up to 243 alphanumeric characters. An asterisk (*) will invoke a default of IN.
ov	is the output data stub variable name; up to 243 alphanumeric characters. An asterisk (*) will invoke a default of OUT.
pm	is the well known port number for PORTMAP. An asterisk (*) will invoke a default of 111.

Return codes

On completion, one of the following codes will be set in *&usrretc* to indicate the success or failure of MHRPCLNT:

0 client function completed.

4 invalid parameters.

8 error in input stub variable(s).

12 TCP/IP initialization failed.

16 error during PORTMAP connection.

20 error in PORTMAP reply.

24 error during server connection

28 error in server reply

32 error in output stub variable(s)

36 unable to initialize IUCV environment

Input stub variable

The input stub variable can be appended by one of the following:

0 provides a count of input variables (DATA/FMT pairs)

DATA*n* contents of input data item *n*:
where
n is in the range 1 through 99

FMT*n* is the format of input data item *n*
where
FMT can be CHAR or INT
n is in the range 1 through 99

PORT is the port number in use by server. If this is set, processing will not contact PORTMAP for the remote server port.

TRACE sets the internal trace settings. Set to Y to enable tracing.

TO is the timeout value in seconds. This value will be used on all receive operations. If omitted or invalid, the default is 10. If the value is entered as 0, no timeout processing will be performed.

The credential and verification fields are set to zero.

Output stub variable

The output stub variable can be appended by one of the following:

0 provides a count of input variables (DATA/FMT pairs)

DATA*n* contents of output data item (returned by server)
where
n is in the range 1 through 99

FMT*n* is the format of data item to be returned by server
where
FMT can be CHAR or INT
n is in the range 1 through 99

PORT is the port number in use by server as returned by PORTMAP.

Example

To call the NCI/XF set procedure running on host 1.2.3.4 as program 9999:

```
SET   &in0 = 2
SET   &indata1 = *demo
SET   &infmt1 = char
SET   &indata2 = allo
SET   &infmt2 = char
SET   &out0 = 1
SET   &outfmt = int
EXEC MHRPCLNT  1.2.3.4  9999  1  3
LOG  'Server at port &outport; return code is &usrretc'
```

Chapter 7 - NC-XMS routines and macros

Foreword 7.2

NC-XMS COMMAND ROUTINES
XMSTART 7.3
XMSTOP 7.5
XMSYS 7.6
Modification of the cross memory system name 7.6
XMSTATUS 7.7
XMCLEAN (MVS only) 7.8
XMFIXES 7.9
XMTRACE 7.11

STARTUP OPTIONS EXEC ROUTINE
XMOPTION 7.13
XMEXIT01 .....7.14
XMEXIT02 .....7.15
XMEXIT03 .....7.16
XMEXIT04 .....7.17

THE XMDIALOG MACRO
Common XMDIALOG parameters 7.18
Restrictions on the LEN parameter 7.21
Notes on the XMDIALOG macro 7.22
Re-entrancy 7.22
Register addressing 7.22
Return codes from XMDIALOG macro 7.22
CLOSE (VM only) 7.28
EXEC 7.29
LOG 7.31
OPEN (VM only) 7.32
PURGE 7.33
RECEIVE 7.34
RESOLVE 7.35
SET 7.37

Foreword

The information in this chapter is provided in three logical areas as follows:

1. the NC-XMS command routines, XMSTART to XMTRACE, which are executed in the NCI/XF job and are therefore most relevant to NCI/XF programmers
2. the customizable XMOPTION with the optional 4 authorization exits XMEXIT01 to XMEXIT04
3. the XMDIALOG macro, used in the non-NCI/XF assembler module, with one of the 6 options EXEC to SET (this is chiefly of interest to assembler programmers)

This chapter should be consulted only after you have read the NCI/XF User Guide, *Chapter 8 - A guide to NC-XMS*.

XMSTART

Starts an XMS cross memory system.

Format

EXEC XMSTART [*parm1*] [*parm2*] [*parm3*] **.....** [*parm6*]

where

parm1 is the name of the Startup Options Exec (SOE) that will be invoked during this start up to validate or respecify the cross memory system name and to set other options. If *parm1* is omitted, the default name XMOPTION is used for which a working example is supplied.

parm2 is the first parameter passed to the SOE.

If you are using the supplied XMOPTION example, *parm2* is used as the four character cross memory system name to be started and will cause XMSYS to be invoked with *parm2* as the XMSYS operand. This will change the current cross memory system name held in user variable *&%xmsys* to *parm2* for all subsequent NC-XMS command routines issued by this user.

Note: The supplied XMOPTION will return a non-zero return code if *parm2* is not set to one of the following values:

XMS1 (or PROD or 1)

XMS2 (or TEST or 2)

parm3 through *parm6* are user defined parameters for the SOE named in *parm1*. The default XMOPTION does not require these parameters.

WARNING: If any other product uses the same SSCT name, results from NCI/XF (and the other product) are unpredictable.

You cannot have a proc and an SSCT with the same name.

Example

```
EXEC XMSTART &sysnull TEST
```

This will start a cross memory system (an XMS server task) which will service XMDIALOG macro requests that use the XMSYS=TEST parameter.

Return codes

The following codes are returned in *&usrretc*:

0	XMS started successfully
4	serious error occurred during attempt to start the cross memory system, for instance no free NC-XMS storage queue elements
8	*parm1* routine could not be found
12	startup disallowed by non zero return code from startup options EXEC
16	start failed, because, for instance, XMS name is already active
20	NCI/XF not running authorized
24	the cross memory system name could not be started, because XMSTART found an error. This may, for example, be within MVS control blocks (such as SSCT) relating to this particular name. An example of this error occurring is when there is an abnormal termination of an earlier NC-XMS system which served this cross memory system name. If this error occurs XMCLEAN should be executed (on MVS only) in order to tidy up the corrupted control blocks, and XMSTART reissued.
28	authorization exit 01 (*&*%xmx01*) not found
32	authorization exit 02 (*&*%xmx02*) not found
36	authorization exit 03 (*&*%xmx03*) not found
40	authorization exit 04 (*&*%xmx04*) not found

XMSTOP

Stops an XMS cross memory system.

Format

EXEC XMSTOP [*parm1*]

where

parm1 is the four character cross memory system name to be stopped.

Note: specification of *parm1* will cause XMSYS to be invoked with *parm1* as the XMSYS operand. This will change the current cross memory system name held in user variable *&%xmsys* to *parm1* for all subsequent NC-XMS command routines issued by this user. See *Modification of the cross memory system name* on page 7.6. If *parm1* is omitted the user variable *&%xmsys* will identify the cross memory system.

Example

```
EXEC XMSTOP
```

This will stop the cross memory system identified by the user variable *&%xmsys*.

Return codes

The following codes are returned in *&usrretc*:

0	cross memory service stopped
8	unknown cross memory system name specified
10	stop has been pending for 1 minute and then abandoned

XMSYS

XMSYS sets or displays on the NCI/XF log the current four character cross memory system name.

Format

EXEC XMSYS [*parm1*]

where

parm1 is the cross memory system name on which subsequent XMS command EXECS invoked by the same user will operate. If *parm1* is omitted, two console log messages containing:

- the name of last XMS system started

and

- the name of current XMS system for the user

will be displayed.

Example

```
EXEC XMSYS xms2
```

This will cause the user variable *&%xmsys* to be set to XMS2, known as the current cross memory system name. All subsequent NC-XMS command routines executed by this user that do not specify *parm1* will use a default *parm1* of XMS2 as a result.

Notes

The following variables may be useful to routines which employ the XMSYS exec.

&%xmsys* contains the name of the last system started

&%xmsys contains the current cross memory system name for this user

Modification of the cross memory system name

For the cross memory services system to work it must have a valid cross memory system name. All NC-XMS command routines will pass the name to the XMSYS routine which will check for and modify invalid names. XMSYS will apply the following algorithm to the name and set the resulting four character name in user variable *&%xmsys*:

- convert to uppercase
- remove leading and trailing blanks and nulls
- concatenate XMS1 to the end
- truncate to four characters

If an NC-XMS command routine is not passed a cross memory system name, it will use the user variable *&%xmsys* as input to the XMSYS routine.

XMSTATUS

XMSTATUS displays the status on the NCI/XF log of one or all XMS systems which have been started in the current NCI/XF environment.

Format

EXEC XMSTATUS [*parm1*]

where

parm1 is the cross memory system name (for instance, XMS1) whose status is to be displayed on the NCI/XF log.

Note: specification of *parm1* will cause XMSYS to be invoked with *parm1* as the XMSYS operand. This will change the current cross memory system name held in user variable *&%xmsys* to *parm1* for all subsequent NC-XMS command routines issued by this user. See *Modification of the cross memory system name* on page 7.6. If *parm1* is omitted all active cross memory systems on this NCI/XF job will be listed.

Example

```
EXEC XMSYS xms2
```

This causes all cross memory systems on this NCI/XF job to be listed on the NCI/XF log.

Return codes

The following codes are returned in *&usrretc* when *parm1* is specified:

0 cross memory system inactive

4 (MVS only) error occurred while determining status of cross memory server task, for instance no free NC-XMS storage queue elements

8 (MVS only) cross memory system active but running in a different NCI/XF job

16 cross memory system active for this NCI/XF job

20 (MVS only) an error was found within MVS control blocks relating to the cross memory system name identified by *parm1*. This error might have arisen, for instance, because of the abnormal termination of an earlier NC-XMS system which served this cross memory system name. If this error occurs XMCLEAN should be executed in order to tidy up the corrupted control blocks, and XMSTART reissued.

XMCLEAN (MVS only)

XMCLEAN performs cleanup after abnormal termination.

Format

EXEC XMCLEAN [*parm1*]

where

parm1 is the cross memory system name (for instance, XMS1) to be cleaned up.

Note: specification of *parm1* will cause XMSYS to be invoked with *parm1* as the XMSYS operand. This will change the current cross memory system name held in user variable *&%xmsys* to *parm1* for all subsequent NC-XMS command routines issued by this user. See *Modification of the cross memory system name* on page 7.6. If *parm1* is omitted the default cross memory system name held in user variable *&%xmsys* will be used.

Example

```
EXEC XMSYS xms2
```

This cleans up the failed cross memory system identified by user variable *&%xmsys.*

Return codes

0 XMCLEAN completed successfully

Notes

This routine is only useful if the XMS task has ended without shutting down properly and NCI/XF is still running. It deactivates the MVS cross memory server set up by XMS and so frees the XMSYS name for reuse by this or another NCI/XF system; when NCI/XF is eventually shutdown, the routine permits MVS to reuse its address space. If the XMS task abnormally terminates you must inform your local support office.

XMFIXES

The XMFIXES routine can be used to generate and display the NC-XMS fix grid indicating which fixes, if any, are applied to the NC-XMS subsystem. The EXEC allows SETTING, LOGGING or RESETTING of the variables that contain the fix grid.

An example of the fix grid being logged to the NCI/XF log is shown below:

```
11:59:26 XMFIXES - NC-XMS VERSION 1.2.0 RELEASED 04 APR 93 FOR MVS
11:59:26 XMFIXES - THE FOLLOWING FIXES HAVE BEEN APPLIED
11:59:26 XMFIXES -     0 1 2 3 4 5 6 7 8 9     0 1 2 3 4 5 6 7 8 9
11:59:26 XMFIXES -    --------------------    --------------------
11:59:26 XMFIXES - 0 | . . . . . . . . . .  1 | . . . . . . . . . .
11:59:26 XMFIXES - 2 | . . . . . . . . . .  3 | . . . . . . . . . .
11:59:26 XMFIXES - 4 | . . . . . . . . . .  5 | . . . . . . . . . .
11:59:26 XMFIXES - 6 | . . . . . . . . . .  7 | . . . . . . . . . .
11:59:26 XMFIXES - 8 | . . . . . . . . . .  9 | . . . . . . . . . .
11:59:26 XMFIXES -    --------------------    --------------------
```

Format

EXEC XMFIXES *option1 option2.....option15*

where

option1 provides the prefix (up to eight bytes in length) to be displayed on the log, if the LOG option is specified. Although only required for the LOG function, this is a positional parameter and is therefore required before specifying *option2* to *option15*. The variable *&sysnull* may be used to allow it to default to a value of XMFIXES, for example:

EXEC XMFIXES &SYSNULL SET

will simply set the variables.

option 2 through *option15* can, if specified, consist of any of the following values:

SET	to set the variables
LOG	to log the variables
RESET	to reset the variables

If SET, LOG or RESET are specified, they will override the default settings and will be executed in the order encountered.

Default

If the panel is executed without options (EXEC XMFIXES), it will default to:

EXEC XMFIXES 'XMFIXES' 'SET' 'LOG' 'RESET'

This will set the variables *&xmvfixh*, *&xmvfixn* and *&xmvfix1-9*, which are dependent on the value set in *&xmvfixn*. Having been set, the variables will be logged with the first option (set to XMFIXES) prefixing each line (as above). Finally the variables will be reset and hence unavailable for use.

Exit Variables

These exit variables are dependent upon the option specified leaving the variables set. The following variables are available:

&xmvfixh	provides the version and release date information at the start of the log
&xmvfixn	provides the number of lines of the display
&xmvfix1 through *&xmvfix5*	provide the fix information displayed in these lines. In the example above no fixes are installed hence this area contains only dots (.).

When a fix is applied the dot (.) in the position which corresponds to the fix applied will be replaced by one of the following characters:

P	represents a fix applied in response to a problem
C	represents a fix applied for customization purposes

All other values are set by the panel during the LOG function.

Notes

If any of the options *option2* through *option15* are specified, it must be remembered that they are executed in the exact order they are encountered, for example:

EXEC XMFIXES "MY-NAME" RESET LOG SET

would reset the variables, log nothing and then set the variables, which may not produce the required result.

Return Codes

0	Options requested worked successfully
1-16	The option number (corresponding to the return code number) was in error
60	The XMVERS module could not be loaded

XMTRACE

XMTRACE sets or displays the tracing level or message format for output on the NCI/XF log.

Format

EXEC XMTRACE *parm1*

where

parm1 is one of the following:

- U all XMS console messages should be uppercased before output
- L leave messages in upper or lower or mixed case
- D display level of all messages from XMS EXECs
- N do not display level of messages from XMS EXECs
- I*n* set **I**nitial program trace level before starting XMS; *n* can be 1 through 9, where 1 provides the lowest level of detail and 9 the most detail; the intermediate numbers provide varying degrees of detail between the two extremes. The distributed sample XMOPTION exec invoked by XMSTART issues an EXEC XMTRACE I1. If you are using the distributed sample XMOPTION and require an Initial program trace level other than I1, you will have to modify XMOPTION.
- P*n* change **P**rogram trace level after starting XMS; *n* can be 1 through 9, where 1 provides the lowest level of detail and 9 the most detail; the intermediate numbers provide varying degrees of detail between the two extremes.
- *n* set trace level for XMS EXECs before or after start; *n* can be 1 through 9, where 1 provides the lowest level of detail and 9 the most detail; the intermediate numbers provide varying degrees of detail between the two extremes.

Examples

```
EXEC XMTRACE P9
```

This will display a detailed trace output of the cross memory assembler server task on the NCI/XF log.

```
EXEC XMTRACE 1
```

This will display the least detailed trace information on the NCI/XF log of the NC-XMS command routines

Notes

The following variables can be examined to determine current trace settings:

*&*xmlgl*	trace level for XMS EXECs
*&*xmlgp*	trace level for program
*&*xmlgd*	*&sysnull* if message level is not being displayed
*&*xmlgu*	*&sysnull* if messages are being displayed in upper case

XMOPTION

XMOPTION is the default startup options EXEC routine called by XMSTART. It can have any name if specified as the first parameter of EXEC XMSTART. The XMOPTION (or differently named) exec is supplied between 1 and 5 parameters using parameters 2 through 6 of EXEC XMSTART. This allows user data to be passed from EXEC XMSTART to the XMOPTION exec. The supplied XMOPTION does not require parameters 2 through 5 (parameters 3 through 6 of EXEC XMSTART).

Input variables

&usrex1	same as *&*%xmsys* (see below)
&usrex2	*parm3* of EXEC XMSTART
&usrex3	*parm4* of EXEC XMSTART
&usrex4	*parm5* of EXEC XMSTART
&usrex5	*parm6* of EXEC XMSTART

Output variables

The following variables can be set or changed by XMOPTION

&%xmsys*

cross memory system name for this XMS system (initially based on *parm2* of XMSTART)

Specification of *parm2* on XMSTART will cause XMSYS to be invoked with *parm2* as the XMSYS operand. This will change the current cross memory system name held in user variable &%xmsys to *parm2* for all subsequent NC-XMS command routines issued by this user.

parm2 of XMSTART (or &%xmsys if omitted) is used as the first parameter for the SOE. The SOE can disallow the start up altogether or set *&*%xmsys* to a different cross memory system name causing XMSYS to be called again which will set the default name XMS1 if necessary. See *Modification of the cross memory system name* on page 7.6.

&%xmx01*	name of authorization exit exec for SET (initially null)
&%xmx02*	name of authorization exit exec for EXEC (initially null)
&%xmx03*	name of authorization exit exec for RESOLVE (initially null)
&%xmx04*	name of authorization exit exec for LOG (initially null)
&%xmcor*	number of XMS work elements to create for this system (default 100 work elements, 320 bytes each)

The next four pages contain descriptions of the optional authorization exits for each of: SET, EXEC, RESOLVE and LOG.

Authorization exits

An authorization exit exec is invoked to determine whether an XMDIALOG request should be honored. If the exit ends with return code zero, the request will be attempted. If it does not, the program which contains the XMDIALOG macro is given the return code in register 0 and decimal 20 in register 15.

XMEXIT01

XMEXIT01 is a sample NCI/XF EXEC routine for use as the optional authorization exit for controlling XMDIALOG SET requests. It can have any name as set in variable *&*%xmx01* by the options exec (XMOPTION). The supplied XMEXIT01 prevents alteration of certain XMS variables.

Input variables

&usrex1	name of variable (without leading ampersand) being SET
&%xmjnam	name of job requesting XMDIALOG SET
&%xmfunc	constant SET for use if the authorization exits are merged into one exec
&%xmauth	contains Y if the requesting job is authorized for MVS (the requesting task is APF authorized).

Return code

0	the SET will be allowed to proceed
non-zero	the SET will not be performed and the XMDIALOG macro will return this code in register 0 and decimal 20 in register 15.

XMEXIT02

XMEXIT02 is a sample NCI/XF EXEC routine for use as the optional authorization exit for controlling XMDIALOG EXEC requests. It can have any name as set in variable *&*%xmx02* by the options exec (XMOPTION).

Input variables

&usrex1	name of EXEC routine requested
&%xmjnam	name of job requesting XMDIALOG EXEC
&%xmfunc	constant EXEC for use if the authorization exits are merged into one exec
&%xmauth	contains Y if the requesting job is authorized for MVS (the requesting task is APF authorized).

Return code

0	the EXEC will be allowed to proceed
non-zero	the EXEC will not be performed and the XMDIALOG macro will return this code in register 0 and decimal 20 in register 15.

XMEXIT03

XMEXIT03 is a sample NCI/XF EXEC routine for use as the optional authorization exit for controlling XMDIALOG RESOLVE requests. It can have any name as set in variable *&*%xmx03* by the options exec (XMOPTION).

Input variables

&usrex1	name of variable without the leading ampersand
&%xmjnam	name of job requesting XMDIALOG RESOLVE
&%xmfunc	constant RESOLVE for use if the authorization exits are merged into one exec
&%xmauth	contains Y if the requesting job is authorized for MVS (the requesting task is APF authorized).

Return code

0	the RESOLVE will be allowed to proceed
non-zero	the RESOLVE will not be performed and the XMDIALOG macro will return this code in register 0 and decimal 20 in register 15.

XMEXIT04

XMEXIT04 is a sample NCI/XF EXEC routine for use as the optional authorization exit for controlling XMDIALOG LOG requests. It can have any name as set in variable *&*%xmx04* by the options exec (XMOPTION).

Input variables

&usrex1	first eight characters of LOG message
&%xmjnam	name of job requesting XMDIALOG LOG
&%xmfunc	constant LOG for use if the authorization exits are merged into one exec
&%xmauth	contains Y if the requesting job is authorized for MVS (the requesting task is APF authorized).

Return code

0	the LOG will be allowed to proceed
non-zero	the LOG will not be performed and the XMDIALOG macro will return this code in register 0 and decimal 20 in register 15

Common XMDIALOG parameters

Refer to *Return code table* on page 7.23 and *Reason code table* on page 7.26 if you want to clarify the references to the contents of registers in the following explanations.

ERROR= *error*

where *error* is the label to branch to if the return code in register 15 is non-zero. Even without this parameter, the condition code is set so that a following BNZ error instruction would branch if register 15 was not zero.

Example: If an XMDIALOG request fails (non zero value in R15), then cause control to transfer to label BADXMS:

```
                XMDIALOG........ERROR=BADXMS.......
                B     WORKED
BADXMS          DS    OH
                      .
                      .
                      .
                      .
                      .
        Examine registers 0 and 15 for precise error
```

TRACE=*n*

where *n* is a one digit number 0 through 9, or the name of a one byte area containing EBCDIC 0 through 9, for use as maximum tracing level of messages from the XMDIALOG module. 0 means no tracing and is also the default. The trace messages are output to the NCI/XF log if the program is running under NCI/XF or the MVS console if it is not. Tracing should be restricted to use by one-task programs.

Example: Turn tracing on to highest level:

```
                  XMDIALOG........TRACE=0........
```

TIMEOUT=*time*

where *time* is the name of a fullword area containing the number of one hundredths of a second which the module will wait for a response from the XMS system before timing out and returning decimal 24 in register 15. If omitted, and OPTCD=SYN (synchronous, the default) is in effect, there will be no timeout and the wait could be indefinite. OPTCD=ASY is not permitted with the TIMEOUT operand.

Example: The maximum wait for this synchronous request to complete is 2 seconds:

```
                XMDIALOG........TIMEOUT=TWOSECS.......
                     OPTCD=SYN
                      .
                      .
                      .
                      .
                      .
TWOSECS          DC  AL4(2*100)
```

OPTCD=*optcd* where *optcd* is one of the following (combinations of) options.

SYN this option is the default. The request will be sent to XMS and a wait will be entered until XMS has processed the request. An indefinite wait can be avoided by the use of the TIMEOUT parameter. NC-XMS will generate an internal response, but it is unnecessary to issue a subsequent XMDIALOG RECEIVE because NC-XMS will automatically do so.

SYN,REPLY same as SYN.

ASY the request will be sent to XMS asynchronously; that is, processing will continue without waiting for a response to the request. The return code from XMDIALOG will indicate whether the request was sent or not, but not whether it worked when it arrived there. XMS will generate a response which must be processed by a subsequent XMDIALOG RECEIVE *[TOKEN=t]* macro.

ASY, REPLY same as ASY.

NOREPLY the request will be sent to XMS asynchronously; that is, processing will continue without waiting for a response to the request. The return code from XMDIALOG will indicate whether the request was sent or not, but not whether it worked when it arrived there. NOREPLY tells XMS not to generate a response to a request.

ASY, NOREPLY same as NOREPLY

SYN, NOREPLY invalid.

REPLY invalid.

Example: Initiate an NCI/XF EXEC named MYEXEC, but do not wait for an indication of when the EXEC has run.

```
XMDIALOG   EXEC,
                 OPTCD=NOREPLY,
                 NAME=MYEXEC
```

TOKEN=*token* the name of a fullword area to contain (for a request) or containing (for a RECEIVE) the unique token used to refer to a specific XMDIALOG request. Any XMDIALOG request (SET/EXEC/RESOLVE/LOG) with OPTCD=ASY which returned zero in register 15 will also have set register 0 to the 4 byte token given to the request by XMS. If *TOKEN=token* was specified the token is also put in storage. For an XMDIALOG RECEIVE, if TOKEN is specified then only that one previous request can satisfy the RECEIVE and return zero in register 15. If a response from XMS could not be found, register 15 is set to 8. If TOKEN is omitted from a RECEIVE any outstanding OPTCD=ASY response can satisfy the RECEIVE and register zero is set to the token of the request whose response has been returned; if no response is available the register 15 is returned with 8 as before.

Example 1

Initiate an NCI/XF EXEC named MYEXEC. Do not wait for an indication of when the EXEC has run, but associate the cross memory request with a token number to be returned in the four byte field MYTOKEN:

```
                 XMDIALOG.....EXEC, NAME=MYEXEC,
                      OPTCD=ASY
                      TOKEN=MYTOKEN
                        .
                        .
                        .
                        .
                        .
MYTOKEN           DS  F
```

Example 2

Determine if the EXEC named MYEXEC in the above example has been executed (MYTOKEN still contains the token returned):

```
RETRY    DS    OH
         XMDIALOG   RECEIVE,TOKEN=MYTOKEN.
         LTR   R15, R15
         BZ    EXECOK
                .
                .
                .
                .
                .
     Wait before trying again
          B    RETRY
                .
                .
                .
                .
MYTOKEN    DS  F
```

MF=(E, *list*)

This invokes the execute form which uses a remote list form. E is a literal meaning execute; *list* is a 1 to 7 character *label* for the list form (that is, *list* equals the *lab* of an XMDIALOG MF=L list form of the macro), or in brackets a register such as (4), or register equate such as (R4), pointing to the list form. Refer to the subsection entitled *Re-entrancy* later in this chapter.

XMSYS=*xmsys*

where *xmsys* is one of the following:

- the name of a four byte area containing the name of the XMS system to use. For example:

  ```
  XMDIALOG XMSYS=MARK,......
  ```

  points to:

  ```
  MARK DC  CL4 'TSG1'
  ```

- a register in brackets, such as (4), or a register equate, such as (R4), pointing to a four byte area containing the name of the XMS system to use. For example, if the register is set up as:

  ```
  LA  R4,SAM
  ```

  then:

  ```
  XMDIALOG XMSYS= (R4),......
SAM  DC  CL4'XMS2'
  ```

- the name of the XMS system to use in quotes, for example:

  ```
  XMSYS='TSG1'
  ```

If *xmsys* is omitted, the value used by the previous macro is used again. If *xmsys* is omitted from the *first* XMDIALOG of the program, the hardcoded value XMS1 is used.

Restrictions on the LEN parameter

With the exception of the XMDIALOG LOG function, which is limited to the number of bytes supported by the NCI/XF LOG statement (See *LOG* on page 1.77), the LEN parameter is restricted to 232 bytes. For performance reasons the maximum amount of available cross memory transaction buffer space is 256 bytes. Of this, 24 bytes are consumed by header data, leaving 232 bytes of user data.

Notes on the XMDIALOG macro

Re-entrancy

For a program to be re-entrant, the list form *lab* XMDIALOG MF=L should be defined in a DSECT which maps GETMAINed storage, and XMS functions should be invoked via the execute form XMDIALOG...MF=(E,*label*).

The list form of the macro can also be used to good effect in the CSECT of a non re-entrant program to generate a common work area; all other XMDIALOG invocations can then use the execute form so that they can all use the same non re-entrant work area, rather than having to generate their own areas which would occur if the MF= parameter was omitted.

Register addressing

When register addressing is being used on an XMDIALOG macro, the value may be specified as the number in brackets or an equated symbol in brackets.

Return codes from XMDIALOG macro

The macro will always set a return code in register 15. A return code of zero always indicates the function was performed without error. A return code of 4 in register 15 indicates the function was performed correctly but a non-zero return code in register 0 was generated while the function was being performed (for example, non-zero return code from the requested EXEC or a variable was resolved as *&sysnull*). Higher return codes in register 15 indicate more serious errors. Register 0 is also returned with either an additional return code or a 4 byte token value associated with the request, as described in the tables on the following pages.

Return codes, reason codes, result codes

Register 15 is a 4 byte buffer, the bytes being numbered 0 to 3, where 3 is in the least significant and 0 in most significant position. When control is returned after an XMDIALOG macro, the least significant byte (byte 3) of register 15 contains one of the return codes tabulated on the next page, and the previous byte, byte 2, contains a reason code or zero. Bytes 0 and 1 always contain zero. Register 0 holds the token associated with the XMDIALOG request, or other values where indicated

APF authorization check

Prior to routines being invoked by XMDIALOG EXEC, variable *&%xmauth* is set either to null (reset) if the caller of the routine is not APF authorized, or non-null (for instance APF) if the caller is APF authorized.

For CKS products which run with NCI/XF acting as an XMS server and which require to reject requests from non-authorized callers, the status of variable *&%xmauth* is checked by the XMDIALOG EXEC routine before proceeding. If the variable is null, the call will be rejected.

Return code table

The first column in the table below shows hexadecimal reason codes in byte 2 of register 15 (*xxxxXXxx*). The second column shows the hexadecimal return code held in byte 3 (*xxxxxxYY*). The third column lists the decimal result code or binary token or abend code in register 0. The fourth column gives a short description and the fifth column a full explanation.

Reason code	Return code	Result code	Description	Explanation
00	00	token	successful	XMDIALOG function completed normally. For a request (SET/EXEC/RESOLVE/LOG) R0 contains the (new) token assigned to the request by the XMS task. For a RECEIVE or PURGE R0 holds the token which was assigned when the original request was made.
00	04	result	USER RC	Function gave non-zero result code, for instance user exec gave non-zero return code which is returned in register 0 or RESOLVE was for a reset variable.
00	0c		EXEC INIT	XMS SYSTEM EXEC did not initialize (internal error).
00	10		function	Invalid function code found in the parameter list supplied to module XMDIALOG.
00	14	result	SYSTEM RC	XMS system exec gave bad result code; R0 contains the decimal result code.
		less than 4096		From user authorization exit, an installation-defined reason code for disallowing the request.
		more than 4096		From SYSTEM EXEC:
		5104		failed while examining the function.
		5108		function requested is not supported.
		5112		error processing SET.
		5116		error processing EXEC.
		5120		error processing RESOLVE.
		5124		error processing LOG.
		5128		error processing user EXEC; EXEC ended with an invalid return code greater than 4095
00	18	token	timeout ok	The timeout specified on a synchronous request expired and the request was purged.
				continued....

Reason code	Return code	Result code	Description	Explanation
XX	1C	token	timeout failed	The timeout specified on a synchronous request expired but the request could not be purged.
00	20	abcc	abend	XMDIALOG abended (for example S0C4). R0 contains the abend code returned by the operating system in the field SDWAABCC of the system diagnostic work area, that is, in the form XXSSSUUU where XX=FLAGS, SSS=SYSTEM ABEND CODE, UUU=USER ABEND CODE.
XX	20	token	no XMS	The XMS system name specified via XMDIALOG (...*XMSYS=xxxx*) has not been started or is shutdown.
XX	24	token	QUERY SYNC	error on synchronous (OPTCD=SYN) request.
XX	28	token	QUERY ASYNC	error on asynchronous (OPTCD=ASY) request.
XX	2C	token	QUERY NOREPLY	error on noreply exec (OPTCD=NOREPLY) request.
XX	30	token	RECEIVE *token*	error on RECEIVE specific (TOKEN=*token*). Reason code 04, return code 30 can occur for an OPTCD=SYN request, indication that the request has been processed but that a RECEIVE should be performed separately to obtain the result.
XX	34	token	RECEIVE *any*	error on RECEIVE *any* (NO token)
XX	38	token	not purged	PURGE not done SPECIFIC-(TOKEN=*token*)
XX	3C	token	not purged	PURGE not done ANY- (NO TOKEN)
00	40	IUCV code	cross memory error in the IUCV component (VM only)	During function=OPEN, MAXCONNS exceeded no paths available. For IUCV codes refer to the following IBM manuals (see note on next page): • GCS Command and Macro Reference • CMS Macro Reference • CP Programming Services
00	4C	IUCV code	cross memory error in the IUCV component (VM only)	XMDIALOG function performed before OPEN. For IUCV codes refer to the following IBM manuals (see note below): • GCS Command and Macro Reference • CMS Macro Reference • CP Programming Services

Reason code	Return code	Result code	Description	Explanation
00	50	IUCV code	cross memory error in the IUCV component (VM only)	error in user directory; probably IUCV not allowed. For IUCV codes refer to the following IBM manuals (see note below): • GCS Command and Macro Reference • CMS Macro Reference • CP Programming Services

Note on IUCV codes (VM only)

The above XMDIALOG codes are listed in hexadecimal format. Take care when checking the IUCV codes in IBM manuals, as these may be in decimal. When IUCV codes are listed in decimal, they often take the form 1*xxx*, where *xxx* is the IPRCODE returned by IUCV to aid diagnosing the error.

Example

A common return code from XMDIALOG is as follows;

R15=00000044 R0=000003F3

The XMDIALOG reason code is 00 and the return code is 44. This indicates that a cross memory error has occurred in the IUCV component (see above table).

R0 contains the result code/IUCV code. If this hexadecimal code (3F3) is converted to decimal (1011), the IPRCODE can be seen to be 011. The explanation for this code is:

'The virtual machine with which you are establishing a path is not logged on'

The appropriate action is to check that the virtual machine name coded on the XMDIALOG OPEN macro as the NCIVM = parameter is logged on and that NC-XMS has started successfully.

Reason code table

Reason code	Return code	Description	Explanation
04	YY	storage	No free XM storage queue elements. Alter XMOPTION to specify more storage elements and restart the XMS system or reduce the number of concurrent outstanding requests. Can be caused by a job that issues OPTCD=ASY requests with no corresponding RECEIVEs or PURGEs.
08	YY	not found	Element to RECEIVE or PURGE was not found; an outstanding request (for PURGE), or result (for RECEIVE) for the jobstep which issued the XMDIALOG macro could not be found for the given token if one was specified, or with any token if not, for reasons such as: • the OPTCD=ASY request has not yet been processed, so retry the RECEIVE later or PURGE the request • token is invalid - no XMDIALOG request has been given the specified token • the request has been PURGEd already • the result has already been RECEIVEd explicitly via XMDIALOG RECEIVE • the result has already been RECEIVEd implicitly as part of OPTCD=SYN processing • the request was OPTCD=NOREPLY
0C	YY	fast purge	Not fast purged - XMPOST in progress. An explicit XMDIALOG PURGE or one done implicitly because a timeout period was specified on a synchronous request (OPTCD=SYN) or omitted) and expired, did nothing because the XMS server task was processing the original request and could not free the storage until finished. Try PURGE or RECEIVE later.
10	YY	PC function	Bad PC function code requested via register 0. Invalid attempt to use the program call services provided by the XMS task owing to internal error within module XMDIALOG or unsupported direct call.
20	YY	no XMS	XMS subsystem communication table not found or marked inactive, The XMS system name specified via XMDIALOG..., XMSYS=*xxxx* has not been started or is shutdown.

Reason code	Return code	Description	Explanation
1C	24	ECB address	An OPTCD=SYN was made specifying a work area above the 24mb line. The work area must be resident below 24mb.
24	00	macro	VM ONLY - XMSYS= your own userid is not permitted on open function.
28	00	sever	VM ONLY - other side severed connection, but data has been returned.
	04	sever	VM ONLY - as above but NO data returned.

CLOSE (VM only)

Description

This function is used to shut down the XMS environment. It must be issued to terminate cross memory functions.

Note: Do not release XMDIALOG work areas (defined by XMDIALOG MF=L) before this function is issued.

Format

XMDIALOG CLOSE,

MF=(E, *list*)

Return codes

After execution of an XMDIALOG CLOSE request register 15 will contain one of the following:

0000	XMS shut down successfully.
0044	IUCV error occurred. Error is in register zero.

For a full list of return codes see *Notes on the XMDIALOG macro* on page 7.22.

EXEC

Description

This function causes the XMS task to run the named EXEC routine, with the option of a parameter string supplied in AREA=*area*. If a string is specified then so must its length, in the LEN=*len* option. The maximum number of operands passed to the user EXEC routine is as supported by the NCI/XF EXEC statement.

Format

XMDIALOG EXEC,
NAME=*name*,
[AREA=*area*,
LEN=*len*,]
[MF=(E, *list*),] (mandatory for VM)
[XMSYS=*xmsys*,]
PARSE=NO/YES,
[ERROR=*error*]

where

name can be:

- the name of an eight byte field which contains the name of the EXEC routine
- a register, for instance (4)
- a register equate, for instance (R4), pointing to such an area
- assigned the name of a routine enclosed in quotes.

area can be:

- the name of an area containing a string to be passed to the EXEC'd routine (see the PARSE= operand)
- a register, for instance (4)
- a register equate, for instance (R4), pointing to such an area.

area should be defined at least as long as the following parameter *len*.

len can be:

- the length of area to be used
- a register, for instance (4)
- a register equate, for instance (R4), containing such a length.

area should be defined at least as long as *len*.

Note: if AREA=*text* is used, *len* may be omitted. *len* defaults to the length between the quotes. The maximum permitted length of the parameter string in *area* is 232 bytes (see *Restrictions on the LEN parameter* on page 7.21)

PARSE=NO The string defined as AREA= will be passed as one parameter in variable *&usrex1* to the user EXEC routine *name*.

PARSE=YES (DEFAULT) The string defined as AREA= will be parsed into separate parameters (*&usrex1*, *&usrex2*, ... , *&usrexn*) using blanks as delimiters, and these variables passed to the user EXEC routine *name*.

