SOFTWARE RELEASE BULLETIN
NOS/VE R1.1.2
LEVEL 630

NOS/VE and its product set are intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features and parameters.

Revision Date: 3/29/85

1.0 INTRODUCTION

1.0 INTRODUCTION

The NOS/VE R1.1.2 level 630 Software Release Bulletin (SRB) is to be used with the NOS/VE Installation and Upgrade Manual (Publication 60463913) for installing NOS/VE and its software products. Control Data recommends that the SRB be read in its entirety prior to software installation.

The SRB is the vehicle used to document any changes to the Installation and Upgrade Manual after it has gone to print. The SRB also documents a number of system deficiencies. It is necessary to install NOS 2.4.1 Level 630 or NOS/BE 1.5 level 627, Level 628 of the Common Products, and CIP 003 before installing NOS/VE R1.1.2.

The following warnings and restrictions are especially important to this release:

1. A Backup_Permanent_File operation must be done using the old system before upgrading to R1.1.2 and the files restored using level R1.1.2. See Section 3.1.2 for more information.

2. As of release 1.1.2, permanent file labels will be written in a new format which older systems will not be able to read. This presents no problem unless permanent files created on a R1.1.2 system or later must be used on a system older than R1.1.2. Should this situation arise, contact Central Software Support for guidance.

3. A new version (V1.1) of object code library format has been implemented to support improved message template capabilities. Message template modules are now created through the OCU CREATE_MESSAGE_MODULE subcommand which places message template modules on the object library. The OCU on the R1.1.2 system can accept object modules from both old (V1.0) and new (V1.1) libraries but will only generate new (V1.1) format libraries. Libraries created by the R1.1.2 OCU cannot be used by previous versions of the system or previous versions of OCU. Object modules (but not SCL procedures or program descriptions) can be moved from a V1.1 library to a V1.0 library as follows:

   a. Using the V1.1 CREOL, do the following:

```
CREOL
ADDM v11_library "version V1.1 library"
```

```
GENL object_file f=f "generates an object file"
```

b. Using the V1.0 CREOL, do the following:

```
CREOL
ADDM object_file "generated as above"
GENL v10_library "version V1.0 object library"
```

4. Previous versions of message template modules cannot be used by the R1.1.2 system. They must be rebuilt using the GENERATE_MESSAGE_TEMPLATE procedure. A MODULE statement must be placed at the beginning and a MODEND statement at the end of the message template definitions in order to provide the name for the message template module on the OCU library. The message template module should be placed on the same library as the program description and CYBIL modules for the application program.

5. A new version (V1.1) of source code library format has been implemented to support current and future performance improvements. It will be necessary to convert all older (pre R1.1.2) source libraries with a utility described in Section 5.5.

6. Changes to the program interface require the recompilation of all CYBIL modules. Modules which have not been recompiled may encounter 'declaration mismatch' errors when they are loaded. In addition, any modules bound with runtime libraries will have to be rebound.

7. Tape usage is restricted to permanent file backup/restore, and dump analyzer operations.

8. R1.1.2 changes the operator interface to NOS/VE from the combination of the K-display and MDD interface to a single CDC CC634B terminal connected via cable AV117A to the mainframe's two port multiplexor. The NOS/VE system console should be connected to port 0 of the two port multiplexor to allow RTA access via port 1.

   The new NOS/VE system console provides access to the operator facility, system core debugger, the refreshing displays under VEDISPLAY, and an operator actions display. This console allows use of full ASCII for all key-ins.

   A single CC634B can be used for both NOS and NOS/VE on the model 810/830 systems. However, only one operating system can use the

system console at a given instant. The F6 key must be used to switch ownership of the system console between NOS and NOS/VE.

For all other model 8xx mainframes supported by NOS/VE, the C170 state operating system will continue to use the CC545 console and NOS/VE will use the CC6348 as its system console.

In several sections of this document, PSR numbers are given following a problem description. Additional information about the problem may be obtained from SOLVER by using this number.

2.0 FEATURES AND PRODUCTS

2.0 FEATURES AND PRODUCTS

The products associated with NOS/VE R1.1.2 level 630 are documented in detail in the Software Availability Bulletin. Please refer to that document for this information.

3.0 INSTALLATION AND OPERATIONS NOTES

3.1 INSTALLATION

3.1.1 NOS INSTALLATION

1. Both PASSON and IRHF must run out of the SYSTEMX account in order to access NOS output queue files which are transmitted to the NOS/VE input queue. If your site has changed the password of the SYSTEMX user, then you must alter the USER statements in the RUNJOBS procedure. Enter the following statements under the user name from which NOS/VE is being deadstarted:

   ```
   COMMON,SYSTEM.
   GTR,SYSTEM,NVELIB,U.ULIB/NVELIB.
   GETPROC,RUNJOBS.
   *Change all USER statement passwords to your site's value
   FSE,RUNJOBS.
   *NVEffff.  adds DSTLIB to the library set
   REPPROC,RUNJOBS,L=DSTLIB.
   *Change DSTLIB from semi-private to private
   CHANGE,DSTLIB/CT=P.
   ```

2. In order to use the ST=NVE parameter to route a file to the input queue, the user must have CUST validation flag set.

3. The NOS system must have a LID table defined containing a PID of NVE with a LID of NVE. Neither entry has attributes.

4. A skeleton file (ACCOUNT) is created during deadstart and used by Interim Remote Host and Interstate Communications to generate partner jobs that are submitted to NOS. This file is created within a procedure named ACCFILE which is located in user library NVELIB. Currently, the ACCOUNT file created will contain the following:

   ```
   &JOB.
   USER,&USER,&PASSWORD.!
   CHARGE,&CHARGE,&PROJECT.
   /EOR
   ```

   When an Interstate Communications job (GETF, REPF, CREIC) is executed, the partner job is created using the ACCOUNT file to set

up NOS accounting for the NOS batch job. The field '&JOB' causes the job card that was created within Interstate Communications to be put in its place. The fields '&USER', '&PASSWORD', '&CHARGE', and '&PROJECT' are filled in with the accounting information entered by the user when executing a SET_LINK_ATTRIBUTES command. Interim Remote Host uses the ACCOUNT file in the same way.

If the charge and project numbers were not specified on the SET_LINK_ATTRIBUTES command the characters '*.' are substituted for '&CHARGE' and blanks for '&PROJECT'. This implies that default charge will be used by NOS and the user must have the correct charge and project numbers specified under MODVAL.

Changes can be made to the ACCOUNT file so that a site may create their own accounting file for Remote Host and Interstate Communications jobs.

a. Get the user library NVELIB from the system in order to change the ACCFILE procedure.

b. Extract the procedure ACCFILE from NVELIB.

```
GETPROC,ACCFILE,NVELIB.
```

c. Modify the procedure defined in this file by inserting the control cards necessary for accounting validation.

d. Place the procedure onto OSTLIB.

```
REPPROC,ACCFILE,OSTLIB.
```

3.2 R1.1.1 - R1.1.2 INSTALLATION AND UPGRADE DIFFERENCES

This section documents the differences between NOS/VE 1.1.1 and NOS/VE 1.1.2. It concentrates on the changes affecting those sites which tailored:

1. system start up commands
2. system shut down commands
3. command libraries
4. system prologs and epilogs

according to the NOS/VE 1.1.1 Installation and Upgrade Manual (60463913B). Reloading of permanent files at NOS/VE 1.1.2 is also discussed.

3.2.1 SYSTEM START UP COMMANDS

Previously you modified record STAUC in the NOS file NVELIB. At 1.1.2, you should place your system start up commands in the file:

$system.start_up_commands

This file is created only by an installation deadstart, with files being reloaded from the REQUIRED tape (i.e., the first time you install NOS /VE 1.1.2).

Examine the contents of $system.start_up_commands, and if you want to change the commands, simply overwrite this file with the commands you want executed. Possible commands include:

1. set_default_family
2. set_job_class_limits
3. set_command_list (for the operator command list)

You may also wish to execute certain commands based on the type of deadstart being performed. You can determine the type of deadstart by testing the value of the variable rav$load_option, which may have these values.

'INSTALL'	Installation deadstart was performed, the value of set_file_loaindg_options was set to install (the default value for an installation deadstart).
'RELOAD'	Installation deadstart, the value of set_file_loading_options was set to reload.
'NORMAL'	Continuation deadstart, the value of set_file_loading_options was set to normal (the default value for a continuation deadstart).
'UPGRADE'	Continuation deadstart, the value of set_file_loading_options was set to upgrade.
'INSTALL_DEFERRED_FILES'	Continuation deadstart, the value of set_file_loaindg_options was set to install_deferred_files.

3.2.2 SYSTEM SHUT DOWN COMMANDS

At NOS/VE 1.1.1, this file was record SHUDC on the NOS file NVELIB. The file

```
$system.shut_down_commands
```

is created the first time you install NOS/VE 1.1.2. Overwrite this file with any commands you wish to be executed during system shut down.

3.2.3 SITE COMMAND LIBRARY

At NOS/VE 1.1.1, the file

```
$system.osf$site_command_library
```

was created from record OSFSCL on the NOS file NVELIB. Any procedures you wished users to have as part of their command lists could be placed in this file, as it was added by the system prolog to users' command lists.

This file is now reserved for CDC, and the system is released such that this fle is no longer automatically added to users' command lists.

Procedures you want to make accessible to your users should be placed on another command library, and the system prolog should be modified appropriately (see below).

3.2.4 OPERATOR COMMAND LIBRARY

At NOS/VE 1.1.1, the file

```
$system.osf$operator_command_library
```

was created from record OSFOCL on the NOS file NVELIB. Any procedures you wished your operators to have as part of their command lists could be placed in this file, as it was part of the operator's command list.

This file is now reserved for CDC, but it is still in the operator's command list.

Procedures you want to make accessible to your operators should be placed on another command library, and the system start up commands (see above) should be modified appropriately to also add your operator command library to the operator's command list.

Should you have to reload permanent files, your operator command library will not be available until files are reloaded. This

should not be a problem, unless you want to use a restore procedure different from the one supplied by CDC. In this case, place your procedure on the NOS file SITECP (just like the CDC restore procedure) and rebuild your deadstart file with REPRECS. This procedure will then be made a permanent file under $SYSTEM for any installation deadstarts.

3.2.5 SYSTEM PROLOGS AND EPILOGS

At NOS/VE 1.1.1, the system prolog executed commands which added:

1. $system.scu.maintenance.command_library
2. $system.osf$site_command_library

The released system prolog no longer contains these commands. Also, the first library has been renamed to:

$system.scu.command_library

If you want to add these (or other) libraries to users' command lists, create a new cycle of the file

$system.prologs_and_epilogs.system_prolog

and include commands to add these libraries to the user's command list.

3.2.6 RELOADING FILES

Perform the following steps if you have to reload your permanent file base.

1. Execute an installation deadstart, but using a NVE procedure file which pauses for operator system core commands. Enter the system core commands:

   ```
   initdd ??????
   GO
   ```

   Note that the GO must be uppercase, and you do not enter the auto system core command.

2. Deadstart will eventually pause with the set file loading options display. Enter:

   ```
   set_file_loading_option load_option=reload
   GO
   ```

3. After deadstart completes, execute the permanent file reload procedure.

```
attach_file $system.restore
restore
```

Note that if you write your own permanent file reload procedure (see the Operator Command Library section) you would attach your procedure instead.

4. After files are reloaded, enter:

```
create_family nve 'administrator-user-name'
include_file $system.start_up_commands
```

3.0 INSTALLATION AND OPERATIONS NOTES
3.2.7 NOS/VE INSTALLATION

3.2.7 NOS/VE INSTALLATION

1. Table 1-1 and 2-2 of the NOS/VE Installation and Upgrade Manual should be expanded to include the larger memories supported by release 1.1.2. The expanded table is shown below:

CM SIZE	CM WORDS (OCTAL)	M WORDS (DECIMAL)	CM WORDS (HEX)	CM BYTES (DECIMAL)	CM BYTES (HEX)
1 MB	4000000	131072	20000	1048576	100000
2 MB	10000000	262144	40000	2097152	200000
3 MB	14000000	393216	60000	3145728	300000
4 MB	20000000	524288	80000	4194304	400000
5 MB	24000000	655360	0A0000	5242880	500000
6 MB	30000000	786432	0C0000	6291456	600000
7 MB	34000000	917504	0E0000	7340032	700000
8 MB	40000000	1048576	100000	8388608	800000
9 MB	44000000	1179648	120000	9437184	900000
10 MB	50000000	1310720	140000	10485760	0A00000
11 MB	54000000	1441792	160000	11534336	0B00000
12 MB	60000000	1572864	180000	12582912	0C00000
13 MB	64000000	1703936	1A0000	13631488	0D00000
14 MB	70000000	1835008	1C0000	14680064	0E00000
15 MB	74000000	1966080	1E0000	15728640	0F00000
16 MB	100000000	2097152	200000	16777216	1000000
17 MB	104000000	2228224	220000	17825792	1100000
18 MB	110000000	2359296	240000	18874368	1200000
19 MB	114000000	2490368	260000	19922944	1300000
20 MB	120000000	2621440	280000	20971520	1400000
21 MB	124000000	2752512	2A0000	22020096	1500000
22 MB	130000000	2883584	2C0000	23068672	1600000
23 MB	134000000	3014656	2E0000	24117248	1700000
24 MB	140000000	3145728	300000	25165824	1800000
25 MB	144000000	3276800	320000	26214400	1900000
26 MB	150000000	3407872	340000	27262976	1A00000
27 MB	154000000	3538944	360000	28311552	1B00000
28 MB	160000000	3670016	380000	29360128	1C00000
29 MB	164000000	3801088	3A0000	30408704	1D00000
30 MB	170000000	3932160	3C0000	31457280	1E00000
31 MB	174000000	4063232	3E0000	32505856	1F00000
32 MB	200000000	4194304	400000	33554432	2000000

2. When upgrading from an earlier NOS/VE release to NOS/VE Release 1.1.2 you must backup your existing NOS/VE permanent files, deadstart NOS/VE Release 1.1.2, and restore your NOS/VE permanent

files. System and product files contained on your permanent file backup tapes may have higher cycle numbers than those installed at NOS/VE Release 1.1.2. Restoring files into the $SYSTEM catalog from your existing permanent file backup tapes can result in a mixture of old and new installed versions, since the highest cycle is used by default as the installed version. If your permanent file backup tapes do not contain $SYSTEM catalog files, then follow the instructions outlined in chapter two of the Installation and Upgrade manual (publication number 60463913). The following steps describe how to upgrade to NOS/VE Release 1.1.2 from an earlier NOS/VE release, and how to reconcile differences between old and new $SYSTEM files:

a. Once you have generated the NOS/VE Release 1.1.2 Deadstart Input file as described in chapter two of the Installation and Upgrade manual, enter the following command from the operator console to terminate your current NOS/VE system:
TERMINATE_SYSTEM

b. Deadstart NOS 2.4.1 level 630 or NOS/BE 1.5 level 627 and NOS/VE Release 1.1.2. Be sure to use a deadstart procedure file for NOS/VE that enables operator intervention prior to system core command processing. Refer to the SETVE procedure documentation in the Installation and Upgrade manual when creating this deadstart procedure file.

c. When "ENTER SYSTEM CORE COMMANDS" appears on the operator console, enter:
INITDD <vsn> "where <vsn> identifies the deadstart device"
GO (GO must be upper case)

d. When the file loading option display appears on the operator console, enter:
SET_FILE_LOADING_OPTION RELOAD"
GO

e. When the message "DEADSTART COMPLETE" appears on the operator console, restore your permanent file base from the previous level of NOS/VE. Details of permanent file restoration may be found in the NOS/VE Operations Manual (60463914).

If the $SYSTEM files backed up from your prior NOS/VE release were created or modified by your site, they may be incompatible with NOS/VE Release 1.1.2. You should identify

which of these $SYSTEM files are to be retained from the prior release, and be prepared to verify these files once NOS/VE Release 1.1.2 has been installed.

f. Enter the Create_Family command for all families which you restored, excluding the :$SYSTEM family. Ignore the diagnostic issued by the Create_Family command which indicates that the family user administrator's master catalog already exists for the family you restored.

g. Enter the following command from the operator console:
TERMINATE_SYSTEM

h. Deadstart NOS/VE Release 1.1.2 a second time. Use the same deadstart procedure file that was used for the previous NOS/VE Release 1.1.2 deadstart.

i. When "ENTER SYSTEM CORE COMMANDS" appears on the operator console, enter:
GO

j. When the file loading option display appears, enter:
SET_FILE_LOADING_OPTION UPGRADE
GO

Follow the directions outlined in Chapter 8 of the NOS/VE Installation and Upgrade Manual (60463913).

k. To determine $SYSTEM catalog content, and the sizes of files installed in the $SYSTEM catalog, enter these commands from the operator console:
DISPLAY_CATALOG $SYSTEM DO=C O=ALL O=XX
PRINT_FILE XX

l. Login to NOS/VE Release 1.1.2 from an interactive terminal and enter the commands:
setcl a=($system.osf$site_command_library, $system.scu.command_library)
Verify_Installed_Software
One or more messages will be displayed for each product you have installed.

3. Remote installation and maintenance of the operating system requires an analyst or operator be present at the operator's console.

3.0 INSTALLATION AND OPERATIONS NOTES
3.2.7 NOS/VE INSTALLATION

4. If your site has the CYBER 170 APL2 installed, and desires to migrate APL applications to the CYBER 180, AFIFIX should be replaced under user number APLO. AFIFIX, a 170 resident permanent file, now has the added capability of being a pre-processor for the 180 APL workspace CONVERT_APL2. Installation instructions are described in this section; refer to the APL section in Product Set Notes and Cautions for usage information.