Return codes

After execution of an XMDIALOG EXEC request register 15 will contain one of the following:

0000 EXEC routine ran successfully.

0004 user EXEC routine exited with non-zero return code, which is returned in register 0.

0014 authorization exit (*&*%xmx02*) disallowed the user EXEC to be run by exiting with a non-zero return code which is returned in register 0.

0018 TIMEOUT period specified was exceeded before the user EXEC had finished. Register 0 will still contain the token value as if *OPTCD=ASY* had been specified. This can be used in a subsequent XMDIALOG RECEIVE.

For a full list of return codes see *Notes on the XMDIALOG macro* on page 7.22.

Example

```
XMDIALOG EXEC,
             NAME=JMSEXEC,
             AREA=MYDATA,
             LEN=L'MYDATA,
             XMSYS=PROD,
             PARSE=YES
```

Before the user EXEC JMSEXEC is run, *area* (MYDATA) will be parsed into separate parameters (*&usrex1* and *&usrex2*), using blanks as delimiters, *unless* the value in *area* contains quoted substrings in which case the characters between quotes count as one word.

If *area* is defined as DC C'DO ''THE RIGHT THING''' - that is, it has the first word DO and the second word THE RIGHT THING - then *&usrex1* will be DO (two characters) and *&usrex2* will be THE RIGHT THING (fifteen characters).

LOG

Description

This function allows the user to write a line to the current NCI/XF log.

Format

XMDIALOG LOG,
AREA =*area*,
LEN =*len*,
[MF=(E, *list*),] (mandatory for VM)
[XMSYS=*xmsys*,]
[ERROR=*error*]

where

area contains a string to be written to the LOG; *area* should be at least as long as the following parameter *len*.

len can be:

- the length of the area to be used
- a register, for instance (4)
- or a register equate, for instance (R4), containing such a length.

area should be at least as long as *len*. The maximum length of the parameter string in *area*, and so the value of *len*, is determined by the number of bytes supported by the NCI/XF LOG statement (see *LOG* on page 1.77).

Return codes

After execution of an XMDIALOG LOG request register 15 will contain one of the following:

0000 LOG command successful

0014 user authorization exit for LOG function (XMEXIT04) has returned a non-zero result code (in register 0)

For a full list of return codes see *Notes on the XMDIALOG macro* on page 7.22.

Example

```
XMDIALOG LOG,
               AREA=EGMSG,
               LEN=L'EGMSG
```

where

EGMSG DC C'MODNAME-TEXT'

is a LOG message

OPEN (VM only)

Description

This function is used to set up and initiate the XMS environment. It must be issued before any other cross memory service functions can be used (in any virtual machine)

Format

XMDIALOG OPEN,

NCIVM=*nci-vmname*,

MF=(E,*list*)

where

nci-vmname is the name of an eight byte area containing a 1 to 8 byte VM NCI/XF virtual machine id. It is mandatory and there is no default.

Example

```
LA   R2,=CL8'NCI310  '
XMDIALOG OPEN,
               NCIVM=(R2),
               MF=(E,MFAREA)
```

This will set up an XMS environment enabling a virtual machine (running either CMS or GCS) to communicate with the NCI/XF virtual machine whose userid is NCI310.

Return codes

After execution of an XMDIALOG OPEN request register 15 will contain one of the following:

0000 XMS initiated successfully

0044 IUCV error occurred. Error is in register zero.

0048 issuing userid is XMSYS and NCIVM. This is not allowed.

0050 error in user directory; probably IUCV.

For a full list of return codes see *Notes on the XMDIALOG macro* on page 7.22.

PURGE

Description

This function purges a previous request from the queue of XMS work elements. Any type of request can be PURGEd, either before it has been processed (for example an XMDIALOG ..., OPTCD=ASY or OPTCD=NOREPLY), or after (in place of a RECEIVE for an OPTCD=ASY request), or after a timeout failed return code (28) from a previous synchronous request.

Format

XMDIALOG PURGE,

[AREA=*area*,
LEN=*len*,]
[TOKEN=*token*,]
[MF=(E, *list*),] (mandatory for VM)
[ERROR=*error*,]
[TRACE=*n*]

where

area is the name of the area which contains the request to be purged; *area* should be at least as long as the following parameter *len*.

len can be:

- the length of area to be used
- a register, for instance (4)
- a register equate, for instance (R4), containing such a length.

area should be at least as long as *len*. The length of the contents of area is limited to 232 bytes - for an explanation of this see *Restrictions on the LEN parameter* on page 7.21.

token the token of the request to be purged. See *Common XMDIALOG parameters* on page 7.18.

Return codes

After execution of an XMDIALOG PURGE register 15 contains one of the following:

0000	PURGE command successful
04xx	no free XM storage QUEUE elements
08*xx*	The token was not found or has not yet finished processing.
0C*xx*	Not purged, still active. (MVS only).

For a full list of return codes see *Notes on the XMDIALOG macro* on page 7.22.

Examples of return and reason codes in register 15

0838	not PURGEd (xx38) because request was not found (08xx)
083C	not PURGEd (xx30) because nonspecific request was not found (VM only).
0C38	not PURGEd (xx38) because the request was in the process of being completed by the XMS server task (0Cxx). The PURGE request should be retried.

RECEIVE

Description

This function receives the response, if it is ready, generated by XMS to a previous request where OPTCD=ASY was specified.

Format

XMDIALOG RECEIVE,

[AREA=*area*,
LEN=*len*,]
[TOKEN=*token*,]
[MF=(E, *list*),] (mandatory for VM)
[ERROR=*error*,]
[TRACE=*n*]

where

area is the name of the area to receive value of variable requested by previous RESOLVE; *area* should be at least as long as the following parameter *len*.

len can be:

- the length of area to be used
- a register, for instance (4)
- a register equate, for instance (R4), containing such a length.

area should be at least as long as *len*. The length of the contents of *area* is restricted to 232 bytes; for an explanation of this see *Restrictions on the LEN parameter* on page 7.21.

token the token of the request whose response is wanted now. If omitted, any outstanding response may be returned. See *Common XMDIALOG parameters* on page 7.18.

Return codes

After execution of an XMDIALOG RECEIVE register 15 contains one of the following:

0 RECEIVE command successful

4 no free XM storage QUEUE elements (04 might for instance precede return code 30 in which case there are no more storage queue elements.)

08xx The token was not found or has not yet finished processing.

For a full list of return codes see *Notes on the XMDIALOG macro* on page 7.22.

Examples of return codes in R 15:

0830 specific receive not found

0834 nonspecific receive not found (VM only)

RESOLVE

Description

This function is used to return the current value of a variable to the program.

Format

XMDIALOG RESOLVE,
VAR=*var*,
AREA=*area*,
LEN=*len*,
[MF=(E, *list*),] (mandatory for VM)
[XMSYS=*xmsys*,]
[ERROR=*error*]

where

var can be:

- the name of an eight byte area which contains the name of the variable to be resolved
- a register, for instance (4)
- a register equate, for instance (R4), pointing to such an area
- the name of the variable in quotes preceded by two ampersands: '&&jim'.

area is the name of the area where the RESOLVEd value is to be put, *area* should be at least as long as the following parameter *len*.

len can be:

- the length of area to be used
- a register, for instance (4)
- a register equate, for instance (R4), containing such a length.

area should be at least as long as *len*. The length of the contents of *area* is restricted to 232 bytes - for an explanation of this see *Restrictions on the LEN parameter* on page 7.21.

Return codes

After execution of XMDIALOG register 15 contains one of the following:

0000 RESOLVE worked

0004 RESOLVE worked but variable had a NULL value

For a full list of return codes see *Notes on the XMDIALOG macro* on page 7.22.

Examples

```
XMDIALOG RESOLVE,
                  VAR=VARNAME,
                  AREA=DATAREA,
                  LEN=20
```

The first 20 bytes of variable *&userid* will be put in the work area called DATAREA; the program work areas are defined as follows:

```
VARNAME DC CL8'&&USERID'
DATAREA DS CL20
```

```
XMDIALOG RESOLVE,
                  VAR=(5),
                  AREA=D(6),
                  LEN=20
```

The value of the variable whose name is pointed to by register 5 will be put in the storage area indicated by register 6.

Note: It is especially important that *area* be at least as long as *len* because whatever follows would be overwritten if the variable is actually longer than the defined length of *area*. Any user variables RESOLVEd by XMDIALOG belong to the XMS-started task of NCI/XF and so are common to all users of XMDIALOG RESOLVE; but they cannot be accessed by any other tasks or users of NCI/XF unless copied into system variables by an XMDIALOG EXEC. It is therefore good practice to set the system variables to start with a colon (:) or some sort of convention which provides each caller with a unique variable name.

If RESOLVE worked, R1 contains the true length of the data, which can be larger than that specified in the LEN= parameter. This allows truncation to be detected and the end point of the resolution variable for the RESOLVE to be reissued using a larger area.

SET

Description

The SET function of XMDIALOG sets a variable to a value.

Format

XMDIALOG SET,

VAR=*var*,
AREA=*area*,
LEN=*len*,
[MF=(E, *list*),] (mandatory for VM)
[XMSYS=*xmsys*,]
[ERROR=*error*]

where

var can be:

- the name of an eight byte area containing the name of the variable to SET
- a register, for instance (4)
- a register equate, for instance (R4), pointing to such an area
- the name of the variable preceded by a double ampersand and enclosed in quotes: '*&&fred*'.

area can be:

- the name of the area which contains the value to be set
- a register, for instance (4)
- a register equate, for instance (R4), pointing to this area.

area should be defined at least as long as the following parameter *len*.

len can be:

- the length of area to be used
- a register, for instance (4)
- a register equate, for instance (R4), containing such a length.

area should be defined at least as long as *len*. The maximum length of variable which may be SET is 232 bytes. See *Common XMDIALOG parameters* on page 7.18.

Return codes

After execution of an XMDIALOG SET request, register 15 will contain one of the following:

0000 variable SET successfully

0014 set failed. Register 0 will contain a value less than 4096 if authorization exit for SET function (XMEXIT01) has refused the SET, and a value of 5112 if NCI/XF could not set the variable because of parm error (for instance invalid variable name)

For a full list of return codes see *Notes on the XMDIALOG macro* on page 7.22.

Examples

```
XMDIALOG SET,
                  VAR=VARNAME,
                  AREA=DATAREA,
                  LEN=5
```

The first five bytes of data stored in DATAREA will be put in variable *&cinit*, where the program work areas are defined as follows:

```
VARNAME DC CL8'&&CINIT'
DATAREA DC C'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

After execution of the NCDIALOG macro, *&cinit* will be set as if the following language statement had been executed

```
SET &CINIT = 'ABCDE'
```

```
XMDIALOG SET,
                  VAR=(5),
                  AREA=(6),
                  LEN=200
```

The 200 bytes of data which start at the location indicated by register 6 will be stored in the variable whose name is pointed to by register 5.

Notes

Any user variables SET by XMDIALOG belong to the XMS server task of NCI/XF and so are common to all users of XMDIALOG SET; but they cannot be accessed by any other users of NCI/XF unless copied into system variables by an XMDIALOG EXEC. It is therefore good practice to set system variables to start with a colon (:) or some sort of convention which provides each caller with a unique variable name.

Chapter 8 - NELOGON and NC-EDIT variables

NELOGON 8.2
NC-EDIT variables (MVS) 8.6
NC-EDIT variables (VM) 8.10

This chapter should be read in conjunction with the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*.

NELOGON

If NC-EDIT is invoked from an XF application the NELOGON EXEC is used.

A successful call to NELOGON passes control to the NC-EDIT subsystem. On exit, control is passed to the panel that contained the EXEC NELOGON statement and not to the statement following the EXEC call.

Format

EXEC NELOGON *&userid* [*&neopts*]

where

&userid defines the userid (maximum length 8) or userid/password if *&*<secm1* has been set. (For security reasons, if userid/password is specified, the password must have previously been encrypted with the NCI/XF ENCRYPT verb)

&neopts defines the EDIT environment for the user. *&neopts* is set to a string of positional keywords, separated by commas, which are placed in the following variables by NELOGON. Where trailing operands are not required, trailing commas are not necessary. The numbers indicate position.

4. *&<mode* edit mode:

   M (default) NC-EDIT PRIMARY OPTION menu to be displayed (xNE100n). *&usrproj*, *&usrlib* and *&usrtype* are the displayed qualifiers on the edit entry screen.

   E edit of a preprogrammed dataset is required where operand 5 defines fully qualified DSN and member name

   B browse of preprogrammed dataset required

   If *&<mode* is not set to M (Display Menu) then *&usrproj* (operand 5) should contain a preprogrammed fully qualified DSN and member, and *&usrlib* (operand 6) should contain the volume id if dataset not catalogued.

5. *&<toexec* user timeout exec

6. *&<tocmd* timeout command, for example SAVE

7. *&scq* scroll quantity

8. *&usrproj* file's first level qualifier. This is the displayed qualifier on the edit entry screen if *&<mode* is set to M (Display Menu). If *&<mode* is not set to M then *&usrproj* should contain a preprogrammed fully qualified DSN and member.

9. *&usrlib* — file's second level qualifier. This is the displayed qualifier on the edit entry screen if *&<mode* is set to M (Display Menu). If *&<mode* is not set to M then *&usrlib* contains the volume id if the dataset is not catalogued.
10. *&usrtype* — file's third level qualifier. This is the displayed qualifier on the edit entry screen if *&<mode* is set to M (Display Menu).
11. *&usrelim* — edit storage limit
12. *&usrnull* — editor NULLS status
13. *&usrstat* — editor STATS status
14. *&usrhex* — editor HEX support status
15. *&usrcaps* — editor CAPS status
16. *&<enqmod* — set to Y or N (user is or is not authorized to issue ENQUEUE or DEQUEUE)

Notes:

There is an option of using the)PROC exit (NE100PEX) of the NC-EDIT PRIMARY OPTION menu (NE100) to examine, set and override the above keywords.

The dataset name will be prefixed by *&usrproj* (first level qualifier) unless the dataset name is enclosed in quotes.

If *&neopts* is enclosed in quotes, use double quotes to avoid a parameter error during the passing of data to NELOGON. See the first example below.

Examples

```
set &neopts = "E,,SAVE,,'CKDNGF1.DATA(MEMBER1)'"
```

This causes the edit of a preprogrammed dataset name CKDNGF1.DATA(MEMBER1). By specifying E as the first operand the user has requested the edit of a preprogrammed dataset. There is no user timeout EXEC because the second operand space is empty. If the session times out, the editor will do an automatic save because of the third operand. No fourth operand is specified so the operation defaults to the default scroll quantity PAGE. See variable *&SCQ* on page 8.6.

```
set &neopts = 'M,MYTIMEO,SAVE,CSR,,,,E250K'
```

In this example, the second operand specifies a user exec driven on a timeout **after** the standard editor timeout is driven. See variable *&SCQ* on page 8.6 for details of CSR. The final operand specifies a limit on the size of the dataset or file to be edited. (There is no comma after the final operand, since there is no trailing operand.)

Return codes

Variables *&<retc* and *&¢<retc* are set on completion of processing. Variable *&<retc* contains any parameter error discovered in NC-EDIT. When *&<retc* is 56, 60 or 84, *&¢<retc* contains further information on the nature of the error.

&<retc	&¢<retc	Description
0		successful return
12		mandatory variable(s) not passed to NELOGON. Refer to the NCI/XF log for variable name.
20		mismatched quotes on dataset name
24		mismatched brackets on dataset name
28		library name not entered
32		library type not entered
36		member list screen requested by NELOGON, but fully qualified dataset implies request for entry at Member screen level
40		editor screen requested by NELOGON, but dataset definition implies request for entry at Member List screen level
44		dataset in use
48		member in use
52		user enqueue name has invalid format/incorrect length
56	(4)	dataset not sequential
56	(8)	invalid dataset name
56	(12)	invalid member name
56	(16)	dataset not partitioned
56	(20)	invalid volume serial or invalid CMS mode
56	(24)	invalid dataset org
56	(28)	unable to allocate dataset
56	(32)	unable to open dataset
56	(40)	unprintable characters in dataset
56	(48)	dataset too large to edit
56	(52)	invalid dataset recfm
56	(56)	invalid record length
56	(60)	member name not found during browse of preprogrammed dataset
56	(64)	module abended - abend code held in *&¢abcode*

continued...

&<retc	&¢<retc	Description
60	(8)	invalid dataset
60	(12)	invalid member name
60	(16)	dataset not partitioned
60	(20)	invalid volume serial number or invalid CMS mode
60	(24)	invalid dataset organization
60	(28)	unable to allocate dataset
60	(32)	unable to open dataset
60	(64)	module abended - abend code held in *&¢abcode*
64		assembler module name(s) not passed to NELOGON
68		variable passed to NELOGON has invalid length. Refer to the NCI/XF log for variable name
72		variable passed to NELOGON is not numeric; refer to NCI/XF log for variable name
80		Invalid number of operands passed to NELOGON or invalid number of options specified in operand 2 on the execution of NELOGON; *&usrretc* is also set to 80
84	(4)	RACINIT return code 4; the user profile is not defined to RACF
84	(8)	RACINIT return code 8; the password is not authorized.
84	(12)	RACINIT return code 12; the password has expired.
84	(3997)	EDRACFI return code 3997; invalid userid
84	(3998)	EDRACFI return code 3998; invalid password
84	(3999)	EDRACFI return code 3999; program not running authorized

Variable	Description
&SCQ	The default scroll amount, as displayed in the top right of the main edit panel. Possible values: (default) PAGE Scroll 1 page at a time. CSR Scroll to cursor position. HALF Scroll half a page at a time. MAX Scroll as far as possible in the chosen direction. *nnnn* Scroll by *nnnn* lines or cols.
&USRCAPS	Indicates whether any data entered is to be automatically translated to UPPERCASE or left as entered. Possible values: Y Translate data to uppercase. (default) N Leave data as entered.
&USRELIM	Specifies the amount of virtual storage a user can own during an edit session. Possible values for X*nnn*Y: Y=B maximum *nnn* bytes of storage will be available for the dataset. Y=K maximum *nnn* K of storage will be available for the dataset. X=E datasets which will not fit nnnY cannot be edited X=W datasets which will not fit nnnY storage will be truncated If X is omitted, it defaults to E. If *&usrelim* is not defined, it defaults to E250K.
&USRFDSN	The name to be inserted in the *Dataset name* field of the Edit Entry Panel. Possible value: a valid dataset name qualifier
&USRHEX	Indicates whether editing of data containing non-displayable characters is permitted. Possible values: E Refuse the edit request (default) W Allow the edit session

Variable	Description
&USRLIB	The name to be inserted in the library field of the edit entry panel. Possible value: a valid dataset name qualifier.
&USRMEM	The name to be inserted in the member field of the edit entry panel. Possible value: a valid PDS member name.
&USRNULL	Indicates whether the data fields on the screen are to be padded with trailing blanks or nulls. Possible values are: Y Insert trailing nulls (default) N Insert trailing blanks
&USRPROJ	The name to be inserted in the project field of the edit entry panel. Possible value: a valid dataset name qualifier. The default is to set equal to the user's *userid*.
&USRCRL	Defines the number of editable data lines to be displayed. Possible value: any integer between 1 and 40
&USRSCRM	Defines the number of editable data lines to be displayed. Possible value: any integer between 1 and 40
&USRCRW	Defines the required width of the edit panel. Possible value: any integer between 1 and 132.
&USRSTAT	Indicates whether SPF type statistics are to be generated when a PDS member is created or updated. Possible values: (default) Y Statistics generated N Statistics not generated
&USRTYPE	The name to be inserted in the type field of the edit entry panel. Possible value: a valid volume serial number)

Variable	Description
&USRV	The name to be inserted in the volume serial field of the edit entry panel. Possible value: a valid volume serial number
&USRPACK	Contains the pack status. Y means that dataset is saved packed (the default), N that it is saved unpacked. Possible values: (default) Y(=PACK ON) N(=PACK OFF)
&<MODEL	Default 2 to 5 according to terminal characteristics. Used to determine which panels are displayed. Can be modified to cause, for instance, model 5 screens to display in model 2 mode.
&<TOPM	Contains the value for the top margin setting, for use with cursor scrolling on the edit panel. Possible value: any valid screen row. The default value is 2.
&LEFTM	Contains the value for the left margin setting, for use with cursor scrolling on the edit panel. Possible value: any valid screen column The default value is 8
&<CALLTO	Name of an EXEC routine to be executed if the user is timed out. This is a user routine and would, typically, log a message to say what has happened before recovering the user. Possible value: any valid member name
&<TOCMD	Command to be executed on the current edit session if the user is timed out. Possible values: CANCEL (default) SAVE
&<ENQMOD	User is allowed to cancel existing enqueues and issue enqueue/dequeue. Possible values: (default) N issue warnings only Y cancel enqueues

Variable	Description
&<COPYEX	Exit routine called before COPY, CREATE or REPLACE command. Routine is passed *userid*, *DSname*, *member* and *volume* to allow decision on action. These parameters are passed in the standard *&usrexn* variables to the EXEC routine. Sample routine NECOPYEX supplied.
&<EDITEX	Exit routine called before any edit primary command. Routine is passed *userid* and *command* to allow decision on action. These parameters are passed in the standard *&usrexn* variables to the EXEC routine. Sample routine NEEDITEX supplied.
&<READEX	Exit routine called before reading dataset or member. Routine is passed *userid*, *DSname*, *member* and *volume* to allow decision on action. These parameters are passed in the standard *&usrexn* variables to the EXEC routine. Sample routine NEREADEX supplied.
&<WRITEX	Exit routine called before writing dataset or member. Routine is passed *userid*, *command*, *DSname*, *member* and *volume* to allow decision on action. These parameters are passed in the standard *&usrexn* variables to the EXEC routine. Sample routine NEWRITEX supplied.
&<MODE	Edit mode Possible values: (default) M NC-EDIT primary option menu to be displayed (xNE100n) E means that the user wants to edit a preprogrammed dataset where operand 5 defines fully qualified *dsn* and member name, and operand 6 defines optional volume. B means browse of preprogrammed dataset required.(

Variable	Description
&SCQ	The default scroll amount, as displayed in the top right of the main edit panel. Possible values: (default) PAGE Scroll 1 page at a time. CSR Scroll to cursor position. HALF Scroll half a page at a time. MAX Scroll as far as possible in the chosen direction. *nnnn* Scroll by nnnn lines or cols.
&USRCAPS	Indicates whether any data entered is to be automatically translated to UPPERCASE or left as entered. Possible values: Y Translate data to uppercase. (default) N Leave data as entered.
&USRELIM	Specifies the amount of virtual storage which a user may own during an edit session. Possible values: E*nnnn*K Allow use of up to nnnK bytes. Files which will not fit can not be edited. W*nnnn*B Allow use of up to nnnn bytes. Files which will not fit will be truncated. (default) E250K
&USRFDSN	The name to be inserted in the Dataset name field of the Edit Entry Panel. Possible value: a valid CMS filename
&USRHEX	Indicates whether editing of data containing non-displayable characters is permitted. Possible values: E Refuse the edit request (default) W Allow the edit session

Variable	Description
&USRMEM	The name to be inserted in the member field of the edit entry panel. Possible value: a valid CMS filetype
&USRNULL	Indicates whether the data fields on the screen are to be padded with trailing blanks or nulls. Possible values: Y Insert trailing nulls (default) N Insert trailing blanks
&USRCRL	Defines the number of editable data lines to be displayed. Possible value: any integer between 1 and 40
&USRSCRM	Defines the number of editable data lines to be displayed. Possible value: any integer between 1 and 40
&USRSCRW	Defines the required width of the edit panel. Possible value: any integer between 1 and 132.
&USRFVOL	The name to be inserted in the *CMS Filemode* field of the Edit Entry Panel. Possible value: any integer between 1 and 132
&USRTYPE	The name to be inserted in the type field of the edit entry panel. Possible value: a valid volume serial number
&<TOPM	Contains the value for the top margin setting, for use with cursor scrolling on the edit panel. Possible value: any valid screen row. The default value is 2.

Variable	Description
&LEFTM	Contains the value for the left margin setting, for use with cursor scrolling on the edit panel. Possible values: any valid screen column The default value is 8
&<SCURR	Contains the current cursor row, for use with cursor scrolling on the Edit panel. Possible value: any valid row within the data
&<SCURC	Contains the current cursor column, for use with cursor scrolling on the Edit panel. Possible value: any valid column within the data
&<RL	The value to be inserted in the *Record Length* field of the Edit Entry panel when creating new CMS files. Possible value: any valid record length
&<RECFM	The value to be inserted in the *Record Format* field of the Edit Entry panel when new CMS files are being created. Possible values: F Fixed V Variable

Chapter 9 - NCI/XF panels and distributed routines

PANELS
NCI/XF product panels 9.2
USE Panels 9.3
Demonstration Panels 9.6
Other panels 9.8

DISTRIBUTED COMPILED FUNCTION EXECS
@UTC2X 9.9
@UTIBMC 9.10
@UTOFF 9.11
@UTROUTE 9.12
@UTSLOG 9.13
@UTSMF 9.14
@UTSORT 9.15
@UTSTATS 9.16
@UTX2C 9.17
CPUTEST 9.18

DISTRIBUTED CALL ROUTINES
NCALERT 9.19
NCDIALC 9.22
NCENCDEC 9.24
NCPEEK 9.26
NCQLOG 9.27
NCRACFX 9.28
NCRAND 9.30
NCSLOG 9.31
NCVLOG 9.32
NCWAIT 9.33
NCWTOR 9.34

NCI/XF product panels

These panels are used mainly by the NCI/XF assembler load modules. They are distributed in non-compiled format and can be modified if required.

Panel	Description
@XFBASE	run if MODE = ENHANCED, TDT = ENGLISH and USE not active
@XFLOGON	default LOGON EXEC MODE = ENHANCED
@XFSTART	sample INITIAL-EXEC MODE = ENHANCED
NCERROR	panel error reporting and recovery
NCERR1	unable to locate/open panel
NCERR1A	unable to locate/open panel
NCERR10	panel not authorized
NCERR10A	panel not authorized
NCERR2	errors encountered on panel build
NCERR2A	errors encountered on panel build
NCERR3	screen size incompatibility
NCERR3A	screen size compatibility
NCERR4	GOTO or EXEC limit exceeded
NCERR4A	GOTO or EXEC limit exceeded
NCERR5	increment/decrement error
NCERR5A	increment/decrement error
NCERR6	GOTO statement in error
NCERR6A	GOTO statement in error
NCERR7	variable generated invalid
NCERR7A	variable generated invalid
NCERR8	abend in call
NCERR8A	abend in call
NCERR9	terminal no longer in TDT
NCERR9A	terminal no longer in TDT

USE Panels

These panels are used by the User Support Environment. They are distributed in non-compiled format and can be modified if required.

Panel	Description
DNCMOD1	verify model terminal in German
DNCMOD2	verify model terminal in German
DNCMOD3	verify model terminal in German
DNCMOD4	verify model terminal in German
DNCMOD5	verify model terminal in German
DNC9311	send/receive message panel in German
DNC9312	send/receive message panel in German
DNC9313	send/receive message panel in German
DNC9314	send/receive message panel in German
DNC9315	send/receive message panel in German
ENCMOD1	verify model terminal in English
ENCMOD3	verify model terminal in English
ENCMOD4	verify model terminal in English
ENCMOD5	verify model terminal in English
ENCSETHD	set panel name heading
ENC0032	screen burnout
ENC0052	main logo
ENC0062	change password
ENC0082	logo clock
ENC0102	menu for multi-product environment
ENC1002	selection menu
ENC1602	system options menu
ENC1702	system options max users help

continued...

Panel	Description
ENC1712	system options date formats help
ENC1732	system options idle timeout help
ENC1742	system options terminal logoff help
ENC1752	system options vtam logoff help
ENC1762	system options disable clock help
ENC1772	system options multi logon help
ENC1792	system options exits help
ENC2002	MHO menu
ENC2102	MHO wait panel
ENC4002	user profile definition
ENC4102	user profile definition help
ENC4112	user profile definition language support help
ENC4122	user profile definition authority/group help
ENC4132	user profile definition primary product help
ENC4142	user profile definition password validation help
ENC4152	user profile definition logoff to vtam help
ENC4352	user profile definition list of all users
ENC9002	operator menu
ENC9102	list logged on users
ENC9311	send receive messages
ENC9312	send receive messages
ENC9313	send receive messages
ENC9314	send receive messages
ENC9315	send receive messages

continued...

Panel	Description
ENC9322	CAA browse
ENC9332	trace options menu
ENC9982	shutdown panel
ENC9992	shutdown panel
ENGLISH	initial TDT panel
EXF0052	NCI/XF logo
E0052	system not initialized logo
FNCMOD1	verify model terminal in French
FNCMOD2	verify model terminal in French
FNCMOD3	verify model terminal in French
FNCMOD4	verify model terminal in French
FNCMOD5	verify model terminal in French
UEXIT03	message format exit
UEXIT07	startup/shutdown exit
UEXIT09	post-security exit
UEXIT10	pre-security exit

Demonstration Panels

These panels are used by the demonstration environment and show examples of the types of features NCI/XF could provide. They also show documented use of NCI/XF statements. The panels are supplied in non-compiled format and are not intended for modification.

Panel	Description
MCP	demonstration master control panel
ZZAUTOIN	demonstrate AUTOINPUT verb with a series of animated screens
ZZBAT1	BATTLESHIPS using TREFRESH verb
ZZBRLOG	browse NCI/XF log
ZZBRQUE	browse NCI/XF queues
ZZCALCUL	calculator using CALCULATE and CURSOR verbs
ZZCALCVB	demonstrate FCALCULATE verb
ZZCALL	demonstrate CALL VERB to invoke user exit
ZZCHAINS	demonstrate QUEUE verb
ZZCURSOR	demonstrate CURSOR verb
ZZDATEFM	demonstrates date information derived from NCI/XF reserved system variables
ZZDEMO	master control panel to demonstrate some NCI/XF verbs
ZZDEMO*nn*	rolling demonstration panels
ZZDYNAM1	demonstrate dynamic application menu building
ZZENQDEQ	demonstrate ENQUEUE/DEQUEUE verbs
ZZEXATTR	provides four hexbyte values for use with NCDIALOG ATTR macro
ZZEXCOLR	demonstrate NCI/XF color screen support for devices which permit extended data streams

continued...

Demonstration Panels - continued

Panel	Description
ZZFIXPAN	provides a table of applied NCI/XF assembler fixes
ZZHANG	HANGMAN using supplied HANGMAN assembler CALL routine
ZZIOSTAT	display VSAM statistics for any NC-VSAM file
ZZLOG	demonstrate LOG verb
ZZMENU	demonstration menu
ZZNCIMON	NCI/XF system monitor
ZZPAD	demonstrate PADRIGHT/PADLEFT/PADCENTER verbs
ZZPARSE	demonstrate PARSE verb
ZZQUEUE	demonstrate QUEUE verb
ZZREF	demonstrate PREFRESH and TREFRESH verbs
ZZSELP	demonstrate NCI/XF support of a selector pen
ZZSPEC	demonstrate special features menu
ZZSTRMON	demonstrate NCI/XF storage monitor
ZZTERMUP	demonstrate TERMUPDATE verb
ZZTRACE	demonstrate TRACE and TRACETERM verbs
ZZTRVERB	demonstrate TRACEVERB verb
ZZUPDATE	demonstrate UPDATE verb
ZZVERIFY	demonstrate VERIFY verb for data verification
ZZVEXEC	general purpose utilities panel to execute dynamic code and display variables

Other panels

These panels show documented use of NCI/XF statements. The panels are supplied in non-compiled format and are not intended for modification.

Panel	Description
BLANK	demonstrate screen burnout
BRODCAST	demonstrate broadcast panel for an NCI/XF system operator
CARTOON	demonstrate animated cartoon using AUTOINPUT verb
ENCRYPT	demonstrate ENCRYPT/DECRYPT
NCLUSTAT	sample panel used with LUSTAT = NCLUSTAT startup option
NCRELREQ	sample panel used with RELEASE = NCRELREQ startup option
OPERATOR	sample operator menu for controlling NCI/XF system tasks
PANPRT	sample panel for display on an *lutype 1* device, for instance a printer
PSERVIC	display screen related attributes, eg whether device is queriable

@UTC2X

This EXEC routine converts a character string to its hex equivalent (unpacked). The input data can be any length.

Format

EXEC @UTC2X *parm1* [*parm2* [*parm3*]]

where

parm1 is the input character string

parm2 is the stub of the variable (the name without &) used to return the derived hex string. If option V - for vertical output string - is chosen in *parm3*, the stub will be suffixed by the characters 1 or 2 to represent the high and low order nibbles respectively of the calculated hex values, otherwise only the stub will be used.

parm3 is either

V - resolve output as vertical string

or

H - (default) resolve output as horizontal string

Examples

```
EXEC @UTC2X 'ABC1' OPSTR
```

In this case *&opstr* will be returned with 'C1C2C3F1' - no value was specified for *parm3* so the output defaulted to a horizontal format.