   The first step of installation is to create the following NOS/VE job to move the NOS/VE file to NOS. It may be created as a NOS or NOS/VE text file. This text file is then accessed from the operator console, using the attach_file or get_file command. The include_file command is executed to submit this job as a $system family batch job.

   (Note: User APLO has an installation definable password; the default, APLO, is being used for example. It is assumed that the CYBER 170 AFIFIX already properly exists in user APLO as an indirect and public file.)

```
JOB,MOVEAPL

WHEN any_fault DO
  display_value osv$status output=$response
  request_operator_action message=' AFIFIX MOVE FAILED '
  logout
WHENEND

set_link_attributes user=(APLO NOS) password=APLO
replace_file from=$system.aol.maintenance.afifix ..
  to=afifix dc=B60

JOBEND
```

5. If you have been installing and supporting NOS/VE systems prior to Release 1.1.2, then you may be aware of the steps required to upgrade to a newer level of NOS/VE (specifically template installation). This process has become obsolete with this system. NOS/VE now uses the system core and the job template from the deadstart media (NOS tape file or NOS disk file representing TPXXXK) as the source for any instance of NOS/VE deadstart. Consequently, the system core commands 'USESJT' and 'USERJT' (and their associated long forms) have been deleted from the system core command repertoire. Therefore, these commands need to be removed from a site's DCFILE.

This means that to perform an upgrade deadstart, one merely needs to use the new deadstart file and perform a continuation deadstart.

6. Interim Remote Host partner jobs are submitted as batch origin jobs with the advent of NOS 2.3. Problems with the commands GET_FILE and REPLACE_FILE occur when the NOS system batch job limit is set too low. This could cause GET_FILE and REPLACE_FILE to fail with the error message "UNABLE TO COMMUNICATE WITH NOS 170 REMOTE HOST". What happens is that when doing one of these commands, a partner job gets spun off and is submitted as a batch job to NOS. If there are enough batch jobs running to reach the batch job limit, the partner job will be put in the NOS input queue to wait for one of the batch jobs to finish. The 180 side waits for the partner job to finish. If the 180 side does not get a response from the partner job within 10 minutes, then it will time out and issue the error message above. To get around this problem, we recommend that the batch limit be set to 20 to 50 or more depending on the use of file transfer through Remote Host and Interstate Communication in your system. The maximum number of Remote Host and Interstate Communication jobs that can be running at one time is 50.

   NOTE: Interstate communication jobs do not time out on the 180 side. They will wait until the partner job gets rolled in and finishes before they continue one.

7. The NOS/VE Analysis Manual (60463915) contains discussion on the events that take place within NOS/VE for the different types of deadstarts performed by NOS/VE. One of the types of deadstarts is a recovery deadstart which is performed primarily after NOS/VE has been terminated by the TERMINATE_SYSTEM command. The purpose of the recovery deadstart is to assure that all mass storage transactions in central memory have been processed to update the mass storage device (i.e., central memory and mass storage agree on the state of a file).

   The NOS/VE Analysis Manual incorrectly implies that one enters the 'INITDD' command when the 'ENTER SYSTEM CORE COMMANDS' prompt appears during a recovery deadstart (if operator intervention is selected). Note that this is incorrect and will not be accepted by NOS/VE. An appropriate error message will be displayed ('INITDD NOT ALLOWED DURING RECOVERY') and NOS/VE then awaits any additional system core commands (i.e., 'AUTO').

3.2.8 NOS/BE INSTALLATION

1. To login to NOS/VE from an interactive terminal, you must first be validated to log in to NOS/BE INTERCOM 5.X. INTERCOM users are validated by use of the PASSWRD utility as described in the NOS/BE Installation Handbook. In order for an INTERCOM user to log in to NOS/VE, he must have a valid NOS/VE user name as one of his INTERCOM password parameters. The user must enter the following command after logging in to INTERCOM in order to log in to NOS/VE:

   ```
   veiaf
   ```

   The VEIAF utility is an INTERCOM multi user job which will take the NOS/VE user name for the current INTERCOM user from the Password file and initiate a login sequence with the NOS/VE Interactive Facility. After successful login, NOS/VE messages similar to the following will be displayed:

   ```
   Welcome to the NOS/VE Software System.
   Copyright Control Data 1984.
   C170-855 SN2.  NOS/VE R1.
   September 13, 1984.  1:34 PM.
   ```

   VEIAF will continue to function as the interface between the INTERCOM user and the NOS/VE Interactive Facility until the session is complete.

2. Interactive processing on NOS/VE in a NOS/BE dual-state environment is as described in the SCL for NOS/VE System Interface Manual with the exception of the interactive control of commands.

   NOS/VE recognizes both a pause break and a terminate break condition. INTERCOM 5.X recognizes only one break condition which is caused by entering '%A' at the terminal. Use of '%A' will cause the currently executing NOS/VE command to enter a pause break condition In order to effect a terminate break condition, the user must first enter '%A' to enter pause break followed by a TERMINATE_COMMAND(TERC) to terminate the suspended command.

3. The NOS/VE GET_FILE and REPLACE_FILE commands used to access 170-side permanent files have been modified to include NOS/BE cycles, passwords, and ID's. NOS/BE files accessed with these commands must be resident on the default permanent file set.

   The format of the GET_FILE command is as follows:

```
GET_FILE(GETF)
TO=file reference
[FROM=name]
[DATA_CONVERSION=keyword value]
[ID=name]
[PASSWORD=list 1..2 of name]
[CYCLE=integer 1..999]
[STATUS=status variable]
```

The FROM(F), ID(ID), PASSWORD(PW) and CYCLE(C) parameters specify the same information that would normally be supplied on a NOS/BE ATTACH command. That is, FROM specifies the file to be accessed, ID specifies the file id, PASSWORD specifies a list of up to two passwords that may be supplied in accessing the file (turnkey and read permissions are the only passwords that would be meaningful for a GET_FILE command), and CYCLE specifies the NOS/BE file cycle number. The system generates the NOS/BE ATTACH command using these values and, if access is granted, copies the file to a NOS/VE file.

The TO(T) parameter specifies an SCL file reference. If the FROM parameter is omitted, the file specified by the TO parameter is used (since the NOS/BE permanent file structure uses only one catalog, just the file name part of the file reference is used to access the NOS/BE file).

The ID parameter is not required if the NOS/BE file id is the same as the current NOS/VE user name, or user name specified in the last SET_LINK_ATTRIBUTE command. The PASSWORD parameters are required only if turnkey and/or read permissions are required to access the file. The CYCLE parameter is required only if the file being acquired is not the highest cycle cataloged.

The REPLACE_FILE command is used to copy a NOS/VE file to a NOS/BE file. The file on the NOS/BE system can be an existing or new permanent file.

If no NOS/BE file of the same name and id exists, a new file is created.

If a NOS/BE file of the same name and id exists, a new cycle of the file is created. If creation of a new cycle would result in more than 5 cycles, the lowest cycle of the file will be purged as part of the REPLACE_FILE operation.

The format of the REPLACE_FILE command is as follows.

```
REPLACE_FILE(REPF)
FROM=file reference
[TO=name]
[DATA_CONVERSION=keyword value]
[ID=name]
[TURNKEY=name]
[XR=name]
[CYCLE=integer 1..999]
[STATUS=status variable]
```

The TO(T), ID(ID), CYCLE(C), TURNKEY(TK) and XR parameters specify the same information that would normally be supplied on a NOS/BE CATALOG command. That is, TO specifies the file to be cataloged, ID specifies the file id, TURNKEY and XR(control, extend and modify) specify NOS/BE permanent file permissions, and CYCLE specifies a NOS/BE file cycle number. The system copies the file to the NOS/BE default permanent file set and generates a CATALOG macro to make the file permanent.

The FROM parameter specifies an SCL file reference. If the TO(T) parameter is omitted, the file specified by the FROM parameter is used and the file name part must conform to NOS/BE permanent file naming conventions, except the length will be limited to 31 characters.

The ID parameter is not required if the NOS/BE file id is the same as the current user name, or name specified in the last SET_LINK_ATTRIBUTE command. The CYCLE parameter is required only if a specific cycle of the file is to be replaced.

The TURNKEY parameter is required only if the user desires to prevent vent all access to the file unless the turnkey password is specified. The XR parameter is required if the user desires to allow read permission only unless the XR parameter is specified. If turnkey and/or XR passwords are specified when a NOS/BE file is initially created, they must be specified in all subsequent uses of REPLACE_FILE on this file.

4. A NOS/VE job can be submitted from NOS/BE by creating a NOS/BE file containing a NOS/VE job image and using the NOS/BE ROUTE command with a logical id of NVE to submit the job to the NOS/VE input queue. The format of the ROUTE command for this operation is as follows.

```
ROUTE(file,DC=IN,ST=NVE)
```

The NVE LID must not be defined for the host NOS/BE system or it will attempt to process the NOS/VE jobs as NOS/BE jobs.

5. NOS/BE support of the SET_TERMINAL_ATTRIBUTES command is as follows:

   a. The following parameters have fixed values that can not be changed:

ABORT_LINE_CHARACTER	%S
BACKSPACE_CHARACTER	BS
CANCEL_LINE_CHARACTER	CAN
CARRIAGE_RETURN_IDLE	0
ECHOPLEX	0
INPUT_DEVICE	keyboard
LINE_FEED_IDLE	0
NETWORK_CONTROL_CHARACTER	ESC
OUTPUT_DEVICE	display
OUTPUT_FLOW_CONTROL	none
OUTPUT_TRANSLATION	none
PAUSE_BREAK_CHARACTER	%A
TRANSPARENT_DELIM_SELECTION	(true,false,false)
TRANSPARENT_END_CHARACTER	CR
TRANSPARENT_CHARACTER_COUNT	2044

   b. The following parameters are set by INTERCOM according to the terminal class prior to logging in to NOS/VE, and cannot be changed:

      HOLD_PAGE
      PAGE_LENGTH
      PAGE_WIDTH
      TERMINAL_CLASS

   c. The PARITY parameter is based on terminal class, it may be changed by the NOS/VE user, but has no effect unless transparent mode has been selected. True transparent mode requires a parity setting of NONE, any other setting will force the default terminal parity to be used. In all cases, the parity bit is set to zero before input data is delivered.

   d. The TERMINATE_BREAK_CHARACTER is not supported.

e. TYPE_AHEAD is always true.

3.3 NOS/VE EXECUTIVE

1. NOS/VE will not support more than 10 tasks concurrently defined per job. The system does not enforce this limit; users must be careful not to exceed this limit or they may cause the system to hang.
   RSE NVEA133

3.4 OPERATOR_COMMANDS

1. Restarting IRHF and IFEXEC on the 180 Side

   IRHF moves NOS/VE input and output from and to NOS. It has a NOS partner job also called IRHF. IFEXEC is the NOS/VE interactive task. It has a NOS partner job called PASSON which is the NAM application VEIAF.

   IRHF should automatically restart if it aborts. To bring up IRHF manually, the operator must first bring down IRHF, then bring it up again. The same holds true for IFEXEC.

   To bring down IRHF, enter at the 180 NOS/VE operator's console:

```
DEACTIVATE_SYSTEM_TASKS TASK_NAMES=RHINPUT
DEACTIVATE_SYSTEM_TASKS TASK_NAMES=RHOUTPUT
```

   Then to bring IRHF back up, enter at the 180 NOS/VE operator's console:

```
ACTIVATE_SYSTEM_TASKS TASK_NAMES=RHINPUT
ACTIVATE_SYSTEM_TASKS TASK_NAMES=RHOUTPUT
```

   To bring down IFEXEC, enter at the 180 NOS/VE operator's console:

```
DEACTIVATE_SYSTEM_TASKS TASK_NAMES=IFEXEC
```

   Then to bring IFEXEC back up, enter at the 180 NOS/VE operator's console:

```
ACTIVATE_SYSTEM_TASKS TASK_NAMES=IFEXEC
```

2. Restarting IRHF and PASSON on the NOS Side

   To make provisions for restarting IRHF and PASSON, two procedures have been provided. These are MSSIRHF and MSSPASS and are located in User Index 377777.

   IRHF is restarted by simply typing the following in at the 170 operator's console:

```
MSSIRHF.
```

   PASSON is restarted by entering:

MSSPASS.

Note: Changes were made to procedure RUNJOBS to allow for PASSON to restart itself whenever it aborts abnormally. If PASSON aborts abnormally, the system will attempt to restart PASSON automatically. If NAM goes down, PASSON will not be automatically restarted.

3. The NOS application (VEIAF) that connects NOS/VE jobs to NAM will sometimes produce the flashing console message:

   "PASSON ABNORMAL 18"

   This message means PASSON is discarding an unsolicited message generated between VEIAF and the first read request from NOS/VE.

   This should be ignored. Clear the message with 'GO,jsn.'

   See Section 3.9.1 for PASSON diagnostic messages.

4. The operator should not enter any command at the operator's console that requires a magnetic tape. This will generally yield a deadlock condition. All requests should come from other than the system job. This will permit assignment of equipment from the operator's console.

5. Restarting IRHF on the NOS/BE side.

   To make provision for restarting IRHF, an operator initiated procedure has been provided. Simply type:

   nn.X RUNIRHF.

   where nn is a clear control point number.

6. PASSON Abnormal Processing (NOS/BE)

   During periods of heavy interactive usage, the message:

   PASSON ABNORMAL

   may appear on the NOS/BE job status (B) display. Unless related to a problem that a user has reported, this message does not necessarily indicate an error condition.

When PASSON ABNORMAL appears on the system console screen, record the number of the message for your site analyst's use.

If your analyst asks you to examine the dayfile of the PASSON job, enter the following command:

A=NN.

NN is the control point number of the PASSON job.

Examine the dayfile for the message. The message format is:

HEX DATA FOR PASSON CONDITION=MESSAGE NUMBER

The message is followed by one or more lines of hexadecimal data which your site analyst may ask you to record.

See Section 3.9.1 for PASSON diagnostic messages.

7. The INITIALIZE_DEADSTART_DEVICE system core command and the INITIALIZE_MS_VOLUME Logical Configuration Utility subcommand do not warn the operator that the volume has been previously initialized. This is of particular concern in INITIALIZE_MS_VOLUME because the operator may misspell the element_name of the device and inadvertently destroy permanent files.

8. In order to freeze the display on the NOS/VE operator's console while it is rapidly updating, we suggest using the 'SETUP' key. Pressing the 'SETUP' key will cause the console to stop accepting data from the host which will immediately halt the display. The lower two lines will be temporarily overwritten with the setup option display. Pressing the 'F1' key will allow the display to resume updating.

9. If the operator accidentally assigns a VE tape to the wrong VE job, then no indication of this situation is given and the tape remains attached to the (wrong) job until it ends.
PSR NVOE854

10. There are two operator commands available for manual job swapping. These are
SWAPOUT <JSN> and
SWAPIN <JSN>
<JSN> is the system job name of the job to be swapped.

If the job is to be swapped in, using the DISSJ command will provide the JSN necessary for the SWAPIN command. When a job is swapped out using the SWAPOUT command, it will remain swapped out until the SWAPIN command is executed.

11. When an operator makes an error on an ASSIGN_DEVICE (ASSD) command, the tape request message will be replaced by an error message. The operator should acknowledge the error by entering the REPLY_ACTION (REPA) command with the appropriate job name. The error message will be cleared, and the original tape request message will reappear, allowing the ASSIGN_DEVICE to be retried.

3.5 CONFIGURATION MANAGEMENT

1. The physical and logical configuration utilities can only be invoked by name from the system console. All other users must use the EXECUTE_TASK command as shown here:

   EXECUTE_TASK STARTING_PROCEDURE = MANAGE_PHYSICAL_CONFIGURATION

2. The name ALL is a reserved element name and should not be used in any element definitions.

3. When running the Physical Configuration Utility (PCU) from an interactive terminal, the include_file command will return an error stating that the included file is not callable. In order to test the configuration, enter the PCU, then copy the test file to another local file. You now can include_file the new local file.

4. Serial number uniqueness is now enforced by the Physical Configuration Utility. When defining elements in the configuration prolog, care must be taken to specify a unique serial number for each element. The only exception to this is for multi-spindle devices such as 885-1x.

5. For 830 dual CPU installation to run NOS/VE dual CPU, when "ENTER SYSTEM CORE COMMANDS" appears on the NOS/VE operator console, enter the following:

   SETSA ENABLE_MULTIPROCESSING 1

   This command is temporary and will be removed in 1.1.3.

6. NOS/VE cannot run on CPU 1 of a dual CPU configuration if CPU 0 is off or down.

7. The following error messages are not included in the diagnostic manual for R1.1.2:

   --ERROR--THE ACTIVE LOGICAL CONFIGURATION IS EMPTY.

   Product Identifier: CM condition: 350410
   Condition Identifier: CME$LCM_EMPTY_LC

   Description: The logical configuration was defined before the physical configuration in the configuration prolog.

User Action: Define the physical configuration in the configuration prolog before you define the logical configuration.

--ERROR--INCOMPATIBLE LOGICAL CONFIGURATION FILE.

Product Identifier: CM condition: 350405
Condition Identifier: CME$LCM_INCOMPATIBLE_LC

Description: Encountered an internal file problem.

User Action: Reexecute the logical configuration utility. If the error reoccurs, execute both the physical and logical configuration utilities to define the physical and logical configurations. If the error persists, contact Control Data field support

--ERROR--PERMISSION NOT GRANTED FOR {text}.

Product Identifier: CM condition: 350430
Condition Identifier: CME$LCM_RING_VALIDATION_ERROR

Description: User does not have ring privilage to execute the program interfaces or command.

User Action: Request the appropriate ring privilege from the family administrator.

--ERROR--{text} MISSING IN PHYSICAL ADDRESS SPECIFIER SET.

Product Identifier: CM condition: 350435
Condition Identifier: CME$LCM_MISSING_PA_SET_MEMBER

Description: Missing required set value in the physical address specifier set.