```
EXEC @UTC2X 'ABC1' OPSTR V
```

In this case the following variables will be returned

&opstr1 'CCCF'

&opstr2 '1231'

@UTIBMC

This EXEC routine invokes C language programs from NCI/XF. Only C programs compiled and linkage edited using the IBM C program product Version 2 common library are supported. C programs cannot be invoked by the NCI/XF CALL language statement.

Format

EXEC @UTIBMC *parm1* [*varn* .....]

where

parm1 is the name of the C program

varn..... is up to 14 NCI/XF variable names without the leading ampersand which contain program parameter data.

Notes

Runtime parameters can optionally be passed to the C environment by setting variable *&ibmcprm*.

The IBM C runtime library must be available to the NCI/XF address space before this interface can be used. The SYSPRINT, STDERR, STDOUT and STDIN DD names should be allocated.

IBM C programs invoked by @UTIBMC can call the NCDIALC subroutine to access a number of NCI/XF services. Refer to *NCDIALC* on page 9.22.

User programs must contain the following compiler directive:

```
#pragma runopts(PLIST(MVS))
```

Return codes

The following return codes are returned in *&usrretc*:

0 program invoked successfully.

4 program name omitted.

8 variable name exceeds 255 characters.

12 variable name is null.

@UTOFF

This EXEC routine can be used to return the terminal to the logo or to VTAM when the user logs off. It can also be coded in the)UPDATE section of all panels to handle user time out. All user variables are reset and the user is removed from the list of logged on users. @UTOFF should always be used if the logon was via USE.

Format

EXEC @UTOFF

Note

The contents of the following variables will determine where the terminal will return.

&#lgoff
1 - return to logo
2 - return to VTAM (assuming system option *&*#lgof* = Y)

The meaning of this variable can be overridden by *&#lgof2*.

&#lgof2
1 - return to logo
2 - return to VTAM

Use of *&#lgof2* overrides *&#lgoff*.

&#lgof*
Y - return to VTAM allowed
N - return to VTAM not allowed (this will force the LOGOFF back to the logo)

Normally the variables listed have been preset during USE logon.

@UTROUTE

The NCI/XF PANEL statement is used to execute a specified panel member from the panel library (see the NCI/XF Reference Manual, *Chapter 1 - NCI/XF language statements*). NCI/XF does not automatically keep track of the panels executed, but only remembers the last panel called (in variable *&usrret* - see the NCI/XF Reference Manual, *Chapter 3 - NCI/XF defined user variables*). The EXEC routine @UTROUTE provides a mechanism whereby it will automatically branch to a panel name on the basis of the top entry of an NCI/XF queue named *#linkq*.

Format

EXEC @UTROUTE

Note

The contents of the following variables will determine where the terminal will return.

&#lang the user's national language (the first character of the panel name)

&#ttype the terminal model type (the seventh character of the panel name)

The above variables will be set if you are in USE.

Example

An NCI/XF queue called *#linkq* is used to store the names of the panels executed. This queue may be maintained by the user, who adds an entry (consisting of the two-character product code and three-character reference number of the panel) to the top for every panel required

&#lang	-	E
&#ttype	-	2
#linkq	-	EM260 (Top of queue)
		EM255
		EM230
		EM200
		EM100 (Bottom of queue)

@UTROUTE then uses the first entry on this queue (together with *&#lang* and *&#ttype*) to construct the panel name for execution. In the above example, to return to panel EEM1002 and visit every other panel in turn @UTROUTE is called five times, the top entry on the queue being deleted by the user between calls.

Error codes

This routine never exits back to the caller, so if any errors occur (owing to an incorrectly formatted or undefined *#linkq*) a message is written to the NCI/XF log and the routine initiates logoff processing.

@UTSLOG

This EXEC routine can be executed to call the distributed load module NCSLOG which prints on the NCI/XF log the number of times that an individual statement within a panel/routine has been executed. Information is printed in the following format.

PANEL *pan-name-section*

execution-count *statement*

execution-count *statement*

Format

EXEC @UTSLOG *parm1*

where

parm1 is a compulsory panel name mask to indicate the panel(s) whose statement execution counts are to be displayed. Two special mask characters can be used:

- * generic group of panels based on the characters before the asterisk (*)
- + any character may be present at this position in the panel name

Note

This module depends on the NCI/XF SXM (Statement eXecution Monitor) being enabled. See *SXM=YES/NO* on page 3.13 of the NCI/XF Installation Manual.

Also see *NCSLOG* on page 12.9 of the NCI/XF User Guide.

@UTSMF

This EXEC routine provides an SMF interface to enable SMF records to be written directly from NCI/XF. It requires NCI/XF to run APF authorized.

Format

EXEC @UTSMF *parm1 parm2*

where

parm1 is a numeric record id *nnn* in the range 128 to 255. This value forms the SMF record id stored in offset 5 of the SMF record.

parm2 is SMF data stored in the user area of the SMF record at offset X'12' onwards.

Example

```
EXEC @UTC2X 'ABC1' OPSTR
```

This will create an SMF record with X'D0' at offset 5 and user data (the NCI/XF jobname and the current NCI/XF date, at offset X'12' in the SMF record).

Return codes

The following return codes are returned in *&usrretc*:

0 record successfully written to SMF file.

4 error during the writing of the SMF record. An additional return code is available in temporary variable *&¢smf*; it is set equivalent to the return code from the SMFWTM macro documented in the IBM manual entitled MVS/XA SPL:SMF (GC28-1153-7) or equivalent.

16 invalid parameter list, for instance operand not numeric or not in range 128 to 255.

36 SMF record not written because the record identifier (passed in *parm1*) is not being recorded because of installation specific parameters defined in SYS1.PARMLIB(SMFPRMxx).

99 NCI/XF job not APF authorized.

@UTSORT

This EXEC routine performs a bubble sort on a specified NCI/XF queue and returns the specified queue sorted as requested.

Format

EXEC @UTSORT *parm1* [*parm2*] [*parm3*] [*parm4*]

where

parm1 is the name of the NCI/XF queue to be sorted.

parm2 is the position within each queue element that marks the start of the data to compare. If this parameter is omitted, then the starting position defaults to 1.

parm3 is the position within each queue element which marks the end of the data to compare. If this parameter is omitted, then the end position defaults to 255.

parm4 specifies the order in which to sort the queue. GT stipulates that the queue be sorted in ascending order; LT denotes descending order. If this parameter is omitted, then the sorting defaults to GT.

Example

If, for instance, the queue USERID contained a series of *userids*:

CKDUSR4 (top of queue)
CKDUSR1
CKATSG2
CKAUSR9
CKDUSR6
CKATSG8
CKAUSR3 (bottom of queue)

and the entries were required in sorted descending order, then calling @UTSORT as follows:

```
EXEC @UTSORT USERID 1 255 LT
```

would perform the function, leaving USERID sorted as required.

@UTSTATS

This EXEC routine can be executed to produce a summary on the NCI/XF log of NCI/XF statistics similar to the NCI/XF termination report.

Format

EXEC @UTSTATS

Note

The @UTSTATS routine will display, for instance, information relating to:

- the storage that NCI/XF is currently using
- storage used for NCI/XF load modules
- whether VTAM buffer shortages have occurred
- whether any VTAM RPL user field corruptions have occurred
- total number of terminals connected

Please also see the NCTERM log messages starting on page 11.22.

Statistics about the number of CALL routine invocations will also be displayed. These statistics are obtained when the @UTSTATS routine issues the MODSTATS verb. Executing @UTSTATS will therefore reset any *&usrmods* variables which you may have created.

All information written to the log is obtained from NCI/XF variables as described in the Reference Manual, *Chapter 2 - NCI/XF defined system variables* and *Chapter 3 - NCI/XF defined user variables*.

@UTX2C

This EXEC routine converts a hexadecimal input string to a character output string. The hexadecimal string will be padded with a leading 0 if necessary to make an even number of digits.

Format

EXEC @UTX2C *parm1* [*parm2*]

where

parm1 is the input hexadecimal string

parm2 is an optional variable stub (a variable name without a leading *&*) identifying a variable in which the output character string will be placed. If omitted, temporary variable *&¢chrstr* is used.

Examples

```
EXEC @UTX2C 'C1C2C3F1' opstr
```

In this case *&opstr* will be set to 'ABC1'

```
EXEC @UTX2C 'C1C2C3F1'
```

In this case the temporary variable *&¢chrstr* will be set to 'ABC1'.

CPUTEST

A distributed non-compiled EXEC routine which can be used to determine the efficiency of coding NCI/XF language statements in various ways. It requires that variable *&num* be set to a number indicating how many times the particular instruction(s) should be executed. This number is effectively the sample size and should be set high enough to give a reliable average. This exec may be useful in fine-tuning important elements of NCI/XF panel logic, in situations where CPU usage is critical.

Format

EXEC CPUTEST *language-statement* .....

where

language-statement ..... is a series of executed NCI/XF language statements (maximum 9), separated by spaces.

Exit variable

&cpu is set to the average amount of CPU time, in seconds, used to execute the series of instructions.

Examples

Set a variable to a null value using three different techniques:

```
set &num = 5000
EXEC CPUTEST 'set &sysamp.a = &sysamp.sysnull'
```

&cpu was set to 0.0001576.

```
set &num = 5000
EXEC CPUTEST 'set &sysamp.a = ""'
```

&cpu was set to 0.000089.

```
set &num = 5000
EXEC CPUTEST 'reset &sysamp.a'
```

&cpu was set to 0.000057.

It is clear from these three examples that RESET is by far the fastest way of clearing a variable - it is almost three times as fast as setting the variable to *&sysnull*.

Note: in the examples it is necessary to represent the variable *&a* as *&sysamp.a* to prevent *&a* being resolved before entry to CPUTEST.

NCALERT

NCALERT is an NCI/XF call routine which sends a data buffer, in a predefined format, to NetView. If this is recognized, an alert will be issued and a series of panels built from the data received will explain the nature of the alert.

Format

CALL NCALERT *sender-id data*

where

sender-id is an eight character literal appearing as the resource name on the NetView alerts dynamic screen. It can consist of characters A-Z, 0-9 and national characters ¢, %, @ and #. It is not uppercased by NCALERT.

See Netview Application Programming: Program to Program Interface (SC31- 6093-1) for information regarding this field.

data is the buffer sent to NetView which provides the information required to build the alert panels. It should be in Network management Vector Transport (NMVT) format as detailed in the SNA formats manual GA27-3136-12 and should take the form of an Alert MS Major Vector.

The buffer should be a maximum of 512 bytes in length and include the standard NMVT header. The data is not uppercased by NCALERT.

Return Codes

On completion, *&usrretc* contains one of the following codes. When *&usrretc* is 32 or 36, *&¢#nrvc* gives the return code from the NetView interface program CNMNETV.

&usrretc	*&¢#nvrc*	
0		The call routine completed successfully.
4		Invalid parameters encountered.
8		Sender-id exceeds eight characters in length.
12		Unable to load NetView's program to program interface, CNMNETV, into virtual storage. This may be due to CNMNETV not being in the STEPLIB concatenation.
32		Bad return code from CNMNETV request type 1. Return code from CNMNETV is held in NCI variable &¢#NVRC.
32	24	The NetView subsystem is inactive.
32	28	The NetView release does not support user requests.
36		Bad return code from CNMNETV request type 12. The NetView alert receiver task is inactive. The NetView subsystem has received a copy of the alert buffer. Return code from CNMNETV is held in NCI variable &¢#NVRC.
36	22	The program issuing the request is not executing in primary addressing mode.
36	26	The alert receiver task is not defined.
36	32	No NetView storage available.
36	33	Invalid buffer length.
36	35	The alert receiver buffer queue is full.
36	36	ESTAE recovery cannot be established as requested.
36	40	Invalid sender-id.
36	90	A processing error has occurred.

NMVT construction

Guidelines for constructing the NMVT:

The NMVT should start with the standard header:

X'410380D5002000000'

The following Alert MS subvectors must be included in the NMVT:

1. Date/time (X'01')
   - local date/time subfield
2. Product set ID (X'10')
3. Product identifier (X'11)
   (Product classification = X'C' (non-IBM software)
   - Software product common name subfield (X'06')
   - Software product program number (X'08')
4. Generic Alert Data (X'92')
5. Probable causes (X'93')
6. User causes (X'94'). Install causes (X'95) or failure causes (X'96')

Example

An example of an NMVT:

X'41038D5002000000'	NMVT header
X'0070'	Length of Alert MS major vector
X'0000'	Alert major vector key
Date/time MS common subvector:	
X'0A'	Subvector length
X'01'	Subvector key
X'08'	Local date/time subfield length
X'10'	Local date/time subfield key
X'5D0110120323'	Local date/time (YYMMDDHHMMSS)
Generic alert subvector:	
X'07'	Subvector length
X'92'	Subvector key
X'0000'	Flags
X'01'	Alert type
X'C007'	User defined code point
Product set ID	
X'17'	Subvector length
X'10'	Subvector key
X'00'	
X'14'	Product identifier subvector length
X'11'	Product identifier subvector key
X'0C'	Product classification
X'08'	Software product common name length
X'06'	Software product common name key
X'D5C3D7C1E2E2'	Software product common name
X'09'	Product ID subfield length
X'08'	Product ID subfield key
X'E4E2C5D9F14040'	Product ID
Probable cause subvector	
X'04'	Probable cause subvector length
X'93'	Probable cause subvector key
X'7004'	User defined code point

User causes

X'19'	User cause subvector length
X'94'	User cause subvector key
X'04'	User cause subfield length
X'01'	User cause subfield key
X'E001'	User defined code point
X'04'	Recommended actions subfield length
X'81'	Recommended actions subfield key
X'310C'	Recommended action code point
X'0F'	Detailed data extended subfield length
X'85'	Detailed data extended subfield key
X'00'	Product ID code
X'00'	
X'006A'	Data ID
X'11'	Data encoding
X'D3E4D5C1D4C540404'	Detailed data

Self defining text message

X'27'	Subvector length
X'31'	Subvector key
X'06'	Coded character set ID subfield length
X'02'	Coded character set ID subfield key
X'11340500'	Coded character set ID
X'05'	National language set ID subfield length
X'12'	National language set ID subfield key
X'C5D5C7'	National language set ID
X'03'	Sender ID subfield length
X'21'	Sender ID subfield key
X'12'	Sender ID code (control program)
X'17'	Text message subfield length
X'30'	Text message subfield key
X'E2C5D3C660C4C5C6C9D5C9D5C740D4C5E2E2C1C7C5'	Text message

The NetView program to program interface routine should be included in the NCI/XF steplib chain.

The NetView tasks DSICRTR and CNMCALRT should be defined and started, see NetView Application Programming: Program-to-Program Interface (SC31-6093-1) for instructions.

If NCALERT exits with a return code of 0, but no alert is generated by NetView, it is possible that the NMVT is not in the correct format. If the command BR NETLOGA is issued on the NetView command line, the log will be displayed identifying the offending vector of the erroneous NMVT.

If user-defined message code points are being sent on an NMVT, they should be defined by the NetView administrator, see NetView Customization Guide (SC31-6016-2) for instructions.

NCDIALC

NCDIALC is a subroutine that can be called from IBM C programs executed by @UTIBMC (Refer to *@UTIBMC* on page 9.10) and is used to:

- set an NCI/XF variable
- resolve an NCI/XF variable
- write a message to the NCI/XF log
- execute an NCI/XF EXEC

The syntax of the subroutine is:

rc=NCDIALC[*parm1*,*parm2*[,*parm3*[,*parm4*]]];

where

rc is an integer value which contains the return code.

parm1 is the NCI/XF function required. It must be a null terminated string containing the literal in EBCDIC. This can be LOG, SET, EXEC or RESOLVE.

parm2 is a null terminated string containing a variable name for SET and RESOLVE, an EXEC name for EXEC, and the message for LOG.

parm3 is a null terminated string containing the variable value to be SET, or the reply area for RESOLVE. This parameter has no meaning for LOG or EXEC.

parm4 is an integer value containing the length of *parm3* for RESOLVE. This parameter has no meaning for SET, LOG or EXEC.

Return Codes

The following return codes are set by this routine:

0 operation completed successfully.

4 NCDIALOG was not invoked from a program executed by @UTIBMC.

8 insufficient parameters supplied.

12 unrecognized first operand.

16 variable to be resolved is null.

20 variable to be resolved is larger than area supplied by caller.

Example

```
)ROUTINE
Set &inparm='ALLO'                          Set input data to C program
Exec @UTIBMC CPGM inparm                    Invoke C program
Exit


#pragma runopts(PLIST(MVS))                 Required by this interface
typedef int funcI();                        Define dynamic link to subroutine
#pragma linkage(funcI,OS);
main(argc,argv)
int argc;
char **argv;
{
int arc;
funcI*ncdial;
char parm1[8];
char parm2[10];
char parm3[10];
int parm4;
printf("Data from NCI is %s\n",argv[1]);    Echo data from calling EXEC
ncdial=(funcI *) fetch("NCDIALC");          Load subroutine
strcpy(parm1,"SET    ");                    Operation is SET
strcpy(parm2,"&VAR   ");                    Variable to SET
strcpy(parm3,"ALLOFROMC");                  Data to SET
arc=(*ncdial)(parm1,parm2,parm3);           SET the variable
printf("NCDIALC SET completed with rc %d\n",arc);
strcpy(parm1,"LOG   ");                     Operation is LOG
strcpy(parm2,"ALLOFROMC");                  Message text
arc=(*ncdial)(parm1,parm2);                 LOG the message
printf("NCDIALC LOG completed with rc %d\n",arc);
strcpy(parm1,"EXEC   ");                    Operation is EXEC
strcpy(parm2,"ANEXEC   ");                  Name of EXEC
arc=(*ncdial)(parm1,parm2);                 EXEC the EXEC
printf("NCDIALC EXEC completed with rc %d\n",arc);
strcpy(parm1,"RESOLVE");                    Operation is RESOLVE
strcpy(parm2,"&VAR   ");                    Variable to RESOLVE
memset(parm3,0,sizeof(parm3));              Clear return area for RESOLVE
parm4=sizeof(parm3)-1;                      Size of return area
arc=(*ncdial)(parm1,parm2,parm3,parm4);     RESOLVE the variable
printf("NCDIALC RESOLVE completed with rc %d\n",arc);
printf("Resolved data is %s\n",parm3);
release(ncdial);                            Delete subroutine
}
```

NCENCDEC

This module is distributed in load format only.

It uses IBM's data encryption standard (DES) to encrypt or decrypt a given block of text with a specific 8-byte key.

If an 8-byte cipher key is given as a parameter, or the block of text given is not eight bytes in length, then the data is encrypted or decrypted using a cipher-block chaining technique to guarantee the random appearance of encrypted text.

Format

CALL NCENCDEC *function key text icv cs*

where

function is one of the following three values:

- ENCRYPT encrypt clear text. Resultant cipher text is set in variable *&$text*.
- DECRYPT decrypt cipher text. Resultant clear text is set in variable *&$text*.
- MAC produce an eight-byte message authentication code (often used as a checksum for a block of text). The resultant message authentication code is set in *&$mac*.

key is an eight byte key which is used to encrypt or decrypt the text block. To decrypt the text, the same key should be provided as that used to encrypt the text.

text is the block of text on which to operate. It can be from 1 to 32767 bytes in length.

icv the initial chain value is optional if the text is eight bytes long and mandatory if the text is any other length. If specified, it must be eight bytes in length. The same chain value used to encrypt the data should be used to decrypt the data.

cs is the character set into which the incoming text is to be converted for encrypt functions or the outgoing text for decrypt functions.

Valid values are A (ASCII) or E (EBCDIC)

The text passed to the routine is assumed to be in EBCDIC format.

Return Codes

On completion, *&usrretc* contains one of the following codes.

4	warning, semiweak key used
8	warning, weak key used
12	insufficient parameters provided
16	des key zero length
20	des key not eight bytes in length
24	text block zero length
28	function zero length (ENCRYPT/DECRYPT/MAC)
32	given function is invalid (ENCRYPT/DECRYPT/MAC)
36	initial chain value zero length
40	initial chain value not eight bytes in length
44	character set zero length (ASCII/EBCDIC)
48	character set invalid
64	catastrophic error, GETMAIN failure
68	catastrophic error, FREEMAIN failure

NCPEEK

This module is distributed in load format only.

The NCPEEK routine prints selected areas of storage on the NCI/XF log.

Format

FCALL NCPEEK *title address length* [*offset*] [*limit*]

where

title is the title line written to the NCI/XF log, a maximum of 50 characters.

address is the address in hex of the area to be displayed.

length is the length in hex of the area to be displayed.

offset the offset in hex, within an area of the address, of the next area to display (for displaying a series of areas chained together)

limit the maximum number, in decimal, of chained areas to display. The default is 10. Chain printing will halt when either the limit is reached or a chain address of zero is encountered.

NCQLOG

This module is distributed in load format only.

The NCQLOG routine prints NCI/XF QUEUE information on the NCI/XF log. Information is printed in the following format (*number* is the number of elements on the queue).

** NEW CHAIN **
queue name = *number*
queue name = *number*
** NEW CHAIN **
queue name = *number*

The queues are listed together on the hash chain in which they reside, the number of such chains being defined by the UQUE and SQUE prestartup options (see the NCI/XF Installation Guide, page 3.5 for further details of these prestartup options).

Format

FCALL NCQLOG *termmask* [*]

where

termmask is a compulsory mask (or variable containing a mask) to indicate the terminal(s) whose user queues are to be displayed. Two special mask characters can be used:

- * a generic group of terminals based on the characters before the asterisk (*)
- + any character may be present at this position in the terminal name

* is an optional character to indicate that system queues are to be displayed.

Examples

FCALL NCQLOG *&usrterm*

FCALL NCQLOG *termmask* *

Note: To display system queues only, ensure the termmask operand does not match any terminal.

NCRACFX

This module is distributed in load format only.

The NCRACFX routine extracts information from a RACF database, via the RACROUTE REQUEST = EXTRACT macro on RACF release 1.8 onwards. NCI/XF must be running APF authorized.

Format

CALL NCRACFX *P1 P2 P3 P4.....Pn*

where

P1 is the RACF class name. The RACF class name can be USER, GROUP, CONNECT, DATASET or any general resource class defined in the class descriptor table.

P2 is the field name in the RACF profile of the data that is to be extracted. The specified field name is assumed to be a repeat group. If it is not, the field name must be prefixed with a plus sign (+). For a list of valid field names and whether they are repeat groups, see the appropriate RACF manual for the templates that define each field in the profiles. If bit 0 of flag 1 is on, this is a repeat group.

P3 is the RACF resource name. It identifies the specific profile from which the data will be extracted.

For a class of.....	This is the....	Maximum size is....
USER	userid	8 bytes
GROUP	group name	8 bytes
CONNECT	connect name	17 bytes
DATASET	dataset name	44 bytes

P4 is the stub of an NCI/XF variable name. A number is appended as a suffix to the variable name stub. For example, if the value is X, *&X0* will be set (and possibly *&X1*, *&X2* and so on, depending on the amount of data extracted). The ampersand (&) should not be supplied. The variable with the suffix 0 contains the count of the number of data items returned in the variables with the suffixes from 1 to the count number. A zero count can be returned.

Fields *P3* and *P4* can be repeated as pairs up to six times in total. (*Pn* has a maximum value of *P14*)

Return codes

0	normal completion
2000	parameter list error
2001	not running APF authorized
2002	abend occurred, ESTAE routine driven
other	RACROUTE REQUEST=EXTRACT SAF return code. When the SAF return code is in *&usrretc* the RACF return code is set in *&¢rretc* and the RACF reason code in *&¢rreac*. See the RACF macro reference for an explanation of the codes. The *&usrretc*, *&¢rretc* and *&¢rreac* values are decimal.

Examples:

```
CALL NCRACFX USER +PGMRNAME CKDJS1 X
```

two variables will be set:

&x0	contains 1
&x1	contains J SMITH

```
CALL NCRACFX GROUP USERID CANDKD X
```

several variables will be set:

&x0	contains the number of userids extracted
&x1 to *&xnn*	contains the userids in group CANDKD. (*nn* is the number contained in *&x0*)

NCRAND

This module is distributed in load format only.

The NCRAND routine produces a random number in the range 1 to *upper-limit*. If the *upper-limit* parameter is not specified then it defaults to 6.

Format

FCALL NCRAND [*upper-limit*]

where

upper-limit must be 1000 or less. If it is greater than 1000 then the system defaults to 6.

Note: The random number is returned in *&usrretc*.

NCSLOG

This module is distributed in load format only.

The NCSLOG routine prints on the NCI/XF log the number of times that an individual statement within a panel/routine has been executed. Information printed is in the following format:

** PANEL *pan-name-section* **

execution-count	*statement*
execution-count	*statement*

Format

FCALL NCSLOG *panmask*

where

panmask is a compulsory mask to indicate the panel(s) whose statement execution counts are to be displayed. Two special mask characters can be used:

* generic group of panels based on the characters before the asterisk (*)

+ any character may be present at this position in the panel name

Note

This module depends on the NCI/XF statement execution monitor being enabled - see the SXM= startup parameter; otherwise no statistics will be available.

Only statements from panels or routines in the PANEL library and with a non-zero execution count are displayed.

This module can also be executed through use of the @UTSLOG routine. See *@UTSLOG* on page 9.13.

NCVLOG

This module is distributed in load format only.

The NCVLOG routine prints NCI/XF variables information to the NCI/XF log. Information is printed in the following format

** NEW CHAIN **
variable name = *variable contents*
variable name = *variable contents*
** NEW CHAIN **
variable name = *variable contents*

The variables are listed together on the hash chain in which they reside, the number of such chains being defined by the UVAR and SVAR prestartup options.

Format

FCALL NCVLOG *termmask* [*]

where

termmask is a compulsory mask to indicate the terminal whose user variables are to be displayed. Two special mask characters can be used

* generic group of terminals based on the characters before the asterisk (*)

+ any character may be present at this position in the terminal name

* is an optional character to indicate that system variables are to be displayed.

Examples

FCALL NCVLOG *&usrterm*

FCALL NCVLOG *termmask* *

Note: To display the system variables only, ensure the *termmask* operand does not match any terminal.

NCWAIT

This module is distributed in load format only.

The NCWAIT routine waits a specified period of time.

Format

FCALL NCWAIT *nnnu*

where

nnn is the number of units to wait

u is the unit of time as follows:

- X 100ths of a second
- S seconds (this is the default)
- M minutes
- H hours

Return codes

0 wait completed satisfactorily.

4 invalid parameter.

NCWTOR

This module is distributed in load format only.

The NCWTOR routine allows the NCI/XF user to issue WTORs from NCI/XF panels. (Write-To-Operator-with-Reply from NCI/XF)

Format

FCALL NCWTOR *P1*..... *Pn* NOREPLY

where

P1 to *Pn* are the message lines to be written to the operator console. A maximum of 11 lines can be displayed and each line has a maximum of 70 characters.

Call processing is suspended until an operator reply is received.

The operator response is returned in the variable *&usrwtor* with a maximum length of 115 characters set to uppercase.

The length of the response is returned in *&usrretc* with a return code of zero indicating a processing error.

NOREPLY can be specified as the last parameter (uppercase or lowercase); if specified, the call will generate a non roll-deletable multi-line WTO, that is, no reply expected, no suspension of processing, no modification of *&usrwtor* and *&usrretc* set to zero.

Chapter 10 - Macros in (F)Call routines

Common NCDIALOG parameters 10.2
ATTR 10.3
CURSOR 10.6
DECRYPT 10.8
ENCRYPT 10.9
ESTAE 10.10
ESTAER 10.12
EXEC 10.13
LOG 10.14
RESOLVE 10.15
SET 10.21
XRESOLVE and XSET 10.26

This chapter should be consulted only after you are familiar with the NCI/XF User Guide, *Chapter 12 - A guide to NCI/XF (F)CALL routines*.

Common NCDIALOG parameters

When the parameters on an NCDIALOG macro instruction in register notation are being specified, the following registers are supported for each parameter:

PARM	-	registers 0 - 15
VAR	-	registers 2 - 14
AREA	-	registers 2 - 13
LEN	-	registers 2 - 13
COUNT	-	registers 2 - 13
RETRY	-	registers 2 - 13
BASE	-	registers 2 - 13
SAVE	-	registers 2 - 13
NAME	-	registers 2 - 13

When register addressing on an NCDIALOG macro is being used, the value specified must be a numeric value enclosed in brackets. Register equates are not supported. Note especially the following common parameter:

ERROR=*error* where *error* is the label to branch to if there is a return code in register 15

ATTR

Description

ATTR allows field attributes in NCI/XF to be controlled by allocating special characters to represent the required combination of attributes.

Formats

Single attribute format

NCDIALOG ATTR,
PARM=*parm1*,
VAR=*char*,
AREA=*value*,
LEN=*vallen*

where

parm1 is the name of a fullword which contains the value in the first 4 bytes of the parameter list passed to this exit in register 1, or a register containing this value.

char is the address of a single character in storage, or a register containing this address. The character must be a special character which is in use as an attribute character in the current panel.

value is the address in storage, or a register containing this address, of an area which contains the list of attributes to be assigned to the character indicated by *char*. The keywords recognized by NCDIALOG are the same as those used on an ATTR language statement (for instance high protected numeric blue).

vallen is the length of the data pointed to by *value*, or a register containing this length.

```
NCDIALOG ATTR,
               PARM=(2),
               VAR=EXCLAIM,
               AREA=VALUES,
               LEN=L'VALUES
```

The single exclamation mark (!) is redefined to mean high-brightness, protected and numeric (autoskip), red, blinking, where the program work areas are defined as:

```
EXCLAIM DC   C'!'
VALUES  DC   C'HIGH PRO NUM RED BLI'
```

Multiple attribute format

NCDIALOG ATTR,
PARM=*parm1*,
COUNT=*num*,
VAR=*chartab*,
AREA=*hextab*

where

num is the number of attribute characters to be redefined on this call to NCDIALOG, or a register containing this number.

chartab is the address in storage, or a register containing this address, of num bytes which contain the special characters to be redefined.

hextab is the address in storage, or a register containing this address, of num fullwords which contain the 3270 datastream hex values of the attributes to be assigned to the characters pointed to by *chartab*. Each fullword contains:

byte 0	-	X'00' = no cursor
	-	X'80' = cursor
byte 1	-	normal attribute values
byte 2	-	extended attribute values
byte 3	-	color attribute values.

```
NCDIALOG ATTR,
               PARM=R1SAVE,
               COUNT=3,
               VAR=VARTAB,
               AREA=HEXTAB
```

The attribute values assigned to the following characters are redefined as:

! - high-brightness, unprotected, cursor, blue
* - normal-brightness, protected, red, blinking
+ - non-display, protected, numeric (autoskip)

where the program work areas are defined as:

```
R1SAVE DS F
HEXTAB DC X'80C800F1' 3270 attr values for !
       DC X'0060F1F2' 3270 attr values for *
       DC X'007C0000' 3270 attr values for +
VARTAB DC C'!*+'      NCI panel attr chars.
```

Notes

Default attributes for a panel are assigned with ATTRIBUTE statements in the)PANEL section; these include the special characters assigned at the relevant places in the panel)BODY definition. For a full description of this process, see *ATTRibute* on page 1.14 of the NCI/XF Reference Manual,. The default attribute values assigned may be varied for a particular panel with the ATTR function of NCDIALOG. New special characters cannot be assigned: this can only be done by using an ATTRIBUTE statement in the)PANEL section. NCDIALOG can vary the characteristics assigned to a particular special character.

A single attribute character may be redefined with an explicit definition of the value required (for example, high, protected, numeric, turquoise). The keywords recognized by NCDIALOG in this mode are the same as those which can be specified in the ATTRIBUTE language statement.

Multiple attribute values may be redefined in a single invocation of NCDIALOG. In this case the attribute values to be assigned to the special characters specified must be given as the hex values defined by the 3270 datastream protocol.

ATTR help panel

To assist users, a panel is supplied with NCI/XF that will accept input in the same form as the ATTRIBUTE language statement, (for instance high unprotected blue) and convert the parameters specified to the required 3270 hex values. This panel can be accessed by selecting the appropriate option from the SPECIFIC FUNCTIONS menu in DEMO mode. DEMO mode is obtained by typing a userid of DEMO, with no password, into the NCI/XF DEMO SYSTEM LOGO panel.

CURSOR

Description

The CURSOR function of NCDIALOG sets the cursor position for the next refresh of the current panel. The cursor can be set to a specific row and column location on the screen, or over the first character of a variable which appears on the panel to be displayed.

Format

NCDIALOG CURSOR,
PARM=*parm1*,
AREA=*curloc*

NCDIALOG CURSOR,
PARM=*parm1*,
VAR=*varname*

NCDIALOG CURSOR,
PARM=*parm1*

where

parm1 is the name of a fullword which contains the value in the first four bytes of the parameter list passed to this exit in register 1, or a register containing this value.

varname is the name of an four byte area in storage, or a register pointing to this area, which contains the name of the variable. The cursor will be set over the first position of the variable.

curloc is the name of an eight byte area in storage, or a register pointing to this area, which contains the row and column numbers where the cursor is to be placed. The first four bytes (4 EBCDIC digits) give the row number and the next four bytes (4 EBCDIC digits) give the column number. The values should be specified right-justified and zero padded.

Examples

```
NCDIALOG CURSOR,
                PARM=R1SAVE,
                VAR=NCIVAR
```

On the next refresh of the current panel, the cursor will be placed over the first character of variable *&userid*, where the program work areas are defined as:

```
R1SAVE DS F
NCIVAR DC CL8'&&USERID'
```

```
NCDIALOG CURSOR,
                PARM=(3)
                AREA=CURLOC
```

On the next refresh of the current panel, the cursor will be placed in line 17, column 23, where the program work areas are defined as:

```
CURLOC DS 0D
CURROW DC C'0017'
CURCOL DC C'0023'
```

Note: If the CURSOR function is invoked with neither the *area* nor *var* operand specified, the cursor will be positioned according to the current attribute characters set in the panel.

DECRYPT

Description

You can use the DECRYPT function of NCDIALOG to RESOLVE and DECRYPT the contents of a variable which has been previously encrypted. One or more variables may be processed in a single call to NCDIALOG.

Formats

Single list format

NCDIALOG DECRYPT,
PARM=*parm1*,
VAR=*vname*,
AREA=*varea*,
LEN=*vlen*

Variable list format

NCDIALOG DECRYPT,
PARM=*parm1*,
COUNT=*numb*,
VAR=*vtab*,
AREA=*dtab*

Example

```
NCDIALOG DECRYPT,
                PARM=(1),
                VAR=VARNAME,
                AREA=DATAREA,
                LEN=6
```

The first six bytes of variable *&passwd* will be decrypted and placed in the work area called DATAREA. The program work areas are defined as:

```
VARNAME     DC CL8'&&PASSWD'
DATAREA     DS CL255
```

Note: All parameters and return codes are as described for *RESOLVE* on page 10.15.

ENCRYPT

Description

You may use the ENCRYPT function of NCDIALOG to SET and ENCRYPT the contents of a variable. One or more variables may be processed in a single call to NCDIALOG.

Format

Single variable format

NCDIALOG ENCRYPT,
PARM=*parm1*,
VAR=*vname*,
AREA=*varea*,
LEN=*vlen*

Variable list format

NCDIALOG ENCRYPT,
PARM=*parm1*,
COUNT=*numb*,
VAR=*vtab*,
AREA=*dtab*

Example

```
NCDIALOG ENCRYPT,
                PARM=(1),
                VAR=VARNAME,
                AREA=DATAREA,
                LEN=5
```

The first five bytes of data stored in DATAREA will be encrypted and placed in variable *&passwd*. The program work areas are defined as

```
VARNAME     DC CL8'&&PASSWD'
DATAREA     DC C'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

After execution, *&passwd* will be set as if the language statement

```
ENCRYPT &PASSWD = 'ABCDE'
```

had been processed.

Note: All parameters and return codes are as described for *SET* on page 10.21.

ESTAE

Description

This function extends the ESTAE environment around a call routine to provide a retry address facility. The retry address must be within your exit routine.

You may want to perform certain cleanup functions if your exit abends, for example close and deallocate files or release enqueues. You may also control NCI/XF ESTAE dump processing, either separately or in conjunction with a retry routine.

Formats

Controlling dump processing

```
NCDIALOG ESTAE,
        PARM=parm1,
        DUMP=YES|NO|0CX|(x)
```

Invoking a retry routine

```
NCDIALOG ESTAE,
        PARM=parm1,
        RETRY=retry-addr,
        BASE=base-addr,
        SAVE=save-addr,
        DUMP=YES|NO|0CX|(x)
```

where

parm1	is the name of a fullword which contains the value in the first 4 bytes of the parameter list passed to this exit in register 1, or a register containing this value.
retry-addr	is the address within your exit routine to which ESTAE will branch after an abend (the address of your retry routine).
base-addr	is the base address used by your retry routine. The base address is usually in register 12.
save-addr	the address of the save area used by your retry routine. The base address is usually in register 13.
YES|NO|0CX|(*x*)	controls the circumstances in which the NCI/XF ESTAE processor will take a dump. The meaning of these options is defined on the next page.

DUMP operand

The options which may be specified for the DUMP operand are

YES requests the NCI/XF ESTAE processor to take a dump if your exit abends for any reason. This is the default if the DUMP operand is not specified. If retry-addr is also specified, the dump is taken before branching to the specified address.

NO requests the NCI/XF ESTAE processor to always bypass dump processing.

0CX requests the NCI/XF ESTAE processor to only take a dump if your exit has abended with an 0Cx abend code. If *retry-addr* is also specified, the dump is taken before branching to the specified address.

x this option allows the dump parameter to be selected by the least significant byte of the register (x) specified. The binary values which are applicable are

0 - DUMP=YES

1 - DUMP=NO

2 - DUMP=0CX

If a value greater than 2 is specified by the register, NCDIALOG returns with RC equal to 8.

Examples

```
NCDIALOG ESTAE,
                PARM=R1SAVE,
                RETRY=RETRYLAB,
                BASE=(10),
                SAVE=(13),
                DUMP=0CX
```

Dumps are suppressed for abends other than 0Cx abends. If the CALL routine abends for any reason, processing branches to label RETRYLAB within the CALL program with the current content of register 10 as the base address. Register 13 contains the save area address.

```
NCDIALOG ESTAE,
                PARM=R1SAVE,
                DUMP=NO
```

All dumps are suppressed if the CALL routine terminates abnormally.

Note: To prevent recursive abends, an NCDIALOG ESTAE environment is cancelled at the first branch to the retry routine. If your program finds that the error is not fatal and processing is to continue, you may issue NCDIALOG ESTAE again to re-establish your retry environment.

ESTAER

Description

This function allows a user to determine the value currently assigned to the NCDIALOG ESTAE function.

Format

NCDIALOG ESTAER,

PARM=*parm1*,
AREA=*value*

where

parm1 is the name of a fullword which contains the value in the first four bytes of the parameter list passed to this exit in register 1, or a register containing this value.

value is the address in storage, or a register containing this address, of a 13 byte work area.

Return codes

On return from the function, register R15 contains one of the following codes:

0	function performed
4	error returned by function routine
8	invalid parms on entry
12	internal error
16	internal error
20	dialog not allowed by issuing exit types
32	invalid PARM parameter

If the code returned is 0, the area referred to by the area parameter will contain:

bytes 0-3	user's retry address
bytes 4-7	user's base address (R12)
bytes 8-11	user's save area address (R13)
byte 13	user's dump flag

00	YES
01	NO
02	0CX

EXEC

Description

This function causes an NCI/XF EXEC routine to be run as selected by the NAME=*addr* parameter.

Format

NCDIALOG EXEC,
PARM=*parm1*,
NAME=*addr*

where

parm1 is the name of a fullword which contains the value in the first four bytes of the parameter list passed to this exit in register 1, or a register containing this value.

addr is the address of an 8 byte field which contains the name of the EXEC routine.

Return codes

After execution of an NCDIALOG EXEC request, register 15 will contain one of the following codes:

0 EXEC routine ran successfully

4 error running EXEC routine (build error or run time error)

LOG

Description

This function allows the user to write a line to the current NCI/XF log.

Format

NCDIALOG LOG,
PARM=*parm1*,
AREA =*area addr*,
LEN =*len*

where

parm1	is the name of a fullword which contains the value in the first 4 bytes of the parameter list passed to this exit in register 1, or a register containing this value.
area addr	is the address of the message to write to the NCI/XF log, or a register containing the address of the message.
len	is the length of the message to write to the NCI/XF log, or a register containing the length.

Example

```
NCDIALOG LOG,
               PARM=R1SAVE,
               AREA=EGMSG,
               LEN=L'EGMSG
```

where

```
EGMSG DC C'FILE HAS BEEN OPENED SUCCESSFULLY'
```

is an example of a LOG message.

RESOLVE

Description

The RESOLVE function of NCDIALOG can be used to obtain the current setting of a single variable or a list of variables.

Format

Resolving a single variable

```
NCDIALOG RESOLVE,
          PARM=parm1,
          VAR=vname,
          AREA=varea,
          LEN=vlen
```

where

parm1 is the name of a fullword, which contains the value in the first four bytes of the parameter list pointed to by register 1 on entry to the exit, or a register containing this value.

vname is the name of an eight byte area, left justified and padded with blanks, that contains the name of the variable to be resolved; or if the variable name is greater than eight bytes, the area contains two fullwords containing the length and address of the variable name respectively,

varea is the name of the area where the contents of the variable will be stored, or a register pointing to such an area.

vlen is the length of the area pointed to by *varea*, or a register containing the length.

Please see the next page for the format for resolving a list of variables.