User Action: Specify the appropriate set value in the physical specifier, i.e. to query controller name, you need to specify cmc$channel and cmc$channel_address on the physical address specifier set.

--ERROR--{text}

Product Identifier: CM condition: 350455
Condition Identifier: CME$LCM_NOT_AVAILABLE

Description: User is trying to use a feature not yet implemented.

User Action: Contact Control Data Central Software Support.

3.6 DEVICE MANAGEMENT/RECOVERY

1. If any disk volume being used by NOS/VE becomes full, the following message will periodically appear on the console:

   AAAAAA - out of space (date) (time)
   (AAAAAA is the vsn of the volume)

   Any task that is requesting space on a full volume will hang waiting for space. Some space may be obtained by asking users (if they are able) to delete permanent files and detach local files. If the disk full condition persists, the NOS/VE system should be taken down and brought back up. This action will release most of the temporary file space that was in use. If the disk full was caused by permanent files, then the disk will be nearly full after the recovery, and the archiving of permanent files or deleting of some files must be done as soon as the system is up.

   It is possible that a disk full situation will occur that cannot be recovered. This will have happened if the "out of space" message appears during a deadstart before the system is up (this usually will occur during the permanent file reorganization phase of deadstart -- recognized by the "pf recovery" message at the console). In this case permanent file volumes must be initialized and reloaded from a previous backup dump.

2. If a continuation deadstart fails after disk full with a NOS/VE CPU monitor fault, it is possible that the failure is due to unprinted files in the output queue, one or more of which are no longer printable. The failure is a job mode software failure and does not leave the Monitor Control Register set in monitor mode. The message indicating this failure is "HR - MONITOR FAULT".

   Remedial action:

   Perform a continuation deadstart (with W=TRUE so you can enter system core commands) and enter the following system core command:

   SET_SYSTEM_ATTRIBUTE HALTRING 0

This command will allow deadstart to complete. If you find that task RHOUTPUT has terminated (by looking at the system job log with DISSL), and files exist in the job output queue (DISC $SYSTEM.$JOB_OUTPUT_QUEUE) you can be reasonably sure that you have encountered this problem.

Delete the contents of the job output queue (DELCC $SYSTEM.$JOB_OUTPUT_QUEUE), terminate NOS/VE, and do another continuation deadstart.

3.7 PERMANENT_FILE_UTILITIES

1. The power of the family administrator has been extended to allow backing up and deleting files for users in his family.

3.8 COMMAND_LANGUAGE_STATISTIC

The data that gets emitted for this statistic is the time and page faults for the task that calls the command. If the command spins off a task, then the data for that task is found under the task statistics and the command statistic will only contain the time and page faults for the initiating task.

Statistic Name	Description
CIL7002	Command Resources. This statistic will collect and generate data for every command executed. It should only be used for testing or performance information gathering. The information generated by this statistic includes: DESCRIPTIVE DATA - command name COUNTER 1 - job mode CP time COUNTER 2 - monitor mode CP time COUNTER 3 - total page faults COUNTER 4 - total page-ins COUNTER 5 - total page-reclaims COUNTER 6 - total page-assigns

Due to the large overhead incurred by using this statistic, it is conditional and must be activated in the following manner:

FROM THE CONSOLE ENTER:

```
SETSA COMMAND_STATISTICS_ENABLED TRUE
ACTS CL170002
```

To deactivate the statistic, use the opposite of the above sequence.

```
DEAS CL170002
SETSA COMMAND_STATISTICS_ENABLED FALSE
```

3.9 ONLINE_MANUALS

1. The installation and upgrade of online manuals is documented in the NOS/VE Installation and Upgrade Usage Manual for Release 1.1.2. The maintenance process is described in this section.

   The source for online manuals is being distributed to allow customers the capability of tailoring the manuals for their site. Source libraries contain a SCU feature reflecting the revision level documentation changes (example: feature VER_1_1_1 for Release 1.1.1) This not only gives the benefit of a "revision packet" to identify new capabilities and changes, but also aids the update process for the site version of manuals.

   To tailor a manual for a site:

   a. Create a new source library or additional cycle of $system.manuals.maintenance.source_library reflecting the site modifications. (Note that each manual is contained in a deck. Decks are grouped by product.)

   b. Execute $system.manuals.maintenance.binding_procedure within the $system user or maintenance user name to create new manuals.

      ```
      PROC bind_manuals, binm, bind_manual
        manual, manuals, m: LIST OF NAME or KEY all = all
        source_catalog, sc: FILE = $SYSTEM.MANUALS.MAINTENANCE
        catalog, c: FILE = $USER.MANUALS
        status)
      ```

      where,

manual = scu deckname(s) of manual(s) desired to be created
source_catalog contains the source libraries used for compilation
catalog = resultant catalog for new manual(s)

3.10 ERROR_MESSAGES

The following diagnostic messages have not been included in NOS/VE manuals for R1.1.2. They will be included in a future release.

3.10.1 PASSON DIAGNOSTIC MESSAGES

There are 28 distinct errors that PASSON indicates by issuing a 'PASSON ABNORMAL' message at its control point. The cause of each error and its associated error code (issued in the 'PASSON ABNORMAL' message) are listed below. The error explanations assume a knowledge of the network concepts evinced in the NAM/CCP Reference Manual (publication number 60499500).

1. PASSON ABNORMAL 0

   PASSON's attempt to sign on to the memory link failed with a fatal error status. PASSON terminates.

2. PASSON ABNORMAL 1

   The memory link reported a fatal error status when PASSON attempted an mip$add_sender to permit itself to receive messages from any NOS/VE application. PASSON terminates.

3. PASSON ABNORMAL 2

   PASSON does not begin interactive processing until NOS/VE's Interactive Facility's executive signals it to do so. In this case, the memory link reports a fatal error status while PASSON is awaiting the executive's signal. PASSON terminates.

4. PASSON ABNORMAL 3

   An attempt to activate network message logging has failed because the logging code is not available. PASSON terminates.

5. PASSON ABNORMAL 4

An attempt to activate network statistics accumulation has failed because the statistics accumulation code is not available. PASSON terminates.

6. PASSON ABNORMAL 5

   The memory link reported a fatal error status when PASSON attempted to receive a downline message from NOS/VE. PASSON issues the abnormal message and continues interactive processing.

7. PASSON ABNORMAL 6

   After receiving a message from NOS/VE, PASSON has discovered that the message is neither a data message nor a supervisory message (the only 2 possibilities for application messages). PASSON issues the abnormal message and continues interactive processing.

8. PASSON ABNORMAL 7

   NAM has signaled to PASSON (via the supervisory status word) that an upline supervisory message is available to relay to NOS/VE, but PASSON ascertains that the message is not a supervisory message. PASSON issues the abnormal message and continues interactive processing.

9. PASSON ABNORMAL 8

   Not used.

10. PASSON ABNORMAL 9

    PASSON has received a connection request supervisory message (CON/REQ/R) from NAM for a connection which is already in use. The connection request is ignored. PASSON issues the abnormal message and continues interactive processing.

11. PASSON ABNORMAL 10

    PASSON has received an initialized-connection (EC/INIT/N) or an initialized-connection request (FC/INIT/R) supervisory message from NOS/VE or NAM, respectively, and the connection receiving the message is not at the appropriate stage of the connection initialization sequence. PASSON places a connection in a state of waiting for an FC/INIT/R (after a CON/REQ/N) or an FC/INIT/N (after an FC/INIT/R). In this case the connection is not in the

wait state when the FC/INIT/R or FC/INIT/N is received. The message is ignored (which will hang an uninitialized connection) and PASSON continues interactive processing after issuing the abnormal message.

12. PASSON ABNORMAL 11

PASSON has received a network shutdown supervisory message (SHUT/INSD/R) from NAM indicating that NAM shutdown is in progress. PASSON terminates immediately if the shutdown is immediate, otherwise the shutdown message is relayed to NOS/VE and PASSON continues interactive processing after it issues the abnormal message.

13. PASSON ABNORMAL 12

A supervisory message has been received by PASSON from NAM that is not in the list of supervisory messages which PASSON can recognize and receive from NAM. The message is ignored, PASSON issues the abnormal message and continues interactive processing.

14. PASSON ABNORMAL 13

The status of PASSON is normal and processing continues.

15. PASSON ABNORMAL 14

The memory link reported a fatal error status when PASSON attempted to send a queued upline supervisory message to NOS/VE. The message is not sent and PASSON continues interactive processing after issuing the abnormal message.

16. PASSON ABNORMAL 15

The memory link reported a fatal error status when PASSON attempted to send an upline supervisory message to NOS/VE that was not queued. The message is not sent and PASSON continues interactive processing after issuing the abnormal message.

17. PASSON ABNORMAL 16

PASSON tried to send a supervisory message from NAM to NOS/VE but had to queue the message for later transmission, but there was no allocatable space to queue the message. The message is not sent and PASSON continues interactive processing after issuing the

abnormal message.

18. PASSON ABNORMAL 17

    The memory link reported a fatal error status when PASSON attempted to send a queued upline data message to NOS/VE. The message is not sent and PASSON continues interactive processing after issuing the abnormal message.

19. PASSON ABNORMAL 18

    The memory link reported a fatal error status when PASSON attempted to send an upline data message to NOS/VE. The message is not sent and PASSON continues interactive processing after issuing the abnormal message.

20. PASSON ABNORMAL 19

    PASSON has received a connection-rejected supervisory message (CON/REQ/A) from NOS/VE for a connection which was not in the wait state for a CON/REQ/N. PASSON places a connection in a state of waiting for a CON/REQ/N after the connection has sent a CON/REQ/R upline to NOS/VE. The connection is ended and PASSON continues interactive processing after issuing the abnormal message.

21. PASSON ABNORMAL 20

    PASSON has received a connection accepted supervisory message (CON/REQ/N) from NOS/VE for a connection which was not in the wait state for a CON/REQ/N. PASSON places a connection in a state of waiting for a CON/REQ/N after the connection has sent a CON/REQ/R upline to NOS/VE. The connection is ended and PASSON continues interactive processing after issuing the abnormal message.

22. PASSON ABNORMAL 21

    PASSON has received a supervisory message from NOS/VE to stop interactive processing. PASSON terminates.

23. PASSON ABNORMAL 22

    PASSON has received a supervisory message from NOS/VE which is not in the list of supervisory messages which PASSON recognizes and can receive from NOS/VE. The message is ignored and PASSON continues interactive processing after issuing the abnormal

message.

24. PASSON ABNORMAL 23

    Not used.

25. PASSON ABNORMAL 24

    PASSON has received a data message from NAM whose message header has the ibu (input block undeliverable) bit set. The message is not sent to NOS/VE and PASSON continutes interactive processing after issuing the abnormal message.

26. PASSON ABNORMAL 25

    PASSON has received a supervisory message from NAM whose message header has the ibu (input block undeliverable) bit set. The message is not sent to NOS/VE and PASSON continues interactive processing after issuing the abnormal message.

27. PASSON ABNORMAL 26

    The memory link reported a fatal error status when PASSON attempted to send a message (data, supervisory, queued data or queued supervisory) to a NOS/VE application not signed on to the memory link. The message is not sent to NOS/VE and PASSON continues interactive processing after issuing the abnormal message.

28. PASSON ABNORMAL 27

    PASSON has received an error-logical supervisory message (ERR/LGL/R) from NAM for a connection. The message is reported to the job dayfile for the connection and then relayed to the connection's associated NOS/VE application. After issuing the abnormal message PASSON continues interactive processing.

3.10.2 MEMORY LINK FATAL ERROR CODES

The fatal memory link error codes are listed below with corresponding descriptions. Those error codes not listed are not fatal errors.

1. MEMORY LINK FATAL ERROR 1

   This status value indicates that the application name of the

receiver is not valid. Currently, the only invalid value is -1. The request is ignored.

2. MEMORY LINK FATAL ERROR 2

   This status value indicates that the application name of the sender is not valid. Currently, the only invalid value is -1. The request is ignored.

3. MEMORY LINK FATAL ERROR 3

   This status value indicates that the receiver application is not currently signed on to MLI. The request is ignored.

4. MEMORY LINK FATAL ERROR 4

   This status value indicates that the application name specified is signed on to MLI, but by a different task than the one making the request, i.e., the requesting task is not the 'owner' of the application. The request is ignored.

5. MEMORY LINK FATAL ERROR 7

   This status value indicates that an ADD_SENDER request is being performed, but the permit list of the receiver is full. The maximum number of permits is specified by mic$max_permits. The request is ignored.

6. MEMORY LINK FATAL ERROR 8

   This status value indicates that the sender application being referenced is not currently signed on to MLI. The request is ignored.

7. MEMORY LINK FATAL ERROR 9

   This status value indicates that a SEND_MESSAGE is being attempted between two applications, both of which are C170 applications. The request is ignored.

8. MEMORY LINK FATAL ERROR 10

   This status value indicates that the sender application has not been granted permission by the receiver application to send messages to it (CONFIRM_MESSAGE or SEND_MESSAGE request) or that a

DELETE_SENDER) request is being performed for an application not currently permitted. The request is ignored.

9. MEMORY LINK FATAL ERROR 15

   This status value indicates that a RECEIVE_MESSAGE request has been made, but the buffer that is to receive the message text is smaller than the actual message. The request is completed normally, but only as much of the message is transferred as will fit in the buffer.

10. MEMORY LINK FATAL ERROR 17

    This status value indicates that the length of the message specified in a SEND_MESSAGE request is greater than mic$max_message_length. The request is ignored.

11. MEMORY LINK FATAL ERROR 21

    This status value indicates that the application has tried to SIGN_ON to MLI more than mic$max_signons_per_appl times. The request is ignored.

12. MEMORY LINK FATAL ERROR 22

    This status value indicates that the max_messages parameter on a SIGN_ON request is larger than mic$max_queued_messages. The request is ignored.

13. MEMORY LINK FATAL ERROR 23

    This status value indicates that the maximum number of applications are currently signed on to MLI. The request is ignored.

14. MEMORY LINK FATAL ERROR 24

    This status value indicates that a SIGN_ON request has attempted, but the task making the request has already signed on mic$max_signons_per_system_name unique applications. The request is ignored.

15. MEMORY LINK FATAL ERROR 25

This status value indicates that some sort of internal error has occurred within MLI. The request may be retried, but the results are unpredictable.

16. MEMORY LINK FATAL ERROR 26

    This status value indicates that a C170 MLI request has specified an illegal MLI function. The request is ignored.

17. MEMORY LINK FATAL ERROR 27

    This status value indicates that a C170 parameter passed to MLI by a C170 job was invalid. This can be due to a parameter value out of range or because of an address beyond the C170 field length.

18. MEMORY LINK FATAL ERROR 28

    This status value indicates that a memory link request was performed by a C170 job and that the NOS/VE system was not running at the time. The request is ignored.

3.10.3 CONFIGURATION MANAGEMENT ERROR CODES

The following Configuration Management error codes are not documented in the Diagnostic Message manual for R1.1.2.

1. Recorded vsn {text} contains more than 6 characters.

   Product Identifier: CM Condition: 350660
   Condition Identifier: CME$CMD_NAME_TOO_LARGE

   Description:
   The recorded vsn entered contained more than 6 characters.

   User Action:
   Reenter the command using a vsn of 6 or fewer characters.

2. Channel number {text} is not in range between {text} and {text}.

   Product Identifier: CM Condition: 350665
   Condition Identifier: CME$ILLEGAL_CHANNEL_NUMBER

   Description:
   The channel currently being used by NOS/VE for access to the system device is not supported by NOS/VE.

User_Action:
Change the channel selected on the SETVE command to one supported by NOS/VE. Currently, NOS/VE supports channel numbers from 0 through 27 (10).

3. File {file} must have page width of at least {text} characters.

   Product Identifier: CM Condition: 350735
   Condition Identifier: CME$PCU_ILLEGAL_OUTPUT_FILE

   Description:
   The output file must have the specified minimum page width.

   User_Action:
   Change the file attributes to have a page width of at least the specified size.

4. Product identification {text} is unknown to the system.

   Product Identifier: CM Condition: 350770
   Condition Identifier: CME$PCU_UNKNOWN_PRODUCT_ID

   Description:
   The product identification specified is not currently supported.

   User_Action:
   Check the supported equipment and verify that you are using a supported product identifier.

5. Unable to allocate the active volume table in procedure {text}.

   Product Identifier: CM Condition: 350602
   Condition Identifier: CME$PC_ALLOCATE_AVT_ERROR

   Description:
   An active volume table for devices besides the system device can not be allocated.

   User_Action:
   Redeadstart the system. If the condition persists, contact a site analyst.

6. Logical unit {text} is not in the physical configuration table.

   Product Identifier: CM Condition: 350640

Condition Identifier: CME$PC_UNIT_NOT_FOUND

Description:
The element described by the given logical unit number is not present.

User Action:
Retry the request.

3.11 CPU MAINTENANCE PROCEDURES

Included with NOS/VE standard products is the Diagnostic Virtual System (DVS). DVS is a utility which controls on-line diagnostic test execution under NOS/VE. Programs used by DVS are enhanced versions of off-line diagnostics. These diagnostics employ the same testing strategy, but contain unique code for on-line interfacing.

Usage

DVS documentation is available in the form of a printed usage guide, publications number 60469720 revision B. It is also documented in an on-line version of the usage guide. The on-line guide is contained on file DVS_USAGE in the HARDWARE_MAINTENANCE subcatalog. To use the guide enter:

```
EXPLAIN M=$SYSTEM.HARDWARE_MAINTENANCE.DVS_USAGE
```

Notes and Cautions

1. The DEBUG hardware test (DBUG), although documented for release 1.1.2, will not be released until 1.1.3.

2. When using DVS with Dual Processor Cyber 830's, it is possible to create a situation where DVS obtains most of the CPU resource when a specific processor is selected for testing. It is recommended for concurrent operation, that you do not select a specific processor.