Resolving a list of variables

NCDIALOG RESOLVE,
PARM=*parm1*,
COUNT=*numb*,
VAR=*vtab*,
AREA=*dtab*

where

parm1 is the name of a fullword which contains the value in the first four bytes of the parameter list passed to this exit in register 1, or a register containing this value.

numb is the number of variables to be resolved, or a register containing this number.

vtab is the address of a list containing *numb* eight byte variable names to be resolved, or, if variable names are longer than eight bytes, the address of a list of *numb* eight byte entries. The two fullwords contain the length and address of the variable name respectively.

dtab is the address of a table containing *numb* entries, where the resolved values will be stored. Each table entry must consist of a byte giving the length of this table entry, a byte giving the length to resolve and an area into which the resolved value of the variable will be written.

Return codes

After execution of an NCDIALOG RESOLVE request, register 15 will contain one of the following codes:

0	last variable resolved successfully
4	last variable not found (may have been RESET)
8	invalid parameters specified
32	error invoking NCDIALOG routine

Examples of resolving a single variable

```
NCDIALOG RESOLVE,
                  PARM=R1SAVE,
                  VAR=VARNAME
                  AREA=DATAREA,
                  LEN=20
```

The first 20 bytes of variable *&userid* will be put in the work area called DATAREA, where the program work areas are defined as:

```
R1SAVE    DS  F
VARNAME   DC  CL8'&&USERID'
DATAREA   DS  CL20
```

```
NCDIALOG RESOLVE,
                  PARM=R1SAVE,
                  VAR=(5)
                  AREA=(6),
                  LEN=20
```

The value of the variable whose name is pointed to by register 5 will be put in the storage area indicated by register 6.

```
NCDIALOG RESOLVE,
                  PARM=R1SAVE,
                  VAR=VN1
                  AREA=DATAREA,
                  LEN=20
```

If the variable name is longer than eight characters, the format of the VAR parameter field is as shown below:

```
VN1       DS OF
          DC F'10'                      Length of var name
          DC A(VN11)                    Address of var name
VN11      DC C'&&LONGVARNM'             Var name
DATAREA   DC CL20'20 bytes of data....'
```

Examples of resolving a list of variables

```
NCDIALOG RESOLVE,
                  PARM=R1SAVE,
                  COUNT=3,
                  VAR=VARTAB,
                  AREA=DATAREA,
```

where the program work areas are defined in the following table; three variables *&name*, *&address* and *&phone* will be resolved in a single call to NC-DIALOG; the ∇ symbol represents a space:

```
R1SAVE   DS F
VARTAB   DC CL8'&&NAME'
         DC CL8'&&ADDRESS'
         DC CL8'&&PHONE'
DATATAB  DC XL1'FF'
         DC XL1'FD'
         DC CL253 '∇'          &NAME stored here
         DC XL1'FF'
         DC XL1'FD'
         DC CL253 '∇'          &ADDRESS stored here
         DC XL1'FF'
         DC XL1'FD'
         DC CL253'∇'           &PHONE stored here
```

```
NCDIALOG RESOLVE,
                  PARM=NCIDAD,
                  VAR=NAMES,
                  AREA=AREAS,
                  COUNT=NUMBER
```

A list of variable names more than eight characters long requires the formatting of the VAR parameter field as shown above. A mixture of eight character names and longer names is allowed.

Examples of fields used to set a list of variables will be found on the next page.

Examples of fields used to set a list of variables

```
NAMES  DS  0F
       DC  F'15'                  Length of first variable name.
       DC  A(VN2)                 Address of first variable name.
       DC  A(LVN3)                Length of second variable name.
       DC  A(VN3)                 Address of second variable name.
       DC  CL8'&&VARNO4'          Third variable name.
NUMBER EQU((*.-NAMES)/8)          Number of variables to set.

VN2    DC  C'&&ZXCVBNMZXCVBNM'    First var name.
VN3    DC  C'&&QWERTYUIOP'        Second variable name.
LVN3   EQU*-VN3                   Length of second variable name.

AREAS  DS  0C
VA2    DC  AL1(LVA2)              1 Byte of first table entry.
       dc  al1(lva2-2)            1-Byte of first variable data.
       DC  CL20'∇'                First variable data.
LVA2   EQU*-VA2                   Length of first table entry.
VA3    DC  AL1(LVA3)              Length of second table entry.
       DC  AL1(LVA3-2)            Length of second variable data.
       DC  CL180'∇'               Second variable data.
LVA3   EQU*-VA3                   Length of second table entry.
VA4    DC  AL1(LVA4)              Length of third table entry.
       DC  AL1(LVA4-2)
       DC  CL5'∇'
LVA4   EQU*-VA4
```

Notes

Alternative to RESOLVE

Variables can be passed to an exit routine as parameters in the (F)CALL statement which invokes an exit. This is the simplest method of passing a limited number of user or local variables to an exit routine. The RESOLVE function of NCDIALOG should be used in the following circumstances where the passing of variables at exit invocation may not be suitable:

- where access is required to more than 11 variables. (A maximum of 11 parameters can be passed in a (F)CALL statement.)
- in general purpose exit routines, where the names of the variables to be resolved are not known when the module is called.
- where system variables are to be accessed. (Since exit routines execute asynchronously from NCI/XF mainline code, the value of a system variable may change after the exit is invoked.)

Zero length

A zero length in the second byte of an entry in the table indicated by the area operand (the data table) means that the variable to which that entry relates was not found because the variable was reset or wrongly specified.

Resolving a single variable

The *vlen* parameter specifies the maximum amount of data which will be moved to the caller's work area during a RESOLVE operation. If the variable data is longer than *vlen*, the data will be truncated to the length specified.

When a variable is being resolved, the true length of the data is returned in register zero. To avoid truncation, first issue the RESOLVE with *vlen* set to zero, and then reissue the RESOLVE with *vlen* set to the value returned in register zero.

Resolving a list of variables

NCDIALOG processing will be more efficient if the list of variable names in *vtab* is arranged in ascending EBCDIC sequence, although this is not mandatory.

The second byte of each table entry pointed to by the *dtab* parameter specifies the maximum length of the data to be resolved. If the data contained in the variable is longer than the length specified in the *dtab* entry, the data will be truncated to the length specified.

Variable length

All variables are variable length, depending on the data assigned to them, up to a maximum length of 32767.

SET

Description

The SET function of NCDIALOG allows the contents of variables to be updated from within an exit routine. A single variable or a list of variables may be SET in a single operation.

Formats

Setting a single variable

NCDIALOG SET,
PARM=*parm1*,
VAR=*vname*
AREA=*varea*,
LEN=*vlen*

where

parm1 is the name of a fullword which contains the value of the first four bytes of the parameter list indicated by register 1 on entry to the exit, or a register containing this value.

vname is the name of an eight byte area which contains the name of the variable to be set, or if the variable name is greater than eight bytes, the area contains two fullwords containing the length and address of the variable name respectively.

varea is the name of the area which contains the data to be stored in *vname*, or a register pointing to this area.

vlen is the length of the data to be copied from *varea* into the variable, or a register containing the length of the data. The maximum value allowed is 32767, which is the maximum length for any variable.

Please see the next page for the format for setting a list of variables.

Setting a list of variables

NCDIALOG SET,
PARM=*parm1*,
COUNT=*numb*,
VAR=*vtab*,
AREA=*dtab*

where

parm1 is the name of a fullword containing the value of the first four bytes of the parameter list passed to this exit in register 1, or a register containing this value.

numb is the number of variables to be set, or a register containing the number.

vtab is the address of a list containing the number of eight byte entries defined by *numb*. These entries contain either the variable names to be set, or, if variable names are longer than eight bytes, two fullwords containing the length and address of the variable name respectively.

dtab is the address of a table containing the number of entries defined by *numb*. These entries contain the data to be set into the variables named in *vtab*. Each table entry must consist of a byte giving the length of this table entry, a byte giving the length to set and an area containing the data to be set.

Return codes

After execution of an NCDIALOG SET request, register 15 will contain one of the following codes:

0	variable SET successfully
8	invalid parameters specified
32	error invoking NCDIALOG routine

Examples of setting a single variable

```
NCDIALOG SET,
                PARM=R1SAVE,
                VAR=VARNAME,
                AREA=DATAREA,
                LEN=5
```

The first five bytes of data stored in DATAREA will be put in variable *&cinit*, where the program work areas are defined as follows:

```
R1SAVE    DS F
VARNAME   DC CL8'&&CINIT'
DATAREA   DC C'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

After execution of the NCDIALOG macro, *&cinit* will be set as if the following language statement had been executed:

```
SET &CINIT = 'ABCDE'
```

```
NCDIALOG SET,
                PARM=(4),
                VAR=(5),
                AREA=(6),
                LEN=255
```

The 255 bytes of data which start at the location indicated by register 6 will be stored in the variable whose name is pointed to by register 5.

```
NCDIALOG SET,
                PARM=NCIDAD
                VAR=VN1,
                AREA=VA1,
                LEN=LVA1
```

If a variable name is longer than eight characters, format the VAR parameter field as shown above.

Examples of fields used to set a single variable

```
VNI     DS    0F
        DC    F'10'                  Length of var name.
        DC    A(VN11)                Address of var name.
VN11    DC    C'&&ASDFGHJKL'         VAR name.
VA1     DC    C'VALUE OF ASDFGHJKL'  VAR data.
LVA1    EQU   *-VA1                  Length of var data.
```

Example of resetting a variable

A variable will be RESET (deleted) and its storage freed by a SET function with a *vlen* of zero.

```
NCDIALOG SET,
                PARM=(4),
                VAR=(5),
                LEN=0
```

Examples of setting a list of variables

```
NCDIALOG SET,
                PARM=R1SAVE,
                COUNT=3,
                VAR=VARTAB,
                AREA=DATATAB
```

Three variables, *&name*, *&address* and *&phone*, will be set in a single call to NCDIALOG. The program work areas are defined as follows:

```
R1SAVE   DS  F
VARTAB   DC  CL8'&&NAME'
         DC  CL8'&&ADDRESS'
         DC  CL8'&&PHONE'
DATATAB  DC  XL1'0C'                Length of entry
         DC  XL1'0A'                Length of name data
         DC  C'JOHN SMITH'          Data for &NAME
         DC  XL1'0A'
         DC  XL1'08'
         DC  C'NEW YORK'            Data for &ADDRESS
         DC  XL1'0D'
         DC  XL1'0B'
         DC  C'01-234-5678'         Data for &PHONE
```

```
NCDIALOG SET,
                PARM=NCIDAD,
                VAR=NAMES,
                AREA=AREAS,
                COUNT=NUMBER
```

When setting a list of variables with names longer than eight characters, you should format the VAR parameter field as shown above. A mixture of eight character names and names longer than eight characters is allowed. Examples of fields used to set a list of variables are shown on the next page.

Examples of fields used to set a list of variables

```
NAMES  DS  0F
       DC  F'15'                                          Length of 1st var name.
       DC  A9VN2)                                         Address of 1st var name.
       DC  A(LVN3)                                        Length of 2nd var name.
       DC  A(VN3)                                         2nd var name address.
       DC  CL8'&&VARN04'                                  3rd var name.
NUMBER EQU((*-NAMES)/8)                                   Number of var to set.

VN2    DC  C'&&ZXCVBNMZXCVBNM'                            1st var name.
VN3    DC  C'&&QWERTYUIOP'                                2nd var name.
LVN3   EQU*-VN3

AREAS  DS  0C                                             Length of 2nd var name.
VA2    DC  AL1(LVA2)                                      1st table entry length.
       DC  AL1(LVA2)                                      Length of 1st var data.
       DC  'VALUE OF 1ST VARIABLE CONTAINING THIS DATA'
LVA2   EQU*-VA2                                           1st table entry length.
VA3    DC  AL1(LVA3)                                      2nd table entry length.
       DC  AL1(LVA3-2)                                    2nd var data length.
       DC  C'VALUE OF QWERTYUIOP'                         2nd var data.
LVA3   EQU*-VA3                                           2nd table entry length.
VA4    DC  AL1(LVA4)                                      3rd table entry length.
       DC  AL1(LVA4-2)
       DC  CLS'ABCDE'
LVA4   EQU*-VA4
```

Notes on setting a list of variables

NCDIALOG processing will be more efficient if the variable names in *vtab* are arranged in ascending EBCDIC sequence, although this is not mandatory.

XRESOLVE and XSET

Description

The XRESOLVE and XSET functions can be used to RESOLVE or SET a list of NCI/XF variables where the variable data is over 255 bytes long.

The standard RESOLVE or SET options can be used for processing a single variable containing over 255 bytes long, but the extended RESOLVE and SET (XRESOLVE and XSET) functions are required when more than one variable containing data over 255 bytes is to be processed.

Data held in variables can have a maximum length of 32767 characters.

Setting and using these variables in the NCI/XF language is transparent.

The NCDIALOG functions XSET and XRESOLVE cater for the extended data in (F)CALLed routines.

These two functions will set and resolve respectively variables with names up to 255 characters.

Both XRESOLVE and XSET can be used to process a single variable, in which case the NCDIALOG parameters are the same as described under RESOLVE and SET.

Format

NCDIALOG XSET,
PARM=*parm1*,
VAR=*vtab*,
AREA=*dtab*,
COUNT=*numb*

where

parm1 is the name of a fullword containing the value in the first four bytes of the parameter list passed to this exit in register 1, or a register containing this value.

numb is the number of variables to be set, or a register containing the number.

vtab is the address of a list containing the number of eight byte entries defined by *numb*. These entries contain either the variable names to be set, or, if variable names are longer than eight bytes, the two fullwords containing the length and address of the variable name respectively.

dtab is the address of a table containing the number of entries defined by *numb*. These entries contain the data to be set into the variables named in *vtab*. Each table entry must consist of two fullwords containing the length and address of the data to be set or resolved.

Example

```
NCDIALOG XSET,
                PARM=NCIDAD,
                VAR=NAMES,
                AREA=AREAS,
                COUNT=NUMBER
```

Example of XSET setting a list of two variables to data containing more than 255 bytes.

```
NAMES  DS  0F
       DC  CL8'&&VAR1'
       DC  CL8'&&VAR2'
NUMBER EQU ((*-NAMES)/8)
AREAS  DS  0F
       DC  AL4(L'DATA10
       DC  AL4(DATA1)
       DC  AL4(L'DATA2)
       DC  AL4(DATA1)