3. When using DVS with Cyber 845/855 systems, diagnostics BIMM and NUMR detect known BDP end-case failures. Using RUN_CPU_DIAGNOSTICS automatically detects the processor type and will not run these tests. However, when manually selecting tests, it is recommended that these tests not be run until the appropriate FCO's are installed.

4. When using DVS with Cyber 845/855 systems, diagnostic RFST detects known BDP end-case failures. When using the RUN_DVS command, you should not alter the RFST_DELETE_BDP parameter. It is IMPARATIVE that this parameter remain at its default setting until appropriate FCO's are installed to correct the end-case failures.

4.0 OPERATING SYSTEM NOTES AND CAUTIONS

4.0 OPERATING SYSTEM NOTES AND CAUTIONS

4.1 SYSTEM COMMAND LANGUAGE

1. A command that depends on the "path name" for a file is affected if the file has been explicitly attached (attach_file, create_file commands).

   The "path name" that such a command will encounter is the "local path name" rather than the path specified when the command was called. For example:

   ```
   ATTACH_FILE $USER.X LFN=Y
   REPLACE_FILE $USER.X
   ```

   will result in a file called Y being referenced in real state rather than X. Users should not mix implicit and explicit use of a file.
   PSR NVOD753

2. The $PROGRAM function does not return the open position designator for the LOAD_MAP file.
   PSR NVOD483

3. The SET_SENSE_SWITCH (SETSS) command will not accept a "user supplied job name" but will only work when given a "system supplied job name".
   PSRs NVOD871, NVOD888

4. The constants $MIN_INTEGER and $MAX_INTEGER have values of -(2**48-1) and 2**48-1, respectively, rather than -(2**63) and 2**63-1.
   PSR NVOD609

5. Any files created by the program interface cannot be displayed with DISC $LOCAL. For example, the default files created in FORTRAN do not appear when subsequently entering DISC $LOCAL.
   PSRs NVEA052, NVEA055, NVEA056

6. JOB/JOBEND reformats data lines which end in two periods. The problem is that JOB/JOBEND treats lines ending in two periods as command lines to be continued and writes the concatenated form of

the command line to the file it submits as a job. This is usually only a problem if the job contains a usage of COLLECT_TEXT. In this case, the file could be created before using JOB/JOBEND and referenced in the job, rather than creating the file within the job.
PSR NV0F327

7. In the process of implementing improvements to DISPLAY_COMMAND_LIST, the command table passed to CLP$PUSH_UTILITY by command utilities has been changed.

   The new tables should be produced by using the new command table generator which is described in APPENDIX D. GENerate_Command_Table does for the new command and function tables what the parameter descriptor table (PDT) generator does for PDTs. It takes as input a specification for the table and produces as output the CYBIL variable declarations that represent the table.

   CLP$PUSH_UTILITY expects entries in the tables it is passed to have been correctly generated, unlike the previous situation in which you could pass a table that was unsorted or contained names in lower case letters.

   New commands for displaying the command list have also been implemented. Thus the information that used to be available via the full mode option of the DISPLAY_COMMAND_LIST command must now be obtained from the DISPLAY_COMMAND_LIST_ENTRY command. See the following section on "Additional Commands" for a description of the new form of the DISPLAY_COMMAND_LIST command, and the new DISPLAY_COMMAND_LIST_ENTRY command.

8. Two parameters have been added to the COLLECT_TEXT command following the UNTIL parameter.

   prompt, p: This parameter specifies the prompt string to be issued for each line if collect_text is getting its input from an interactive terminal. If the null string is specified, no prompt is issued.

   Omission causes 'CT?' to be used.

   substitution_mark, sm: This parameter specifies a character that is used within the text to be collected to delimit text to be substituted, or that no such character be used (NONE).

Corresponding pairs of substitution marks must appear on the same line. The text between such marks is evaluated as an SCL string expression, the result of which replaces the original text including the substitution marks.

If the expression cannot be evaluated or its result cannot be converted to a string, the COLLECT_TEXT command terminates with an appropriate error. If no second substitution mark is found on a line, the end-of-line is treated as the second substitution mark. If two consecutive substitution marks appear, they are replaced by a single substitution mark in the collected text. Substitution cannot be used to construct the line containing the termination string, i.e. the termination string is checked for prior to processing the line for substitution.

Omission causes NONE to be used.

9. A parameter has been added to the JOB/JOBEND command following the JOB_CLASS parameter.

substitution_mark, sm: This parameter specifies a character that is used within the statements to be collected to delimit text to be substituted, or that no such character be used (NONE). Corresponding pairs of substitution marks must appear on the same line. The text between such marks is evaluated as an SCL string expression, the result of which replaces the original text including the substitution marks.

This capability is provided because the submitted job does not have access to any variables or procedure parameters of the job that initiates it. Therefore, if parametric information is to be used in the submitted job, it must be substituted directly into the statements of the job.

If the expression cannot be evaluated or its result cannot be converted to a string, the JOB/JOBEND command terminates with an appropriate error. If no second substitution mark is found on a line, the end-of-line is treated as the second substitution mark. If two consecutive substitution marks appear, they are replaced by a single substitution mark in the collected text. Substitution cannot be used to construct the JOBEND delimiter.

Omission causes NONE to be used.

10. Interactive Command Language Prompting

This release includes the initial phase of interactive prompting. Subsequent releases will include a more comprehensive prompting facility with help modules containing messages and prompts for each command.

For a command entered from an interactive terminal, SCL issues prompting information to the terminal depending upon user actions and the "mode" in which the user is operating.

"NORMAL" MODE

This is the mode under which all commands in procedures, batch jobs and included files are processed by default -- that is, if any error is detected during evaluation of the parameters for the command, processing of the command is terminated immediately.

"LINE" MODE

This mode is applicable only to interactive jobs. Line mode must be explicitly selected by the user via the SET_COMMAND_MODE command.

This mode does not require any special type of terminal but does take into account certain terminal attributes such as page width.

In line mode, the parameters entered with the command name are evaluated and for each detectable error, an error message is written and the user is prompted to enter the correction. The prompting information is derived from the data in the command's parameter description table (PDT).

Once all specified parameters have been accepted, a check is made to insure that all required parameters have been supplied and if not, the user is prompted to enter them.

When entering corrections or omitted parameters, the user may enter more than the parameter being prompted for. This is done in exactly the same way as parameters are normally given to a command.

If the original call to the command did not include any parameters and the command has at least one required parameter, the user is prompted for each parameter of the command, except those with the

VAR attribute that are optional (this includes the STATUS parameter).

In response to any prompt, a terminate break can be entered to terminate processing of the command. Also, a question mark (?) can be entered to obtain information about acceptable responses to the prompt. If a question mark is entered twice in succession in response to a prompt, or a double question mark (??) is entered, information about how to use line mode facilities is displayed.

4.1.1 ADDITIONAL COMMANDS (NOT IN MANUALS)

4.1.1.1 DISPLAY_COMMAND_PARAMETER(S) (DISCP)

This command displays information about the parameters for a command. The information includes the names of the parameters, their types (including allowed keyword values) and their default values (or an indication that the parameter is required).

This command is not intended to be a replacement for information in online manuals, but rather a "memory jogger" for any command. It will work for any command that could be called at the point where this command is called (including system supplied commands, SCL procedures, user programs and utility subcommands).

This command can be used to obtain information about the parameters for any command that uses SCL services to parse its parameters.

```
display_command_parameters command=<command>
      [output=<file reference>]
      [status=<status variable>]
```

command ; c: This parameter specifies the command for which information about parameters is sought.

output ; o: This parameter specifies the file to which the parameter information is written. Omission will cause $OUTPUT to be used.

status: This is the standard status parameter.

4.1.1.2 DISPLAY_COMMAND_LIST (DISCL)

This command displays the names of the entries in and/or the search mode governing the command list.

```
display_command_list [display_options=list of all ;
            entries;entry;e ; search_mode;sm]
      [output=<file reference>]
      [status=<status variable>]
```

display_options ; display_option ; do: This parameter selects the information to be displayed.

all: This option selects all of the other options.

entries ; entry ; e: This option causes the names of the entries in the command list to be displayed.

search_mode ; sm: This option causes the command list search mode to be displayed.

Omission causes entries to be selected.

output ; o: This parameter designates the file to which the information is written.

Omission causes $output to be used.

status: This is the standard status parameter.

4.1.1.3 DISPLAY_COMMAND_LIST_ENTRY(IES) (DISCLE)

This command displays information about one or more command list entries.

```
display_command_list_entry [entry=list of <file> ; all ;
            control_statements;control_statement;cs ;
            first;f]
      [display_options=list of all ; all_names;allname;an ;
            commands;command;c ; functions;function;f
            names;name;n]
      [output=<file reference>]
```

4.0 OPERATING SYSTEM NOTES AND CAUTIONS
4.1.1.3 DISPLAY_COMMAND_LIST_ENTRY(IES) (DISCLE)

[status=<status variable>]

entry ¦ entries ¦ e: This parameter selects the command list entry or entries to be displayed. Any entry that can be put in the command list can be selected regardless of whether it is presently in the command list. In addition the following keywords can be used to make generic selections.

all: This option selects the entire command list for display (this includes the control statements).

control_statements ¦ control_statement ¦ cs: This option selects the control statements and commands inherently part of SCL.

first ¦ f: This option selects the first entry in the command list. This provides a way of selecting a display of the subcommands supplied by a utility when the utility is active.

Omission causes first to be used.

display_options ¦ display_option ¦ do: This parameter selects the form and content of the output.

all: This option selects all of the other options.

all_names ¦ all_name ¦ an: This option causes all of the names (abbreviations and aliases, as well as the "nominal" names) of commands and/or functions and/or control statements to be displayed.

commands ¦ command ¦ c: This option causes information about the individual commands within the selected command list entries (other than catalogs) to be displayed.

functions ¦ function ¦ f: This option causes information about the individual functions within the selected command list entries to be displayed.

names ¦ name ¦ n: This option causes the "nominal" names of commands and/or functions and/or control statements to be displayed. A "nominal" name is the name by which the command, function or statement is normally known.

Omission of this parameter causes commands and names to be

selected. Omission of both commands and functions causes commands to be selected.

output ; o: This parameter designates the file to which the information is written.

Omission causes $output to be used.

status: This is the standard status parameter.

4.1.1.4 SET_DEFAULT_FAMILY (SETDF)

This command provides the operator with the capability of establishing a default family for jobs. This command affects all jobs except those initiated via the JOB/JOBEND command. This includes jobs ROUTEd from the NOS system.

NOTE that prior to the availability of this command, jobs submitted via the SUBMIT_JOB command would have defaulted the family to that of the submittor -- this is no longer true.

If this command is never issued, the default family name is $SYSTEM.

```
set_default_family family_name=<name>
      [status=<status variable>]
```

family_name ; fn: This parameter specifies the name to be used as the default value for the family_name parameter of the LOGIN command.

status: This is the standard status parameter.

4.1.1.5 SET_COMMAND_MODE (SETCM)

This command determines how subsequent commands entered from a terminal will be handled in the event of incomplete or incorrect parameters.

```
set_command_mode mode=line!l!normal!n
      status
```

mode, m: This parameter specifies the command mode being selected.

4.0 OPERATING SYSTEM NOTES AND CAUTIONS
4.1.1.5 SET_COMMAND_MODE (SETCM)

status: See ERROR HANDLING.

4.1.1.6 CHANGE_NATURAL_LANGUAGE (CHANL)

This command determines the preferred natural language to be used for messages, help information, etc.

```
change_natural_language, chanl (
      natural_language, nl: any of
          key
            danish
            dutch
            english
            finnish
            french
            german
            italian
            norwegian
            portuguese
            spanish
            swedish
            us_english
          keyend
          name
        anyend = $required
      status)
```

natural_language: this parameter specifies the natural language being selected.

status: See ERROR HANDLING.

4.1.2 ADDITIONAL FUNCTIONS (NOT IN MANUALS)

4.1.2.1 $COMMAND_SOURCE

This function is used to determine where the processor for the requesting command was found. The source (file or catalog) of the command is returned as a string.

By its nature, this function is not particularly useful when used in the expression for a parameter to a command, since, in that case, it will return the source of that command. Therefore, this function is normally used in an assignment statement.

4.0 OPERATING SYSTEM NOTES AND CAUTIONS
4.1.2.1 $COMMAND_SOURCE

```
<$command_source> ::= $COMMAND_SOURCE [<(> <)>]

Example: "The following proc resides on an"
         "object library in some catalog."
         PROC sample_command
           cs = $command_source
           cat = $path($fname(cs), catalog)
           "The following command executes file"
           "sample_program in the same catalog"
           execute_task $fname(cat//'.sample_program')
         PROCEND
```

4.1.2.2 $PREVIOUS_STATUS

This function is used to obtain the completion status of the previous command.

```
<$previous_status> ::= $PREVIOUS_STATUS [<(> <)>]

Example: collect_text display_status
           PROC display_status, diss (
                 status: status = $previous_status)
             display_value $value(status)
           PROCEND display_status
         **
         create_variable s kind=status
         create_variable x status=s
         display_status
         NORMAL STATUS
         create_variable x status=s
         display_status
         --ERROR-- X is already declared as a variable.
```

Note: there are certain status conditions that do not cause $PREVIOUS_STATUS to return the expected value. Among these are error from control or assignment statement and "xxx is not a command".
PSR NV0F077

4.1.2.3 $QUOTE

This function is used to quote a string.

```
<$quote> ::= $QUOTE <(> <string expr> <)>
```

4.0 OPERATING SYSTEM NOTES AND CAUTIONS
4.1.2.3 $QUOTE

```
Example: s = 'ABC''DEF'
         q = $quote(s)
         display_value s
         ABC'DEF
         display_value q
         'ABC''DEF'
```

4.1.2.4 $SCAN_ANY

This function is used to search a string for any one of a set of characters and return the index in the string of the found character. If no character from the set appears in the string, zero is returned.

```
<$scan_any> ::= $SCAN_ANY <(> <char set> <,¡sp>
                           <string expr> <)>

<char set> ::= <string expr>

Example: digits = '0123456789'
         s = 'TEMP_32'
         display_value $scan_any(digits, s)
         6
```

4.1.2.5 $SCAN_NOT_ANY

This function is used to search a string for any character that is not in a set of characters and return the index in the string of the found character. If only characters from the set appear in the string, zero is returned.

```
<$scan_not_any> ::= $SCAN_NOT_ANY <(> <char set> <,¡sp>
                                  <string expr> <)>

<char set> ::= <string expr>

Example: digits = '0123456789'
         s = 'TEMP_32'
         display_value $scan_not_any(digits, s)
         1
```

4.1.2.6 $SCAN_STRING

This function is used to search a string for another string, called the pattern, and return the index of the first character of the pattern in the string. If the pattern is the null string, a value of one(1) is returned. If the pattern is not found in the string, a value of zero (0) is returned.

```
<$scan_string> ::= $SCAN_STRING <(> <pattern> <,¦sp>
                                 <string expr> <)>

<pattern> ::= <string expr>
```

```
Example: s = '123_abc_9'
         p = 'abc'
         display_value $scan_string(p, s)
         5
```

4.1.2.7 $TRANSLATE

This function is used to change characters in a string according to a translation table. The translation table is a string of 256 characters which is utilized according to the following algorithm.

```
result = ''
for i = 1 to $strlen(string) do
  j = $ord($substr(string, i))
  result = result // $substr(table, j+1)
forend
```

Two standard translation tables are provided. Use of LOWER_TO_UPPER (LTU) produces a string with all lower case letters translated to their upper case counterparts. Use of UPPER_TO_LOWER (UTL) produces a string with all upper case letters translated to their lower case counterparts.

```
<$translate> ::= $TRANSLATE <(> <translation table> <,¦sp>
                                <string expr> <)>

<translation table> ::= <string expr>
                      ¦ LOWER_TO_UPPER ¦ LTU
                      ¦ UPPER_TO_LOWER ¦ UTL
```

```
Example: display_value $translate(lower_to_upper, '123_abc')
         123_ABC
```

4.1.2.8 $TRIM

This function is used to remove trailing space characters from a string.

<$trim> ::= $TRIM <(> <string expr> <)>

```
Example: s = 'STRING '
         display_value '<'//s//'>'
         <STRING >
         display_value '<'//$trim(s)//'>'
         <STRING>
```

4.1.3 ADDITIONAL CONTROL STATEMENTS (NOT IN MANUALS)

4.1.3.1 PUSH_COMMANDS

<push commands> ::= PUSH_COMMANDS

The purpose of this statement is to cause the source (file or catalog) of the issuing command to be pushed onto the top of the "dynamic command list". The entries in the dynamic command list are searched after the commands belonging to any active utilities and before the command list entries manipulated by the SET_COMMAND_LIST command. The effect of this statement is removed (popped) when the issuing command terminates.

4.1.4 ADDITION CYBIL PROGRAM INTERFACES (NOT IN MANUALS)

4.1.4.1 AMP$REPLACE_PREVIOUS_RECORD

The purpose of this request is to replace an entire record. This request is valid only for mass storage files opened for sequential record access. The file position must be at the end of the record to be replaced. The file position remains at the end of the record replaced. The length of the replacement record must be identical to the length of the previous record for a file whose record-type is CDC-variable (V). Note the length of the previous record is exclusive of V record-headers, i.e. only the logical length of the records must be identical. Therefore, a record output with a series of partial PUTs can be replaced. For a file whose record-type is ANSI-fixed (F), the replacement record will be truncated to maximum-record-length, if the replacement record exceeds maximum-record-length. The length of the replacement record must be less than or equal to the maximum-block-length for a file whose record-type is undefined (U) and

whose block-type is user-specified. Note that a record deleted by an amp$delete_previous_record request cannot be replaced because one cannot position to the end of a deleted record. This request is not supported for a file whose record-type is undefined (U) and whose block-type is system-specified.