DATA1  DC  CL400'THIS IS DATA AREA1'
DATA2  DC  CL300'EXAMPLE OF DATA OVER 255 BYTES'
```

This page intentionally left blank

Chapter 11 - Messages and abend codes

NCI/XF operating system messages 11.2
NCI/XF log file messages 11.10
Abend codes 11.26
Local Support Office reporting 11.26
NC-EDIT information messages 11.29
NC-MHO log messages 11.36
NC-VSAM log messages 11.63
NC-XMS log messages 11.70
Useful manuals for reference in diagnosis of problems 11.77

NCI/XF operating system messages

Certain messages may be output to inform the operator of particular events. These messages, output via SVC 35 (WTO), use route codes 2 and 11. References to the system console, for MVS systems, refer to consoles supporting route codes 2 and 11; for VM systems it refers to the console or secondary console of the VM machine where NCI/XF is running.

The table on the following pages lists the WTO messages with an explanation of the WTO and any action that should be taken if the WTO is issued. Each message is prefixed by jobname. On MVS systems, *jobname* is the jobname from the job card (batch jobs), or PROC statement (started tasks); on VM systems it is the virtual machine name where NCI/XF is running.

WTO	Description	Action
***JOBNAME* - ABEND IN PROGRESS**	In response to an Operator MODIFY command which specified the ABEND option, the process is terminating with a dump.	Restart and pass the storage dump to your support office if a problem exists.
***JOBNAME* - ABEND - PANEL STORAGE MANAGEMENT TERMINATED**	This is a non-rollable message sent to the system console when the task responsible for panel storage management abends.	Restart NCI/XF and contact your support office.
***JOBNAME* - ABEND - UPDATE PROCESSING TERMINATED**	The services subtask has encountered an error from which it could not recover. Operation continues, but all update processing will cease.	Update processing for the startup panel and all terminals will cease. Application status information may cease to be updated as well. You should stop and restart at the earliest convenient time.
***JOBNAME* - ABNORMAL TERMINATION IN PROGRESS**	NCI/XF has abended and tidying up is in progress. A dump may be initiated, in which case job termination will be delayed.	This message is for information only. No direct action is required.
***JOBNAME* - BUFF = X PARM INVALID ON NON-XA SYSTEM**	The Operating System is not specified as XA. The BUFF = X parameter is invalid on this system.	Find out whether the system should be XA. Contact your support office if necessary.
***JOBNAME* - BUFF = PARM MUST BE IN THE RANGE 100 - 32,767**	The BUFF = parameter specified in the JCL is not in the range 100 through 32767.	Respecify the parameter within the correct range.
***JOBNAME* - ERROR - ABEND OCCURRED IN PRINT SUBTASK, LOG PROCESSING TERMINATED**	This is a non-rollable message sent to the system console when the task responsible for writing the NCILOG file abends. Normal processing will continue without a log.	No audit or event logging will be done. You should stop and restart at the earliest convenient time, and contact your support office.
***JOBNAME* - ERROR- UNABLE TO LOAD MODULE PR*XXXX* - ABENDING 202**	An error occurred while loading one of the print subsystem modules from the STEPLIB (MVS) or LOADLIB (VM).	Refer to Abend Code 202 on page 11.27.
***JOBNAME* - ERROR- UNABLE TO OPEN LOG FILE, LOG PROCESSING TERMINATED**	This is a non-rollable message sent to the system console when the task responsible for writing the NCILOG file cannot be opened. Normal processing will continue without a log.	No audit or event logging will be done. You should establish why NCI/XF could not open the LOG file, then stop and restart at the earliest convenient time. Contact your support office if the problem persists.
***JOBNAME* - EXTENDED MODIFY DISABLED BY STARTUP OPTION**	The MODIFY command you have just entered is part of the extended control set and has been ignored because EXTENDED MODIFY support has been disabled by the EXTENDED-MODIFY = NO startup option.	

WTO	Description	Action
***JOBNAME* - INVALID MODIFY COMMAND**	The system operator communicated via the MODIFY command, but the command was invalid.	Correct the MODIFY command.
***JOBNAME* - INVALID OPERAND FOUND IN JCL PARM**	An operand within the JCL was not recognized.	Delete or respecify the operand.
JOBNAME* - INVALID RPL USER FIELD DETECTED BY NC*XXXXXX	The indicated module has found that the VTAM RPL USER FIELD (RPLUSFLD) has been corrupted.	NCI/XF will attempt to recover from this problem. Contact your local support office.
***JOBNAME* - MAXIMUM LOG SIZE REACHED**	The number of lines written to the current NCI/XF log has reached the maximum specified in the startup options. NCI/XF will close the current log file and open the alternative.	This message is for information only. No action is required.
***JOBNAME* - MODIFY COMMAND COMPLETED**	The statement entered from a system console as the text of a MODIFY command has been completed.	
***JOBNAME* - MODIFY COMMAND FAILED, EXEC LIMIT EXCEEDED**	An EXEC statement, entered from a system console, has failed. Further details about the error may be found on the NCILOG file.	Check that the routine is not looping and that the MAXEXEC startup option specifies a value high enough to allow the routine to run to completion.
***JOBNAME* - MODIFY COMMAND FAILED, GOTO LIMIT EXCEEDED**	A GOTO statement, entered from a system console, has failed. Further details about the error may be found on the NCILOG file.	Check that the routine is not looping and that the MAXGOTO startup option specifies a value high enough to allow the routine to run to completion.
***JOBNAME* - MODIFY COMMAND FAILED, ILLEGAL VERB FOUND**	An attempt was made to issue an EXTENDED MODIFY command using an unauthorized NCI/XF verb.	If you want this verb, contact your support centre to request a valid authorization code.
***JOBNAME* - MODIFY COMMAND FAILED, INCREMENT/DECREMENT ERROR**	An INCREMENT or DECREMENT statement, entered from a system console as the text of a MODIFY command, or within a routine EXECed from a console has failed.	Ensure that the variable being manipulated has previously been set to a numeric value.
***JOBNAME* - MODIFY COMMAND FAILED, INVALID GOTO LABEL GENERATED**	A variable GOTO statement, entered from a system console as the text of a MODIFY command, or within a routine EXECed from a console, has resolved an invalid label name.	Correct the logic of the routine before reissuing the command. If the routine is modified in the panel library, use the DELETE command to remove it from storage before re-executing it.
***JOBNAME* - MODIFY COMMAND FAILED, INVALID VARIABLE NAME GENERATED**	An invalid variable name has been resolved from a variable variable referenced within a routine invoked by an EXEC command entered from a system operator console.	Correct the logic of the routine before reissuing the command. If the routine is modified in the panel library, use the DELETE command to remove it from storage before re-executing it.

WTO	Description	Action
***JOBNAME* - MODIFY COMMAND FAILED UNABLE TO BUILD EXEC ROUTINE**	A routine invoked by an EXEC statement entered from a system console contains a syntax error. Details of the error may be found on the NCILOG file.	Correct the error and reissue the command.
***JOBNAME* - MODIFY COMMAND FAILED, USER CALL ROUTINE ABENDED**	An exit routine invoked by a CALL or EXEC command from a system operator console has abended. Depending on the setting of the NCDIALOG ESTAE DUMP option, a storage dump may have been taken.	Correct the cause of the abend before reissuing the command.
***JOBNAME* - MODIFY COMMAND INITIATION FAILED**	Owing to a VTAM error it was not possible to process an EXTENDED MODIFY command.	Contact your support centre.
JOBNAME* - MODIFY COMMAND RECEIVED, *command	A command has been accepted from the system operator console and is being processed.	This message is for information only. No action is required.
***JOBNAME* - NCILOG DD STATEMENT NOT CODED ABENDING 201**	An attempt was made to initialize with no NCILOG DD JCL statement (MVS) or NCILOG FILEDEF PROFILE GCS statement (VM).	Refer to Abend Code 201 on page 11.27
***JOBNAME* - NCILOG*n* DD STATEMENT NOT CODED - ABENDING 201**	NCI/XF could not find a DD JCL statement (MVS) or FILEDEF PROFILE GCS statement (VM) for one of its log files.	Include the required DD or FILEDEF statement or use a DD DUMMY statement if necessary.
***JOBNAME* - NCI IS ATTEMPTING RECOVERY**	NCI/XF is attempting to recover from an error it has detected.	See associated messages for further details.
***JOBNAME* - NC-XMS DISCONNECTED BY ESTAE**	NC-XMS main task has abended (for example NCI/XF job cancelled) or system 052 abend occurred.	Refer to MVS system codes if necessary.
***JOBNAME* - NC-XMS SSCT MARKED INACTIVE BY ESTAE**	NC-XMS main task has abended (for example NCI/XF job cancelled) or system 052 abend occurred.	Refer to MVS system codes if necessary.
***JOBNAME* - NO LONGER ACCEPTING TERMINAL LOGONS**	The NCI/XF ACB has closed.	This message is for information only.
***JOBNAME* - NON-NUMERIC CHARACTERS FOUND IN *XXXX* = PARM**	The indicated parm must have a numeric value.	Change the incorrect characters to numeric or delete as necessary.

WTO	Description	Action
***JOBNAME* - NOT LICENSED FOR THIS CPU - ABENDING 207**	This message indicates that the trial/license period for this copy of NCI/XF has expired. This message is also issued on a GCS machine if an attempt is made to run NCI/XF with a production code on a machine that is unauthorized.	If this message occurs during a trial period, contact your support office for a new startup authorization CODE. If this message occurs outside a trial period, contact your support office to find out why the license has expired. On a GCS machine, ensure your machine is authorized.
***JOBNAME* - NOW ACCEPTING TERMINAL LOGONS**	The NCI/XF ACB is open and accepting logons.	This message is for information only.
***JOBNAME* - PLEASE CONTACT YOUR LOCAL SUPPORT CENTRE**	Your local support centre may be able to supply details of corrective maintenance.	See associated messages for further details.
***JOBNAME* - RUNTIME ERROR IN STARTUP PANEL, EXEC LIMIT EXCEEDED, ABENDING 203**	The)INIT or)UPDATE section of the startup panel has failed. Because startup panel processing is often critical to the correct operation of the system, an Abend 203 message is issued. Details of the error may be found on the NCILOG file.	Correct the error and restart.
***JOBNAME* - RUNTIME ERROR IN STARTUP PANEL, GOTO LIMIT EXCEEDED, ABENDING 203**	The)INIT or)UPDATE section of the startup panel has failed. Because startup panel processing is often critical to the correct operation of the system, an Abend 203 message is issued. Details of the error may be found on the NCILOG file.	Correct the error and restart.
***JOBNAME* - RUNTIME ERROR IN STARTUP PANEL, ILLEGAL VERB FOUND. ABENDING 203**	The)INIT or)UPDATE section of the startup panel has failed. Because startup panel processing is often critical to the correct operation of the system, NCI/XF is abending. Details of the error may be found on the NCILOG file.	Correct the error and restart.
***JOBNAME* - RUNTIME ERROR IN STARTUP PANEL, INCREMENT/DECREMENT ERROR, ABENDING 203**	The)INIT or)UPDATE section of the startup panel has failed. Because startup panel processing is often critical to the correct operation of the system, an Abend 203 message is issued. Details of the error may be found on the NCILOG file.	Check that the variable which is being incremented or decremented has been previously SET to a numeric value. Correct the error and restart.
***JOBNAME* - RUNTIME ERROR IN STARTUP PANEL, INVALID GOTO LABEL GENERATED, ABENDING 203**	The)INIT or)UPDATE section of the startup panel has failed. Correct system operation depends on startup panel processing, so an Abend 203 message is issued. Details may be found on the NCILOG file.	Correct the error and restart.

WTO	Description	Action
***JOBNAME* - RUNTIME ERROR IN STARTUP PANEL, INVALID VARIABLE NAME GENERATED**	The)INIT or)UPDATE section of the startup panel has failed. Because startup panel processing is often critical to the correct operation of the system, an Abend 203 message Is issued. Details may be found on the NCILOG file.	Correct the error and restart.
***JOBNAME* - RUNTIME ERROR IN STARTUP PANEL, INVALID VARIABLE NAME GENERATED**	The)INIT or)UPDATE section of the startup panel has failed. Because startup panel processing is often critical to the correct operation of the system, an Abend 203 message is issued. Details may be found on the NCILOG file.	Correct the error and restart.
***JOBNAME* - RUNTIME ERROR IN STARTUP PANEL, UNABLE TO BUILD EXEC ROUTINE, ABENDING**	A routine invoked by an EXEC statement with the)INIT section of the startup panel contains a syntax error. Because startup panel processing is often critical to the correct operation of the system, an ABEND 203 message is issued. Details may be found on the NCILOG file.	Correct the error and restart.
***JOBNAME* - RUNTIME ERROR IN STARTUP PANEL, USER CALL ROUTINE ABENDED, ABENDING 203**	The)INIT or)UPDATE section of the startup panel has failed. Correct system operation depends on startup panel processing, so an Abend 203 message is issued. Depending on the setting of the NCDIALOG ESTAE *dump* option, a storage dump may have been taken.	Correct the error and restart.
***JOBNAME* - SLUS = PARM MUST BE IN THE RANGE 2-32767**	The SLUS = parm specified in the JCL is not in the range 2 through 32767.	Respecify the parameter within the correct range.
***JOBNAME* SQUE ~ PARM MUST BE IN THE RANGE 2-32767**	The SQUE = parm specified in the JCL is not in the range 2 through 32767.	Respecify the parameter within the correct range.
***JOBNAME* - STARTUP PANEL INITIATION FAILED**	Due to a VTAM error NCI/XF can not initiate a startup panel.	Contact your local support office.
***JOBNAME* - STILL PROCESSING PREVIOUS MODIFY COMMAND**	A command has been entered from the system operator console, but a previously issued command is still executing. Only one command can be in progress at once.	Wait for the previous command to respond before reissuing the command.
***JOBNAME* - SUBTASK TERMINATION FAILED**	Owing to a VTAM error NCI/XF can not terminate a subtask. This may cause a terminal hang.	Contact your local support office.

WTO	Description	Action
***JOBNAME* - SUPERFLUOUS CHARACTERS FOUND IN JCL PARM**	Characters were found which were not necessary for the interpretation of the JCL parm.	Delete or respecify the characters.
***JOBNAME* - SUPERFLUOUS CHARACTERS FOUND IN *XXXX* = PARM**	Characters were found which were not necessary for the interpretation of the parm indicated.	Delete or respecify the characters.
***JOBNAME* - SVAR = PARM MUST BE IN THE RANGE 2-32767**	The SVAR = parm specified in the JCL is not in the range 2 through 32767.	Respecify the parameter within the correct range.
***JOBNAME* - SWITCH LOG REQUESTED**	A switch log command has been issued requesting NCI/XF to close one log and switch to the alternative.	This message is for information only. No action is required.
JOBNAME* - SWITCHING LOG FILES FROM ***** TO **********	NCI/XF is switching log processing from one log to the alternate (i.e. NCILOG1 and NCILOG2)	If the log is written to SPOOL, no action is required. However, if the log is written to DISK it could be dumped down and cleared at this point.
***JOBNAME* - UNABLE TO BUILD STARTUP PANEL, ABENDING 203**	A syntax error exists in the startup panel definition which prevents building an in-storage copy of the panel. Because startup panel processing is often critical to the correct operation of the system, an Abend 203 message is issued. Details of the error may be found on the NCILOG file.	Correct the error and restart.
***JOBNAME* - UNABLE TO LOAD INITIALIZE ROUTINE - ABENDING 200**	An error was encountered while loading the initialization routine from the steplib library.	Refer to Abend Code 200 on page 11.27.
***JOBNAME* - UPDATE INITIATION FAILED**	Owing to a VTAM error NCI/XF can not initiate an update.	Contact your support office.
***JOBNAME* - UQUE = PARM MUST BE IN THE RANGE 2-32767**	The UQUE = parm specified in the JCL is not in the range 2 through 32767.	Respecify the parameter within the correct range.
***JOBNAME* UVAR = PARM MUST BE IN THE RANGE 2-32767**	The UVAR = parm specified in the JCL is not in the range 2 through 32767.	Respecify the parameter within the correct range.
***JOBNAME* - VTAM TEMPORARY BUFFER SHORTAGE DETECTED**	NCI/XF has found a VTAM temporary buffer shortage. NCI/XF recovers from most of these shortages.	Increase the buffer size.
JOBNAME* - WAITING FOR ACB *acbname	During initialization, NCI/XF tried to open an ACB for use. But the ACB was unavailable.	Establish why the ACB is unavailable, and make it available or restart having specified the name of an available ACB in the ACBNAME = startup parameter.

WTO	Description	Action
***JOBNAME* - WAITING FOR ACB ACTIVATION - COMMAND IGNORED**	Unable to process any operator commands because the ACB is currently not available.	Wait until initialization is complete before reissuing the command.
***JOBNAME* - *WARNING* DEVICE ERROR - INVALID INBOUND RU**	NCI/XF received data in an unexpected format from a terminal. This is caused by an error with the terminal. A SET BUFFER ADDRESS was detected but the two byte value which should follow it did not appear. The panel ERROR-EXEC is executed.	Report this error to your hardware supplier.
JOBNAME* - **WARNING** PANEL STORAGE LIMIT EXCEEDED - NUMBER OF BYTES NEEDED = *nn,nnn,nnn,nnn	NCI/XF cannot maintain the panel storage limit specified at startup. All panels in storage are in use and more space is needed.	Respecify the panel storage limit as soon as possible.
***JOBNAME* - WARNING - PREVIOUS UPDATE FOR STARTUP PANEL INCOMPLETE. PLEASE INFORM ADMINISTRATOR.**	A previous update section has not completed before the next invocation of the panel. If the startup panel issues STATUS requests, this message may indicate a problem in your network.	If this message occurs frequently check that the UPDATE-INTERVAL specified is long enough to allow the processing in the startup panel)UPDATE section to complete. If the startup panel issues STATUS requests, check that all nodes in your network are responding normally.
***JOBNAME* - **WARNING** YOUR LICENSE EXPIRES WITHIN 14 DAYS**	This message indicates that the trial/license period for this copy of NCI/XF has almost expired.	If this message occurs during a trial period, contact your support office for a new startup authorization CODE. If this message occurs outside a trial period, contact your support office to find out why the license has expired.
***JOBNAME* WARNING: PROD-MODE DISABLED FOR "PANEL"**	During initialization NCI/XF could not use the PROD-MODE = YES option for the PANEL library as requested.	No datasets in the PANEL library concatenation should have any secondary space allocation.
***JOBNAME* WARNING: PROD-MODE DISABLED FOR "PBLIB"**	During initialization NCI/XF could not use the PROD-MODE = YES option for the PBLIB library as requested.	No datasets in the PBLIB library concatenation should have any secondary space allocation.
***JOBNAME* WARNING: UNABLE TO RUN NON-SWAPPABLE**	The startup option NON-SWAP = YES has been specified or allowed to default, but NCI/XF cannot make itself non-swappable, usually because it is not running authorized.	Authorize NCI/XF or specify NON-SWAP = NO.

NCI/XF log file messages

During normal running, NCI/XF writes information and error messages (to the file indicated by the NCILOG DD statement or the NCILOG FILEDEF command) which provide useful trace back messages in case of errors. The messages range from simple information types, indicating when terminals connect, to detailed messages issued if NCI/XF fails to initialize successfully. This section lists messages that can be written to the NCILOG file during normal operation, together with descriptions of each message. Messages not included here may be issued under the following circumstances:

- If NCI/XF tracing is active, trace records will be recorded on the NCILOG file. These are primarily intended for support centre personnel.
- If VSM-TRACE was specified at NCI/XF startup, trace records produced by NCI/XF's internal Virtual Storage Manager will be written to the NCILOG file. These are primarily intended for support center personnel.
- The NCI/XF language LOG statement may be used within a panel to write a customer-defined message to the NCILOG file. If you use the LOG statement a lot, you may wish to add a record of your own messages and their meaning.

Message	Description
HEXP *****ERROR*** NEGATIVE LENGTH SPECIFIED**	Indicates that a data string passed to the hexadecimal print routine had a negative length. Any occurrence of this message should be reported to your local support office.
NCATTR **INVALID ATTRIBUTE ASSIGNMENT**	The PANEL build processor has found an invalid keyword in the specification of an attribute character for a panel. See the NCI/XF Reference Manual, *Chapter 1 - NCI/XF language statements*, for a complete list of keywords for this verb. The invalid card will be written to the NCILOG before this message.
NCDIRB **ERROR: ABOVE DIRECTORY CARD IS INVALID**	During either the initialization stage or an on-line update of the System Directory of Users, NCI/XF discovered a syntax error in the DIRECTORY member. The invalid card will be written to the NCILOG before this message.
NCDIRB **ERROR: INVALID USERID ON ABOVE CARD**	During the construction of the System Directory, NCI/XF found a USER statement which contained an invalid USERID field. The statement in error is written to the NCILOG before this message.
NCDIRB **ERROR: INVALID PASSWORD ON ABOVE CARD**	During the building of the System Directory, NCI/XF found a USER statement containing an invalid PASSWORD field. The card in error will be written to the NCILOG before this message.
NCDIRB **DIRECTORY BUILD SUCCESSFUL**	This information-only message indicates that NCI/XF has successfully built the table of authorized users that it stores.
NCDIRB **ERROR: UNABLE TO OPEN DIRECTORY**	Either during the initialization stage or an on-line update of the System Directory of Users, NCI/XF could not open the dataset containing the directory member owing to an I/O error or device malfunction.
NCESTAE **ABEND OCCURRED AT *psw-addr* (OFFSET *offset*) IN MODULE *module-name* CODE S*nnn* U*nnnn* CALLED FROM *address* (OFFSET *offset*) IN MODULE *module-name***	NCI/XF has abended. To help find the problem, the ESTAE routine has printed all involved modules with relevant abend codes etc. Any occurrence of this message should be reported to your local support office immediately.
NCESTAE **===== NCESTAE COMPLETE =====**	ESTAE error recovery processing has completed successfully and all relevant fields have been printed.
NCEXEC **SUBROUTINE *name* CREATED SUCCESSFULLY {LAST FIX *xxxxxxxx* : LEVEL *x.x.x*}**	NCI/XF has successfully built an incore copy of the EXEC routine *name*. The optional suffix to the message providing fix information will appear for distributed PBLIB routines. This suffix will not appear for EXEC routines compiled with NC-COMPILE. *xxxxxxxx* shows the fix number, or **BASE** if no fixes have been applied; LEVEL *x.x.x* is for CKS internal use only.
NCEXEC **ERROR: UNABLE TO BUILD SUBROUTINE *member-name***	NCI/XF could not build the EXEC routine called *member-name* because of error(s). Check the NCILOG file for any error messages before this one.

Message	Description
NCEXEC **ERROR: ALLOCATION OPEN OR RECFM ERROR**	NCI/XF could neither open nor allocate the library indicated by the PANEL DD statement or PANEL FILEDEF command. This library must be cataloged and only record formats (RECFM) F, FB, V or VB are supported.
NCEXEC **SUBROUTINE CONTAINS INVALID STATEMENTS - SEE PREVIOUS NCILOG MESSAGES FOR ERROR**	During the construction of an incore copy of a subroutine at least one invalid statement was encountered. The statement(s) in error will precede this message on the NCILOG file.
NCEXEC **SUBROUTINE CREATION FAILED - SEE PREVIOUS NCILOG MESSAGES FOR ERROR**	During the building of an incore copy of a subroutine, at least one error condition occurred. A detailed description of the error will precede this message on the NCILOG file.
NCEXEC **ERROR: PANEL LIBRARY FIXED RECORD LENGTH GREATER THAN 132 CHARACTERS**	The maximum LRECL which may be specified for a panel library with RECFM of F or FB is 132.
NCEXEC **ERROR: PANEL LIBRARY VARIABLE RECORD LENGTH GREATER THAN 136 CHARACTERS**	The maximum LRECL which may be specified for a panel library with RECFM of V or VB is 136.
NCEXEC **ERROR: NO OR INVALID SECTION DEFINED**	An EXEC routine contains an invalid section heading, or a mandatory section is omitted.
NCEXEC **ERROR: NO OR INVALID FUNCTION SPECIFIED**	Within an EXEC routine, NCI/XF has detected a statement with invalid or missing operands.
NCEXEC **ERROR: NO OR INVALID OPERANDS SPECIFIED**	Within an EXEC routine, NCI/XF has found a statement with invalid or missing operands.
NCEXEC **ERROR: ATTRIBUTE CHARACTER INVALID**	An EXEC routine has found an invalid character used as an attribute. See *ATTRibute* on page 1.14 for the rules for attribute assignment.
NCEXEC **ERROR: ATTRIBUTE ASSIGNMENT INVALID**	An EXEC routine has found an invalid character used as an attribute. See *ATTRibute* on page 1.14 for the rules governing attribute assignment.
NCEXEC **ERROR: CHARACTER MISSING OR INVALID**	The panel build processor could not locate a certain character on an ATTRIBUTE statement for assigning as an attribute.
NCEXEC **CREATING SUBROUTINE *member-name***	During normal processing an EXEC routine which was not already in storage has been referenced and NCI/XF is building an incore copy of it.
NCEXEC **ERROR: INVALID NUMBER OF OPERANDS**	The NCI/XF verb listed before this message has been specified with the incorrect number of operands.
NCEXEC **ERROR: INVALID SUBSTRING VALUES SPECIFIED**	The SUBSTRING verb listed before this message was specified incorrectly.
NCEXEC **ERROR: OPERATOR NOT EQ,LT,GT,LE,GE OR NE**	The statement listed before this message contains a verb which requires an algebraic operator as part of its syntax. However, the specified parameter was not in the accepted list.

Message	Description
NCEXEC **ERROR: OPERATOR NOT "="**	The statement listed before this message contains a verb which requires the equals sign (=) as its operator for correct syntax, but it was incorrectly specified.
NCEXEC **ERROR: INVALID CONJUNCTION**	An NCI/XF verb which allows the specification of a conjunction, for instance SET, has been coded invalidly.
NCEXEC **ERROR: UNAUTHORIZED FUNCTION SPECIFIED**	An unauthorized NCI/XF verb has been coded in an EXEC routine. If you want to use this EXEC routine contact your support office.
NCEXEC **ERROR: *XXXXXXXX* SUBROUTINE NOT AUTHORIZED**	An unauthorized prebuilt EXEC routine has been called. If you want to use this EXEC routine contact your support office.
NCEXEC **SESSION TERMINATED - ERROR COUNT EXCEEDED**	NCI/XF encountered an error within a subroutine. While attempting to display the error messages a second error was found. To avoid a loop condition NCI/XF will permit only three errors to occur before rejecting the terminal.
NCEXEC ****WARNING** PANEL STORAGE UNIT EXCEEDED - NUMBER OF BYTES NEEDED = *nn,nnn,nnn,nnn***	NCI/XF cannot maintain the panel storage limit specified at startup. All panels in storage are in use. More space is needed. If this problem is frequent, panel storage must be increased.
NCI **ACB HAS BEEN CLOSED EITHER BY OPERATOR OR SYSTEM - ABENDING 205**	NCI/XF has discovered that the ACB under which it is running has been made inactive either by VTAM itself or more probably by a system operator command.
NCI **ERROR DURING OPEN OF ACB - ERROR = X'*nn*'**	NCI/XF found an error while trying to open its ACB. Possible reasons are: • a spelling mistake • the ACB is inactive • the ACB is not defined to VTAM • the ACB is in use by someone else. The hexadecimal error code *nn* is set to the value of ACBERFLG. See the *VTAM Programming Guide* for more information.
NCI **NCI/XF AUTHORIZED**	This information only message indicates that the CODE = startup parameter allows you to use the extended version of NCI/XF, but not the full screen edit feature. If you want to use the full screen edit feature, contact your support office.
NCI **NCI/XF PLUS NCEDIT AUTHORIZED**	This information only message indicates that the CODE = startup parameter allows you to use all the features of NCI/XF and the full screen edit utility.
NCI **TERMINATION COMPLETE**	This information only message indicates that NCI/XF has been successfully shutdown in response to an operator command or a NCI/XF SHUTDOWN statement.

Message	Description
NCI **UPDATE PROCESSING DISABLED**	The UPDATE-INTERVAL = NO startup parameter was specified, or NCI/XF could not attach its timer services subtask. No update processing will be done during this execution of NCI/XF. Update processing should not be re-enabled by means of the UPDATE statement.
NCI **VTAM VERSION IDENTIFIER = *XXXXXXXX***	Indicates the VTAM level of the system on which NCI/XF is operating. Always quote this value when reporting problems to your support office.
NCINIT **ADDRESS SPACE ALREADY MARKED NON-SWAPPABLE**	The startup option NON-SWAP = YES has been specified or allowed to default and NCI/XF is already marked as non-swappable.
NCINIT **ADDRESS SPACE MARKED NON-SWAPPABLE**	This informational message indicates that NCI/XF has made itself non-swappable.
NCINIT **ADDRESS SPACE NOT AUTHORIZED**	This informational message indicates that NCI/XF is not running authorized.
NCINIT **ADDRESS SPACE RUNNING AUTHORIZED**	This informational message indicates that NCI/XF is running authorized.
NCINIT **ADDRESS SPACE RUNNING SWAPPABLE**	This warning message indicates that NCI/XF was loaded from a non authorized library and is running swappable. Execution will continue but response times may fluctuate in a heavy paging/swapping environment as NCI/XF is swapped in and out.
NCINIT ***XXXXXXXX* AUTHORIZED BY CODE STARTUP OPTION**	This informational message indicates that the NCI/XF subproduct is authorized for use.
NCINIT **ERROR: ACB PARAMETER NOT SPECIFIED**	The ACBNAME startup parameter was omitted or misspelled. This parameter is mandatory. Update the file indicated by the STARTUP DD statement or STARTUP FILEDEF command to include the ACBNAME option before restarting NCI/XF.
NCINIT **ERROR: ATTACH FAILED FOR NCSUBT**	NCI/XF could not attach a subtask. This error should be reported immediately to your local support office.
NCINIT **ERROR: DATE FORMAT INCORRECT**	The DATE = startup option was specified incorrectly.
NCINIT **ERROR: ESTAE INVOCATION FAILURE - RESTART**	During initialization, NCI/XF could not establish its ESTAE recovery environment. Check your installation procedure before restarting NCI/XF. If the problem persists contact your local support office.
NCINIT **ERROR: INVALID DUMP MAP**	The DUMP-MAP startup option was specified incorrectly.
NCINIT **ERROR: INVALID MAX-LOG-SIZE**	An invalid value was specified for the MAX-LOG-SIZE startup option. Restart NCI/XF having specified a numeric value in the range 0 to 999999.

Message	Description
NCINIT ERROR: INVALID OR MISSING "EXEC" VALUE - MUST BE NUMERIC 1 - 32767	The MAXEXEC startup option was specified incorrectly.
NCINIT ERROR: INVALID OR MISSING "GOTO" VALUE - MUST BE NUMERIC 1 - 999999	The MAXGOTO startup option was specified incorrectly.
NCINIT ERROR: INVALID OR MISSING LOG-PAGE-LENGTH	An invalid value was specified for the LOG-PAGE-LENGTH startup parameter. Restart NCI/XF having specified a numeric value between 1 and 99.
NCINIT ERROR: INVALID OR MISSING MAX SUBTASK VALUE - MUST BE NUMERIC 1 TO 250	An invalid value was specified for the MAXIMUM-SUBTASKS startup parameter. Restart NCI/XF having specified a numeric value in the range 1 to 250.
NCINIT ERROR: INVALID OR MISSING PANEL STORAGE	An invalid value was specified for the panel storage startup parameter. Restart NCI/XF having specified a valid value.
NCINIT ERROR: INVALID OR MISSING TERMINAL-UPDATE	An invalid value was specified for the. TERMINAL-UPDATE startup option. Restart NCI/XF having specified a valid numeric value.
NCINIT ERROR: INVALID STATEMENT EXECUTION MONITOR MASK	An invalid value was specified for the SXM-PANEL startup option. Restart NCI/XF, having specified a valid mask value.
NCINIT ERROR: INVALID TRACE EXEC MASK	An invalid value was specified for the TRACE-EXEC startup option. Restart NCI/XF having specified a valid mask value.
NCINIT ERROR: INVALID TRACE TERMINAL MASK	An invalid value was specified for the TRACE-TERM startup option. Restart NCI/XF having specified a valid mask value.
NCINIT ERROR LOADING MODULE *module-name*	During initialization NCI/XF could not load *module-name*. Ensure that the steplib you are running contains all the NCI/XF modules before contacting your support office.
NCINIT ERROR: MISSING QUEUE-MASK CHARACTER	The optional QUEUE-MASK startup parameter was specified, but no character was coded to the right of the equals sign (=).
NCINIT ERROR: "PANEL" DD STATEMENT NOT CODED	A PANEL DD statement is not present in the JCL used to invoke NCI/XF, or under VM there is no PANEL FILEDEF command in the PROFILE GCS. NCI/XF can not function without any panel definitions.
NCINIT ERROR: "PANEL" LIBRARY OPEN FAILURE	During initialization NCI/XF encountered an error while opening the PANEL library. NCI/XF cannot function without any panel definitions.
NCINIT ERROR: "TDT" DD STATEMENT NOT CODED	During initialization NCI/XF could not locate the library containing the Terminal Definitions Table which defines the terminals that NCI/XF is to control.

Message	Description
NCINIT **ERROR: *XA-MODE = YES* INVALID FOR THIS OPERATING SYSTEM**	If XA-MODE = YES is specified in the startup options, NCI/XF must be running on an XA operating system.
NCINIT **INITIALIZATION COMPLETE**	NCI/XF has successfully built an incore table of terminals to control, processed any startup parameters, opened its ACB, issued SETLOGON and is scheduling the send of initial panels to all LOGAPPL'd terminals.
NCINIT **INVALID OR MISSING UPDATE - INTERVAL**	An invalid value was specified for the UPDATE-INTERVAL startup parameter. Restart NCI/XF having specified a valid value.
NCINIT **INVALID STARTUP-PANEL/EXEC**	The panel name specified in the STARTUP-PANEL startup parameter is invalid. Restart NCI/XF having specified the name of a member which exists in the library indicated by the PANEL DD statement or PANEL FILEDEF command.
NCINIT **INVALID STARTUP PARAMETER CODED**	During initialization NCI/XF found an error in the startup options specified in the STARTUP DD statement or the STARTUP FILEDEF command.
NCINIT **NCI ABENDING DUE TO ABOVE ERRORS**	Execution of NCI/XF has been stopped because of errors during initialization. The error(s) have been written to the NCILOG file before this message.
NCINIT **NCI VERSION *X.X* EXECUTING ON CPU *XXXXXX***	This message details the version of NCI/XF and the CPUID of the machine on which it is executing.
NCINIT **NOT LICENSED FOR THIS CPU**	This message indicates that the trial/licence period for this copy of NCI/XF has expired. Contact your support office. This message is also issued on a GCS machine if an attempt is made to run NCI/XF with a production code on a machine that is unauthorized.
NCINIT ***** PASSWORD SUPPRESSED *****	A PASSWORD startup parameter has been encountered and processed. For security reasons, the password is not printed.
NCINIT *****PRECEDING CODE EXPIRED OR INVALID FOR CPUID *XXXXXX***	An invalid authorization code was specified on a CODE = startup parameter. The code in error is not suppressed and is displayed on the line before this message.
NCINIT **THE FOLLOWING FIXES HAVE BEEN APPLIED**	This message precedes a matrix which indicates those fixes that have been applied to your version of NCI/XF.
NCINIT *****VALID CODE SUPPRESSED*****	A CODE startup parameter has been encountered and processed. For security reasons, valid codes are not printed.

Message	Description
NCINIT **WARNING: "DIRECT" DD STATEMENT NOT CODED**	This warning is issued if NCI/XF cannot locate a directory of users at startup. This optional directory allows simple userid security checking and should only be used if network security is required and no alternative security system is available at your installation.
NC INIT *****WARNING*** NO CURRENT STARTUP CODE FOUND** *****WARNING*** PLEASE CONTACT YOUR LOCAL SUPPORT OFFICE**	The code specified on the CODE = startup parameter is valid but has expired or is not valid for this CPU. NCI/XF will not be allowed to start up. Contact your local support office.
NC INIT *****WARNING*** NO CURRENT STARTUP CODE FOUND** *****WARNING*** EXPIRED CODE ACCEPTED TEMPORARILY** *****WARNING*** COURTESY STARTUP GRANTED. PLEASE CONTACT** *****WARNING*** YOUR LOCAL SUPPORT OFFICE URGENTLY**	The code specified on the CODE = startup parameter is valid but has expired or is not valid for this CPU. A courtesy NCI/XF startup will be allowed on a temporary basis. Contact your local support office as soon as possible. The entered code will be printed to the NCI/XF log and indicated as invalid.
NCINIT **WARNING: "PBLIB" LIBRARY OPEN FAILURE**	During initialization NCI/XF encountered an error while opening the PBLIB library. The PBLIB library is not essential for NCI/XF execution and the job will continue to execute.
NCINIT ****WARNING** YOUR LICENCE EXPIRES WITHIN 14 DAYS**	The code specified on the CODE = startup parameter is valid but is about to expire. Without a valid code NCI/XF will not start up.
NCINIT **XA-MODE NOT LICENSED FOR THIS CPU**	NCI/XF was asked to use storage above the 16mb line owing to the assignment of BUFF = X or XA-MODE = YES. This is impossible on a CPU which is not licensed for XA-mode. Remove BUFF=X or change XA-MODE = YES to XA-MODE = NO, or contact your support office for a valid code.
NCINIT **31 BIT STORAGE ENABLED**	This informational message says the XA-MODE = YES startup option message was specified. NCI/XF will use storage above the 16mb line.
NCLANG **ATTACH OF STARTED TASK FAILED**	If this message occurs contact your support office.
NCLANG **PANEL *XXXXXXXX* MARKED FOR DELETION**	The NCI/XF panel *XXXXXXXX* has been marked for deletion because of an NCI/XF delete command.
NCLANG **MAXIMUM STARTED TASKS LIMIT REACHED**	If this message occurs contact your support office.
NCLANG **USER EXIT ABENDED - SDWA FOLLOWS AND DUMP PRODUCED**	A user exit, invoked via the VALIDATE verb, has abended. To help find the error NCI/XF has produced a dump and listed the SDWA. Note that an abend in an exit will not cause NCI/XF to fail.

Message	Description
NCLANG **TRACE OPTIONS SET *option option option...***	In response to a TRACE verb, NCI/XF has invoked the trace options listed.
NCLANG **TERMINAL *luname* FORCED OFF - REMOVED FROM DIRECTORY**	A user at terminal *luname* has issued an NCI/XF command that requires information from the TDT. But the entry for the terminal has been removed since NCI/XF was started.
NCLANG **TRACE EXEC MASK SET - *mask***	Owing to a TRACEEXEC statement, the Exec tracing mask has been set to the indicated value. Subsequent tracing will be limited to the resource(s) designated.
NCLANG **TRACE TERMINAL MASK SET - *mask***	Owing to a TRACETERM statement, the terminal tracing mask has been set to the indicated value. Subsequent tracing will be limited to the resource(s) designated.
NCLANG **TRACEVAR MASK SET FOR VARIABLE - *&xxxxx***	The TRACEVAR statement has been used to specify that variable *&xxxxx* should be traced.
NCLANG **TRACEVAR MASKS RESET**	Any tracing of selected variables through use of the TRACEVAR statement has been reset.
NCLANG **SWITCH LOG REQUESTED**	A switch log command has been issued to request NCI/XF to close the active log and switch to the alternate one. This message is for information purposes only.
NCLANG ***termid* - GOTO LIMIT EXCEEDED**	A GOTO statement, within a routine EXEC'd through NCDIALOG has exceeded the MAX-GOTO limit established at startup. *termid* identifies the terminal that has encountered the problem.
NCLANG ***termid* - EXEC LIMIT EXCEEDED**	An EXEC statement, within a routine EXEC'd through NCDIALOG has exceeded the MAX-EXEC limit established at startup. *termid* identifies the terminal that has encountered the problem.
NCLANG ***termid* - INCREMENT/DECREMENT ERROR**	An INCREMENT or DECREMENT statement within a routine EXEC'd through NCDIALOG has failed. *termid* identifies the terminal that has encountered the problem.
NCLANG ***termid* - INVALID GOTO LABEL GENERATED**	A variable GOTO statement, within a routine EXEC'd through NCDIALOG has resolved an invalid label name. *termid* identifies the terminal that has encountered the problem.
NCLANG ***termid* - INVALID VARIABLE NAME GENERATED**	An invalid variable name has been resolved from a variable variable, within a routine EXEC'd through NCDIALOG. *termid* identifies the terminal that has encountered the problem.
NCLANG ***termid* - USER CALL ROUTINE ABENDED**	An exit routine invoked by a CALL statement within a routine EXEC'd through NCDIALOG has abended. *termid* identifies the terminal that has encountered the problem.

Message	Description
NCLANG ***termid* - ILLEGAL VERB FOUND**	An attempt has been made to use a verb that is not available when running a routine EXEC'd through NCDIALOG. *termid* identifies the terminal that has encountered the problem.
NCOPER **ABEND IN PROGRESS**	F NCI,ABEND has been issued from a system operator console and is being processed.
NCOPER **EXTENDED MODIFY DISABLED BY THE STARTUP OPTION**	A command from the extended control set was entered from a system operator console, but EXTENDED MODIFY support was not enabled by specification of the EXTENDED-MODIFY = YES.
NCOPER **INVALID MODIFY COMMAND**	An invalid command was entered from a system operator console.
NCOPER **MODIFY COMMAND RECEIVED: *command***	The indicated command was entered from a system operator console.
NCOPER **STILL PROCESSING PREVIOUS MODIFY COMMAND**	A command was entered from a system operator console but a command was already being processed.
NCPANB **CREATING PANEL *member-name***	During normal processing a panel which was not already in storage has been referenced and NCI/XF is building an incore copy of it.
NCPANB **PANEL *member-name* CREATED SUCCESSFULLY {LAST FIX *xxxxxxxx* : LEVEL *x.x.x*}**	NCI/XF has successfully built an incore copy of the panel *member-name*. The optional suffix to the message providing fix information will appear for distributed PBLIB routines. This suffix will not appear for PANEL routines compiled with NC-COMPILE. *xxxxxxxx* shows the fix number, or **BASE** if no fixes have been applied; LEVEL *x.x.x* is for CKS internal use only.
NCPANB **ERROR: UNABLE TO BUILD PANEL *member-name***	NCI/XF could not build the indicated panel. Check the NCILOG file for any error messages before this one.
NCPANB **ERROR: ALLOCATION, OPEN OR RECFM ERROR**	NCI/XF could neither open nor allocate the library indicated by the PANEL DD statement or the PANEL FILEDEF command. This library must be cataloged and only record formats (RECFM) F, FB, V or VB are supported.
NCPANB **PANEL CONTAINS INVALID STATEMENTS - SEE PREVIOUS NCILOG MESSAGES FOR ERROR**	During the building of an incore copy of a panel at least one invalid statement was encountered. The statement(s) in error precede this message on the NCILOG file.
NCPANB **PANEL CREATION FAILED - SEE PREVIOUS NCILOG MESSAGES**	During the construction of an incore copy of a panel at least one error condition occurred. A detailed description of the error precedes this message on the NCILOG file.
NCPANB **ERROR: PANEL LIBRARY FIXED RECORD LENGTH GREATER THAN 132 CHARACTERS**	The maximum LRECL which may be specified for a panel library with RECFM of F or FB is 132.

Message	Description
NCPANB ERROR: PANEL LIBRARY VARIABLE RECORD LENGTH GREATER THAN 136 CHARACTERS	The maximum LRECL which may be specified for a panel library with RECFM of V or VB is 136.
NCPANB ERROR: NO OR INVALID SECTION DEFINED	During PANEL build processing, NCI/XF found an invalid section header, or a required section header was missing.
NCPANB ERROR: NO OR INVALID FUNCTION SPECIFIED	During panel build processing NCI/XF found a statement with either no function or an invalid one.
NCPANB ERROR: NO OR INVALID OPERANDS SPECIFIED	During panel build processing, NCI/XF found a statement either without operands or with invalid operands.
NCPANB ERROR: INVALID FUNCTION FOR PANEL SECTION	During panel build processing, NCI/XF found a statement which is not valid in the)PANEL section.
NCPANB ERROR: ATTRIBUTE CHARACTER INVALID	The panel build processor has found an Invalid character for use as an attribute.
NCPANB ERROR: ATTRIBUTE ASSIGNMENT INVALID	The panel build processor has found an error in the specification of an attribute character for a panel. The invalid statement is listed before this message.
NCPANB ERROR: BODY OF PANEL EXCEEDS PANEL LENGTH	During panel build processing, NCI/XF found that the length of a panel)BODY section exceeded the size defined by the ROWS statement.
NCPANB ERROR: PANEL NOT DELIMITED BY)END	As a precaution against missing statements within panel definitions NCI/XF insists on all panels being delimited by an)END statement.
NCPANB ERROR: CHARACTER MISSING OR INVALID	The panel PANEL build processor could not locate a single character on an ATTRIBUTE statement to assign as an attribute.
NCPANB ERROR: PANEL WIDTH NOT 40, 80 OR 132	The width of the panel defined by the COLUMNS statement is invalid. NCI/XF supports panels with a width of 40, 80 or 132.
NCPANB ERROR: PANEL LENGTH NOT 12,24,27,32 OR 43	The length of the panel as defined by the ROWS statement is invalid. NCI/XF supports panels 12, 24, 27, 32 or 43 lines long.
NCPANB ERROR: INVALID NUMBER OF OPERANDS	The NCI/XF verb listed before this message has been specified with the incorrect number of operands.
NCPANB ERROR: PANEL SECTION NOT DEFINED FIRST	During panel build processing, NCI/XF found an error in the order of panel sections.
NCPANB ERROR: INVALID SUBSTRING VALUES SPECIFIED	The statement listed before this message contains an invalid substring.
NCPANB ERROR: OPERATOR NOT EQ,LT,GT,LE,GE OR NE	The statement listed before this message contains a verb which requires an operator as part of its syntax, but the specified parameter was not in the accepted list.

Message	Description
NCPANB **ERROR: OPERATOR NOT '='**	The statement listed before this message contains a verb which requires the equals sign (=) as its operator for correct syntax, but it was incorrectly specified