This request requires 'modify' access to the file.

FORMAT:

```
AMP$REPLACE_PREVIOUS_RECORD                    (FILE_IDENTIFIER,
  WORKING_STORAGE_AREA, WORKING_STORAGE_LENGTH, STATUS)
```

FILE_IDENTIFIER: (input) This parameter specifies the file access identifier established when the file was opened.

WORKING_STORAGE_AREA: (input) This parameter specifies the address of the record which is to replace an existing record of the file.

WORKING_STORAGE_LENGTH: (input) This parameter specifies the length of the record to be output.

STATUS: (output) This parameter specifies the request status.

condition identifiers:

```
ame$ring_validation_error,
ame$improper_file_id,
ame$improper_access_attempt,
ame$improper_wsl_value,
ame$improper_file_position,
ame$unrecovered_write_error,
ame$file_organization_conflict,
ame$improper_device_class,
ame$conflicting_access_level,
ame$record_exceeds_mbl,
ame$record_unequal_to_previous,
ame$unsupported_operation;
```

procedure declaration:

```
PROCEDURE [XREF] amp$replace_previous_record (file_identifier:
  amt$file_identifier;
  working_storage_area: ^cell;
  working_storage_length: amt$working_storage_length;
  VAR status: ost$status);
```

```
??  PUSH (LISTEXT := ON) ??
*copyc AMT$FILE_IDENTIFIER
*copyc AMT$WORKING_STORAGE_LENGTH
*copyc OST$STATUS
*copyc AMC$CONDITION_CODE_LIMITS
*copyc AME$CONFLICTING_ACCESS_LEVEL
*copyc AME$IMPROPER_FILE_ID
*copyc AME$PUT_VALIDATION_ERRORS
*copyc AME$TERMINAL_VALIDATION_ERRORS
*copyc AME$RING_VALIDATION_ERRORS
*copyc AME$IMPROPER_WSL
*copyc AME$PUT_PROGRAM_ACTIONS
??  POP ??
```

4.2 PROGRAM MANAGEMENT

1. Loading files into multiple rings may cause improper ring attributes to be assigned.
   PSR NVOC789

2. The load time for large programs is too high.

   Solution: Pre-linked object modules.

   Large programs take a long timed to load. Analysis shows the loader spending much of its time building addresses in static data sections. This is especially true in large FORTRAN programs.

   A process of "Pre-linking" has been created to completely eliminate the time spent in the loader building static addresses. A bound program and its destination ring brackets are given to the virtual environment linker (LINVE) which pre-links the product using a set of reserved segment numbers. All addresses in static data sections are built and stored with the static data for the section. This data can be simply copied at load time into its reserved segment.

   The output of the linker is a single object module which can be placed on an object library and/or executed. The Object Code Utilities handle this new module as it would any other object module. All OCU commands may be used with a "pre-linked" module.

   Example

```
/
/link_virtual_environment
VEL/set_link_options  mode=PRODUCT
VEL/add_object_file  file=BOUND_PRODUCT  ring_brackets=(11,11,11)
VEL/generate_virtual_memory  virtual_image=PRE_LINKED_PRODUCT
VEL/quit
/
/create_object_library
COL/add_module  library=PRE_LINKED_PRODUCT
COL/generate_library  library=PRODUCT_LIBRARY
COL/quit
/
/execute_task  sp=PRODUCT_ENTRY_POINT  file=PRE_LINKED_PRODUCT
/execute_task  sp=PRODUCT_ENTRY_POINT  library=PRODUCT_LIBRARY
/
```

Notes and Cautions

Care must be used when building programs that are made up of multiple "pre-linked" modules. All pre-defined segment numbers must be unique for the entire load sequence. The STARTING_SEGMENT parameter on the linker command SET_LINK_OPTIONS can be used to specify the first reserved segment number for a module. This parameter allows modules that have been "pre-linked" separately to be used together at execution time. By default, the first reserved segment number is currently 36.

The operating system reserves segments 36 to 63 for "pre-linked" products. Each product must fit into these segments. Linker directives can be used to collect sections with similar attributes into one segment to reduce the total number of segments.

With the current implementation, sporatic initialization of read/write data will increase both file space and load time. During the coding and binding of a program, efforts should be made to collect all initialized data together near the beginning of a segment.

A warning diagnostic is issued if the linker finds a text imbedded library that has not been specified on a USE_OBJECT_LIBRARY command. This warning can be ignored and the loader will attempt to satisfy them at load time.

A map is generated by the linker containing diagnostics and information on how the program was "pre-linked". The default file

name for the link map is $LOCAL.LINKMAP.

With the current implementation, only one module should be "pre-linked". A program should be bound into a single module and then "pre-linked".

Conclusions

The process of "pre-linking" programs can significantly reduce the time spent loading products. "Pre-linking" one of the system products resulted in a 35-40 percent reduction in page faults, CP time, and elapsed time during loading. The reduction should be even greater when loading large FORTRAN products.

4.3 PHYSICAL_I/O_(TAPE)

1. Tape usage is restricted to permanent file backup/restore and dump analyzer operations for R1.1.2.

4.4 BASIC_ACCESS_METHOD

1. The SET_WORKING_CATALOG (SETWC) command has no effect upon files whose names are created within applications. For example:

```
SETWC .jxt
execute_task
   task generates file name a
```

   Files created by programs are not subject to the working catalog. The program should specify a complete path name. Files referenced as parameters of commands (i.e., files specified at command level) will end up using working catalogs.
   PSR NVEA027

2. Although the preset_value file attribute can be set bye the set_file_attribute command, this feature tis not supported. New pages assigned to segment access files are always initialized to zero.
   PSR NV0B204

3. File attribute MIN_RECORD_LENGTH is not set properly for record_type F files. MIN_RECORD_LENGTH should default to MAX_RECORD_LENGTH if it is allowed to default, but it is currently defaulting to 0. This affects SORT/MERGE when the sort key is

omitted and the default is taken. The user should explicitly set min_record_length = max_record_length.
PSR NV0D398

4. Skipping forward/backward by records and partitions does not always report the encounter of boundary conditions with the correct exception condition.
PSR NV0D424

5. If a FAP returns abnormal status to AMP$OPEN during an OPEN (new), then BAM still creates the local file. In simpler terms, a user who makes a mistake in the initial OPEN of an indexed sequential file will receive a proper diagnostic, but BAM wrongly goes ahead and creates the local file anyway.
PSR NV0E150

6. File attributes that can be changed cannot be reassigned a NIL value.
PSR NVEA003

7. The SET_FILE_ATTRIBUTE command for an existing file accepts the specification of a preserved attribute even if it conflicts with the present one. No error will be seen and the preserved attribute will not be changed. Use DISPLAY_FILE_ATTRIBUTES to verify attributes were set. Also, a subset of preserved attributes may be changed with the CHANGE_FILE_ATTRIBUTE command.
PSRs NV0C839, NV0D617, NV0D549, NV0D536

8. COPY_FILE does not detect a copy of a file to itself via circular file connections.
PSR NVEA118

9. Files implicitly attached are not implicitly detached.
PSRs NV0E322, NV0E296

10. Attributes in the NOS/VE file label cannot be altered after creation. This could affect some migrating applications using indexed sequential files because FILE_LIMIT could be updated on C170 but not on C180.
PSR NVEA034

11. If a file has a file-access-procedure (FAP) associated with it, the file can not be accessed from within the job monitor task. If this is attempted, a message will report the inability to load the FAP from the library list. To get around the problem, enter a

utility environment of your choice and then access the file. As an example, if you do a COPY_FILE involving an indexed sequential file and the error "unable to load FAP ..." occurs, then you have just experienced this problem.
PSR NV0C941

12. If a file was created by COLLECT_TEXT at NOS/VE 1.1.1, it was created with file content = legible and line number = (0, 0) (length and location). Under 1.1.2, if you attempt to copy this file to another file which does not yet exist, the copy aborts with AM 160044 - line number length and location are unequal. Note that it is impossible to create these files under 1.1.2 since (0 0) are out of range values for line number and the default is (1 1).
PSR NV00521

4.5 LOADER

1. The length of large data segments and files cannot exceed 100,000,000 bytes.

4.6 INTERACTIVE

1. If an output line for a display type terminal (CRT) is exactly equal to the page width attribute, overprinting of the line will occur if more output lines follow. To remedy this, set the page width differently or set the output_device attribute (OD) to printer.

2. Users are now able to inform the OS of their terminal model by using SETTA's terminal_model (TRM) parameter and specifying a 1-25 character name. This attribute can only be changed by SETTA but can be retrieved by IFP$FETCH_TERMINAL, IFP$GET_TERMINAL_ATTRIBUTES, IFP$GET_DEFLT_TERMINAL_ATTRIBUTES and can be displayed by DISTA. The login defaults for this attribute are 'CDC721' for terminal class 3, 'VT100' for terminal class 7, and the null name for all other terminal classes. The restrictions which define valid SCL names are the only restrictions for specifying the terminal_model parameter.

3. A new error code has been added to Interactive Facility's repetoire to indicate that a task has aborted a get operation because an interactive condition was received.

   IFE$ABORT_GET --ERROR-- get operation aborted because

interactive condition received

4.7 PERMANENT_FILE_UTILITIES

1. A new function $BACKUP_FILE is available under the restore_permanent_files utility. The purpose of this function is to interrogate the information on a backup_file produced by the backup_permanent_file utility. All characters returned in the string are in upper case. The $backup_file function causes the backup file to be rewound, so only the first item on the backup file may be queried.

   $BACKUP_FILE(<file> <attribute>)

   The file parameter specifies the backup file to query.

attribute	result	
IDENTIFIER	string	Returns the path name of the first item on the backup file. This is the default attribute.
IDENTIFIER_TYPE	string	Returns the type of the item. Either: 'SET' 'CATALOG' 'FILE' or 'CYCLE' is returned.

   Currently no abbreviations are supported for either the function Or for the attribute keywords.

   An example use for this function is in attempting to restore from a backup file for which the name of the catalog or file backed up is not known, but the destination of the restore is known. Note that the value returned is a string, so that $fname needs to be used to convert this to a file name, if this is to be used as a parameter on the restore_catalog or restore_file subcommands.

   Example:
   Suppose someone has sent you a backup tape produced by a single BACKUP_CATALOG command, and that you want to restore the catalog to your own catalog $USER.MY_CATALOG. You could do the following:

```
restore_permanent_files l=list_file
  restore_catalog $fname($backup_file(backup_file,identifier)) ..
    bf=backup_file ncn=$user.my_catalog
```

```
quit
```

5.0 PRODUCT SET NOTES AND CAUTIONS

5.1 CYBIL

1. When using DEBUG to breakpoint on a multi-line statement, the line number supplied to DEBUG must be the last line number of the statement.
   PSR CILA498

2. Run-time error messages, such as RANGE ERROR, do not display the bad value.
   PSR CILA575

3. If binary maintenance is required, the CCM parameter must be specified as $SYSTEM.COMMON.MAINTENANCE.OLD_UNBOUND_CCM. This CCM interface will be updated for Release 1.1.3.

4. The largest negative integer value causes STRINGREP to abort with arithmetic overflow.
   PSR CILA838

5. It is not possible to share data variables between CYBIL and another language (say FORTRAN), because CYBIL does not have any mechanism to create a common block.

6. Static initialization treats integers as 60 bits instead of 64 bits.
   PSR CILA880

7. The RUNTIME_CHECKS parameter is not accepted by the CYBIL compiler. Currently, the CYBIL compiler looks for RUNTIME_CHECK (no 'S').
   PSR CIL0005

8. Tag checking does not work.
   PSR CILA886

9. Using a null string inside an inline procedure gives a length mismatch.
   PSR CILA969

10. Using unimplemented features may cause CYBIL to abort with CG1 error 38.

5.0 PRODUCT SET NOTES AND CAUTIONS

5.1 CYBIL

PSR CILA679

5.2 COBOL

1. A deficiency in the COBOL/DEBUG implementation does not allow the MODULE and PROCEDURE parameter names to contain hyphens when specified by the user. This problem can be avoided by omitting hyphens in the PROGRAM-ID name in the COBOL source program, or by relying on the default values for MODULE and PROCEDURE when using the DISPLAY_PROGRAM_VALUE, CHANGE_PROGRAM_VALUE and SET_BREAK commands. (DISPLAY_MEMORY does not support default MODULE names.) COBOL data_names may contain hyphens, however, when specified as the NAME parameter of the DISPLAY_PROGRAM_VALUE and CHANGE_PROGRAM_VALUE commands.
   PSRs NV0F043, DB8A054

2. The KEY_TYPE attribute cannot be overridden any more.
   PSR CB8A431

3. The utility routine CBP$SET_FILE_ATTRIBUTES has been deleted. File attributes can be set in the COBOL program by doing ENTER "CLP$SCAN_COMMAND_LINE" using "SETFA....", STATUS. The COBOL usage guide contains an example of the CLP$SCAN_COMMAND_LINE routine.

4. COBOL cannot replace data items referenced by indices in the "INSPECT ... REPLACE" statement.
   PSR CB8A537

5. When compiling with a working catalog set, and the compile file contains COPY statements, the reference decks are not expanded.
   PSR CB80006

5.3 FILE_MANAGEMENT_UTILITY

1. Do not use the file name INPUT in batch mode. It is always a null (empty) file with no data in it. Choose a different name. This affects migration of jobs from NOS, where the name INPUT was the job's input deck. On NOS/VE use the COLLECT_TEXT command to create the input data.

2. When using FMU and inside Interstate Communication Facility, do not do a CLEAR on the EXECUTE_INTERSTATE COMMAND. This will clear the FASLAVE procedure file and render CREIC inoperable.
   PSR NV0F516

3. The ICF connection will be terminated without any indication (i.e., diagnostic or informational message) if an FMU job aborts within ICF. This occurs in batch jobs only.
   PSR NV0F320

4. When using VAX migration commands (CREATE_VAX_REQUEST, DISPLAY_VAX_REQUEST, DELETE_VAX_REQUEST), it is recommended that parameter specification be limited exclusively to keyword specification to allow for the addition of new parameters in a subsequent release.

5.4 ADVANCED_ACCESS_METHODS

1. A job using an indexed sequential file may fail due to entry point errors from the loader.

   The following command executed earlier in the job will correct this:

   ```
   SET_PROGRAM_ATTRIBUTES TERMINATION_ERROR_LEVEL=FATAL
   or SETPA TEL=F
   ```

   PSR NV0E276

2. When the FILE_LIMIT attribute is set too small for accommodating all updates or inserts, the user runs the risk of the access method aborting with an error AAE$FILE_AT_FILE_LIMIT. In this case, the file is destroyed.
   PSR AA8A380

3. All of the XREF decks relevant to indexed_sequential file usage should be in one base library. At present, some of them are in $SYSTEM.COMMON.PSF$EXTERNAL_INTERFACE_SOURCE and some of them are in $SYSTEM.CYBIL.OSF$PROGRAM_INTERFACE.
   PSR NV0F197

4. The XREF deck AMP$GET_KEY_DEFINITIONS forgets to copy in type declaration AMT$OPTIONAL_KEY_ATTRIBUTES. The workaround is to use ARRAY [1..n] of AMT$OPTIONAL_KEY_ATTRIBUTE.
   PSR NV0F938

5. The GET routine does not honor $KEY_RELATION nor update the value in $PRIMARY_KEY_ADDRESS.
   PSRs AA8A377

6. COPY_KEYED_FILE aborts when copying any empty file.
   PSR AA8A357

7. COPY_KEYED_FILE specifying output to a terminal causes the first character of each line to be lost.
   PSR AA8A353

8. COPY_KEYED_FILE aborts when copying a file containing a nonstandard collated key; the message is "CAN'T LOAD COLLATING TABLE".
   PSR AA8A338

5.5 SOURCE_CODE_UTILITY

1. A CYBIL range error occurs at line 4747 of SCM$CONVERT_UPDATE_TO_SCU.
   PSR SC8A130

   The workaround is to take out the LIST parameter.

2. A new source library format has been implemented to support present and future performance improvements. This will require conversion of existing libraries with the command that follows:

   This command reads an SCU source library in 1.0 format and writes it in 1.1 format. There is no procedure to convert 1.1 back to 1.0.

   convert_scu10_to_scu11, CONS10TOS11 [base=<file>]
   [result=<file>]
   [status=<status_variable>]

   base : b This is the name of a file containing an SCU source_library in the 1.0 library format. If this parameter is omitted an attempt is made to access a file named SOURCE_LIBRARY.

   result : r : This is the name of the file to receive the converted library in version 1.1 format. If this parmeter is omitted the library will be written on file SOURCE_LIBRARY.$NEXT.

   status : Status variable.

3. With the new library format the lists of features, groups and modifications for a library will be maintained in alphabetic

sequence. The list of modifications and groups associated with decks will be kept in the same order as previously used. In connection with this change the display_option parameter has been changed on the display_modification_list command:

```
display_modification_list [output=<file_reference>]
     [display_options=alphabetic|chronological]
     [status=<status_variable>]
```

output : o : This is the name of the file which receives the display. Default is $OUTPUT.

display_options : display_option : do : If this parameter is given the keyword value CHRONOLOGICAL or C, modifications will be order by date and time of modification with the least recent first. If this parameter is omitted or value ALPHABETIC or A the order will be alphabetic.

status : Status variable.

This command displays a list of all the modifications for a source library.

4. The deck_description parameter on the create_deck and change_deck comamnds, the library_description parameter on the create_library and change_library commands and the modification_description parameter on the create_modification and change_modification commands all accept as their values a list of strings. The corresponding $deck_header, $library_header and $modification_header functions have been modified to return an array of strings as the value for the description fields when it is appropriate.

5.6 PRODUCT SET = DEBUG

1. When an unselected event occurs in a ring lower than the ring in which it is running, DEBUG incorrectly informs the user that the error occurred in the procedure which called the faulty procedure in the lower ring.
   PSR DB8A098

2. DEBUG and CYBIL do not have a run-time interface. Therefore, a program which aborts due to "CYBIL run time error" cannot be

properly trapped by DEBUG. Instead, the DEBUG trap occurs in the CYBIL error processor. To reference variables in the failing procedure, MODULE and PROCEDURE parameters need to be explicitly specified. However, dynamic variables disappear as soon as the run-time error occurs.

3. The operation of DEBUG in a multi-ring environment is not well-defined. DEBUG only gets control when the program executes in the same ring as set by the SCL command Set_Debug_Ring.

4. DEBUG does not coordinate its prompting activities between two asynchronous tasks.
PSR DB8A076

5. DEBUG does not accept hyphenated names for MODULE or PROCEDURE parameter values. COBOL users may use the default value for these parameters in line mode DEBUG. Full screen DEBUG users must remove the hyphen from the program id statement name and recompile.
PSR DB8A054

5.7 APL

1. AFIFIX is needed in order for the APL/VE conversion utility to successfully convert "APL structured files" from APL2 format to APL/VE format. Installation of AFIFIX is discussed in Section 3.1.2 of this SRB.

   APL2 structured files are NOS random files with a format that only APL2 understands. If the user attempts to bring a random file over to NOS/VE using get_file, the records get packed together, and as a result the random file directory becomes meaningless, and can no longer be used to locate the records. AFIFIX copies files to get rid of EORs so that they can be brought across to NOS/VE without their random file directories becoming out of synch with the data. So the user must apply AFIFIX to all APL structured files <u>on the NOS side, before</u> using the APL/VE conversion utility on them. To use AFIFIX, enter:

```
GET,AFIFIX/UN=APLO.
and then either
  AFIFIX,LFN1.                    (to only get the control byte
                                   set, which is why current
```

or AFIFIX,NOEOR,LFN1. APL2 users use AFIFIX). (to get the control byte set and to get the file copied without EORs so it can be brought over to APL/VE via the conversion utility).

5.8 FILE_MIGRATION_AID

1. Do not use 'CLEAR' or 'RETURN,*' to clean up NOS files. This will delete internal FMA files and cause FMA to fail. Also, executing an EXIT control card will cause FMA to fail. This item is a warning only and will also appear in the Migration Guide.

5.9 SORT

1. When a predefined key name, e.g., COBOL6, is used for a user specified collating sequence name, diagnostic message 610110 may show up twice to indicate the error though one error message is sufficient.
   PSR SM8A105

2. If a nonexistent file is specified by the DIR parameter of SORT/MERGE, an empty file by that name will be created by the system.

5.10 EQBIRAN

In order to allow comparisons of Hollerith data stored in real variables, the code for .EQ. and .NE. relational expressions with real operands has been changed to do a bit-by-bit comparison of the operands, rather than a floating point comparison. This change emphasizes the need to maintain all real values as standard normalized floating point numbers or zero in order to ensure correct compulations. The compiler automatically normalizes all real constants and the results of all floating point operations with normalized operands will be normalized, but nonstandard or unnormalized values can be introduced by means of boolean expressions, equivalencing to other data types or various input operations.

5.11 LISP

1. At this release, LISP/VE is a partial implementation of Common Lisp. The functions implemented for R1.1.2 were chosen to provide the user an adequate working base for common programming

applications. The list of functions currently supported is available in the LISP/VE Language Definition Usage Supplement, publication number 60486213.

2. Performance will not be optimal until the compiler is available at a later release. In particular, the print function, when implemented with the full capabilities was slow. LISP/VE therefore provides a faster (but less capable) version of the print function, named "print", and the complete version has been named "slow-print". Patience is important until the compiler is complete.

3. When the system editor is called from within LISP/VE, the editor parameter I=COMMAND must be used. For example:
(VE-COMMAND "EDIF MYFILE I=COMMAND").

4. NCONC causes LISP to hang if passed a circular LIST. User break can be used to terminate LISP when this happens.

5. When a file F contains an unmatched vertical bar, for example: (setq a |x), loading that file causes LISP to hang.

6. LISP aborts if it runs out of available space. This happens because NOS/VE aborts LISP instead of giving it control when a segment overflows.

7. NOS/VE will hang when it is given a line of more than 2000 or so characters. To exit when this occurs one must disconnect the terminal session, sign back on, and execute a terminate_job NOS/VE command to kill the job. The displaying of the stack sometimes causes this to happen.

5.12 OBJECT_CODE_UTILITY

1. A new version (V1.1) of object code library format has been implemented to support improved message template capabilities. Message template modules are now created through the OCU CREATE_MESSAGE_MODULE subcommand which places message template modules on the object library. The OCU on the R1.1.2 system can accept object modules from both old (V1.0) and new (V1.1) libraries but will only generate new (V1.1) format libraries. Libraries created by the R1.1.2 OCU cannot be used by previous versions of the system or previous versions of OCU. Object modules (but not SCL procedures or program descriptions) can be moved from a V1.1 library to a V1.0 library as follows:

a. Using the V1.1 CREOL, do the following:

```
CREOL
ADDM v11_library "version V1.1 library"
GENL object_file f=f "generates an object file"
```

b. Using the V1.0 CREOL, do the following:

```
CREOL
ADDM object_file "generated as above"
GENL v10_library "version V1.0 object library"
```

5.13 MESSAGE_TEMPLATES

1. Previous versions of message template modules cannot be used by the R1.1.2 system. They must be rebuilt using the GENERATE_MESSAGE_TEMPLATE_MODULE procedure. A MODULE statement must be placed at the beginning and a MODEND statement at the end of the message template definitions in order to provide the name for the message template module on the OCU library. The message template module should be placed on the same library as the program description and CYBIL modules for the application program.

5.14 PROGRAM_INTERFACE

1. Changes to the program interface require the recompilation of all CYBIL modules. Modules which have not been recompiled may encounter 'declaration mismatch' errors when they are loaded. In addition, any modules bound with runtime libraries will have to be rebound.

6.0 HOST SYSTEM NOTES

6.0 HOST SYSTEM NOTES

6.1 NOS R2.4.1

1. The BATCHIO printer driver does not correctly handle long lines (over 135 characters). The extra characters are either dropped, overprinted, or cause the page to be spaced erratically.
   PSR NS2C121

6.2 NOS/BE L627 OPERATING SYSTEM

1. The NOS/BE terminal driver always issues a line feed (LF) in response to a carriage return (CR) sent from a terminal. This causes a problem when using the NOS/VE Full Screen Editor (FSE) with terminals which consider the home position (command line) to be the last line of the display and do not have a page mode. For terminals like this, such as the CDC 722-10, the user should clear the screen (followed by a CR) to cause the NOS/VE FSE to repaint the entire screen.

2. NOS/VE fails to deadstart after a NOS/BE checkpoint/recovery deadstart due to checkpoint flags set in the Environment Interface face Communication Block.

   Solution:

   Install NBOE477 from SOLVER.

3. NVE subsystem control point hangs after a level 3 recovery deadstart due to coding error in 1AB.

   Solution:

   Install NBOE481 from SOLVER.

4. The DST1 job on the installation decks PL will not install NOS/VE binaries correctly.

   Solution:

   Install the following modification to the installation decks:

```
=IDENT C7535DST
=B HISTORY.2
```

6.0 HOST SYSTEM NOTES
6.2 NOS/BE L627 OPERATING SYSTEM

```
C7535DST  CORRECT VE ROUTINE EDITLIB.
          JWB    03/15/85      DST1

=C HISTORY
=D C7535DK.7                        NEAR DST1.315
REPLACE(FASLAVE,NUC,AL=1,FL=23100)
=D C7535DK.18                       NEAR DST1.629
REPLACE(DSMRUN,NVE,AL=1,FL=44000)
=D C7535DK.21,31
SETAL(CLOSLNK,0)
SETAL(GETLNK,0)
SETAL(GETNLNK,0)
SETAL(GETPLNK,0)
SETAL(MLIF,0)
SETAL(OPENLNK,0)
SETAL(PUTLNK,0)
SETAL(PUTNLNK,0)
SETAL(PUTPLNK,0)
SETAL(WREPLNK,0)
SETAL(IC7MMLI,0)
=C DST1
```

7.0 FCA LEVELS

7.0 FCA LEVELS

NOS/VE V1.1.2 level 630 was tested in an environment containing the following hardware and software components.

COMPONENT	RELEASE LEVEL
7155/855 (FMD)	MA721-A10
7021/67X (FIRM66X)	MB434-A14
CYBER Initialization Package	CIP003
NOS	V2.4.1 level 630
CML	Level 170
NOS/BE	V1.5 level 627

The following components are certified at the indicated levels:

MODEL	FCA Index #	CIP	CML	NOS	NOS/VE	NOS/BE
CYBER 810	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627
CYBER 815	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627
CYBER 825	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627
CYBER 830	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627
CYBER 835	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627
CYBER 840	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627
CYBER 845	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627
CYBER 850	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627
CYBER 855	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627
CYBER 860	TBD	003	170	V2.4.1 L630	V1.1.2 L630	V1.5 L627

8.0 FUTURES

8.0 FUTURES

This section is to communicate some future customer impacts expected in and prior to NOS/VE R1.1.2., allowing the customer to be better prepared at the time of that release.

8.1 NOS/VE RELEASE 1.1.2 DUAL STATE COMBINATIONS

NOS/VE R1.1.2 will be certified with NOS 2.4.1 as its dual state partner. Based on Control Data's software support policy, NOS/VE R1.1.2 will also be supported in critical PSR mode only with the dual state partner NOS Release 2.3 Level 617.

8.2 ROUTE PARAMETER CHANGE

IRHF allows a user on NOS to route a NOS/VE batch job as follows:

```
    ROUTE,filename,DC=LP,FC=RH
  or
    ROUTE,filename,DC=IN,ST=NVE
```

For NOS/VE R1.1.3, the capability to use the first route (route to output queue) will be removed.

8.3 SCU

At NOS/VE R1.1.3, the INFORMATION_LEVEL, IL, parameter of all SCU commands will be changed to DISPLAY_OPTION, DO.

A1.0 SAMPLE PROCEDURES FOR DISPLAYING FAILURE DATA

A1.0 SAMPLE_PROCEDURES_FOR_DISPLAYING_FAILURE_DATA

The following SCL procedures are provided as an example of how to use DISPLAY_BINARY_LOG to obtain failure data. It is necessary to include $system.osf$site_command_library in the user's command list to use DISPLAY_BINARY_LOG.

A1.1 DISPLAY_DISK_FAILURE_DATA

```
PROC display_disk_failure_data ,disdfd ( ..
    log,i: file =$local.$engineering_log; ..
    output,o: file = $output)
display_binary_log i=$value(log) o=$value(output)
define_group system_action statistic=CM350000
define_group software_failure statistic=CM350001
define_group $7155_failure statistic=CM350003
dump_group group=system_action cf=((1..4,8),(5..7,10))
dump_group group=software_failure cf=((1..4,8),(5..8,10))
dump_group group=$7155_failure ..
  cf=((1..4,8),(5..10,10),(11..16,8),(17,10),(18..60,8))
quit

PROCEND display_disk_failure_data
```

A1.1.1 DISPLAY_TAPE_FAILURE_DATA

```
PROC display_tape_failure_data ,distfd ( ..
    log,i: file =$local.$engineering_log; ..
    output,o: file = $output)

display_binary_log i=$value(log) o=$value(output)
define_group group=$7021_failure statistic=CM351002
dump_group group=$7021_failure ..
    cf=((1..4,8),(6..9,10),(10..25,8),..
    (26..28,10),(29,16),(30,10),(31..62,8))
quit

PROCEND display_tape_failure_data
```

A2.0 RECONFIGURATION

A2.1 ASSUMPTIONS

1. The reader is already familiar with the NOS/VE installation and deadstart process but is not familiar with what to do in particular failure situations.
2. The system analyst will look at the NOS/VE operator console for failure information in the event of a system failure. If the system has not failed but peripheral failures are suspected, the system analyst will look at the Engineering Log for failure information.
3. The site has generated a deadstart tape with the specified physical and logical configurations as depicted below. Unless otherwise specified, assume the site has installed NOS/VE, has been running the system, has permanent files on disk and has backup tapes of the permanent files.

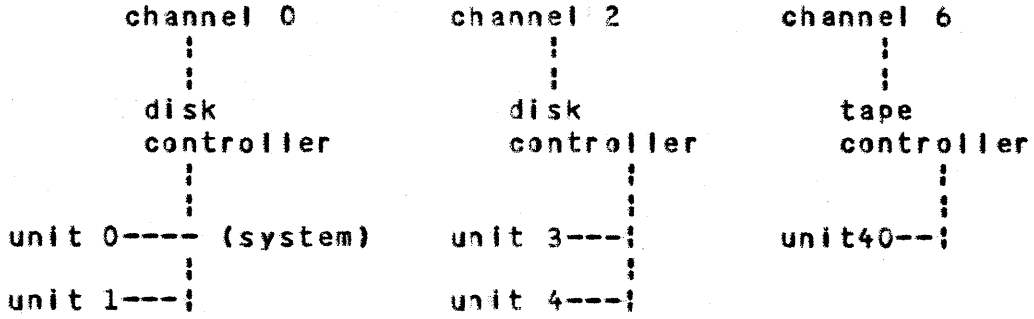
A2.2 FAILURES NOT REQUIRING RECONFIGURATION

4. If the performance of the system is degrading and there is an MDD console, see if unrecovered disk failure(s) have been reported. Alternatively, use the DISPLAY_BINARY_LOG command to determine whether unrecovered disk/tape failures have been reported in the Engineering Log.
5. The first alternative to consider when a peripheral failure is suspected is to terminate NOS/VE (if still running) and revert to NOS (single-state execution). When NOS/VE is terminated, all devices remain DOWN in the NOS configuration. Therefore MALET may be used on NOS to isolate and repair peripheral hardware problems. If the problem is a hardware logic failure and is repaired, one should be able to perform a continuation deadstart and resume normal operation. If this alternative is not chosen and the failing element is the system deadstart controller or disk drive, any modified data in NOS/VE real memory at the time of the failure will be lost.
6. If repair will take too long or files will be lost as a result of the failure, the discussion below is pertinent.

A2.0 RECONFIGURATION
A2.3 CONFIGURATION #1 - SINGLE CONTROLLER

A2.3 CONFIGURATION #1 - SINGLE CONTROLLER

Note that the disk and tape units are only accessible from one controller. This is not a recommended configuration from the standpoints of fault tolerance and system performance. Compare the following scenarios with those of configuraton #2.



A2.3.1 DISK RECONFIGURATION SCENARIOS

Listed below are scenarios for possible disk failures which require a NOS/VE deadstart or which occur while deadstarting NOS/VE. Assume that in each case the desired recovery action is to change the configuration and resume operations as soon as possible. In the discussion below a 'bad HDA' refers to the disk pack mounted on a disk drive; failures such as head-crash or a bad head may cause data to be recorded improperly or to be incapable of being read.

1. Assume 1) a bad HDA on disk #1 (not system device) or 2) a failure in disk drive #1 which will take too long to repair and no spare disk drive. The desired result is to delete disk #1 and continue degraded operation with only 1 disk (#0) on channel #0.

    a. Rebuild the NOS/VE deadstart tape with a new configuration prolog. The new prolog should have the failing device in the physical configuration but not in the logical configuration, i.e. the INSTALL_LOGICAL_CONFIGURATION subcommand of the LOGICAL_CONFIGURATION_UTILITY (LCU) should 'exclude' the failing element. In addition the configuration prolog may need to be changed if it was set up to INITIALIZE_MS_VOLUME and ADD_VOLUME_TO_SET for the failing element; these subcommands must be deleted.
    b. Perform an 'installation deadstart' of NOS/VE.
    c. Reload permanent files. Since you now have only 3 disks,

you may need to reload files for a subset of the users and tell the rest of your users to wait for the fourth disk to be repaired.

d. Resume operation in a degraded mode.
e. When the failing disk has been repaired, execute the LCU to re-install the logical configuration to 'include' the repaired element. Note that the newly installed logical configuration will not become active until the next continuation deadstart.
f. When convenient do a TERMINATE_SYSTEM and then a continuation deadstart. This will activate the logical configuration installed above.
g. Execute the LCU to INITIALIZE_MS_VOLUME and ADD_VOLUME_TO_SET for the repaired element.
h. Reload any permanent files not previously reloaded.
i. Continue with normal system operation.

2. Assume 1) a bad HDA on disk #0 (system device) or 2) a failure in disk drive #0 which will take too long to repair and no spare disk drive. The desired result is to delete disk #0 and continue degraded operation with only 1 disk (#1) on channel #0.

a. Rebuild the NOS/VE deadstart tape on the NOS system. The SET_DEADSTART_DEVICE system-core command must be changed to reflect the type of unit and the unit-number (1) for the new system disk. The configuration prolog must also be rebuilt. The new prolog should have the failing device (0) in the physical configuration but not in the logical configuration, i.e. the INSTALL_LOGICAL_CONFIGURATION subcommand of the LCU should 'exclude' the failing element. In addition the configuration prolog may need to be changed if it was set up to INITIALIZE_MS_VOLUME and ADD_VOLUME_TO_SET for the new system device; these subcommands must be deleted.
b. Perform an 'installation deadstart' of NOS/VE.
c. Reload permanent files. Since you now have only 3 disks, you may need to reload files for a subset of the users and tell the rest of your users to wait for the fourth disk to be repaired.
d. When the failing disk has been repaired, execute the LCU to re-install the logical configuration to 'include' the repaired element. Note that the newly installed logical configuration will not become active until the next continuation deadstart.
e. When convenient do a TERMINATE_SYSTEM and then a continuation deadstart. This will activate the logical

configuration installed above.
f. Execute the LCU to INTITIALIZE_MS_VOLUME and ADD_VOLUME_TO_SET for the repaired element.
g. Reload any permanent files not previously reloaded.
h. Continue with normal system operation.