NCPANB **ERROR: INVALID CONJUNCTION**	An NCI/XF verb which allows the specification of a conjunction, for example SET, has been coded invalidly.
NCPANB **SESSION TERMINATED - ERROR COUNT EXCEEDED OR PROBABLE LOOP**	During panel build processing NCI/XF found an error. While NCI/XF was attempting to display the error messages, a second error was found. To avoid looping NCI/XF will only permit three panel display errors to occur before rejecting the terminal.
NCPANB ****WARNING** PANEL STORAGE LIMIT EXCEEDED - NUMBER OF BYTES NEEDED = *nn,nnn,nnn,nnn***	NCI/XF cannot maintain the panel storage limit specified at startup. All panels in storage are in use and more space is needed. If this problem is frequent, panel storage must be increased.
NCPANEL **SESSION TERMINATED - ERROR COUNT EXCEEDED OR PROBABLE LOOP**	Sixteen PANEL statements have been executed without a)BODY section being sent to the terminal. The error message is output and the session terminated to avoid a possible loop condition.
NCREF **PREFRESH ISSUED FOR PANEL *panel-mask***	A PREFRESH *panel-mask* statement has been executed.
NCREF **PREFRESH ISSUED FOR PANEL *mask1* WITH *mask2***	A PREFRESH *mask1 mask2* statement has been executed.
NCREF **TREFRESH ISSUED FOR TERMINAL *term-mask***	A TREFRESH *term-mask* statement has been executed.
NCREF **TREFRESH ISSUED FOR TERMINAL *term-mask* WITH *panel-mask***	A TREFRESH *term-mask panel-mask* statement has been executed.
NCTDTB **ERROR: ABOVE TDT CARD IS INVALID**	During initialization or an on-line update of the Terminal Definitions Table, NCI/XF found syntax errors in the member containing the definitions. The card(s) in error will be written to the NCILOG file before this message.
NCTDTB **ERROR: ABOVE CARD NOT MEMBER OF GROUP**	During initialization or an on-line update of the Terminal Definitions Table, NCI/XF encountered a card not preceded by a GROUP definition and cannot process it.
NCTDTB **ERROR: NO PANEL AND OR TERMINAL STATEMENTS DEFINED FOR GROUP**	During initialization or an on-line update of the Terminal Definitions Table, NCI/XF encountered a GROUP definition that contained no associated TERMINAL or PANEL definitions.
NCTDTB **T.D.T. BUILD SUCCESSFUL**	NCI/XF has successfully built an incore copy of the TDT (Terminal Definitions Table). This is an information-only message.

Message	Description
NCTDTB **ERROR: MULTIPLE PANEL STATEMENTS DEFINED**	During initialization or an on-line update of the Terminal Definitions Table, NCI/XF encountered a GROUP definition that contained more than one associated PANEL definition.
NCTDTB **ERROR: UNABLE TO OPEN TDT LIBRARY**	During initialization or an on-line update of the Terminal Definitions Table, NCI/XF could not open the dataset that contains the member. This can be due to a device malfunction or an I/O error.
NCTERM **MAX CONCURRENT ASYNC STATUS COMMANDS = *value***	*value* displays the maximum number of asynchronous STATUS commands which were processed concurrently.
NCTERM **MAX CONCURRENT MODIFY COMMANDS = *value***	*value* displays the maximum number of MODIFY commands which were processed concurrently.
NCTERM **MAX CONCURRENT STARTED TASKS = *value1*,** **MAX NCDIALX GETPARM'S = *value2***	values displayed are for CKS internal use only.
NCTERM **MODULE *XXXXXXXX* TOTAL CALLS ~ *value1*,** **CALLS REQUIRING LOAD = *value2***	Module *XXXXXXXX* was called a total of *value1* times and had to be loaded when called *value2* times.
NCTERM **PANEL-STORAGE BYTES LIMIT = *value1*, MAXIMUM BYTES USED = *value2***	*value1* displays the maximum amount of storage in bytes that NCI/XF was allocated to use for building panels, as defined by the PANEL-STORAGE= startup parameter. *value2* displays the highest amount of storage in bytes that NCI/XF actually used for building panels.
NCTERM **STATEMENT EXECUTION MONITOR**	This message precedes the output produced by the NCI/XF Statement Execution Monitor.
NCTERM **TOTAL AVAILABLE TO BE SAVED ABOVE/BELOW LINE = *value1*, TOTAL SAVED/WASTED = *value2***	*value1* displays the total storage above or below the 16mb line available for saving by the NCI/XF virtual storage manager. *value2* displays the total storage saved or wasted by the NCI/XF virtual storage manager.
NCTERM **TOTAL GETMAIN/FREEMAIN SVC'S ISSUED = *value***	*value* displays the total number of GETMAIN/FREEMAIN SVCs issued by the NCI/XF virtual storage manager.
NCTERM **TOTAL GETMAIN/FREEMAIN SVC'S SAVED = *value***	*value* displays the total number of GETMAIN/FREEMAIN SVCs saved by the NCI/XF virtual storage manager.
NCTERM **TOTAL GETMAINED BYTES USED ABOVE / BELOW LINE *value***	*value* displays the amount of GETMAINED storage, in bytes, that NCI/XF was using above or below the 16mb line at shutdown. This does not include the storage required to load the actual NCI/XF modules, or storage obtained by user call routines for their own use.
NCTERM **TOTAL LOAD MODULE BYTES ABOVE/BELOW LINE = *value***	*value* displays the amount of storage used by NCI/XF modules above or below the 16mb line. This does not include storage used by call routines.

Message	Description
NCTERM **TOTAL MODIFY COMMANDS RECEIVED = *value***	*value* displays the total number of MODIFY commands received.
NCTERM **TOTAL NUMBER OF PANELS DELETED = *value***	*value* displays the total number of panel deletions.
NCTERM **TOTAL NUMBER OF VEXEC'S ISSUED = *value1*,** **TOTAL NCDIALOG EXEC'S = *value2***	*values* displayed are for CKS internal use only.
NCTERM **TOTAL NUMBER SYSTEM QUEUES = *value1*, TOTAL USER QUEUES = *value2***	*value1* and *value2* display the total number of system and user queues that were defined to NCI/XF at shutdown.
NCTERM **TOTAL NUMBER SYSTEM VARIABLES = *value1*,** **TOTAL USER VARIABLES = *value2***	*value1* and *value2* display the total number of system and user variables which were defined to NCI/XF at shutdown.
NCTERM **TOTAL PANELS HELD IN STORAGE = *value1*, TOTAL EXEC ROUTINES = *value2***	*value1* and *value2* display the total number of panels and EXEC routines built and held in storage at shutdown.
NCTERM **TOTAL SUCCESSFUL SCAVENGES = *value1*, TOTAL FAILED SCAVENGES = *value2***	*value1* and *value2* display the total number of successful and failed panel scavenges that occurred within the duration of the NCI/XF job.
NCTERM **TOTAL SYSTEM QUEUE BYTES = *value1*, TOTAL USER QUEUE BYTES = *value2***	*value1* and *value2* display the total amount of storage in bytes used for system and user queues respectively at NCI/XF shutdown.
NCTERM **TOTAL TERMINALS CONNECTED = *value***	*value* displays the total number of terminals which connected to NCI/XF within the duration of the NCI/XF job.
NCTERM **TOTAL VTAM BUFFER SHORTAGES = *value1*, VTAM RPLUSFLD ERRORS = *value2***	*value1* displays the total number of VTAM buffer shortages and *value2* the total number of VTAM RPL USER FIELD errors that were found by NCI/XF within the duration of the NCI/XF job.
NCTIMER **DELETING PANEL *XXXXXXXX***	Panel *XXXXXXXX* has been marked for deletion and is no longer in use. NCI/XF is recovering the storage.
NCTIMER **SCAVENGING PANEL *XXXXXXXX***	The panel storage limit has been reached. NCI/XF is trying to recover some panel storage and panel *XXXXXXXX* has been selected.
NCTIMER **UNABLE TO DELETE PANEL *XXXXXXXX*, USER COUNT IS *nnnnn***	Panel *XXXXXXXX* has been marked for deletion but is still in use. NCI/XF cannot recover the storage until it is no longer in use.
NCTIMER ****WARNING** PANEL STORAGE LIMIT EXCEEDED - NUMBER OF BYTES NEEDED = *nn,nnn,nnn,nnn***	NCI/XF cannot maintain the panel storage limit specified at startup. All panels in storage are in use and more space is needed. If this occurs frequently panel storage must be increased.
NCTIMERU **UPDATE INITIATED BY REQUEST**	An immediate update cycle has been initiated as a result of an UPDATE statement with no operands specified.

Message	Description
NCTIMERU **UPDATE INTERVAL CHANGED, NEW INTERVAL = *nnn***	An UPDATE *nnn* statement has been executed. The timer interval used for controlling)UPDATE processing has been changed to *nnn* seconds.
NCTIMERU **UPDATE PROCESSING ENABLED AND INTERVAL = *nnn***	UPDATE-INTERVAL = *nnn* was specified at NCI/XF initiation, or allowed to default to *nnn*.
NCTIMERU **UPDATE PROCESSING ENABLED BUT QUIESCED**	UPDATE-INTERVAL = 0 was specified at NCI/XF initiation. Update processing may be subsequently started using the UPDATE *nnn* statement.
NCTIMERU **UPDATE PROCESSING RESTARTED, INTERVAL = *nnn***	Update processing, which was previously quiesced, has been restarted as a result of an UPDATE *nnn* statement.
NCTIMERU **UPDATE PROCESSING QUIESCED**	Update processing has been halted as a result of an UPDATE 0 statement. Update processing may be restarted by issuing UPDATE with a non-zero value.
NCVERS **VERSION *X.X* RELEASED *XX/XX/XX* FOR *XXX***	This message details the release level of this version of NCI/XF. This information should be quoted when reporting problems to your local support office.
NCVLOGON **TERMINAL *xxxxxxxx* REJECTED-DISABLED**	Terminal *xxxxxxxx* has previously issued a disable command. NCI/XF is therefore rejecting this logon. The next logon attempt will not be rejected.
NCVLOGON **TERMINAL *luname* REJECTED - NOT DEFINED**	The terminal *luname* attempted to connect to NCI/XF which has rejected the connection because no entry could be found for the terminal in the TDT.
NCVLOSTE **TERMINAL *luname* LOST - CODE *XXX***	NCI/XF has lost control of the terminal luname for one of the following reasons: • a device malfunction • the operator has varied the terminal inactive to VTAM • the user has issued the SYSREQ LOGOFF command.
NCVNSEXI **SESSION TERMINATION - TERMINAL *luname***	NCI/XF tried to regain control of the terminal *luname* via the SIMLOGON macro, but failed because of an error condition at the terminal.
NCVNSEXI **SESSION SUCCESS - TERMINAL *luname* - APPLICATION *applid***	This information only message indicates that NCI/XF successfully passed control of the terminal *luname* to the application *applid*.
NCVNSEXI **SESSION FAILURE - TERMINAL *luname* - APPLICATION *applid***	NCI/XF tried to connect the terminal *luname* to application *applid*, but failed. NCI/XF will try to regain control of the screen via the SIMLOGON macro.
NCVNSEXI **SESSION CLEANUP - TERMINAL *luname***	An error condition that is classed as non-recoverable by VTAM occurred on terminal *luname*. NCI/XF will try to regain control of the terminal by creating a new session with it.

Message	Description
NCVNSEXI **TERMINAL INACTIVE TERMINAL *luname* - APPLICATION *applid***	An error condition that is classed as non-recoverable by VTAM occurred on terminal *luname*. NCI/XF will not try to regain control of the terminal. This is the result of a link error.
NCVSCIP **INVALID CONNECTION ATTEMPT BY *luname***	An application tried to start a session with NCI/XF when it is the SLU. This is not permissible.
NCVTPEND **VTAM HALT COMMAND ISSUED**	The system operator has requested VTAM to close. NCI/XF cannot function without VTAM and will terminate soon after this message is issued.
NCVTPEND **VTAM HALT QUICK OR V INACT I OR F ISSUED**	The system operator has requested VTAM to close or has varied NCI/XF inactive. In either case NCI/XF will terminate soon after this message is issued.
NCVTPEND **VTAM HALT CANCEL ISSUED**	The system operator has requested VTAM to close. NCI/XF cannot function without VTAM and will terminate soon after this message is issued.
NCVXOPND **RECOVERY FOR TERMINAL *luname* FAILED**	NCI/XF has failed to recover terminal *luname* probably because of a device malfunction.
NCVXOPND **TERMINAL *luname* CONNECTED - LOGMODE *mode-name***	This information only message is issued for every terminal that connects to NCI/XF unless the CONNECT-MESSAGE = NO startup option was specified.
NCVXOPND **TERMINAL *luname* INVALID BIND PARAMETERS**	Terminal *luname* has tried to connect to NCI/XF with bind parameters that do not match any of the devices currently supported by NCI/XF.
NCVXRECV ***WARNING* DEVICE ERROR - INVALID INBOUND RU**	NCI/XF received data in an unexpected format from a terminal. This is caused by an error with the terminal. A SET BUFFER ADDRESS was detected but the two byte value which should follow it did not appear. The panel ERROR-EXEC is executed. The problem should be reported to your hardware supplier.
NCVXRECV **DEVICE ERROR - INVALID RU** **CONTACT YOUR HELPDESK WITH THE FOLLOWING INFORMATION:** **NO BUFFER ADDRESS AFTER SBA** **CONTACT YOUR HARDWARE SUPPLIER**	NCI/XF received data in an unexpected format from a terminal. This is caused by an error with the terminal. A SET BUFFER ADDRESS was detected but the two byte value which should follow it did not appear. The panel ERROR-EXEC was executed resulting in these messages. The problem should be reported to your hardware supplier.
NCVXUPDP **WARNING - PREVIOUS UPDATE FOR STARTUP PANEL INCOMPLETE WITHIN INTERVAL**	A previous)UPDATE cycle of the startup panel has not yet completed, and it is time for a new cycle to start. Ensure that the update interval specified is sufficient to allow all processing to finish in time. If the startup panel is issuing regular application STATUS requests, this message may indicate a problem or slowdown in the network.

Abend codes

This section describes all abend codes that occur when NCI/XF detects an error condition. Abend codes for this version of NCI/XF are in the range 200 - 216, 500 - 501. The following table details possible abend codes and the action that should be taken if one occurs.

By default, NCI/XF will produce a dump for all abends. For abend codes 205 to 216, the dump can be suppressed by using the DUMP-MAP startup option. This may be useful for self-evident error conditions where a dump will not significantly assist problem determination, such as abend 205.

Local Support Office reporting

In the event of an abend, the following items of information should be made available to your local support office:

- MVS NCILOG listing or VM console listing at time of error
- dump produced by NCI/XF
- NCILOG listing produced by NCI/XF
- the JCL or PROFILE GCS used to run NCI/XF
- the current user directory and the current TDT

Any abend code not defined in the table should be reported directly to your support office.

Code	Description	Action
200	NCI/XF could not locate a module called NCINIT during initialization processing. Check that the STEPLIB DD card in the STARTUP JCL, or the STEPLIB FILEDEF command in the PROFILE GCS, indicates the correct dataset or that the library is specified correctly in the LNKLSTXX member of SYS1.PARMLIB. If no error can be found the module must have been inadvertently deleted or an error occurred during product installation.	If an invalid library has been specified correct the error, otherwise reinstall the product using the installation tape supplied. If the problem persists contact your support office.
201	An error occurred during initialization processing because NCI/XF could not locate both NCILOG DD statements. Probable missing DD statement/FILEDEF command or typing error.	Check that the NCI/XF STARTUP deck contains a DD statement/FILEDEF command with a DDNAME of NCILOG1 or NCILOG2 and that it relates to a SYSOUT file/printer or a valid sequential dataset. After correcting the DD statement/FILEDEF command restart NCI/XF. If the problem persists contact your support office.
202	An error was encountered while loading an NCI/XF processing routine PR*xxxx*.	Check the installation procedure before restarting NCI/XF. If the problem persists contact your support office.
203	NCI/XF encountered an error during initialization from which it could not recover. An information message about the error has been written to the NCILOG DD/NCILOG FILEDEF file. A startup panel runtime error was also encountered.	Correct the error indicated and restart NCI/XF. If the problem persists contact your support office.
204	During initialization NCI/XF encountered an error trying to establish an ESTAE environment.	Check the installation procedure before restarting NCI/XF. If the problem persists contact your support office.
205	The ACB under which NCI/XF is running has been deactivated. Probable operator error or VTAM failure.	Check the reason for the ACB closure, reactivate the node which defines the ACB and restart NCI/XF.
206	The system operator issued a MODIFY command requesting that NCI/XF terminate and produce a dump.	This action should only be done on advice from your support office. Pass the dump to them as soon as possible.
207	The CODE parameter specified on startup indicated that NCI/XF is not licensed for your CPU or that XA-MODE = YES was used but not authorized.	Contact your support office for assistance.
208	An error was encountered when NCI/XF issued the SETLOGON VTAM macro.	Check the installation procedure before restarting NCI/XF. If the problem persists contact your support office.

Code	Description	Action
209	NCI/XF encountered an error condition from which it could not recover.	Check the installation procedure before restarting NCI/XF. If the problem persists contact your support office.
210	An internal logic error has occurred resulting in the posting of an invalid ECB.	Any occurrence of this should be reported to your support office immediately.
211	An error occurred during the ATTACH of the NCI/XF timer subtask. This may be caused by an error during installation or may indicate that the NCI/XF region size is too small.	Check the installation procedure before restarting NCI/XF. If the problem persists increase the virtual storage size on the EXEC statement before restarting NCI/XF. If the problem persists contact your support office.
212	An error occurred during the initialization of the NCTIMER routine. This is probably caused by an error during initialization.	Check the installation procedure before restarting NCI/XF. If the problem persists increase the virtual storage size on the EXEC statement before restarting NCI/XF. If the problem persists contact your support office.
213	During initialization NCI/XF encountered an error during the loading of its subroutines. This is probably caused by an error during installation or may indicate that the NCI/XF region size is too small.	Check the installation procedure before restarting NCI/XF. If the problem persists increase the virtual storage size on the EXEC statement before restarting NCI/XF. If the problem persists contact your support office.
214	A CALL routine abended during a DIALOG function.	Check the parameters given to DIALOG MANAGER, and correct errors. If the problem persists contact your support office.
216	Corruption of the VTAM RPL USER FIELD has been found.	Any occurrence of this should be reported to your support office.
500	An error occurred during the ATTACH of the NCI/XF log processing subtask. This may be caused by an error during installation or may indicate that the NCI/XF region size is too small.	Check the installation procedure before restarting NCI/XF. If the problem persists increase the virtual storage size on the EXEC statement before restarting NCI/XF. If the problem persists contact your support office.
501	An error occurred during the initialization of the NCI/XF log processing subtask. This is probably caused by an error during initialization.	Check the installation procedure before restarting NCI/XF. If the problem persists increase the virtual storage size on the EXEC statement before restarting NCI/XF. If the problem persists contact your support office.

NC-EDIT information messages

If an invalid command is entered or an error condition occurs, NC-EDIT will overlay the column indicator (top right of the main edit panel) with an appropriate informational message.

This section lists all informational messages, describes why the message was issued and suggests corrective action.

Message	Help text
Abend *nnn*-rc	An operating system abend occurred during the processing of an NC-EDIT request. The most frequent cause is lack of space during a SAVE operation.
Allocation error	NC-EDIT could not dynamically allocate the requested dataset. Ensure that the dataset name and volume serial (if specified) were typed correctly and that NC-EDIT is authorized to access the requested dataset before you reissue the request.
Bottom of data	While processing a FIND or CHANGE command, NC-EDIT reached the end of the edit dataset without encountering the specified string. This message does not necessarily indicate an error condition. It may show that the requested string does not exist in the data. Press <F5> or <F6> as appropriate to reinvoke the FIND or CHANGE, searching from top of data.
Command conflict	Contradictory line commands have been entered in the line number area of the display, for instance a D2 command immediately followed by another line command. Remove the offending command or refer to *Chapter 9 - A guide to NC-EDIT* for further information.
Copy cancelled	A request to merge the contents of another dataset or member with the data being edited has been cancelled by the user.
Copy pending	A COPY command has been entered without identifying the position in the dataset where the copied data is to be put. Enter an A (after) or a B (before) line command and press <Enter> to proceed with the copy request.
Create pending	A CREATE command has been entered without identifying the data lines to be used to build the new dataset or member. Enter a C (copy) or M (move) line command and press <Enter> to continue the create operation.
Dataset copied	A message confirming that the requested copy merge operation has been successfully completed.
Dataset created	A message confirming that the requested create operation has been successfully completed.
Dataset in use	The dataset specified for an EDIT, COPY or REPLACE operation is in use by another person. Reissue the request later.
Dataset not PDS	During a COPY, CREATE or REPLACE operation, a member name was entered but the specified dataset is not a partitioned dataset. Correct the dataset name or remove the member name to continue with the requested operation.
Dataset not partitioned	During an edit operation, a member name was entered but the specified dataset is not a partitioned dataset. Correct the dataset name or remove the member name to continue with the requested operation.
Dataset replaced	A message confirming that the requested replace operation has been successfully completed.
Dataset saved	A save operation was requested by either the SAVE command or <F3> or <F4>. The updated contents of the sequential dataset have been written to disk.
Dataset too large to edit	The sequential dataset you are trying to edit is too large to fit into the maximum amount of virtual storage you are allowed to use. Your edit request is refused. The amount of storage available to a user is defined in variable *&usrelim* which is described in the NCI/XF Reference Manual, *Chapter 8 - NELOGON and NC-EDIT variables* and in the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*.

Message	Help text
Dataset truncated	The sequential dataset you are attempting to edit is too large to fit into the maximum amount of virtual storage you are allowed to use. NC-EDIT has loaded as much data as will fit within the allowable virtual storage. If the dataset is subsequently saved, the data NC-EDIT could not load will be lost. The amount of storage available to a user is defined in variable *&usrelim* which is described in the NCI/XF Reference Manual, *Chapter 8 - NELOGON and NC-EDIT variables* and in the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*.
Editor not licensed	The CODE = parameter specified when NCI/XF was started does not authorize the use of NC-EDIT, which is an optional extension to NCI/XF. Restart NCI/XF having specified an appropriate CODE= parameter or contact your local marketing representative for assistance.
Enter FIND string	A FIND command has been entered without specifying the character string to be searched for. Respecify the command or refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for further information.
Enter library name	An incomplete dataset name has been specified for an edit or copy merge operation. If no data is entered in the Dataset name field, NC-EDIT expects a fully qualified dataset name to be entered in the Project/Library/Type fields. Reissue the request having specified a complete dataset name.
Enter member name	The dataset specified for a COPY merge, CREATE or REPLACE operation is a partitioned dataset and no member name has been specified. Reissue the request having specified the name of the member to be processed.
Enter type qualifier	An incomplete dataset name has been specified for an edit or copy merge operation. If no data is entered in the Dataset name field, NC-EDIT expects a fully qualified dataset name to be entered in the Project/Library/Type fields. Reissue the request having specified a complete dataset name.
Inv. dataset name	During a COPY, CREATE or REPLACE operation, the dataset name specified is found to be invalid. Enter a valid dataset name to continue with the requested operation.
Inv. dataset org	During a COPY, CREATE or REPLACE operation, the organization of the dataset specified is not supported by NC-EDIT. Refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for details of supported dataset types.
Inv. member name	During a COPY, CREATE or REPLACE operation, the member name entered is invalid. Enter a valid member name to continue with the requested operation.
Inv. recfm	During a COPY, CREATE or REPLACE operation, the record format of the dataset specified is not supported by NC-EDIT. Refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for details of supported dataset types.
Inv. vol ser	During a COPY, CREATE or REPLACE operation, the volume serial entered was invalid. Respecify the data correctly to continue with the requested operation.
Invalid amount	An invalid scroll amount has been entered in the Scroll field or requested in the primary command area. Refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for details of valid scroll amounts.
Invalid command	Data has been entered in the primary command input area which NC-EDIT does not recognize as a valid command. Correct or remove the offending data or refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for further information.

Message	Help text
Invalid dataset name	During an edit operation, the dataset name specified is invalid. Enter a valid dataset name to continue with the requested operation.
Invalid dataset org	During an edit operation, the organization of the dataset specified is not supported by NC-EDIT. Refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for details of supported dataset types.
Invalid dataset recfm	The dataset you are attempting to edit is in a record format not supported by NC-EDIT. Refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for details of supported dataset types.
Invalid member name	During an edit operation, the member name entered is invalid. Enter a valid member name to continue with the requested operation.
Invalid operands	A command has been entered in the primary command input area with operands which are not valid for the specified command. Respecify the command correctly or refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for further information.
Invalid select code	An invalid character has been entered against a member name on the member select list. Members may be selected from the member select list by typing the character S against the required members(s). Remove any characters other than S and reissue the request.
Invalid volume serial	During an edit operation, the volume serial number entered was invalid. Respecify the data correctly to continue with the requested operation.
Member copied	A message confirming that the requested copy merge operation has been successfully completed.
Member created	A message confirming that the requested create operation has been successfully completed.
Member exists	The member name specified on a CREATE request already exists. The create request is abandoned. If required, the existing member may be overwritten with the REPLACE command.
Member in use	The member specified for an edit, copy or replace operation is in use by another person. Reissue the request later.
Member name too long	The member name specified on a SELECT or CREATE command exceeds the operating system imposed limit of eight characters. Reissue the command having specified a valid member name.
Member not found	The member specified on a copy merge operation does not exist. Reissue the request having specified a valid member name.
Member replaced	A message confirming that the requested replace operation has been successfully completed.
Member saved	A save operation was requested either by the SAVE command or use of <F3> or <F4>. The updated contents of the partitioned dataset member have been written to disk.
Member too large to edit	The partitioned dataset member you are attempting to edit is too large to fit into the maximum amount of virtual storage you are allowed to use. Your edit request is refused. The amount of virtual available to a user is defined in variable *&usrelim* which is described in the NCI/XF Reference Manual, *Chapter 8 - NELOGON and NC-EDIT variables* and in the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*.

Message	Help text
Member truncated	The member being edited is too large to be loaded in to the maximum amount of virtual storage you are allowed to use. NC-EDIT has loaded as much of the dataset as can be accommodated in your storage limit. Note that data will be truncated if it is saved. The maximum amount of virtual storage available to a user is controlled by NCI/XF variable *&usrelim*. Refer to the NCI/XF Reference Manual, *Chapter 8 - NELOGON and NC-EDIT variables*.
Mismatched brackets	The data entered contains an inconsistent number of brackets. Respecify the command or data, having stipulated the correct number of brackets.
Mismatched quotes	The data entered contains an inconsistent number of quotation marks. Respecify the command or data having specified the correct number of quotation marks.
No edit header	This is a **CRITICAL ERROR.** NC-EDIT cannot continue. NC-EDIT stores data essential to its successful operation in an NCI/XF local variable called *&usredit*. This variable has been reset or overwritten. If you have customized the supplied panel definitions, check that you have not accidentally RESET or modified variable *&usredit*, perhaps by using the RESET command with no operands. Otherwise contact your local support office.
No 'from' string	A CHANGE command has been entered with no operands. Respecify the command having included all required operands or refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for further information.
No member name	During a COPY, CREATE or REPLACE operation, the dataset name entered specifies a partitioned dataset but no member name. Enter the name of the required member to continue with the requested operation.
No 'to' string	A CHANGE command has been entered without specifying any replacement data. Respecify the command including all required operands or refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for further information.
Not enough space	A command has been issued to cause the amount of virtual storage in use to expand beyond the maximum amount you are allowed to use. The requested operation is abandoned at the point where the storage limit was reached. Refer to the NCI/XF Reference Manual, *Chapter 8 - NELOGON and NC-EDIT variables* and to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for details of specifying the maximum storage limit for a user.
Not on this line	A valid line command has been entered in the line number area of the screen. But the specified command is not allowed on the selected line, for instance a C (copy) command on the bottom of data line. Move the command to a permitted line or refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for further information.
Nothing saved	A save operation was requested by either the SAVE command or <F3> or <F4>. NC-EDIT discovered that the dataset or member being edited had not been amended and the save operation was bypassed.
No usredit link	This is a **CRITICAL ERROR.** NC-EDIT cannot continue. NC-EDIT stores data essential to its successful operation in an NCI/XF local variable called *&usredit*. This variable has been reset or overwritten. If you have customized the supplied panel definitions, check that you have not accidentally RESET or modified variable *&usredit*, perhaps by use of the RESET command with no operands. Otherwise contact your support office.
Open error	An error occurred during an attempt to open the requested dataset. Ensure that the dataset name and volume serial (if specified) were typed correctly and that NC-EDIT is authorized to access the requested dataset before reissuing the request.
Replace cancelled	A request to replace another dataset or member with data selected from the dataset being edited has been cancelled by the user.

Message	Help text
Replace pending	A command has been entered without identifying the data lines to be replaced. Enter a C (copy) or M (move) line command and press <Enter> to continue with the replace operation. Multiple lines may be selected by using multiple C or M commands, with or without a numeric suffix to select blocks of lines.
Saved but no room for stats	A request to SAVE, CREATE or REPLACE a member of a partitioned dataset has been successfully completed, but NC-EDIT had insufficient space to write directory statistics.
Saved but without stats	A request to SAVE, CREATE or REPLACE a member of a partitioned dataset has been successfully completed, but NC-EDIT had insufficient space to write directory statistics.
String changed	A CHANGE command has been processed and at least one occurrence of the specified string has been located and replaced with the specified data. If CHANGE ALL was stipulated, all lines changed have been marked with a == CHG > marker.
String found	In response to a FIND command, NC-EDIT has found a match for the search argument. The line of data which contains the string has been scrolled to the top of the display area.
String not found	In response to a FIND or CHANGE command, NC-EDIT has searched the data from top to bottom without encountering the search argument. The specified data does not exist within the dataset being processed.
Too few operands	A command has been entered in the primary command input area but a required operand has been omitted. Respecify the command having included the required operands or refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for further information.
Too many operands	A command has been entered in the primary command input area containing more than the permissible number of operands. Remove any unwanted operands or use quotation marks to delimit the operands.
Unable to alloc	During a COPY, CREATE or REPLACE operation, NC-EDIT could not dynamically allocate the specified dataset. Check that the name and volume serial (if any) were correctly specified and that NC-EDIT is authorized to access the specified dataset.
Unable to allocate dataset	NC-EDIT could not dynamically allocate the requested dataset. Ensure that the dataset name and volume serial (if specified) were typed correctly and that NC-EDIT is authorized to access the requested dataset before reissuing the request.
Unable to allocate member	NC-EDIT could not dynamically allocate the requested partitioned dataset member. Ensure that the dataset and member names and the volume serial (if any) were typed correctly and that NC-EDIT is authorized to access the requested dataset before reissuing the request.
Unable to open	During a COPY, CREATE or REPLACE operation, an error occurred when an attempt was made to open the requested dataset or member. Check that NC-EDIT is authorized to access the specified dataset.
Unable to open dataset	An error occurred during an attempt to open the requested sequential dataset for editing. Check that NC-EDIT is authorized to access the specified dataset.
Unable to open member	An error occurred during an attempt to open the requested partitioned dataset member for editing. Check that NC-EDIT is authorized to access the specified dataset.
Unknown command	Data has been entered in the line number area of the screen that NC-EDIT does not recognize as a valid line command. NC-EDIT will position the cursor against the offending data. Correct or remove the data in error or refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for further information.

Message	Help text
Unprintable characters	A COPY command has been issued to cause the contents of another dataset or member to be merged into the data being edited. However, the data being copied contains non-display (hex) data and you are not authorized to edit data containing such characters. The copy operation is abandoned at the point where the hex data was encountered. Refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for details of editing data which contains non-display characters.
Unprintable characters in dataset	You have tried to edit a sequential dataset which contains non-displayable (hex) data. If you are not authorized to edit such data, the edit request is refused. If you are authorized to access such data, the records containing the non-displayable characters are protected from access. Refer to the NCI/XF User Guide, *Chapter 9 - A guide to NC-EDIT*, for details of the authorization options available for such datasets.
Unprintable characters in member	You have tried to edit a partitioned dataset member which contains non-displayable (hex) data. If you are not authorized to edit such data, the edit request is refused. If you are authorized to access such data, the records containing the non-displayable characters are protected from access. Refer to the NCI/XF User guide, *Chapter 9 - A guide to NC-EDIT*, for details of the authorization options available for such datasets.

NC-MHO log messages

During the execution of NC-MHO, messages will occasionally be issued to the NCI/XF log. Each will be prefixed by the name of the routine which issued it.

ALL FUNCTIONS TERMINATED

Explanation: All protocol handlers have terminated; there is no MHO function active in the NCI/XF address space.

System action: No further processing is carried out.

User response: No action is required.

AN ISC ROUTINE HAS ABENDED S*xxx*

Explanation: An internal error has occurred.

System action: The protocol handler will terminate.

User response: Report the message to your local CKS support office.

ATTEMPTING SESSION WITHOUT COMPRESSION

Explanation: Issued by an LU62 based protocol handler during session start as result of an MHOLINK EXEC or similar when the originating node supports compression but the remote node does not.

System action: A further attempt is made to start the session without compression.

User response: No action required unless you require compression; in which case update your SOE EXEC *&mh_comp* variable, restart the protocol handler and reestablish the session.

ATTN(LOSS) DRIVEN FOR *aaaaaaaa/mmmmmmmm*

Explanation: Issued by an LU62 based protocol handler if a session is terminated by the remote partner normally or abnormally. *aaaaaaaa* is the name of the session partner, and *mmmmmmmm* is the mode name used.

System action: The session loss EXEC will be driven if defined.

User response: Determine the cause of the session termination; correct if necessary and re-establish the session if required.

C-APPCMD-ERROR

Explanation: An error has been detected during a session using the LU62 protocol. This message is part of a group which will also show the contents of the RPL and the RPL extension.

System action: The session is terminated.

User response: Report the message to your local CKS support office.

C-APPCCMD-NO-RETRY

Explanation: Issued by an LU62 based protocol handler if an APPCCMD macro fails with an unrecoverable error. A hexadecimal dump of the failing RPL and RPL extension will be displayed.

System action: The results depend on the macro involved and may be considered unpredictable.

User response: Report this message to your local CKS support office.

CINIT REJECTED FROM *aaaaaaaa*

Explanation: Issued by an VTAM based protocol handler if it receives CINIT data that it considers invalid. *aaaaaaaa* is the name of the application that was attempting contact with this handler.

System action: Processing continues; the remote partner EXEC will terminate with a non-zero return code.

User response: This message will be preceded by another message describing the exact reason for the reject but you will need to reproduce the problem with tracing active. This is done using the MHOTRACE TRACE EXEC.

CONTROL VALUE (RPL6REQ) ..... *xxxx*

Explanation: This message is issued with a group of other messages. Refer to the group of messages starting with message:

ERROR DURING LOGICAL PROCESSING; SESSION BEING TERMINATED

ERROR DURING LOGICAL PROCESSING; SESSION BEING TERMINATED
CONTROL VALUE (RPL6REQ) ..... *xxxx*
QUALIFY VALUE (RPL6QUAL) ..... *xxxx*
PRIMARY RETURN CODE (RPL6RCPR) .... *xxxx*
SECONDARY RETURN CODE (RPL6RCSC) .. *xxxx*
SENSE CODE (RPLSNSI) ..... *xxxxxxxx*

Explanation: These six messages are issued together. An error occurred during an LU6.2 session.

System action: The current session is terminated.

User response: Use the IBM 'VTAM LU6.2 Programming' manual appendices and the 'VTAM Messages and Codes' to interpret the RPL6-prefixed fields and the sense code. Correct and retry the session.

ERROR EXECUTING @MHWORK

Explanation: Issued by protocol handler subtask if the execution of the internal EXEC @MHWORK fails.

System action: The current unit of work is ignored; results are unpredictable.

User response: Report this message to your local CKS support office.

ERROR EXECUTING FINISH ROUTINE

Explanation: An error occurred executing the finish EXEC defined by the *&mh_fini* variable.

System action: The protocol handler terminates.

User response: Correct the value of *&mh_fini* or the statements within the EXEC, and retry.

ERROR EXECUTING START ROUTINE

Explanation: An error occurred executing the start EXEC defined by the *&mh_soe* variable.

System action: The protocol handler terminates.

User response: Correct the value of *&mh_soe* or the statements within the EXEC, and retry.

ERROR RUNNING PARM EXEC ROUTINE

Explanation: An error occurred during a protocol handler initialization routine.

System action: The protocol handler is terminated.

User response: Ensure that the EXEC defined by the *&mh_soe* variable contains statements with a valid syntax. Correct and retry.

EXTERNAL INTERRUPT DOES NOT ADDRESS AN SPL

Explanation: Issued by a sockets protocol handler interrupt exit if it receives a valid IUCV buffer but it does not address an internal control block correctly.

System action: The data is ignored; processing continues.

User response: Report this message to your local CKS support office.

I-APPCCMD-RESETRCV *** UNSUPPORTED***

Explanation: The LU62 APPCCMD MACRO completion routine has been driven for an invalid value in RPL6REQ.

System action: The data associated with this flow is ignored; processing continues.

User response: Report the message to your local CKS support office.

I-APPCCMD-SETSESS *** UNSUPPORTED***

Explanation: The LU62 APPCCMD MACRO completion routine has been driven for an invalid value in RPL6REQ.

System action: The data associated with this flow is ignored; processing continues.

User response: Report the message to your local CKS support office.

INITIALIZATION FAILED

Explanation: An error occurred during a protocol handler initialization routine.

System action: The protocol handler terminates.

User response: This message will be preceded by at least one other error message indicating more precisely what the error is. See that message.

INITIAL_MESSAGE NAME_FIELD INCORRECT

Explanation: Issued by a sockets protocol handler interrupt exit if it receives a connection request from a remote partner (as a result of an EXEC such as MHOLINK) but the node name on the EXEC does not match the name used to MHOOPEN the protocol handler.

System action: The connection request is rejected; the current EXEC will terminate with a non-zero return code.

User response: Use the MHOSIDE EXEC to define the name of the remote partner with a value that matches that used to MHOOPEN it. Reissue the MHOLINK EXEC.

INITIAL_MESSAGE_REPLY INVALID

Explanation: Issued by a sockets protocol handler interrupt exit if an invalid reply is received during the execution of a connection setup (from such as MHOLINK).

System action: The connection process is terminated; the current EXEC will receive a non-zero return code.

User response: Report this message to your local CKS support office.

INVALID APPCCMD ENTERED

Explanation: The LU62 APPCCMD MACRO completion routine has been driven for an unexpected value in RPL6REQ.

System action: The data associated with this flow is ignored; processing continues.

User response: Report the message to your local CKS support office.

INVALID HEXADECIMAL DIGITS IN SIGNAL DATA

Explanation: Issued by a protocol handler in response to sockets specific data from the MHOSOCK EXEC.

System action: The MHOSOCK operation is ignored and results are likely to be unpredictable.

User response: Report this message to your local CKS support office.

INVALID NOTIFY CONTROL VECTOR KEY - REPORT THIS MESSAGE

Explanation: Invalid notify status.

System action: The request unit is ignored.

User response: Report this message.

INVALID NUMBER OF PARAMETERS

Explanation: An MHSTATUS EXEC has been issued with a syntax error.

System action: The EXEC terminates with a non-zero return code.

User response: Correct and retry.

INVALID OR MISSING PARM EXEC ROUTINE NAME

Explanation: An error occurred during a protocol handler initialization routine.

System action: The protocol handler is terminated.

User response: Ensure that the *&mh_soe* variable is set correctly. Correct and retry.

INVALID PROTOCOL *xxxx* FOUND IN MH_ORDR FIELD

Explanation: An MHOSTART EXEC has been issued with a value other that LU0, LU62, or SOCK in the *&mh_ordr* variable.

System action: The EXEC terminates with a non-zero return code.

User response: Correct and retry.

INVALID RPL6REQ FIELD

Explanation: Issued by an LU62 based protocol handler if an RPL extension is received with an unsupported value in the RPL6REQ field.

System action: The data is ignored; processing continues.

User response: Report this message to your local CKS support office.

INVALID SIGNAL