3. Assume 1) a bad HDA on disk #3 (not system device) or 2) a failure in disk drive #3 which will take too long to repair and no spare disk drive. The desired result is to delete disk #3 and continue degraded operation with only 1 disk (#4) on channel #2.

   a. Rebuild the NOS/VE deadstart tape on the NOS system. The configuration prolog will need to be changed. The new prolog should have the failing device in the physical configuration but not in the logical configuration, i.e. the INSTALL_LOGICAL_CONFIGURATION subcommand of the LOGICAL_CONFIGURATION_UTILITY (LCU) should 'exclude' the failing element. In addition the initialization prolog may need to be changed if it was set up to INITIALIZE_MS_VOLUME and ADD_VOLUME_TO_SET for the failing device; these subcommands must be deleted.
   b. Perform an 'installation deadstart' of NOS/VE
   c. Reload permanent files. Since you now have only 3 disks, you may need to reload files for a subset of the users and tell the rest of your users to wait for the fourth disk to be repaired.
   d. When the failing disk has been repaired, execute the LCU to 'include' the failing element in the installed logical configuration. Note that the newly installed logical configuration will not become active until the next continuation deadstart.
   e. When convenient do a TERMINATE_SYSTEM and then a continuation deadstart. This will activate the logical configuration installed above.
   f. Execute the LCU to INITIALIZE_MS_VOLUME and ADD_VOLUME_TO_SET for the repaired element.
   g. Reload any permanent files not previously reloaded.
   h. Continue with normal system operation.

4. Assume the controller on channel #0 requires repair and that the repair will take too long. The desired result is to delete the failing controller from the configuration and continue degraded operation with the single controller on channel #2.

   a. Have a Customer Engineer recable units 0 and 1 to the

controller on channel 2.

b. Rebuild the NOS/VE deadstart tape on the NOS system. A new configuration prolog is required. The new prolog should show units 0 and 1 being connected to the controller on channel 2. The configuration prolog should also be changed to 'exclude' channel 0 from the installed logical configuration. The SET_DEADSTART_CONTROLLER system-core command may need to be changed if the controller on channel #2 has a different product_identification than the one on channel #0.
c. Do a SETVE to change the deadstart channel number from 0 to 2.
d. Perform an 'installation deadstart' of NOS/VE.
e. Reload permanent files.
f. Continue degraded system operation.
g. When the failing controller has been repaired, use the INSTALL_PHYSICAL_CONFIGURATION subcommand of the PHYSICAL_CONFIGURATION_UTILITY to re-install the original physical configuration. Note that the newly installed physical configuration will not become active until the next continuation deadstart.
h. When convenient do a TERMINATE_SYSTEM. Recabling cannot occur while the NOS/VE system is executing.
i. Recable to match the installed physical configuration.
j. Perform a continuation deadstart.
k. Continue with normal system operation.

5. Assume the controller on channel #2 requires repair and the repair will take too long. The desired result is to delete the failing controller from the configuration and continue with the remaining controller on channel #0.

a. Have a Customer Engineer recable units 3 and 4 to the controller on channel #0.
b. Rebuild the NOS/VE deadstart tape on the NOS system. A new configuration prolog is required. The new prolog should show units 3 and 4 being connected to the controller on channel #0. The configuration prolog should also be changed to 'exclude' channel 2 from the installed logical configuration.
c. Perform an 'installation deadstart' of NOS/VE.
d. Reload permanent files.
e. Continue degraded system operation.
f. When the failing controller has been repaired, use the INSTALL_PHYSICAL_CONFIGURATION subcommand of the

PHYSICAL_CONFIGURATION_UTILITY to re-install the original physical configuration. Note that the newly installed physical configuration does not become active until the next continuation deadstart.

g. When convenient do a TERMINATE_SYSTEM. Recabling cannot occur while the NOS/VE system is executing.
h. Recable to match the installed physical configuration.
i. Perform a continuation deadstart. This will activate the physical configuration installed above.
j. Continue with normal system operation.

6. Assume 1) the HDA on disk #1 (not system device) is not damaged and 2) the failure in disk drive #1 will take too long to repair and 3) a spare disk drive. The desired result is to delete the failing device, to move the disk pack from the failing drive to the spare drive, and to continue normal operation.

   a. This assumes that a spare disk drive has been physically configured and that the disk pack (HDA) normally mounted on the drive is not currently used by NOS/VE, i.e. the disk drive is defined in the installed physical configuration but not in the installed logical configuration. If these assumptions are not valid, then follow one of the earlier scenarios concerning a failing disk drive.
   b. Remove the disk pack (HDA) from drive #1 and mount it on the spare drive.
   c. Perform a continuation deadstart and request operator intervention.
   d. From within the LCU use the INSTALL_LOGICAL_CONFIGURATION subcommand to 'exclude' drive #1 and 'include' the spare drive.
   e. Continue normal system operation.

7. Assume 1) the HDA on disk #0 (system device) is not damaged and 2) the failure is in disk drive #0 which will take too long to repair and 3) a spare disk drive. The desired result is to delete the failing device, to move the disk pack (HDA) from the failing device to the spare device and to continue normal operation. Note that 885-1x (FMD) HDAs can be removed by a CE and moved to a spare drive. Operators can move 844-4x drives themselves.

   a. This assumes that a spare disk drive has been physically configured and that the HDA normally mounted on the storage device is not currently used by NOS/VE, i.e. the disk drive is defined in the installed physical configuration but not

in the installed logical configuration. If these assumptions are not valid, then follow one of the earlier scenarios concerning a failing disk storage element.

b. Remove the disk pack (HDA) from drive #0 and mount the pack on the spare drive.
c. If the disk pack was moved to channel #2 use the SETVE command to change the deadstart channel from 0 to 2.
d. Perform a continuation deadstart and request operator intervention.
e. If necessary change the SET_DEADSTART_CONTROLLER system-core command to specify the product-identification of the controller.
f. Change the SET_DEADSTART_DEVICE system-core command to identify the unit number of the spare drive.
g. From within the LCU use the INSTALL_LOGICAL_CONFIGURATION subcommand to 'exclude' drive #0 and 'include' the spare drive.
h. Continue normal system operation.

A2.3.2 TAPE RECONFIGURATION SCENARIOS

Listed below are scenarios for possible hardware failures which may occur (or be detected) while using tape peripherals as configured in configuration #1.

1. Assume a failure in the tape controller has caused a NOS/VE interruption. The desired result is to remove the failing tape controller from the NOS/VE configuration to allow diagnosis and repair to occur in the NOS state concurrent with NOS/VE execution.

    a. Perform a continuation deadstart and request operator intervention.
    b. From within the LCU, re-install the logical configuration and 'exclude' channel 6.
    c. Continue operation without the tape controller.
    d. When the tape controller has been repaired, execute the LCU to re-install the logical configuration and 'include' channel #6. Note that the newly installed logical configuration will not become active until the next continuation deadstart.
    e. When convenient do a TERMINATE_SYSTEM and perform a continuation deadstart to activate the newly installed logical configuration.
    f. Continue with normal operation.

2. Assume a failure in the tape controller which does not cause a NOS/VE interruption. The desired result is to remove the failing controller from the NOS/VE configuration to allow diagnosis and repair of the element to occur in the NOS state concurrent with NOS/VE execution.

    a. Execute the LCU to re-install the logical configuration and 'exclude' channel #6. Note that the newly installed logical configuration does not become effective until the next continuation deadstart.
    b. When convenient do a TERMINATE_SYSTEM and perform a continuation deadstart.
    c. Continue operation without the tape controller.
    d. When the tape controller has been repaired, repeat the process above to 'include' channel #6.

3. Assume that tape unit #40 has failed. Also assume that this failure has not caused a NOS/VE interruption. The desired result is to replace unit #40 with a different tape unit from the NOS configuration, use the NOS state for diagnosis and repair of the failing element and continue NOS/VE execution.

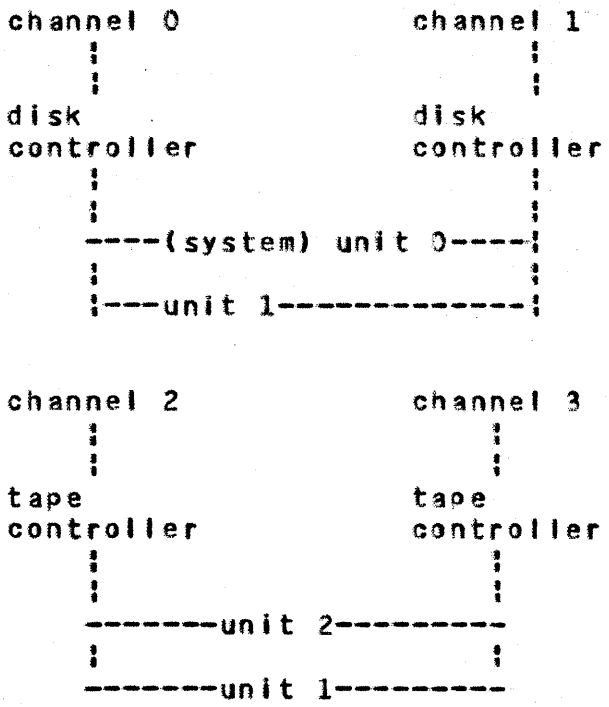
    a. Use the PCU to re-install the physical configuration omitting the failing tape unit and adding the replacement tape unit. Note that the newly installed physical configuration will not become active until the next continuation deadstart.
    b. Use the LCU to re-install the logical configuration. Note this is necessary because the LCU captures some of the information in the installed physical configuration which changed above.
    c. When convenient do a TERMINATE_SYSTEM and a continuation deadstart.
    d. Continue with normal system operation.

4. Assume that tape unit #40 has failed and the failure caused a NOS/VE interruption. The desired result is to replace unit #40 with a different tape unit from the NOS configuration, use the NOS state for diagnosis and repair of the failing element and continue NOS/VE execution.

    a. Perform a continuation deadstart.
    b. After deadstart run the PCU to install a new physical configuration which omits the failing tape unit and defines the replacement.

c. Do a TERMINATE_SYSTEM and perform another continuation deadstart to activate the newly installed physical configuration.
d. Continue normal system operation.

A2.4 CONFIGURATION #2 - MULTIPLE CONTROLLERS



Listed below are senarios for possible hardware failures that may require a NOS/VE deadstart or may occur while performing a NOS/VE deadstart. The following scenarios can also be applied to the case where the failure did not cause a NOS/VE interruption. In this case one can execute the LCU to 'exclude' a failing controller, do a TERMINATE_SYSTEM when convenient and then follow the steps below omitting the operator intervention and consequent LCU execution.

A2.4.1 DISK RECONFIGURATION SCENARIOS

The following describes the recommended approach for dealing with the failure of one of the two disk controllers in configuration #2 above.

1. Assume the site has configured to channel #0 unit #0 for the deadstart device and the controller on channel #0 fails. The desired result is to return the controller to the NOS state for diagnosis and repair and to continue degraded operation minus the failing controller.
   a. Execute SETVE command to change the deadstart channel number

from 0 to 1.
b. Perform a continuation deadstart and request operator intervention.
c. Change the SET_DEADSTART_CONTROLLER system-core command if the product_identification of the controller on channel #1 is different than that of the controller on channel #0.
d. From within the LCU use the INSTALL_LOGICAL_CONFIGURATION subcommand to 'exclude' channel #0.
e. Continue degraded system operation.
f. When the failing controller has been repaired, use the LCU to re-install the logical configuration and 'include' channel #0. Note that the newly installed logical configuration will not become active until the next continuation deadstart.
g. When convenient do a TERMINATE_SYSTEM and perform a continuation deadstart to activate the newly installed logical configuration.
h. Continue normal system operation.

2. Assume the controller on channel #1 fails. The desired result is to return the failing controller to the NOS configuration for diagnosis and repair and to continue with degraded system operation minus the failing controller.

a. Perform a continuation deadstart and ask for operator intervention.
b. From within the LCU use the INSTALL_LOGICAL_CONFIGURATION subcommand to 'exclude' channel #1.
c. Continue degraded system operation.
d. When the failing controller has been repaired, use the LCU to re-install the logical configuration and 'include' channel #1. Note that the newly installed logical configuration will not become active until the next continuation deadstart.
e. When convenient do a TERMINATE_SYSTEM and perform a continuation deadstart to activate the newly installed logical configuration.

A2.4.2 TAPE RECONFIGURATION SCENARIOS

The following is the recommended approach for dealing with the failure of one of two tape controllers in configuration #2 above.

1. Assume one of the controllers on either channel #2 or channel #3 fails. The desired result is to return the failing tape

controller to the NOS state for diagnosis and repair and to continue degraded operation minus the failing controller.

a. Perform a continuation deadstart and ask for operator intervention.
b. From within the LCU use the INSTALL_LOGICAL_CONFIGURATION subcommand to 'exclude' the channel with the failing controller.
c. When the failing controller has been repaired, use the LCU to re-install the logical configuration and 'include' the channel with the failing controller. Note that the newly installed logical configuration will not become active until the next continuation deadstart.
d. When convenient do a TERMINATE_SYSTEM and perform a continuation deadstart to activate the newly installed logical configuration.

B1.0 PERIPHERAL MAINTENANCE PROCEDURES

B1.0 PERIPHERAL_MAINTENANCE_PROCEDURES

B1.1 INTRODUCTION

Failure data for peripheral devices dedicated to NOS/VE are emitted as statistics to $ENGINEERING_LOG in binary format (Section A1-1). The statistics facility command DISPLAY_BINARY_LOG provides a mechanism for monitoring those errors encountered during system operation. While this facility can be used by anyone via SCL commands, the format of the output does not describe the status fields in sufficient detail to permit easy interpretation. Three independently executable SCL procedures are provided that produce several reports by editing the output of DISPLAY_BINARY_LOG. These procedures are included in library $SYSTEM.OSF$SITE_COMMAND_LIBRARY. A certain amount of execution time overhead is incurred as a result of using SCL to do the editing, but the command structure is easily modified to include new statistics that may be added in subsequent releases.

B1.2 USAGE

These procedures are initiated interactively by the commands DISPLAY_HPA_SUMMARY (DISHS), DISPLAY_HPA_DISK_DETAIL (DISHDD), and DISPLAY_HPA_TAPE_DETAIL (DISHTD). There are no required parameters. Default output is $OUTPUT. Default input is the current system $ENGINEERING_LOG. Alternative files many be specified by the following parameters:

```
input, i : file
output, o : file
```

These procedures should be initiated immediately prior to terminating $ENGINEERING_LOG via the TERMINATE_LOG command. Alternatively, the file created by the TERMINATE_LOG command should be used as input to these procedures to insure that all statistics have been evaluated. See the SAMPLE BATCH PROCEDURE described later in this section.

B1.3 DISPLAY_HPA_SUMMARY (DISHS)

This procedure displays summary information in the following format:

B1.0 PERIPHERAL MAINTENANCE PROCEDURES
B1.3 DISPLAY_HPA_SUMMARY (DISHS)

<DISPLAY_BINARY_LOG header>

descriptive data	count
descriptive data	count
..	

The descriptive data has the following format:

<mf>.<pp>.<channel>.<controller>.<unit>*<vsn>*<severity>*<symptom>

where: severity can be UF (unrecovered failure) or RF (recovered failure). Entries will appear in ASCII collating sequence of the descriptive data string.

This report uses the DISPLAY_BINARY_LOG display_descriptive_data subcommand. The report provides up to five sections depending on the types of statistics contained in $ENGINEERING_LOG. These sections can be used to determine if the number of statistics reported against a subsystem warrant further inspection or maintenance activity.

The descriptive data in this report is also the first line in the tape and disk detail reports. It will be truncated at 56 characters and result in loss of message text when long <mf>, <controller>, or <unit> values are used. PSR NVOE985 documents this problem. A value of 99 for PP number will occur when the failure occurs on the system deadstart device.

B1.4 DISPLAY_HPA_DISK_DETAIL (DISHDD)

This procedure displays disk statistics in the following format:

B1.0 PERIPHERAL MAINTENANCE PROCEDURES
B1.4 DISPLAY_HPA_DISK_DETAIL (DISHDD)

<DISPLAY_BINARY_LOG header>

<statistic number> <julian date>

FCN	TY	E	UN	CYL	TK	SC	GEN STAT	DETAIL STATUS 1	2	3	4	5	6	7	8	9	10
hh:mm:ss descriptive data																	
fcn	ty	e	un	cyl	tk	sc	stat/	/	/	/	/	/	/	/	/	/	/
ERROR POS				cyl	tk	sc	/	/	/	/	/	/	/	/	/	/	/
FINAL STAT		UNREC ERR					stat/	/	/	/	/	/	/	/	/	/	/
							/	/	/	/	/	/	/	/	/	/	/

OPCD= REQUEST RTY= SECTOR RTY= RESIDUAL CNT=

where: descriptive data is the same as in the summary report (up to 80 characters long), and the remaining lines are described as follows:

<fcn><type><eq><unit><cyl track sector(INITIAL)><gen status> + det stat 1-10
ERROR POS <cyl track sector(FAILURE)> + det stat 11-20
FINAL STAT UNREC ERR <gen stat(FAILURE)> + 2 X 10 det stat (only when UF)

This report uses the DISPLAY_BINARY_LOG generate_group_file subcommand. The report provides up to four sections depending on the types of statistics contained in $ENGINEERING_LOG. The entries in each section appear in chronological order. The values in the descriptive data, counts, opcd and equipment type are decimal. All other values are octal. All fields are initialized to blanks, and only valid data is quantified.