Explanation: An internal error has occurred.

System action: The protocol handler will terminate.

User response: Report the message to your local CKS support office.

INVALID SIGNAL CODE 05

Explanation: Issued by a protocol handler in response to LU62 specific data from the MHOAPPC EXEC.

System action: The MHOAPPC operation is ignored and results are likely to be unpredictable.

User response: Report this message to your local CKS support office.

INVALID SOCKET FUNCTION CODE

Explanation: Issued by a protocol handler in response to sockets specific data from the MHOSOCK EXEC.

System action: The MHOSOCK operation is ignored and results are likely to be unpredictable.

User response: Report this message to your local CKS support office.

ISCMAIN SHUTTING DOWN

Explanation: A protocol handler is shutting down due to an error or shutdown request.

System action: Termination continues.

User response: No action required.

IUCV EXIT ALREADY DEFINED

Explanation: A sockets based protocol handler has failed in initialization due to an internal error.

System action: The protocol handler terminates.

User response: Report this message to your local CKS support office.

IUCV EXIT CLOSING

Explanation: The IUCV interface is closing in response to an NCI/XF SHUTDOWN command.

System action: Termination continues; all sockets based protocol handlers should be terminating as well.

User response: No action required.

IUCV EXIT DISABLED

Explanation: The IUCV interface has terminated in response to an NCI/XF SHUTDOWN command.

System action: No further processing using the IUCV environment will take place.

User response: No action required.

IUCV EXIT IS NOT INITIALIZED

Explanation: A sockets based protocol handler has failed in initialization due to an internal error.

System action: The protocol handler terminates.

User response: Report this message to your local CKS support office.

IUCV EXIT TRACE DISABLED

Explanation: Issued by the IUCV interface in response to an MHIUCV TRACE OFF command to confirm that the tracing of IUCV events has been discontinued.

System action: Processing continues.

User response: No action required.

IUCV EXIT TRACE ENABLED

Explanation: Issued by the IUCV interface in response to an MHIUCV TRACE ON command to confirm that the tracing of IUCV events has been started.

System action: Processing continues.

User response: No action required.

IUCVMCOM_CONNECT FAILED; RETURN CODE IS X'*xxxxxxxx*'

Explanation: A sockets based protocol handler has failed in initialization due to an error during connection to the TCP/IP address space.

System action: The protocol handler terminates.

User response: Report this message to your local CKS support office.

IUCVMCOM_SEND MACRO FAILED FOR *aaaaaaaa*; ERRNO=X'*xxxx*' FUNC=X'*xxxx*'

Explanation: An error has been detected during an IUCV based sockets operation.

System action: The session is terminated.

User response: The ERRNO may be analyzed using the IBM 'TCP/IP Programmers Reference' manual appendix. The FUNC is the first halfword of the IUCV TRGCLS field. For example, ERRNO=X'0031' for FUNC=X'0004' indicates that a CONNECT has failed because the connection partner cannot be found.

IUCVMCOM_SEND MACRO FAILED FOR *aaaaaaaa*; IPRCODE=X'*xx*' FUNC=X'*xxxx*'

Explanation: An error has been detected during an IUCV based sockets operation.

System action: The session is terminated.

User response: Report the message to your local CKS support office.

IUCVMCOM_SEND MACRO REJECTED FOR *aaaaaaaa*; RC=X'*xx*' REASON=X'*xx*' FUNC=X'*xxxx*'

Explanation: An error has been detected during an IUCV based sockets operation.

System action: The session is terminated.

User response: Report the message to your local CKS support office.

IUCVMINI_SET FAILED

Explanation: A sockets based protocol handler has failed in initialization due to an error in an IUCVMINI MACRO.

System action: The protocol handler terminates.

User response: Report this message to your local CKS support office.

LINK TO *aaaaaaaa* IS IN UNKNOWN STATE *x*

Explanation: During session establishment/termination an unexpected status was encountered.

System action: Processing for the current operation terminates.

User response: Report the message to your local CKS support office.

LOGIC ERROR

Explanation: An internal error has occurred.

System action: The protocol handler will terminate.

User response: Report the message to your local CKS support office.

LOSTERM DRIVEN (CODE *nnn*) FOR HOST *aaaaaaaa*

Explanation: Issued by a VTAM based protocol handler if a session with a remote partner is lost unexpectedly.

System action: The session is terminated; the user defined loss EXEC will be driven if defined.

User response: Determine the cause of the session termination; correct if necessary and re-establish the session if required.

MHOAPPC EXEC FAILED; RETURN CODE IS *n*

Explanation: An LU62 protocol handler is attempting to return a token but the transaction start request has failed.

System action: The current operation terminates.

User response: Report the message to your local CKS support office.

MHOOPEN EXEC NOT ISSUED FOR ID *xxxx*

Explanation: An error has occurred during MHOSTART processing.

System action: SOE processing terminates.

User response: Report the message to your local CKS support office.

MHOOPEN FAILED; RETURN CODE IS *n*

Explanation: An MHOSTART EXEC issued an MHOOPEN EXEC on behalf of the caller and this failed.

System action: The MHOSTART EXEC terminates with a non-zero return code.

User response: Report the message to your local CKS support office.

MHOSTART EXEC HAS NOT BEEN ISSUED

Explanation: An internal error occurred during MHO processing.

System action: The results of the current EXEC are unpredictable.

User response: Report the message to your local CKS support office.

NCDIALX_WAIT FAILED

Explanation: An internal error has occurred.

System action: The protocol handler will terminate.

User response: Report the message to your local CKS support office.

NO ACTIVE LINK FOR REMOTE NODE *bbbbbbb*

Explanation: An MHOAPPC or MHODLINK EXEC has been issued for a node name that is not in session.

System action: The EXEC is terminated with a non-zero return code.

User response: Ensure the node name passed to the EXEC is correct, and if necessary issued an MHOLINK EXEC first.

NO ENVIRONMENT RECORD FOUND FOR *aaaaaaa* DURING MHOSTART

Explanation: An MHOSTART EXEC has not been issued prior to an MHOOPEN EXEC.

System action: An MHOSTART EXEC is issued on behalf of the caller.

User response: No action required unless the MHOSTART fails, or sets default values for any of the SOE variables. In this case issue the MHOSTART EXEC and then retry the MHOOPEN EXEC.

NO LINKS AVAILABLE; ATTEMPTING CLEANUP

Explanation: An MHOLINK EXEC was issued but none of the available protocol handlers was able to successfully establish a session with the partner application.

System action: MHOLINK processing terminates.

User response: Ensure that the correct protocol handlers are available and active both locally and remotely. Correct and retry.

NO LLAPI COMPLETION EXEC SPECIFIED

Explanation: An MHOSTART EXEC has been issued with the *&mh_auto* variable set to N (for LLAPI) but no completion EXEC (*&mh_sock* or *&mh_appc*) has been specified.

System action: The EXEC terminates with a non-zero return code.

User response: Correct and retry.

NO MHOOPEN HAS BEEN DONE FOR TASK *aaaaaaaa*

Explanation: An internal error occurred during a protocol handler initialization routine.

System action: The protocol handler terminates.

User response: Report the message to your local CKS support office.

NO PROTOCOL SERVER FOUND FOR NODE *aaaaaaaa* DURING MHOOPEN

Explanation: An MHOLINK EXEC has been issued without a preceding MHOOPEN EXEC.

System action: An MHOOPEN EXEC will be performed if variable *&*mh_acb* is defined.

User response: No action required if the MHOOPEN completes successfully. If it does not then set the appropriate variables and issue MHOOPEN before retrying the MHOLINK EXEC.

NOT RUNNING IN HLAPI MODE (MH_AUTO=Y)

Explanation: An MHOEXEC/SET/RESLV/LOG EXEC has been issued for an environment started with *&mh_auto*=N, that is, LLAPI mode.

System action: The EXEC terminates with a non-zero return code.

User response: Ensure the *&mh_id* variable is set to the correct value and reissue the EXEC.

PRIMARY RETURN CODE (RPL6RCPR) .... *xxxx*

Explanation: This message is issued with a group of other messages. Refer to the group of messages starting with message:

ERROR DURING LOGICAL PROCESSING; SESSION BEING TERMINATED

QUALIFY VALUE (RPL6QUAL) ..... *xxxx*

Explanation: This message is issued with a group of other messages. Refer to the group of messages starting with message:

ERROR DURING LOGICAL PROCESSING; SESSION BEING TERMINATED

RECEIVE FAILED, SHUTTING DOWN

Explanation: An error occurred during an LU0 RECEIVE MACRO.

System action: The protocol handler terminates.

User response: Report the message to your local CKS support office.

REMOTE NODE NAME PARAMETER OMITTED

Explanation: The name of a remote partner has not been supplied to an MHO EXEC that requires one.

System action: The current EXEC completes with a non-zero return code.

User response: Retry specifying the remote node name.

REMOTE PROCEDURE CALL SERVER ENDED

Explanation: Issued by an RPC server to confirm shutdown is error free.

System action: The server terminates.

User response: No action required.

REMOTE PROCEDURE CALL SERVER STARTED

Explanation: Issued by an RPC server to confirm initialization is successful and incoming requests will now be accepted.

System action: Processing continues.

User response: No action required.

REMOTE PROCEDURE CALL SERVER TRACE ENDED

Explanation: Issued by an RPC server to confirm internal tracing is now inactive in response to an EXEC MHRPCSRV TRACE STOP command.

System action: Processing continues.

User response: No action required.

REMOTE PROCEDURE CALL SERVER TRACE STARTED

Explanation: Issued by an RPC server to confirm internal tracing is now active in response to an EXEC MHRPCSRV TRACE START command.

System action: Processing continues; all activities will be traced to the NCI/XF log.

User response: No action required.

REPLY EXEC *eeeeeeee* NOT FOUND FOR ID *bbbbbbbb*

Explanation: An MHOEXEC/LOG/RESLV/SET EXEC was issued specifying a reply EXEC which cannot be found.

System action: The MHOEXEC/LOG/RESLV/SET operation has completed but the results of the operation are unavailable to the caller of the EXEC.

User response: Ensure the EXEC name is specified correctly, and it is in the libraries associated with your PANEL and or PBLIB concatenation. Correct and retry.

REQUEST REJECTED; SESSION DOES NOT EXIST

Explanation: Issued by a protocol handler in response to LU62 specific data from the MHOAPPC EXEC.

System action: The MHOAPPC operation is ignored and results are likely to be unpredictable.

User response: Report this message to your local CKS support office.

RPC EXIT HAS ABENDED

Explanation: Issued by an RPC server if an ABEND is detected in the IUCV external interrupt exit code.

System action: The server terminates.

User response: Report this message to your local CKS support office.

RPC MAINLINE HAS ABENDED

Explanation: Issued by an RPC server if an ABEND is detected in mainline code.

System action: The server terminates.

User response: Report this message to your local CKS support office.

SCIP RPL (UNKNOWN)

Explanation: Issued by a VTAM based protocol handler if a session control request unit is received but unrecognized.

System action: The request unit is ignored.

User response: Report this message to your local CKS support office.

SCIUCV FAILED; RETURN CODE IS X'*xxxxxxxx*'

Explanation: A sockets based protocol handler has been started but has failed during initialization of the IUCV environment.

System action: The protocol handler is terminated.

User response: The return code will be one of the following:

00000004	Unable to locate VMCF; report this message to your local CKS support office.
00000008	SCIUCVT not run successfully; report this message to your local CKS support office.
0000000C	CONNECT failed; ensure the TCP/IP address space is active, and its name matches the value in the *&mh_jobn* variable. Correct and retry.
00000010	IUCV initialization was terminated during retry processing waiting for TCP/IP to come active probably due to an MHOCLOSE EXEC or a timeout condition during MHOOPEN processing. Retry when TCP/IP active.
00000014	CONNECT failed and no retry allowed. Ensure the TCP/IP address space is active and is specified correctly on the MHRPCLNT EXEC.
00000018	Too many paths; report this message to your local CKS support office.

SECONDARY RETURN CODE (RPL6RCSC) .. *xxxx*

Explanation: This message is issued with a group of other messages. Refer to the group of messages starting with message:

ERROR DURING LOGICAL PROCESSING; SESSION BEING TERMINATED

SEND COMPLETION EXEC *eeeeeeee* SET RETURN CODE *n*

Explanation: The user defined send completion EXEC (defined by *&mh_send* in the SOE) has set a non-zero return code.

System action: Processing will continue as determined by the statements in the user EXEC.

User response: No action required.

SEND FAILED, SHUTTING DOWN

Explanation: An error occurred during an LU0 SEND MACRO.

System action: The protocol handler terminates.

User response: Report the message to your local CKS support office.

SEND FAILED TERMINATING SESSION FOR *aaaaaaaa*

Explanation: Issued by an LU0 based protocol handler if a VTAM SEND macro fails.

System action: The session with *aaaaaaaa* is terminated; the loss EXEC will be executed if defined.

User response: Report this message to your local CKS support office.

SENSE CODE (RPLSNSI) ..... *xxxxxxxx*

Explanation: This message is issued with a group of other messages. Refer to the group of messages starting with message:

ERROR DURING LOGICAL PROCESSING; SESSION BEING TERMINATED

SERVER TASK *aaaaaaaa* ATTACH FOR SERVER TASK FAILED; RETURN CODE IS 14

Explanation: Initialization of a protocol handler has failed during an ATTACH MACRO.

System action: The protocol handler terminates.

User response: Report the message to your local CKS support office.

SERVER TASK *aaaaaaaa* ERROR DURING INITIALIZATION; RETURN CODE IS 10

Explanation: Initialization of a VTAM based protocol handler failed due to a non-zero return code in a SETLOGON MACRO.

System action: The protocol handler terminates.

User response: Report the message to your local CKS support office.

SERVER TASK *aaaaaaaa* FINISH EXEC NOT SPECIFIED; RETURN CODE IS 4

Explanation: Initialization of a protocol handler has been unable to determine the name of the EXEC to run a protocol handler termination.

System action: The protocol handler terminates.

User response: Correct and retry.

SERVER TASK *aaaaaaaa* INITIALIZATION COMPLETE

Explanation: Initialization of a protocol handler has completed without error.

System action: Processing continues.

User response: No action required.

SERVER TASK *aaaaaaaa* INITIALIZATION IN PROGRESS

Explanation: Issued by a protocol handler when an MHOOPEN EXEC has been received but before the operation has completed.

System action: Processing continues.

User response: No action required.

SERVER TASK *aaaaaaaa* INVALID MODE (MH_AUTO); RETURN CODE IS 13

Explanation: Initialization of a protocol handler has detected a value other than N or Y in the variable *&mh_auto*.

System action: The protocol handler terminates.

User response: Correct and retry.

SERVER TASK *aaaaaaaa* INVALID TRACE VALUE (MH_TRAC); RETURN CODE IS 6

Explanation: Initialization of a protocol handler has detected a value other than N or Y in the variable *&mh_trac*.

System action: The protocol handler terminates.

User response: Correct and retry.

SERVER TASK *aaaaaaaa* NODE (MH_ACBN) HAS NOT BEEN SPECIFIED; RETURN CODE IS 5

Explanation: Initialization of a protocol handler has been unable to determine the name of the node to use as an operand on the MHOOPEN EXEC.

System action: The protocol handler terminates.

User response: Correct and retry.

SERVER TASK *aaaaaaaa* SOCKET SERVER INITIALIZATION FAILED; RETURN CODE IS 16

Explanation: Initialization of a sockets protocol handler has failed.

System action: The protocol handler terminates.

User response: Ensure that the port number specified by the MHOSIDE EXEC is not in use elsewhere in the host. Correct and retry.

SERVER TASK *aaaaaaaa* TCP/IP FOR MVS IUCV INITIALIZATION FAILED; RETURN CODE IS 17

Explanation: Initialization of a sockets protocol handler has failed.

System action: The protocol handler terminates.

User response: Ensure that the TCP/IP address space is active, and its name matches that specified on the *&mh_jobn* variable. Correct and retry.

SERVER TASK *aaaaaaa* TERMINATION COMPLETE

Explanation: Issued during shutdown of a protocol handler to indicate that the shutdown process is complete.

System action: No further processing occurs for this handler.

User response: No action required.

SERVER TASK *aaaaaaaa* TERMINATION DUE TO ABEND; RETURN CODE IS *n*

Explanation: Issued during shutdown if the protocol handler is terminating due to an earlier ABEND.

System action: Termination continues.

User response: Report the message to your local CKS support office.

SERVER TASK *aaaaaaaa* TERMINATION DUE TO ENVIRONMENT BEING CLOSED; RETURN CODE IS *n*

Explanation: Issued during shutdown if the protocol handler is terminating due to VTAM ending/ABENDing, the VTAM application being inactivated (LU0/LU62) or a sever interrupt being received (TCP/IP).

System action: Termination continues.

User response: Restart the appropriate system tasks prior to restarting the NC-MHO protocol handlers.

SERVER TASK *aaaaaaaa* TERMINATION DUE TO INITIALIZATION ERROR; RETURN CODE IS *n*

Explanation: Issued during the shutdown of a protocol handler if an error occurs during initialization.

System action: Termination continues.

User response: The error during initialization will be described by the message(s) that precedes this one in the NCI/XF log. See the appropriate entry in this section.

SERVER TASK *aaaaaaaa* TERMINATION DUE TO UNKNOWN ERROR; RETURN CODE IS *n*

Explanation: Issued during the shutdown of a protocol handler when it completes with an unrecognized return code.

System action: Termination continues.

User response: Report the message to your local CKS support office.

SERVER TASK *aaaaaaaa* TERMINATION IN PROGRESS
TOTAL LINKS ..... *n*
TOTAL OUTBOUND TRANSFERS ..... *n*
TOTAL INBOUND TRANSFERS ..... *n*

Explanation: These four messages are issued together. A protocol handler is terminating. TOTAL LINKS ..... *n* indicates the highest number of sessions concurrently active for the handler.

System action: Processing continues.

User response: No action required.

SERVER TASK *aaaaaaaa* UNABLE TO LOAD MODULE *mmmmmmmm*;RETURN CODE IS 7

Explanation: Initialization of a protocol handler has failed during the loading of a product module.

System action: The protocol handler terminates.

User response: Report the message to your local CKS support office.

SERVER TASK *aaaaaaaa* UNABLE TO OPEN ACB FOR *bbbbbbbb* ;ACBERFLG=*xx*

Explanation: Initialization of a VTAM based protocol handler failed due to a non-zero return code in an OPEN_ACB MACRO.

System action: The protocol handler terminates.

User response: Use the IBM 'VTAM Programming' manual section on the OPEN MACRO to interpret the ACBERFLG field. Correct and retry.

SERVER TASK *aaaaaaaa* UNKNOWN ERROR DURING INITIALIZATION; RETURN CODE IS *n*

Explanation: A protocol handler has failed during initialization with an unrecognized return code.

System action: The protocol handler terminates.

User response: Report the message to your local CKS support office.

SERVER TASK aaaaaaaa UNKNOWN PROTOCOL (MH_PROT); RETURN CODE IS 11

Explanation: Initialization of a protocol handler has detected a value other than LU0, LU62 or SOCK in the variable *&mh_prot*.

System action: The protocol handler terminates.

User response: Correct and retry.

SERVER TASK *aaaaaaaa* VTAM DOES NOT SUPPORT LU6.2; RETURN CODE IS 12

Explanation: Initialization of a protocol handler with a value of LU62 for variable *&mh_prot* has been attempted in a VTAM environment which does not support the LU6.2 API.

System action: The protocol handler terminates.

User response: LU6.2 protocol handlers may only be run in MVS VTAM 3.2 or above.

SESSION ESTABLISHED FROM *aaaaaaaa* TO *bbbbbbbb*

Explanation: A valid session request has been received and the session has been successfully established.

System action: Processing continues.

User response: No action required.

SESSION ESTABLISHED WITH NODE *aaaaaaaa*

Explanation: Issued when a session with a remote partner has ended.

System action: Processing continues.

User response: No action required.

SESSION FROM *aaaaaaaa* TO *bbbbbbbb* NOW INACTIVE

Explanation: This message is issued with groups of other messages. Refer to the group of messages starting with either of the following two messages:

SESSION WITH *aaaaaaaa* HAS ENDED

TOTAL LINKS .....*n*

SESSION REJECTED BY PARTNER X00 EXEC

Explanation: Issued by an LU62 based protocol handler during session start as a result of an MHOLINK EXEC or similar when the destination node logon authorization EXEC (defined by the *&mh_x00* variable in the SOE) rejects the session request.

System action: The session is terminated; the user defined NSE EXEC will be driven if defined.

User response: Determine if this is a valid response from the remote partner. Correct and retry if necessary.

SESSION REJECTED; UNKNOWN SESSION PARTNER ID *xxxx*

Explanation: An internal error occurred during session establishment.

System action: The session for *aaaaaaaa* is rejected.

User response: Report the message to your local CKS support office.

SESSION REJECTED; UNKNOWN SESSION PARTNER NODE NAME *aaaaaaaa*

Explanation: An internal error occurred during session establishment.

System action: The session for *aaaaaaaa* is rejected.

User response: Report the message to your local CKS support office.

SESSION TERMINATED WITH NODE *aaaaaaaa*

Explanation: Issued when a session with a remote partner has ended.

System action: Processing continues.

User response: No action required.

SESSION WITH *aaaaaaaa* HAS ENDED
TOTAL OUTBOUND TRANSFERS ..... *n*
TOTAL INBOUND TRANSFERS ..... *n*
SESSION FROM *aaaaaaaa* TO *bbbbbbbb* NOW INACTIVE

Explanation: These four messages are issued together. A normal session termination has been received and processed successfully.

System action: Processing continues.

User response: No action required.

SETLOGON FAILURE

Explanation: A VTAM based protocol handler has failed in initialization due to a non-zero return code from a SETLOGON MACRO.

System action: The protocol handler terminates.

User response: Report this message to your local CKS support office.

SEVER RECEIVED; TEXT IS *text*

Explanation: Issued by a sockets based application if an IUCV sever dataflow is received. *text* is the text from the IUCV data buffer.

System action: The application terminates.

User response: Report this message to your local CKS support office unless you receive this message during TCP/IP shutdown in which case the message is to be expected.

SIGNAL DATA INVALID OR OMITTED

Explanation: Issued by a protocol handler in response to sockets specific data from the MHOSOCK EXEC.

System action: The MHOSOCK operation is ignored and results are likely to be unpredictable.

User response: Report this message to your local CKS support office.

SIGNAL FROM *aaaaaaaa* TO *bbbbbbbb* FAILED; RETURN CODE IS *n*

Explanation: An internal data flow has failed.

System action: The current operation is ignored.

User response: Report the message to your local CKS support office.

STACK BUFFER TOO SMALL

Explanation: Issued by a Sockets protocol handler interrupt exit if its internal buffer overflows without receiving a complete logical data record.

System action: Subsequent data is ignored but results are likely to be unpredictable.

User response: Report this message to your local CKS support office.

START EXIT NAME NOT SPECIFIED

Explanation: The *&mh_soe* variable has not been defined during a protocol handler startup.

System action: The protocol handler is terminated.

User response: Ensure that the *&iscstar* variable is set correctly. Correct and retry.

STOPPING DUE TO THE ABOVE ERRORS

Explanation: An error occurred during protocol handler initialization.

System action: The protocol handler terminates.

User response: This message will be preceded by at least one other message indicating the reason for failure more precisely; see that message.

TARGET NODE NOT FOUND

Explanation: Issued by a protocol handler in response to sockets specific data from the MHOSOCK EXEC.

System action: The MHOSOCK operation is ignored and results are likely to be unpredictable.

User response: Report this message to your local CKS support office.

TERMSESS REJECTED; SESSION DOES NOT EXIST

Explanation: Issued by a protocol handler in response to a session termination operation (such as MHODLINK) when an internal error occurs.

System action: Processing continues but the session end request will terminate with a non-zero return code indicating a time-out condition.

User response: Report this message to your local CKS support office.

THE FOLLOWING FIXES HAVE BEEN APPLIED

Explanation: This message is followed by the MHO fix matrix showing fixes which have just been applied.

System action: Processing continues

User response: No action required

TIMEOUT DURING MHOCLOSE PROCESSING

Explanation: An MHOCLOSE EXEC has been issued to terminate a protocol handler but termination has not completed within the required timeout interval.

System action: The protocol handler MAY terminate at some later time.

User response: Ensure that the timeout interval has been specified correctly otherwise report the message to your local CKS support office.

TIMEOUT DURING MHODLINK PROCESSING

Explanation: An MHODLINK EXEC has been issued to terminate a session with a remote partner node but the termination has not completed within the required timeout interval.

System action: The session MAY terminate at some later time.

User response: Ensure that the remote partner protocol handler is active and functioning normally, otherwise report this message to your local CKS support office.

TIMEOUT DURING MHOLINK PROCESSING TO *aaaaaaaa*

Explanation: An MHOLINK EXEC has been issued but has not completed within the time allowed.

System action: The MHOLINK EXEC terminates; the session start MAY complete at a later time.

User response: Ensure that the partner identified in the MHOLINK operation is active and available. Also check the timeout period has not been set too low. Retry the MHOLINK operation.

TIMEOUT DURING MHOOPEN EXEC PROCESSING

Explanation: An MHOOPEN EXEC has been issued but has not completed within the time allowed.

System action: The MHOOPEN EXEC terminates; protocol handler initialization MAY complete at a later time.

User response: Ensure the parameters specified for the MHOOPEN EXEC are correct and retry. Also check the timeout period has not been set too low.

TIMEOUT DURING MHOSTOP PROCESSING

Explanation: An MHOSTOP EXEC has been issued to terminate an environment but termination has not completed within the required timeout interval.

System action: The environment MAY terminate at some later time.

User response: Ensure that the timeout interval has been specified correctly, otherwise report the message to your local CKS support office.

TOTAL INBOUND TRANSFERS ..... *n*

Explanation: This message is issued with groups of other messages. Refer to the group of messages starting with either of the following two messages:

SESSION WITH *aaaaaaaa* HAS ENDED

SERVER TASK *aaaaaaaa* TERMINATION IN PROGRESS

TOTAL LINKS ..... *n*

Explanation: This message is issued with a group of other messages. Refer to the group of messages starting with the message:

SERVER TASK *aaaaaaaa* TERMINATION IN PROGRESS

TOTAL OUTBOUND TRANSFERS ..... *n*

Explanation: This message is issued with groups of other messages. Refer to the group of messages starting with either of the following two messages:

SESSION WITH *aaaaaaaa* HAS ENDED

SERVER TASK *aaaaaaaa* TERMINATION IN PROGRESS

TOTAL SHORT ON STORAGE (LU0) .... *n*

Explanation: Issued during the shutdown process for a LU0 protocol handler to indicate the number of VTAM operations that were retried due to a short-on-storage condition across the VTAM API.

System action: Processing continues.

User response: Whilst NC-MHO operation will continue a high non-zero value could indicate a potential performance problem.

TOTAL SHORT ON STORAGE (LU62) ... *n*

Explanation: Issued during the shutdown process for a LU62 protocol handler to indicate the number of VTAM operations that were retried due a short-on-storage condition across the VTAM API.

System action: Processing continues.

User response: Whilst NC-MHO operation will continue a high non-zero value could indicate a potential performance problem.

TRACE OPTION *xxxxxxx* FOR NODE *aaaaaaaa*

Explanation: An MHOTRACE EXEC has been issued for option *xxxxxxxx* and will be interpreted by the protocol handler for node *aaaaaaaa*. More than one of these messages will appear if there are multiple protocol handlers.

System action: Processing continues.

User response: No action required.

UNABLE TO ESTABLISH SESSION WITH HOST *aaaaaaaa*

Explanation: Issued by a protocol handler in response to a session start operation (such as MHOLINK) when an internal error occurs.

System action: Processing continues but the session start request will terminate with a non-zero return code indicating a time-out condition.

User response: Report this message to your local CKS support office.

UNABLE TO ESTABLISH SESSION WITH NODE *aaaaaaaa*

Explanation: Issued when a session initialization request fails.

System action: The current EXEC terminates with a non-zero return code.

User response: Use messages preceding this one in the NCI/XF log to determine the cause of failure. Correct and retry.

UNABLE TO INITIALIZE IUCV ENVIRONMENT; RETURN CODE IS *n*

Explanation: An MHOOPEN EXEC for a Sockets protocol handler has been issued and has failed.

System action: The MHOOPEN EXEC terminates.

User response: All return codes except 12 should be reported to your local CKS support office. Return code 12 indicates that a time-out condition occurred during the initialization of the IUCV environment. Ensure that you have started the TCP/IP address space and that the variable *&mh_jobn* contains the name of that address space.

UNABLE TO LOCATE EXEC *eeeeeeee*

Explanation: An MHOEXEC EXEC has been received by a remote partner but the EXEC name specified cannot be found.

System action: The MHOEXEC EXEC completes with a non-zero return code.

User response: Ensure the EXEC name is specified correctly, and that it is in the libraries associated with your PANEL and or PBLIB concatenation. Correct and retry.

UNABLE TO LOCATE VMCF SSCT

Explanation: A Sockets based protocol handler has been started but the SSCT cannot be found during initialization.

System action: The protocol handler is terminated.

User response: Ensure the system image has been configured (via the sub-system name table) to support the IUCV component of TCP/IP. Correct and retry.

UNABLE TO PROCESS *xxxxxxx* QUEUE; RETURN CODE IS *nn*

Explanation: An error occurred processing an internal queue.

System action: The current process(es) will fail; results may be unpredictable.

User response: Report the message to your local CKS support office.

UNABLE TO REGISTER RPC SERVICE

Explanation: Issued by an RPC server if the process of registering the program to PORTMAP fails due to an unexpected reply data format.

System action: The server terminates.

User response: Report this message to your local CKS support office.

UNABLE TO START MHO TASK FOR ID *xxxx* NODE *aaaaaaaa* ;RETURN CODE IS *n*

Explanation: An MHOOPEN EXEC has been issued but the SOE failed with return code *n*.

System action: The protocol handler for *aaaaaaaa* is terminated during initialization

User response: Report the message to your local CKS support office.

UNEXPECTED NOTIFY STATUS - REPORT THIS MESSAGE

Explanation: Issued by a VTAM based protocol handler if a session notification Request Unit is unrecognized for one of the following reasons:

- unknown RU received
- invalid notify control vector key

System action: The request unit is ignored.

User response: Report this message to your local CKS support office.

UNKNOWN ID *xxxx* ON *MH_LINKS QUEUE FOR *aaaaaaaa* AND USER *bbbbbbbb*

Explanation: A session establishment/establishment failure is in progress but the related user defined EXEC cannot be located.

System action: The user EXEC is not run.

User response: Report the message to your local CKS support office.

UNKNOWN PROTOCOL *xxxx* FOR NODE *aaaaaaaa*

Explanation: An internal error occurred during a session establishment or termination.

System action: The session may be left in an unpredictable state.

User response: Report the message to your local CKS support office.

UNKNOWN RU RECEIVED - REPORT THIS MESSAGE

Explanation: Issued by a VTAM based protocol handler if a session notification Request Unit is unrecognized for one of the following reasons:

- unknown RU received
- invalid notify control vector key

System action: The request unit is ignored.

User response: Report this message to your local CKS support office.

UNKNOWN SESSION END FROM *aaaaaaaa* TO *bbbbbbbb*

Explanation: A session termination request has been received from an unrecognized partner.

System action: The termination request is ignored.

User response: Report the message to your local CKS support office.

UNKNOWN SESSION ESTABLISHMENT FAILURE FROM *aaaaaaaa* TO *bbbbbbbb*

Explanation: A session establishment operation has completed unexpectedly.

System action: The session is ignored.

User response: Report the message to your local CKS support office.

UNKNOWN SESSION REQUEST FROM *aaaaaaaa* TO *bbbbbbbb*

Explanation: A session request has been received for an unrecognized partner.

System action: The session request is ignored.

User response: Report the message to your local CKS support office.

UNKNOWN TPEND CODE

Explanation: Issued by a VTAM based protocol handler if the application is to be ended due to a VTAM environment termination reason.

System action: The protocol handler terminates; any user defined protocol end EXEC will be executed.

User response: This message is to be expected during VTAM termination, or if the protocol handler application is deactivated for any reason. Correct and restart the protocol handler if necessary.

UNRECOGNIZED EXTERNAL INTERRUPT TYPE

Explanation: Issued by a Sockets protocol handler interrupt exit if it receives unrecognized data.

System action: The data is ignored; processing continues.

User response: Report this message to your local CKS support office.

UNRECOGNIZED IUCV FUNCTION CODE

Explanation: Issued by a Sockets protocol handler interrupt exit if it receives a valid IUCV buffer but it does not contain a supported TRGCLS value.

System action: The data is ignored; processing continues.

User response: Report this message to your local CKS support office.

UNSUPPORTED OPRCNTL QUALIFIER

Explanation: The LU62 APPCCMD MACRO completion routine has been driven for an invalid value in RPL6QUAL.

System action: The data associated with this flow is ignored; processing continues.

User response: Report the message to your local CKS support office.

UTILITY *eeeeeeee* FAILED; RETURN CODE IS n

Explanation: An internal utility program has completed with a non-zero return code.

System action: Processing for the current operation terminates.

User response: Report the message to your local CKS support office.

VTAM HALT CANCEL ISSUED

Explanation: Issued by a VTAM based protocol handler if the application is to be ended due to a VTAM environment termination reason.

System action: The protocol handler terminates; any user defined protocol end EXEC will be executed.

User response: This message is to be expected during VTAM termination, or if the protocol handler application is deactivated for any reason. Correct and restart the protocol handler if necessary.

VTAM HALT COMMAND ISSUED

Explanation: Issued by a VTAM based protocol handler if the application is to be ended due to a VTAM environment termination reason.

System action: The protocol handler terminates; any user defined protocol end EXEC will be executed.

User response: This message is to be expected during VTAM termination, or if the protocol handler application is deactivated for any reason. Correct and restart the protocol handler if necessary.

VTAM HALT QUICK OR V INACT Q OR F ISSUED

Explanation: Issued by a VTAM based protocol handler if the application is to be ended due to a VTAM environment termination reason.

System action: The protocol handler terminates; any user defined protocol end EXEC will be executed.

User response: This message is to be expected during VTAM termination, or if the protocol handler application is deactivated for any reason. Correct and restart the protocol handler if necessary.

WARNING OPRCNTL TO SELF IGNORED

Explanation: Issued by a protocol handler in response to a session start operation (such as MHOLINK) but the destination is the same as the origin.

System action: Processing continues but the session start request will terminate with a non-zero return code indicating a time-out condition.

User response: Report this message to your local CKS support office.

WARNING REJECT TO SELF IGNORED

Explanation: Issued by a protocol handler in response to a session termination operation (such as MHODLINK) but the destination is the same as the origin,

System action: Processing continues but the session end request will terminate with a non-zero return code indicating a time-out condition.

User response: Report this message to your local CKS support office.

WARNING REQSESS TO SELF IGNORED

Explanation: Issued by a protocol handler in response to a session start operation (such as MHOLINK) but the destination is the same as the origin.

System action: Processing continues but the session start request will terminate with a non-zero return code indicating a time-out condition.

User response: Report this message to your local CKS support office.

WARNING TERMSESS TO SELF IGNORED

Explanation: Issued by a protocol handler in response to a session termination operation (such as MHODLINK) but the destination is the same as the origin.

System action: Processing continues but the session end request will terminate with a non-zero return code indicating a time-out condition.

User response: Report this message to your local CKS support office.

NC-VSAM log messages

During the execution of NC-VSAM, messages will occasionally be issued to the NCI/XF log. Each will be prefixed by the name of the routine which issued it.

ABEND xxxxxx HAS OCCURRED IN IOVSAM

Explanation: An internal error has occurred.

System action: NC-VSAM will terminate.

User response: Report the message to your CKS support office.

ALREADY OPEN FOR INPUT, OPEN FAILED FOR FILE *xxxxxxxx*

Explanation: File *xxxxxxxx* already open for input.

System action: Open processing terminates.

User response: Ensure file is available for input processing.

ALREADY OPEN FOR OUTPUT, OPEN FAILED FOR FILE *xxxxxxxx*

Explanation: File *xxxxxxxx* already open for output.

System action: Open processing terminates.

User response: Ensure file is available for output processing.

CLOSE SUCCESSFUL FOR *xxxxxxxx*

Explanation: File *xxxxxxxx* close successful.

System action: Processing continues.

User response: No action required.

CLOSE SYNTAX ERROR FOR *xxxxxxxx*

Explanation: Possible internal error closing file *xxxxxxxx*.

System action: Close processing ends.

User response: Report the error to your CKS office.

CLOSE WARNING CODE *nnnn* REG15= *nnnn* FOR *xxxxxxxx*

Explanation: Error detected during close processing for file *xxxxxxxx*.

System action: Close processing continues.

User response: Review error message.

Read manual: Virtual Storage Access Method (VSAM) Programmer's Guide.

DD DUMMY NOT SUPPORTED. *xxxxxxxx*

Explanation: DD of dummy detected during OPEN processing.

System action: Open processing terminates.

User response: Correct and rerun.

DLBL RETURN CODE *xxx*

Explanation: GCS (VM only) follows fail message.

System action: Open processing terminates.

User response: Correct and rerun.

Read manual: VM System Messages and Codes.

DYNAMIC ALLOCATE FAILED FOR *xxxxxxxx*

Explanation: Dynamic allocate failed for file *xxxxxxxx*.

System action: Open processing terminates.

User response: Look for further messages which may follow. Correct and rerun.

DYNAMIC DE-ALLOCATE FAILED FOR *xxxxxxxx*

Explanation: An error was detected in close processing for file *xxxxxxxx*.

System action: Close processing ends.

User response: Review error following message.

ERROR UNLOADING MODULE *yyyyyyyy*. FILE *xxxxxxxx*

Explanation: Internal error detected at close time.

System action: Close processing ends.

User response: Report the error to your CKS office.

(IOCLOSE) FILE *xxxxxxxx* NOT CLOSED

Explanation: File *xxxxxxxx* has not been closed in the required time.

System action: The file may close some time later.

User response: If the problem persists, report to local CKS office.

(IOCLOSE) FREQUENCY *nn* INVALID, DEFAULTED TO 1/10TH SECOND

Explanation: Invalid time parameter on IOCLOSE.

System action: System default (1/10th sec) used.

User response: Correct for future use.

(IOCLOSE) MULTIPLIER *nnn* INVALID, DEFAULTED TO 600 TIMES

Explanation: Invalid repeat value coded on IOCLOSE request.

System action: System default 600 used.

User response: Correct for future use.

(IOVSAMT) UNABLE TO LOCATE SYSPARMS QUEUE ENTRY FOR *xxxxxxxx*

Explanation: Internal error.

System action: Close processing continues.

User response: Report the error to your CKS office.

NCVSAM INITIALIZATION FAILURE

Explanation: Incorrect parameter(s) supplied.

System action: Open processing stops.

User response: Review parameters and correct errors.

NCVSAM TERMINATION FAILURE

Explanation: Error found in termination EXEC.

System action: Termination processing stops.

User response: Report the error to your CKS office.

OPEN OF *xxxxxxxx* ABANDONED

Explanation: Open of file *xxxxxxxx* abandoned.

System action: Open processing terminates.

User response: If problem persists contact CKS.

OPEN OF *xxxxxxxx* HAS BEEN PENDING FOR *n* MINUTES

Explanation: IOVSAM has been trying for *n* minutes to open file *xxxxxxxx*.

System action: Allows 3 minutes before open is abandoned.

User response: Ensure file is available for opening.

OPEN SUCCESSFUL FOR *xxxxxxxx*

Explanation: File *xxxxxxxx* has been opened successfully.

System action: Open processing continues.

User response: No action required.

OPEN SYNTAX ERROR FOR *xxxxxxxx*

Explanation: Internal error during open of file *xxxxxxxx*.

System action: Open processing stops.

User response: Report the error to your CKS office.

OPEN WARNING CODE *yyyy* REG15= *zzzz* FOR *xxxxxxxx*

Explanation: A non-serious error was detected during open processing.

System action: Open processing continues.

User response: Review message.

Read manual: Virtual Storage Access Method (VSAM) Programmer's Guide.

PARAMETER ERROR SUPPLIED TO NCVSAM

Explanation: Bad parameter passed for IOVSAM initialization.

System action: Open processing terminates.

User response: Report the error to your CKS office.

RECORD '*xxxx*' IN USE BY ANOTHER RESOURCE FOR 30 SECONDS
RECORD '*xxxx*' NOT LOADED

Explanation: Internal error.

System action: Processing continues.

User response: Contact your local CKS office.

RECORD DEFINITION ALREADY LOADED *xxxx*

Explanation: Record definition already loaded for *xxxx*.

System action: Processing continues.

User response: No action required.

RECORD DEFINITION LOADED *xxxx*

Explanation: Record definition *xxxx* was successfully loaded.

System action: Processing continues.

User response: No action required.

REUSE OPTION NOT SUPPORTED FOR FILETYPE, FILE xxxxxxxx

Explanation: Reuse is only supported for ESDS.

System action: Open processing terminates.

User response: Correct and rerun.

SVC 99 INF.REASON CODE *yyyy*, ERROR CODE *xxx*

Explanation: This message follows a previous message informing why open failed.

System action: Processing terminates.

User response: Correct and rerun.

Read manual: MVS/ESA Authorized Assembler Programming Guide.

THE FOLLOWING FIXES HAVE BEEN APPLIED

Explanation: This message is followed by the NC-VSAM fix matrix showing which fixes have been applied.

System action: Processing continues.

User response: No action required.

UNABLE TO LOCATE FILE DESCRIPTION FOR xxxxxxxx - OPEN FAILED

Explanation: Unable to locate file description.

System action: Open processing terminates.

User response: Report the error to your CKS office.

UNABLE TO LOCATE SYSPARMS QUEUE ENTRY FOR *xxxxxxxx*

Explanation: Internal error.

System action: Processing terminates.

User response: If problem persists contact CKS.

VERSION *xx*

Explanation: This is the current version of NC-VSAM.

System action: Processing continues.

User response: No action required.

VSAM CLUSTER AND FILETYPE DIFFER FOR *xxxxxxxx*

Explanation: File type and VSAM cluster differ for file *xxxxxxxx*.

System action: Open processing terminates.

User response: Ensure file type and cluster are the same.

VSAM CLOSE ERROR CODE *yyyy* REG15= zzzz FOR *xxxxxxxx*

Explanation: A serious error was detected during close processing.

System action: Close processing ends.

User response: Review error message.

Read manual: Virtual Storage Access Method (VSAM) Programmer's Guide.

VSAM I/O ERROR - RECORD '*xxxx*' NOT DEFINED

System action: Internal error.

System action: Processing continues.

User response: Contact your local CKS office.

VSAM OPEN ERROR CODE *yyyy* REG15= *zzzz* FOR *xxxxxxxx*

Explanation: Error detected during open processing for file *xxxxxxxx*.

System action: Open processing terminates.

User response: Check error codes and correct.

Read manual: Virtual Storage Access Method (VSAM) Programmer's Guide.

NC-XMS log messages

During the execution of NC-XMS, messages will occasionally be issued to the NCI/XF log. Each will be prefixed by the name of the routine which issued it.

NC-XMS VERSION v.r.m. RELEASED xx xxx xx FOR YYY
THE FOLLOWING FIXES HAVE BEEN APPLIED

```
0 1 2 3 4 5 6 7 8 9          0 1 2 3 4 5 6 7 8 0
----------------------------------------------------
```

Explanation: An XMFIXES routine was started and the logged fixes have been applied.

System action: Processing continues.

User response: None.

XXXXXXX (LOG) FAILED. VARIABLES NOT SET

Explanation: An XMFIXES routine was started, but failed due to required variables not being set.

System action: No action taken.

User response: None.

XXXX NOT KNOWN TO BE ACTIVE SO STOP COMMAND NOT ISSUED

Explanation: An XMSTOP routine was issued to shutdown an NC-XMS which was not in the internal queues.

System action: No action taken.

User response: If required, restart the routine with the correct active NC-XMS name.

CURRENT XMS SYSTEM IS STILL XXXX

Explanation: An XMSYS was issued without an XMS name specified.

System action: The name of the current NC-XMS remains unchanged.

User response: None.

CURRENT XMS SYSTEM SET TO XXXX

Explanation: An XMSYS was issued to change the current NC-XMS name.

System action: The NC-XMS name is changed as required.

User response: None.

LAST XMS SYSTEM STARTED WAS XXXX

Explanation: An XMSYS routine was started with no operands.

System action: The current NC-XMS name is provided.

User response: None.

'NC-XMS ESTAE ENTERED FOR XMSYS=XXXX

Explanation: The named NC-XMS system has abended.

System action: The XMS will be marked inactive in SSCT.

User response: Restart the XMS system if required.

NC-XMS V1.3 - XM130 COMPILED ON XXXXX AT YYYYY

Explanation: This is an information message only issued at NC-XMS startup.

System action: Processing continues.

User response: None.

NEW NAME=XXXX

Explanation: The named XMS is being started for the first time and so required a new SSCT.

System action: Startup continues.

User response: None.

NO 2ND PARM ON STARTT - USING XM

Explanation: An internal error has occurred at startup due to no 2nd parameter on STARTT which is normally issued from within the supplied XMSTART PBLIB panel.

System action: XMS starts up with limited function.

User response: Restart job with correct PBLIB files available.

NO FREE SYSTEM LX INDEX - REUSE AN OLD XMS NAME

Explanation: All global system linkage numbers are in use. This can occur on MVS/370 where the limit is 10. XA and ESA have larger limits (for instance 55) so this should not occur.

System action: XMS does not start up.

User response: Restart XMS specifying an NC-XMS name that has previously been stopped (rather than invent a new name).

NOT AUTHORIZED - XMS ABORTED

Explanation: Unauthorized library on STEPLIB concatenation (MVS, not VM).

System action: NC-XMS startup is aborted.

User response: Restart job with all STEPLIB or JOBLIB libraries APF authorized.

OPTIONS HAVE BEEN SET BY XXXXX

Explanation: Information only; provides name of options EXEC.

System action: Processing continues.

User response: None.

OPTIONS NAME AND ONLY 5 PARAMETERS ARE SUPPORTED (NOT X)

Explanation: An XMSTART routine was started but too many parameters are supplied.

System action: XMS does not start up.

User response: Re-issue XMSTART command with less parameters.

PARAMETER NO. X INVALID

Explanation: An XMFIXES routine was started with an invalid parameter.

System action: Processing continues.

User response: A non-standard method of displaying the fix grid has been executed or an invalid parameter was supplied. Reissue with correct parameters.

PROGRAM TRACE LEVEL IS X

Explanation: Trace level for XMS programs has been set.

System action: Processing continues.

User response: None.

ROUTINE TRACE LEVEL IS X

Explanation: Trace level for XMS EXEC routines has been set.

System action: Processing continues.

User response: None.

RUNNING AS TASK XXXXXXXX - XMS SYSTEM YYYY

Explanation: XMSTATUS is attempting to start but is already started within this NCI/XF job.

System action: Additional XMS startup is aborted.

User response: None, or respecify correct second XMS name.

(SET) FAILED. RET. CODE WAS: X

Explanation: An XMFIXES routine failed.

System action: Processing continues.

User response: If necessary, call your support office if the problem persists. (It does not affect processing).

SHUTTING DOWN CROSS MEMORY SYSTEM XXXX TASK YYYYYYYY

Explanation: XMSTOP routine has been invoked to shut down named XMS.

System action: XMS shuts down after a 10 second quiescent period.

User response: None.

SSCT AT XXXXXXXX

Explanation: Issued during startup - giving address of SSCT for this XMS.

System action: Process continues.

User response: None.

STARTING XMS CROSS MEMORY SYSTEM

Explanation: An XMSTART routine was started.

System action: NC-XMS is starting up normally.

User response: None.

STATUS OF ALL XMS SYSTEMS RUNNING IN THIS NCI JOB

Explanation: An XMSTATUS routine has been started with no operands

System action: The status of all NC-XMS's in this job are being listed.

User response: None.

STOP XXXX ABANDONED AFTER 1 MINUTE

Explanation: XMSTOP failed to complete.

System action: Stop abandoned.

User response: Call your support office.

STOP FOR XXXX COMPLETED

Explanation: XMSTOP completed successfully.

System action: The requested NC-XMS has been made inactive.

User response: The stopped NC-XMS can be restarted in this NCI/XF job if required, or in another job after this one ends.

STOPT COMMAND DETECTED BY CROSS MEMORY SYSTEM XXXX

Explanation: The XMS main program acknowledged that it is to shut down, following an XMSTOP command.

System action: XMS main program shuts down after a 10 second quiescent period.

User response: None.

XMS CROSS MEMORY SYSTEM NAME IS XXXX

Explanation: The cross memory system name is logged whenever it is verified or altered by the options EXEC.

System action: None.

User response: None.

XMS EXIT 01 XXXX NOT FOUND - XXXX STARTUP ABORTED

Explanation: The requested exit *&*%xmx01* does not exist.

System action: XMS does not start up.

User response: Add named EXEC routine to PANEL ddname or remove name from OPTIONS EXEC.

XMS EXIT 02 XXXXXXXX NOT FOUND - YYYY STARTUP ABORTED

Explanation: The requested exit *&*%xmx0*2 does not exist.

System action: XMS does not start up.

User response: Add named EXEC routine to PANEL ddname or remove name from OPTIONS EXEC.

XMS EXIT 03 XXXXXXXX NOT FOUND - YYYY STARTUP ABORTED

Explanation: The requested exit *&*%xmx0*3 does not exist.

System action: XMS does not start up.

User response: Add named EXEC routine to PANEL ddname or remove name from OPTIONS EXEC.

XMS EXIT 04 XXXXXXXX NOT FOUND - YYYY STARTUP ABORTED

Explanation: The requested exit *&*%xmx0*4 does not exist.

System action: XMS does not start up.

User response: Add named EXEC routine to PANEL ddname or remove name from OPTIONS EXEC.

XMS LOG MESSAGES WILL BE LEFT ASIS

Explanation: XMTRACE A has been issued.

System action: XMS messages from EXEC routines will be mixed case.

User response: None.

XMS LOG MESSAGES WILL BE UPPERCASED

Explanation: XMTRACE U has been issued.

System action: XMS messages from EXEC routines will be uppercase.

User response: None.

XMS MAIN TASK PROGRAM TRACE OUTPUT WILL APPEAR ON THE VM CONSOLE

Explanation: Tracing has been set to use WTO.

System action: None.

User response: You will have been recommended to do this by support.

XMS OPTIONS XXXXXXXX. DISALLOWED STARTUP OF YYYY - Z

Explanation: During startup, the named OPTIONS EXEC disallowed the quoted XMS SSCT name.

System action: XMS does not start.

User response: Restart specifying an allowed XMS name, or a different OPTIONS EXEC.

XMS OPTIONS EXEC XXXXXXXX NOT FOUND - YYYY STARTUP ABORTED

Explanation: An XMSTART routine was started, but the OPTIONS EXEC *&%xmx00* could not be found.

System action: XMS does not start.

User response: Restart specifying an OPTIONS EXEC that exists, or correct the job.

XMS SYSTEM *XXXX* DID NOT RESPOND WITHIN Y SECONDS

Explanation: NC-XMS did not respond to enquiry from XMSTATUS within the given number of seconds. NC-XMS does exist but there is a problem.

System action: STATUS command abandoned.

User response: If the problem persists, contact your support office.

XMS SYSTEM XXXX RESPONDED AND IS TASK YYYYYYYY

Explanation: NC-XMS has responded successfully to enquiry from XMSTATUS during startup.

System action: Startup continues.

User response: None.

Useful manuals for reference in diagnosis of problems

Product	IBM Manual
TCP/IP for MVS 2.2.1	TCP/IP Programmer's Reference TCP/IP Messages and codes TCP/IP User's Guide
NC-VSAM / NC-SAM	Virtual Storage Access Method (VSAM) Programmer's Guide
MVS/ESA	Authorized Assembler Programming Guide
ACF VTAM	VTAM Programming VTAM Programming for LU6.2

This page intentionally left blank

Index

&, specifying character other than 1.209
&systupd, current setting of terminal update multiple 1.173
)BODY section
  &'s in, with ALIAS 1.10
  characters translated to attribute bytes 1.14
)INIT section
  modifying attributes 1.14
  previous panel, returning control to 1.155
)PANEL section
  &'s in, with ALIAS 1.10
  with ATTRIBUTE 1.14
)PROC section
  driving the)PROC section of next panel 1.18
  modifying attributes 1.14
)UPDATE section
  exiting from without redisplaying the panel 1.52
  interval before invocation
    (INTERVAL statement) 1.73
  modifying attributes 1.14
* Comment statement 1.5
@UTC2X 9.10
@UTC2X routine 9.9
@UTIBMC routine 9.10
@UTOFF routine 9.11
@UTROUTE routine 9.12
@UTSLOG routine 9.13
@UTSMF routine 9.14
@UTSORT routine 9.15
@UTSTATS routine 9.16
@UTX2C routine 9.17

A

A2E (NCI/XF language statement)
  examples 1.7
  format 1.6
ACQUIRE (NCI/XF language statement) 1.8
alarm 1.9, 1.19, 1.89
ALARM (NCI/XF language statement)
  disabling (NOALARM statement) 1.89
  example and format 1.9, 1.20, 1.21, 1.35, 1.37, 1.216
ALIAS (NCI/XF language statement)
  assigning &'s in)BODY an alias name 1.10
  example and format 1.10
alternative log, switching to (SWITCHLOG statement) 1.172
AND construction - see IF statement
APPCCMD completion exit (NC-MHO) 6.66
ASSIGN (NCI/XF language statement) 1.11
ATTR (NCDIALOG macro) 10.3
ATTRIBUTE (NCI/XF language statement)
  defining different settings for specific occurrences of a character 1.14
  examples 1.15
  extended data streams 1.16
  four color terminals 1.17
  modifying attribute bytes 1.17
  reset attributes 1.14
AUTOINPUT (NCI/XF language statement)
  &usraid setting 1.18
  animation on panels 1.18
  examples and format 1.18
  security character sequences at logon 1.18
  sounding terminal alarm before
    queued logon request 1.18
available variables (NC-MHO) 6.50

B

BEEP (NCI/XF language statement)
  alarm for next send 1.19
  disabling (NOBEEP) 1.90

C

C language programs
  accessing NCI/XF services with NCDIALC 9.10
  invoking with @UTIBMC 9.10
C2D (NCI/XF language statement) 1.20
C2X (NCI/XF language statement) 1.21
CALCULATE (NCI/XF language statement)
  embedded variables in 1.22
  example 1.22
  format 1.22
  return codes 1.22
calculator, floating point
  (FCALCULATE statement) 1.54
CALL (NCI/XF language statement)
  GLOBAL LOADLIB (VM) 1.23
  installation specific functions 1.23
  routines
    distributed 9.1
    NCDIALOG macros in 10.2
  STEPLIB library (MVS) 1.23
CHECKFIX (NCI/XF language statement) 1.24
CHECKPANEL (NCI/XF language statement)
  example 1.25
  format 1.25
  location of panels 1.25
  return code 1.25
CLSDST PASS
  RESTART to top level panel after (CONNECTR statement) 1.31
  waiting for notification of success (CONNECTR statement) 1.29
COLUMNS (NCI/XF language statement)
  example 1.26
  format 1.26
  panel width 1.26
  restrictions 1.26
comment statements, see *
compiled function execs 9.1

CONNECT (NCI/XF language statement) 1.27
&usrlogm and logmode 1.28
example 1.28
format 1.27
IBM default table 1.28
ISTINCLM 1.28
logmode
specifying logmode operand 1.28
table 1.28
terminal default 1.28
usrlogn 1.28
MODETAB and logmode table 1.28
NCI/XF
passing terminal to/from application 1.28
return codes 1.27
terminal alarm 1.18
terminal default logmode 1.28
CONNECTR (NCI/XF language statement)
contrasted with CONNECT 1.29
CSA usage 1.31
dial-up lines 1.31
example 1.30
format 1.29
non-logappl'd terminals 1.30
RESTART to a top level panel 1.31
return codes 1.29
return to USSMSG10 at logoff 1.30
terminals
returning to NCI/XF 1.30
SIMLOGONS 1.31
varied inactive while in session 1.30
converting one character in a string to another (TRANSLATE statement) 1.184
CPUTEST
(distributed non-compiled EXEC routine) 9.18
CSA usage (CONNECTR statement) 1.31
CURSOR (NC-DIALOG macro) 10.6
CURSOR (NCI/XF language statement)
examples 1.32
format 1.32
customization of NC-MHO 6.35

D

D2C (NCI/XF language statement) 1.35
D2P (NCI/XF language statement) 1.37
data
format for low level NC-MHO 6.83
format, checking (VERIFY statement) 1.211
line, writing to operator console (WTO statement) 1.215
PIP data (low level NC-MHO) 6.68
type, establishing (TYPE statement) 1.199
DECREMENT (NCI/XF language statement)
contrasted with CALCULATE 1.22
errors 1.38
example 1.38
format 1.38
reducing value of previously
defined variable by 1 1.38
used with INCREMENT statement 1.38
zero value variable 1.38
DECRYPT (NC-DIALOG macro) 10.8
DECRYPTION (NCI/XF language statement)
converting previously encrypted variables 1.39
example 1.39
exit (NC-MHO) 6.49
format 1.39
security 1.39
preventing exposures in event of a dump 1.39
systems 1.39
DEFINE (NCI/XF language statement)
a panel from storage 1.41
adding terminal definitions on-line 1.40
automatic update of TDT variables 1.40
dynamically updating the TDT 1.40
example 1.40, 1.41
format 1.40, 1.41
removing a terminal from the TDT 1.40
return codes 1.40, 1.41
storage recovery and user count 1.41
updating terminal definitions on-line 1.40
DELETE (NCI/XF language statement) 1.41
demonstration panels 9.6
DEQUEUE (NCI/XF language statement)
assigning control of serially
reusable resource to panel 1.42
example 1.42
format 1.42
DIRECT (NCI/XF language statement)
example 1.43
format 1.43
return codes 1.43
update of NCI/XF directory of users 1.43
directory of users
on-line update of (DIRECT statement) 1.43
DISABLE (NCI/XF language statement)
format 1.44
temporarily preventing terminal connect 1.44
terminal default in 8100 DPCX environment 1.44
disabling
the ALARM statement
(with the NOALARM statement) 1.89
displaying error messages (BEEP statement) 1.19
distributed call routines 9.1
distributed compiled function EXECS 9.1
dynamic modification of basic timer interval (UPDATE statement) 1.202
dynamic modification of update interval (TERMUPDATE statement) 1.173

E

E2A (NCI/XF language statement)
examples 1.46
format 1.45
ELSE (NCI/XF language statements)
example and format 1.47
ENABLE (NCI/XF language statement)
format 1.48
reconnecting disabled terminal 1.48
ENCRYPT (NCDIALOG macro) 10.9
ENCRYPTION (NCI/XF language statement)
example and format 1.49
exit (NC-MHO) 6.49
preventing security exposures 1.49
ENQUEUE (NCI/XF language statement)
example and format 1.50
return codes 1.50
ESTAE (NCDIALOG macro) 10.10

ESTAER (NCDIALOG macro) 10.12
EXEC (NCDIALOG macro) 10.13
EXEC exit points 6.36
EXECs for NC-MHO
    EXEC authorization 6.39
    log authorization 6.41
    logon authorization 6.37
    lost session 6.43
    MHOAPPC 6.57
    no session established (NSE) 6.44
    protocol end 6.45
    receive completion 6.47
    resolve authorization 6.40
    send completion 6.46
    session established 6.42
    set authorization 6.38
    subsystem end EXEC 6.48
EXECUTE (NCI/XF language statement) 1.51
    example and format 1.51
    minimizing error checking 1.51
    subroutines,invoking 1.51
    variables on entry 1.51
executing a variable routine
    (VEXECUTE statement) 1.213
EXIT (NCI/XF language statement)
    example
        in a routine 1.53
        in an)UPDATE section 1.52
    exiting from)UPDATE without
        redisplaying the panel 1.52
    returning control from a subroutine 1.52
exit points for NC-MHO 6.35
    encrypt/decrypt 6.49
    EXEC exit points 6.36
    user authorization 6.35
extended datastreams (ATTRIBUTE statement) 1.16

F

fast call (FCALL statement) 1.56
FCALCULATE (NCI/XF language statement)
    example 1.54
    floating point calculator 1.54
    floating point calculator (FCALCULATE statement)
        1.54
    format 1.54
    overflow 1.55
    return codes 1.54
    truncation 1.55
FCALL (NCI/XF language statement)
    contrasted with CALL 1.56
    example 1.56
    format 1.56
    return code 1.56
    rules for 1.56
FMH5 data 6.68
format of data, checking (VERIFY statement) 1.211
four-color terminals (ATTRIBUTE statement) 1.17

G

GETLOG (NCI/XF language statement) 1.57
    retrieving a line from the NCI/XF log buffer 1.57
GLOBAL LOADLIB (VM) (CALL statement) 1.23
GLOBALV (NCI/XF language statement) 1.58
GOTO (NCI/XF language statement)
    example 1.59
    format 1.59
    LABEL branching in the same panel section 1.59
    restrictions 1.60
GRESET (NCI/XF language statement)
    deleting sets of variables 1.61
    examples 1.61
    format 1.61

H

height of panel, setting (ROWS statement) 1.157
hexadecimal
    conversion 9.9, 9.17
    variables containing (&SYSXCnn) 2.8

I

IF (NCI/XF language statement)
    AND construction 1.64
    ASIS
        character comparates 1.68
        logic flow 1.68
        null 1.68
        numeric 1.68
    character operands 1.65
    default comparison type 1.65
    examples 1.70
    false conditions 1.63
    format 1.62
    null operands 1.65
    numeric operands 1.65, 1.65– 1.68
    PADNULL 1.66
    SHORT 1.67
    true conditions 1.63
increasing a previously defined variable by 1 1.71
INCREMENT (NCI/XF language statement)
    contrasted with CALCULATE 1.22
    example 1.71
    format 1.71
    increasing a previously defined variable by 1 1.71
    NCERR5 panel 1.72
    run-time error 1.72
INQUIRE APPSTAT request to VTAM (STATUS
    statement) 1.163
INTERVAL (NCI/XF language statement)
    defining timer units to be completed before update
        is invoked 1.73
    format 1.73
    restrictions 1.73
    TERMINAL-UPDATE value 1.73
    timer unit 1.73
    UPDATE-INTERVAL startup parameter 1.73
IOADD (NC-VSAM routine) 4.17
IOCLOSE (NC-VSAM routine) 4.29
IODELETE (NC-VSAM routine) 4.24
IOFIXES (NC-VSAM routine) 4.27
IOOPEN (NC-VSAM routine) 4.3
IOREAD (NC-VSAM routine) 4.5
IORESET (NC-VSAM routine) 4.32
IOSCLOSE (NC-SAM routines) 5.8
IOSOPEN (NC-SAM routine) 5.2
IOSREAD (NC-SAM) routine 5.4
IOSTATS (NC-VSAM routine) 4.30
IOSWRITE (NC-SAM routine) 5.6

L

LABEL (NCI/XF language statement)
- destination for GOTO 1.74
- example 1.74
- format 1.74

LENGTH (NCI/XF language statement)
- example 1.75
- format 1.75
- length of data 1.75

LOAD (NCI/XF language statement)
- @UTSTATS 1.76
- CALL-STATS=YES 1.76
- format 1.76
- loading copies of re-entrant modules into memory 1.76
- MODSTATS 1.76
- return codes 1.76
- statistics 1.76

load modules, user written, extracting statistics from (MODSTATS statement) 1.86

LOG (NCDIALOG macro) 10.14

LOG (NCI/XF language statement)
- blanks in message 1.78
- delimiters 1.78
- example 1.78
- format 1.77
- recommendations 1.80
- sending user defined messages to NCI/XF log dataset 1.77

log authorization EXEC 6.41

LOGLEVEL (NCI/XF language statement) 1.79

LOGOFF (NCI/XF language statement) 1.80

logon authorization EXEC 6.37

lost session EXEC 6.43

low level APPC environment (NC-MHO) 6.57

low level environment 6.57

low level Sockets environment (NC-MHO) 6.73

M

mathematical functions (CALCULATE) 1.22

MESSAGE (NCI/XF language statement)
- &usrmsg 1.81
- &usrmsgn 1.81
- examples 1.84
- format 1.81
- sending to several terminals 1.81

messages
- abend codes 11.26
- information messages 11.29
- log file messages 11.10
- NC-MHO log messages 11.36
- operating system messages 11.2
- user defined messages to NCI/XF log dataset (LOG statement) 1.77

MHOAPPC EXEC 6.57

MHOCLOSE (NC-MHO routine) 6.10

MHODLINK (NC-MHO routine) 6.11

MHOEXEC (NC-MHO routine) 6.24

MHOLINK (NC-MHO routine) 6.12

MHOLOG (NC-MHO routine)) 6.27

MHOOPEN (NC-MHO routine) 6.14

MHOREPLY (NC-MHO routine) 6.29

MHORESLV (NC-MHO routine) 6.31

MHOSET (NC-MHO routine) 6.33

MHOSIDE (NC-MHO routine) 6.16

MHOSOCK EXEC (NC-MHO) 6.73

MHOSTART (NC-MHO routine) 6.17

MHOSTOP (NC-MHO routine) 6.21

MHOTRACE (NC-MHO routine) 6.22

MHRPCLNT (RPC routine for NC-MHO) 6.86

MHRPCSRV (RPC routine for NC-MHO) 6.86

MHSTATUS (NC-MHO routine) 6.20

MODSTATS (NCI/XF language statement) 1.86
- &usrmods 1.86
- examples 1.88
- format 1.86
- mask characters 1.86
- return codes 1.86
- statistics, extracting 1.86

N

NCALERT 9.19

NCDIALC 9.22

NCDIALOG macros in (F)CALL routines
- ATTR
  - assigning default attributes 10.5
  - ATTR help panel 10.5
  - controlling field attributes 10.3
  - multiple attribute format 10.4
  - redefining attribute characters 10.5
  - single attribute format 10.3
- common NCDIALOG parameters 10.2
- CURSOR
  - invoking without operands 10.7
  - setting cursor position 10.6
- DECRYPT
  - resolving and decrypting variables 10.8
  - single list format 10.8
  - variable list format 10.8
- ENCRYPT
  - setting and encrypting variables 10.9
  - single variable format 10.9
  - variable list format 10.9
- ESTAE
  - cleanup functions 10.10
  - dump operands 10.11
  - format for dump processing 10.10
  - format for invoking a retry routine 10.10
  - preventing recursive abends 10.11
  - retry address facility 10.10
- ESTAER
  - determining value of ESTAE function 10.12
- EXEC
  - running NCI/XF EXEC routines 10.13
- LOG
  - writing a line to the NCI/XF log 10.14
- RESOLVE
  - alternative 10.20
  - examples 10.17
  - obtaining current variable settings 10.15
  - resolving a single variable 10.15, 10.20
  - resolving list of variables 10.16, 10.20
  - return codes 10.16
  - variable
    - current setting of 10.15
    - length 10.20
    - resolving 10.16, 10.20
    - wrongly specified 10.20
    - zero length 10.20

SET
resetting a variable 10.24
setting a list of variables 10.22
setting a single variable 10.21
XRESOLVE and XSET
example 10.27
NC-EDIT
invoking from an XF application 8.2
variables 8.6
NCENCDEC 9.24
NCERR1 panel (EXEC and PANEL statements) 1.97
NCERR1 panel (EXEC/PANEL statements) 1.51
NCERR2 panel (VEXECUTE statement) 1.213
NCERR4 panel (GOTO statement) 1.60
NCERR5 panel (INCREMENT and DECREMENT statement) 1.38, 1.72
NCERR6 panel (GOTO statement) 1.60
NCERR8 panel (CALL statement) 1.23
NCERR9 panel (DEFINE statement) 1.40
NCI/XF product panels 9.2
NC-MHO
customization 6.35
low level APPC environment 6.57
low level Sockets environment 6.73
MHOAPPC EXEC 6.57
MHOCLOSE 6.10
MHODLINK 6.11
MHOEXEC 6.24
MHOLINK 6.12
MHOLOG 6.27
MHOOPEN 6.14
MHOREPLY 6.29
MHORESLV 6.31
MHOSET 6.33
MHOSIDE 6.16
MHOSOCK EXEC 6.73
MHOSTART 6.17
MHOSTOP 6.21
MHOTRACE 6.22
MHSTATUS 6.20
RPC routines 6.86
MHRPCLNT 6.88
MHRPCSRV 6.86
Sockets environment 6.73
TCP/IP 6.73, 6.86, 6.88
variables 6.50
NCPEEK 9.26
NCQLOG 9.27
NCRACFX 9.28
NCRAND 9.30
NC-SAM
changing stem variables 5.4
examples 5.10
return codes 5.11
routines
IOSCLOSE 5.8
IOSOPEN 5.2
IOSREAD 5.4
IOSWRITE 5.6
technical notes 5.11
NCSLOG 9.31
NCVLOG 9.32
NC-VSAM
administration facilities 4.2
control file 4.2
routines
IOADD 4.17
adding a record to KSDS or ESDS files 4.17
entry variables 4.18
exit variables 4.19
return codes 4.20
IOCLOSE
closing a file 4.29
entry variables 4.29
exit 4.29
IODELETE
deleting record from KSDS cluster 4.24
entry variables 4.25
exit variables 4.25
return codes 4.26
IOFIXES
entry variables 4.3
exit variables 4.28
fix grid display 4.27
return codes 4.28
IOOPEN
exit variables 4.4
return codes 4.4
IOREAD
&¢selctn, specifying 4.7
AND and OR selection criteria 4.9
delimitation of variables 4.7
entry variables 4.14
ESDS and KSDS files 4.5
exit variables 4.15
RBA of last record returned 4.13
returning data 4.11
segmented reads 4.13
selecting records 4.6
IOSTATS
entry variables 4.30
exit variables 4.30
extracting VSAM statistics 4.30
IOUPDATE
&¢fields, update fields 4.21
entry variables 4.22
exit variables 4.22
return codes 4.23
VSLOGON 4.2
NCWAIT 9.33
NCWTOR 9.34
NC-XMS
common XMDIALOG parameters 7.18
reason codes (XMDIALOG) 7.26
re-entrancy (XMDIALOG) 7.22
register addressing 7.22
return codes from XMDIALOG macro 7.22
routines
modification of the cross memory system name 7.6

XMCLEAN 7.8
XMEXIT01 7.14
XMEXIT02 7.15
XMEXIT03 7.16
XMEXIT04 7.17
XMFIXES 7.9
XMOPTION 7.13
XMSTART 7.3
XMSTATUS 7.7
XMSTOP 7.5
XMSYS 7.6
XMTRACE 7.11
XMDIALOG macro options
CLOSE 7.28
EXEC 7.29
LOG 7.31
PURGE 7.33
RECEIVE 7.34
SET 7.37
NELOGON and NC-EDIT variables
invoking NC-EDIT 8.2
MVS NC-EDIT variables 8.6
NELOGON 8.2
VM NC-EDIT variables 8.10
NOALARM (NCI/XF language statement)
example 1.89
format 1.89
NOBEEP (NCI/XF language statement)
disabling terminal alarm for next send only 1.90
example 1.90
format 1.90
non-printable characters in a string
change to dots (TRANSLATE statement) 1.184
notational conventions 1.4

O

operator console, writing to (WTO statement) 1.215
OR (NCI/XF language statement) 1.91
format 1.91

P

PADCENTER (NCI/XF language statement)
centering error messages in general purpose panel 1.92
conservation of virtual storage 1.92
example 1.92
format 1.92
padding variable to a requested length and centering 1.92
truncation 1.92
PADLEFT (NCI/XF language statement)
conservation of virtual storage 1.93
example 1.93
fixed length data for queue elements 1.93
format 1.93
padding variable to requested length 1.93
right justification of data 1.93
truncation 1.93
PADRIGHT (NCI/XF language statement)
example 1.94
format 1.94
left justification of data 1.94
padding variable to requested length 1.94
truncation 1.94
panel
attribute characters (ATTRIBUTE statement) 1.14
deleting from NCI/XF storage (DELETE statement) 1.41
demonstration 9.6
establishing height of (ROWS statement) 1.157
lists 9.2
location (CHECKPANEL statement) 1.25
serially reusable resources (DEQUEUE and ENQUEUE) 1.42
suspending panel processing until POST term-id is issued (WAIT statement) 1.214
USE 9.3
PANEL (NCI/XF language statement)
examples 1.96
format 1.95
panel and return statements 1.96
passing from one to another 1.95
PANSTATS (NCI/XF language statement)
&usrpans 1.98
examples 1.100
format 1.98
return codes 1.98
statistics 1.98
PARSE (NCI/XF language statement)
&usrpm00 through &usrpm99 1.102
examples 1.102
format 1.102
passing a terminal from NCI/XF to an application (CONNECT statement) 1.27
POST (NCI/XF language statement)
format 1.104
terminal processing after wait 1.104
PREFRESH (NCI/XF language statement)
asynchronous requests 1.109
displaying a different panel 1.108
examples 1.106
format 1.105
mask characters 1.108
previous releases 1.109
queueing requests 1.109
redisplaying the same panel 1.108
refreshing several panels 1.108
target terminal status 1.109
TURN REFRESH 1.109
preventing a terminal from connecting to NCI/XF (DISABLE statement) 1.44
protocol end EXEC 6.45

Q

QUERYTERM (NCI/XF language statement)
QUEUE (NCI/XF language statement) 1.111
QUEUE ADD
defining a queue 1.114
examples 1.113
fixed length element 1.115
format 1.112
queue elements 1.115
return codes 1.113
variable length element 1.115
QUEUE BLDINDEX
creating an index to a key field 1.117
example 1.118
format 1.117
return codes 1.117

QUEUE COPY
copying from one queue to another 1.119
examples 1.121
format 1.119
return codes 1.120
QUEUE DELETE
examples 1.125
format 1.124
return codes 1.124
QUEUE FIND
and QUEUE BLDINDEX 1.126
and QUEUE-MASK startup parameter 1.133
examples 1.128
format 1.126
partitioning a queue into subqueues 1.136
result variables 1.133
return codes 1.127
QUEUE GFIND
and QUEUE BLDINDEX 1.138
and QUEUE-MASK startup parameter 1.139
example 1.140
format 1.138
result variables 1.142
return codes 1.139
QUEUE GPURGE
deleting entries from a generic queue 1.143
examples 1.144
format 1.143
mask characters 1.144
return codes 1.143
QUEUE PURGE
deleting entries from a queue 1.147
example 1.147
format 1.147
freeing all queue storage 1.147
return codes 1.147, 1.205
QUEUE REPLACE
avoiding QUEUE DELETE followed by QUEUE ADD 1.148
example 1.149
format 1.148
return codes 1.149
QUEUE SORT
example 1.150
format 1.150

R

receive completion EXEC 6.47
reconnecting disabled terminal (ENABLE statement) 1.48
redisplaying a panel (PANEL statement) 1.95
reducing value of variable by 1 (DECREMENT statement) 1.38
re-entrant modules (LOAD statement) 1.76
refreshing all terminals displaying specified panels (PREFRESH statement) 1.105
refreshing terminals with either current panel or another (TREFRESH statement) 1.185
Remote Procedural Call (RPC) routines in MHO 6.86
repeated error checking, minimizing (EXECUTE statement) 1.51
RESET (NCI/XF language statement)
deleting variables 1.151
examples 1.151
format 1.151
freeing variable storage 1.151
reset attributes (ATTRIBUTE statement) 1.14
RESETATTR (NCI/XF language statement) 1.152
RESETEXEC (NCI/XF language statement) 1.153
RESOLVE (NCDIALOG macro) 10.15
resolve authorization EXEC 6.40
resources, serially reusable (DEQUEUE and ENQUEUE statements) 1.42, 1.50
RESTART (NCI/XF language statement)
dynamic update to TDT with NCI/XF active 1.154
example 1.154
format 1.154
restrictions 1.154
returning to TDT defined primary panel 1.154
RETURN (NCI/XF language statement)
example 1.155
format 1.155
retention of reference to one previous panel 1.155
return control from subroutine (EXIT statement) 1.52
returning a terminal to its highest level panel (RESTART statement) 1.155
REVERSE (NCI/XF language statement)
examples 1.156
format 1.156
reversing order of characters in variable 1.156
ROWS (NCI/XF language statement)
establishing height of panel 1.157
example 1.157
format 1.157
restrictions 1.157

S

SCAN (NCI/XF language statement)
examples 1.158
format 1.158
return codes 1.158
searching line of text for character string 1.158
security
character sequences at logon (AUTOINPUT statement) 1.18
database, verifying userid and password against (VALIDATE statement) 1.206
exposures, preventing (DECRYPTION and ENCRYPTION) 1.39, 1.49
send completion EXEC 6.46
session established EXEC 6.42
session limits 6.68
SET (NCDIALOG macro) 10.21
SET (NCI/XF language statement)
examples 1.159
format 1.159
set authorization EXEC 6.38
setting maximum length of variable names (VARNAMEL statement) 1.210
SHUTDOWN (NCI/XF language statement) 1.161
example 1.161
format 1.161
SMF entry and @UTSMF 9.14
Sockets environment 6.73
statistics, extracting from user written load modules invoked by (F)CALL 1.86

STATUS (NCI/XF language statement)
)UPDATE section of startup panel 1.163
asynchronous operation 1.165
CONNECT 1.163
displaying a status message 1.163
examples 1.162
format 1.162
INQUIRE APPSTAT to VTAM 1.163, 1.165
PREFRESH 1.163
recommendations 1.163
result-var operand 1.165
return codes 1.162
sample panels 1.164
UPDATE-EXEC 1.163
status options, terminal, specifying (TURN statement) 1.196
STEPLIB library (MVS) (CALL statement) 1.23
storage, freeing variable (RESET statement) 1.151
STRING (NCI/XF language statement)
example 1.167
format 1.167
STRIPCENT (NCI/XF language statement)
examples 1.168
format 1.168
stripping characters from both sides of a variable 1.168
STRIPEMBED (NCI/XF language statement)
examples 1.169
format 1.169
stripping characters from within a variable 1.169
STRIPLEFT (NCI/XF language statement)
examples 1.170
format 1.170
stripping characters from left of a variable 1.170
STRIPRIGHT (NCI/XF language statement)
examples 1.171
format 1.171
stripping from right of a variable 1.171
subroutines
returning control from (EXIT statement) 1.52
standard,invoking to reduce error checking (EXECUTE statement) 1.51
subsystem end EXEC 6.48
suspending panel processing
(WAIT statement) 1.214
SWITCHLOG (NCI/XF language statement)
example 1.172
format 1.172
switching to alternative log 1.172
system variables
provided by NC-MHO 2.11
provided by NC-XMS 2.12
provided by USE 2.10
reserved 2.2

T

TCP/IP 6.73, 6.86, 6.88
terminal
alarm (ALARM and BEEP statements) 1.9, 1.19
disabling alarm (NOALARM statement) 1.89
definitions table (TDT)
dynamically updating (DEFINE statement) 1.40
returning terminal to primary panel (RESTART statement) 1.154
four color (ATTRIBUTE statement) 1.17
reconnecting DISABLEd terminal (ENABLE statement) 1.48
status options, specifying 1.196
temporarily preventing connection to NCI/XF (DISABLE statement) 1.44
TERMINAL-UPDATE startup option as modified by TERMUPDATE statement 1.173
TERMUPDATE (NCI/XF language statement)
dynamic modification of update interval 1.173
example 1.173
format 1.173
timer units, number before)UPDATE invoked (INTERVAL statement) 1.73
TRACE (NCI/XF language statement)
example 1.175
format 1.174
TRACEEXEC (NCI/XF language statement)
format 1.178
mask characters 1.178
TRACETERM (NCI/XF language statement)
&systrtr, setting of trace mask 1.181
examples 1.180
format 1.179
restricting trace to certain terminals 1.179
TRACEVAR (NCI/XF language statement)
format 1.182
TRACEVERB (NCI/XF language statement) 1.183
TRANSLATE (NCI/XF language statement) 1.184
changing non-print characters into dots 1.184
converting one character in a string to another 1.184
examples 1.184
format 1.184
TREFRESH (NCI/XF language statement) 1.185
asynchronous requests 1.189
displaying a different panel 1.188
examples 1.186
format 1.185
mask characters 1.188
previous releases 1.189
queueing 1.189
refreshing several terminals 1.188
target terminal status 1.189
TRMSTATS (NCI/XF language statement)
examples 1.191
mask characters 1.190
providing terminal statistics 1.190
statistics 1.192
TRUNCCENT (NCI/XF language statement)
examples 1.193
format 1.193
TRUNCLEFT (NCI/XF language statement) 1.194
examples 1.194
format 1.194
removing characters from the left of a string 1.194

TRUNCRIGHT (NCI/XF language statement)
examples 1.195
format 1.195
removing characters from right of a string 1.195
TURN (NCI/XF language statement) 1.196
format 1.196
specifying terminal status options 1.196
TYPE (NCI/XF language statement)
establishing type of data supplied 1.199
format 1.199
type of data, establishing (TYPE statement) 1.199

U

UNLOAD (NCI/XF language statement)
format 1.200
return codes 1.200
UNSTRING (NCI/XF language statement)
example 1.201
format 1.201
UPDATE (NCI/XF language statement)
&systint, indicating new update interval 1.203
format 1.202
UPPERCASE (NCI/XF language statement)
examples 1.204
format 1.204, 1.205
variable variables 1.204
uppercasing contents of a variable (UPPERCASE statement) 1.204
USE
system 2.10
user 3.7
USE panels 9.3
USE variables 2.10, 3.7
user authorization exit points (NC-MHO) 6.35
user variables, reserved 3.2
user-defined messages (LOG statement) 1.77
UVBL (NCI/XF language statement) 1.205

V

VALIDATE (NCI/XF language statement) 1.206
ACF2 return codes 1.207
example 1.207
format 1.206
modifying 1.208
non ACF2 return codes 1.206
register 15 1.206
variable
deleting (RESET statement) 1.151
executing a variable routine (VEXECUTE statement) 1.213
increasing value by 1 (INCREMENT statement) 1.71
maximum length of name, setting (VARNAMEL statement) 1.210
NC-EDIT 8.6, 8.10
NCI/XF defined system 2.1
NCI/XF defined user 3.1
NC-MHO 2.11
NC-XMS 2.12
padding to a requested length (PADCENTER, PADLEFT, PADRIGHT statements) 1.92, 1.93, 1.94
reducing the value of by 1 (DECREMENT statement) 1.38
setting (SET variable) 1.159
stripping characters from various parts of (STRIPCENT, STRIPEMBED, STRIPLEFT, STRIPRIGHT) 1.168, 1.169, 1.170, 1.171
system variables, reserved 2.2
VARIABLE (NCI/XF language statement) 1.209
ampersand (&)
as literal in)BODY 1.209
character other than 1.209
in)INIT,)UPDATE and)PROC 1.209
example 1.209
format 1.209
restrictions 1.209
VARNAMEL (NCI/XF language statement)
restrictions 1.210
setting maximum length of variable names 1.210
startup option VARNAMEL 1.210
VERIFY (NCI/XF language statement)
checking data is in correct format 1.211
default mask table 1.212
modifying 1.212
examples 1.211
order of verification 1.212
verifying userid and password against security database (VALIDATE statement) 1.206
VEXECUTE (NCI/XF language statement)
example 1.213
executing a variable routine 1.213
format 1.213
VSLOGON 4.2
VTAM definitions for NC-MHO routines 6.3
VTAM tracing routines (TRACE statement) 1.174
restricting to certain terminals (TRACETERM statement) 1.179

W

WAIT (NCI/XF language statement)
continue processing after WAIT (POST statement) 1.104
format 1.214
suspension of panel processing until POST issued 1.214
writing data lines to operator console (WTO statement) 1.215
WTO (NCI/XF language statement)
writing specified data line to operator console 1.215

X

X2C (NCI/XF language statement)
    example 1.216
    format 1.216
XMCLEAN (NC-XMS routine) 7.8
XMDIALOG functions (NC-XMS) 7.18
    CLOSE 7.28
    EXEC 7.29
    LOG 7.31
    OPEN 7.32
    PURGE 7.33
    RECEIVE 7.34
    RESOLVE 7.35
    SET 7.37
XMDIALOG macro options
    OPEN 7.32
XMEXIT01 7.14
XMEXIT02 7.15
XMEXIT03 7.16
XMEXIT04 7.17
XMFIXES 7.9
XMOPTION (NC-XMS routine) 7.13
XMSTART (NC-XMS routine) 7.3, 7.13
XMSTATUS (NC-XMS routine) 7.7
XMSTOP (NC-XMS routine) 7.5
XMSYS (NC-XMS routine) 7.6
XMTRACE (NC-XMS routine) 7.11
XRESOLVE (NCDIALOG macro) 10.26
XSET (NCDIALOG macro) 10.26

Reader's Comment Form

If you find any discrepancy in the information contained in this publication, please complete this form and mail it to the address below.

The authors may use, or distribute, any of the information you supply in any way they consider appropriate without incurring any obligation whatsoever.

Publication number NCR0001.002 - Second Edition (September 2001)

Please write your comments below and on the following page and return this form to the

Documentation Manager
PassGo Technologies Ltd.
Horton Manor
Ilminster, Somerset
TA19 9PY
England

NCI XF REFERENCE MANUAL

Reader's Comment Form

NCI XF REFERENCE MANUAL