B1.5 DISPLAY_HPA_TAPE_DETAIL (DISHTD)

This procedure displays tape statistics in the following format:

B1.0 PERIPHERAL MAINTENANCE PROCEDURES
B1.5 DISPLAY_HPA_TAPE_DETAIL (DISHTD)

<DISPLAY_BINARY_LOG header>

<statistic number> <julian date>

								STATUS							
OPCD	TYP	PP	CH	EQ	UN	RTY	DEN	GS1	GS2	DS3	DS4	DS5	DS6	DS7	DS8
hh:mm:ss descriptive data															
opcd	typ	pp	ch	eq	un	rty	den	/	/	/	/	/	/	/	/
FORMAT=								/	/	/	/	/	/	/	/
BLOCK COUNTS		CURR=				READ=				WRITTEN=					

where: descriptive data is the same as in the summary report (up to 80 characters long). Refer to Section A-5.6 for further details.

This report uses the DISPLAY_BINARY_LOG generate_group_file subcommand. Currently the only statistic being logged for the tape subsystem is unrecovered 679 errors. The entries will appear in chronological order. The values in the descriptive data, counts, opcd and equipment type are decimal. All other values are octal. All fields are initialized to blanks, and only valid data is quantified.

B1.6 SAMPLE_BATCH_PROCEDURE

a. Create a permanent file containing the following procedure:

```
PROC run_dish (
  input, i : file = $local.$engineering_log
  output, o: file = $output
  status)

  display_hpa_disk_detail $value(input) $value(output)
  display_hpa_tape_detail $value(input) $value(output)
  display_hpa_summary $value(input) $value(output)

PROCEND run_dish
```

b. To run this procedure from a terminal and display output to the terminal:
   1. Attach the file created in step a. as run_dish
   2. Enter "run_dish"

c. To run this procedure from a terminal and route the output to a printer:

1. Attach the file created in step a. as run_dish
2. Enter "run_dish o=list.$eoi"
3. Enter "print_file list"

d. To run this procedure from a terminal with an alternate input file, and route the output to a printer:
   1. Attach the file created in step a. as run_dish
   2. Enter "run_dish i=<file> o=list.$eoi"
   3. Enter "print_file list"

NOTE - enter the characters between the quotes(") followed by a carriage return, except for <file> which must be a file created by the TERMINATE_LOG command.

C1.0 SUPPORT OF 7154 CONTROLLER

C1.0 SUPPORT OF 7154 CONTROLLER

The following information is provided for those sites installing NOS/VE on a configuration using 7154 controllers. Users should be aware that NOS/VE support of 7154 controllers is planned to be dropped in the future. This information is supplied as an appendix to the SRB and will not be incorporated into any manuals.

To use a 7154 controller in a NOS/VE system, make the following changes:

1. Use $7154_1 as the Product_Identification in Configuration Prolog files.

2. Use $7154_1 when defining the deadstart controller.

D1.0 COMMAND TABLE GENERATOR

D1.0 COMMAND TABLE GENERATOR

D1.1 CALLING THE COMMAND TABLE GENERATOR

The generator is called via the following NOS/VE command.

```
generate_command_table ! generate_command_tables ! genct
     [input=<file reference>]
     [output=<file reference>]
     [status=<status variable>]
```

input ! i: This parameter specifies the file from which the generator is to read the command table defintition.

Omission causes $INPUT to be used.

output ! o: This parameter specifies the file to which the generator is to write the CYBIL declarations that represent the command table.

Omission causes $OUTPUT to be used.

status: See ERROR HANDLING.

D1.2 INPUT TO THE GENERATOR

The input to the generator is in the form of a series "commands" that declare a new table or add entries to the "current table".

There are three different commands: TABLE, COMMAND and FUNCTION. Each TABLE "command" starts a new command or function table and and is followed by any number of COMMAND "commands" or any number of FUNCTION "commands". An empty table may be defined by omitting COMMAND and FUNCTION "commands".

D1.2.1 TABLE

The TABLE "command" declares a new table.

```
table name=<name>
     [type=command ! function]
     [section_name=<name>]
     [scope=local ! xdcl]
```

D1.0 COMMAND TABLE GENERATOR
D1.2.1 TABLE

[module=<name>]

name ; n: This parameter specifies the name of the table. This name can be no longer than 24 characters.

type ; t: This parameter specifies the type of table to be generated. Either a Command (C) or Function (F) table be selected.

Omission causes a Command table to be generated.

section_name ; sn: This parameter can be used to specify a section name for the CYBIL variable declaration. The specified section must be read-only.

Omission causes no section name to be used.

scope ; s: This parameter specifies the scope of the CYBIL variable declaration. Specifying XDCL causes that attribute to be used in the CYBIL variable declaration. Specifying LOCAL causes the STATIC attribute to be used.

Omission causes LOCAL to be used.

module ; m: This parameter specifies the name to be given to a module to contain the command or function table. If this parameter is given, CYBIL module/modend statements are generated around the table declarations along with the appropriate SCU *COPY directives to reference the type definition decks. If the parameter is omitted, no module/modend statements or *copy directives are included in the generated output.

D1.2.2 COMMAND

The COMMAND "command" declares a group of entries in a command table. The order in which the names for the entries are provided has significance when the table is used by the system supplied DISPLAY_COMMAND_LIST command and in other similar contexts. The first entry name is taken to be the name "nominal" name for the command, i.e., the name by which it is usually known. The last entry name is taken to be the "abbreviation" for the command name. Any other entry names are considerred to be aliases for the command name.

command name=list of <name>
processor=<name>

```
        [call_method=local ¦ xref ¦ load ¦ proc ¦ program]
        [availability=advertised ¦ hidden]
        [log=automatic ¦ manual]
```

name ¦ names ¦ n: This parameter specifies the names by which the command can be called.

processor ¦ p: This parameter specifies the name of the processor for the command.

call_method ¦ cm: This parameter specifies how the processor for the command is to be invoked.

LOCAL - indicates that the processor is a CYBIL procedure in the same module as the table.

XREF - indicates that the processor is a CYBIL procedure in a separate module from the table. This option causes an XREF procedure declaration to be generated for the processor.

LOAD - indicates that the processor is a CYBIL procedure that must be "dynamically loaded" (i.e., via pmp$load) before it can be called.

PROC - indicates that the processor is an SCL procedure on one of the object libraries that comprise the program.

PROGRAM - indicates that the processor is designated by a program description on one of the object libraries that comprise the program.

Omission causes LOCAL to be used.

availability ¦ a: This parameter specifies whether the command is Advertised (A) to users or Hidden (H) from them. A "hidden" command will not appear in a display of the command list entry.

Omission causes ADVERTISED to be used.

log ¦ l: This parameter determines whether the logging of a call to the command is Automatic (A), i.e. is done by the SCL interpreter, or Manual (M), i.e. is done by the command processor. Logging by a command processor is done in those situations where secure information potentially passed to the command should not be written

to the log.

Omission causes AUTOMATIC to be used.

D1.2.3 FUNCTION

The FUNCTION "command" declares a group of entries in a function table. The order in which the names for the entries are provided has significance when the table is used by the system supplied DISPLAY_COMMAND_LIST function and in other similar contexts. The first entry name is taken to be the name "nominal" name for the function, i.e., the name by which it is usually known. The last entry name is taken to be the "abbreviation" for the function name. Any other entry names are considerred to be aliases for the function name.

```
function name=list of <name>
        processor=<name>
        [call_method=local ! xref ! load ! proc]
        [availability=advertised ! hidden]
```

name ! names ! n: This parameter specifies the names by which the function can be called.

processor ! p: This parameter specifies the name of the processor for the function.

call_method ! cm: This parameter specifies how the processor for the function is to be invoked.

LOCAL - indicates that the processor is a CYBIL procedure in the same module as the table.

XREF - indicates that the processor is a CYBIL procedure in a separate module from the table. This option causes an XREF procedure declaration to be generated for the processor.

LOAD - indicates that the processor is a CYBIL procedure that must be "dynamically loaded" (e.g., via pmp$load) before it can be called.

PROC - indicates that the processor is an SCL procedure on one of the object libraries that comprise the program.

Omission causes LOCAL to be used.

availability ! a: This parameter specifies whether the function is Advertised (A) to users or Hidden (H) from them. A "hidden" function will not appear in a display of the function list.

Omission causes ADVERTISED to be used.

D1.3 OUTPUT

The output of the command table generator can best be described by an example.

Example Input

```
table example_command_table..
        section_name=oss$job_paged_literal scope=xdcl
command (generate_command_table, generate_command_tables, genct)..
        processor=clp$generate_command_table..
        call_method=xref
command (display_command_list, discl)..
        processor=clp$display_command_list..
        call_method=xref

table example_function_table type=function
function $time processor=clp$$time
function $peek processor=clp$$peek availability=hidden
function $file processor=clp$$file call_method=load
```

Output from Example Input

```
  VAR
    example_command_table: [XDCL, READ, oss$job_paged_literal]
          ^clt$command_table := ^example_command_table_entries,

    example_command_table_entries: [STATIC, READ,
          oss$job_paged_literal] array [1 .. 5] of
          clt$command_table_entry := [
    {} ['DISCL                              ', clc$abbreviation_entry,
          clc$advertised_entry, 2, clc$automatically_log,
```

```
          clc$linked_call, ^clp$display_command_list],
    {} ['DISPLAY_COMMAND_LIST          ', clc$nominal_entry,
          clc$advertised_entry, 2, clc$automatically_log,
          clc$linked_call, ^clp$display_command_list],
    {} ['GENCT                         ', clc$abbreviated_entry,
          clc$advertised_entry, 1, clc$automatically_log,
          clc$linked_call, ^clp$generate_command_table],
    {} ['GENERATE_COMMAND_TABLE        ', clc$nominal_entry,
          clc$advertised_entry, 1, clc$automatically_log,
          clc$linked_call, ^clp$generate_command_table],
    {} ['GENERATE_COMMAND_TABLES       ', clc$alias_entry,
          clc$advertised_entry, 1, clc$automatically_log,
          clc$linked_call, ^clp$generate_command_table]];

?? PUSH (LISTEXT := ON) ??

  PROCEDURE [XREF] clp$generate_command_table (
        parameter_table: clt$parameter_table;
    VAR status: ost$status);

  PROCEDURE [XREF] clp$display_command_list (
        parameter_table: clt$parameter_table;
    VAR status: ost$status);

?? POP ??

  VAR
    example_function_table: [STATIC, READ]
          ^clt$function_table := ^example_function_table_entries,

    example_function_table_entries: [STATIC, READ]
          array [1 .. 3] of
          clt$function_table_entry := [
    {} ['$FILE                         ', clc$nominal_entry,
          clc$advertised_entry, 1, clc$linked_call, ^clp$$file],
    {} ['$PEEK                         ', clc$nominal_entry,
          clc$hidden_entry, 2, clc$linked_call, ^clp$$peek],
    {} ['$TIME                         ', clc$nominal_entry,
          clc$advertised_entry, 3, clc$linked_call, ^clp$$file]];
```

D2.0 COMMAND FORMATTER

D2.0 COMMAND_FORMATTER

D2.1 INTRODUCTION

The command formatter is a tool used to format a file of SCL commands. The formatter will read an input file of SCL commands and will generate a formatted output file. The input file may consist of one or more SCL PROCs or may consist of only a portion of a PROC.

D2.2 CALLING_THE_COMMAND_FORMATTER

The formatter is called via the following NOS/VE command.

```
format_scl_proc ! format_scl_procs ! forsp
      [input=<file reference>]
      [output=<file reference>]
      [page_width=<integer>]
      [initial_indent_column=<integer>]
      [key_character=<character>]
      [utility_definition_file=<file reference>]
      [status=<status variable>]
```

input ! i: This parameter specifies the file from which the formatter is to read the SCL PROCs or commands. This file must have the following file attributes:

```
access_mode = read
file_contents = legible or unknown
file_processor = scl or unknown
file_structure = data or unknown
```

Omission causes $INPUT to be used.

output ! o: This parameter specifies the file to which the formatter is to write the formatted SCL PROCs or commands. This file must be accessible to write and must have the following file attributes:

```
file_contents = legible or unknown
file_processor = scl or unknown
file_structure = data or unknown
```

If the file has never been opened, any of the above 3 attributes specified as unknown will be set to the alternate value.

Omission causes $OUTPUT to be used.

page_width ; pw: This parameter specifies the page width of the output file. The value specified for this parameter is considered to be the right-hand margin of the output file.

Omission causes the value of the page_width attribute of the output file to be used if the value was explicitly specified. If the attribute was not explicitly specified, 110 will be used. NOTE: The value of the page_width parameter must be at least 64 greater than the value of the initial_indent_column parameter.

initial_indent_column ; iic: This parameter specifies the starting column of the first line written to the output file. This parameter is provided so that a section of commands (such as those appearing as input to COLLECT_TEXT) which wil not normally be formatted may be extracted from a PROC, formatted, and replaced in a formatted PROC.

Omission will cause 1 to be used.
NOTE: The value of the page_width parameter must be at least 64 greater than the value of the initial_indent_column parameter.

key_character ; kc: This parameter specifies the character which, if appearing in column 1 of the input line, will cause the entire line to be written to the output file without any other formatting processing.

Omission will cause * to be used.

utility_definition_file ; udf: This parameter specifies the file which contains definitions for additional utility commands which may appear in the input file. The utility names and terminators defined within in this file will be added to those built into the formatter.

Omission will cause only the built-in utilities to be used.

status: See ERROR HANDLING.

D2.3 INPUT TO THE FORMATTER

The input to the formatter is a file containing SCL commands. Usually the input consists of one or more PROCs - starting with a PROC statement and ending with a PROCEND statement; however it is not required that the input be complete PROCs. Any collection of SCL commands may be formatted, but SCL structure blocks within the input file must be complete or the formatter will report errors.

D2.3.1 PRAGMATS

Pragmats are special SCL comments in the input file which control the formatting process. A pragmat is an SCL comment with $ (dollar-sign) as the first character of the comment. That is, a pragmat begins with the characters "$ (double-quote dollar-sign) and is terminated with " (double-quote) or end-of-line. A pragmat may begin in any character position of a line but only blank characters may precede a pragmat in a line. The "command definition" of a pragmat (where "$ may be thought of as the "command") is:

"$ [command=<name>] [format=<boolean>]

command ; c: This parameter specifies the name of a utility command or a utility terminator. The purpose of this parameter is to inform the formatter that the utility name, or terminator, is "hidden" (such as being included in a file). Upon encountering a pragmat with this parameter, the formatter will proceed as though the name had been encountered in the input and will adjust indentation accordingly.

Omission causes no change in indentation.

format ; fmt; f: This parameter specifies formatting action after the pragmat is processed. Note that if there is a command on the line after the pragmat comment and separated from it by a semicolon, the specified action will apply to that command.

Omission causes no change in formatting action.

D2.3.2 UTILITY DEFINITION FILE

The formatter maintains a table of known SCL utility names and terminators. The utilities initially entered into this table are:

D2.0 COMMAND FORMATTER
D2.3.2 UTILITY DEFINITION FILE

Utility Names	Terminator Names
ADMINSTER_USER, ADMU	QUIT, QUI
BACKUP_PERMANENT_FILES, BACPF	QUIT, QUI
BUILD_REAL_MEMORY, BUIRM	QUIT, QUI
CREATE_INTERSTATE_CONNECTION, CREIC	QUIT, QUI, DELIC, DELETE_INTERSTATE_CONNECTION
CREATE_OBJECT_LIBRARY, CREOL	QUIT, QUI
EDIT_FILE, EDIF	QUIT, QUI, END
EDIT_LIBRARY, EDIL	QUIT, QUI, END
LINK_VIRTUAL_ENVIRONMENT, LINVE	QUIT, QUI
MEASURE_PROGRAM_EXECUTION, MEAPE	QUIT, QUI
RESTORE_PERMANENT_FILES, RESPF	QUIT, QUI
SOURCE_CODE_UTILITY, SCU	QUIT, QUI, END

The user may augment this table by generating a utility definition file and specifying that file on the formatter call. The file must consist of one line entries with the following "parameter list".

names ! name ! n: This parameter specifies a list of names for one utility. Utility names must be unique. This parameter is required.

terminators ! terminator ! t: This parameter specifies a list of names which terminate the utility.

Omission causes (quit,qui) to be used.

Utility names must be unique and may not be the same as any of the "built in" names. If the terminator names of a utility include more than QUIT and QUI then all terminators must be specified.

Examples

```
n=(my_utility, mu) t=(end,quit,qui)
names=(your_utility, yu)
its_utility stop
```

D2.4 OUTPUT

The formatter generates lines in the output file according to the following conventions.

1. If any input statement contains continuation lines, all continuation lines will be read and concatenated with the beginning line of the statement before formatting of that statement begins.

2. The "current indent column" will govern the starting column of the statement written to the output file. Initially the current indent column will be column 1 (unless overrriden by the initial_indent_column in the command call to the formatter).

3. If the statement to be output (starting at the current indent column) is of such length as to exceed the specified page width, then that statement will be broken at - if possible - a reasonable place and as many continuation lines as required will be generated.

4. A continuation line will be indented six (6) columns from the indent column of the first line of the statement.

5. Breaking a line at a "reasonable place" means that items which are closely related to adjoining items will not be placed on separate lines unless there is no alternative.

6. If an expression consisting of <operand><operator><operand> must be broken, it will be broken immediately after the operator - or the space following the operator if one exists.

7. In certain cases - a string too long to fit on the entire line, for example - indentation must be suppressed on the continuation line.

8. If a statement consists of an SCL control statement which defines the beginning of an SCL structure block (IF, for example), then all statements following such a statement will be indented two (2) spaces from the control statement until the corresponding end statement (IFEND, for example) is encountered.

9. If a statement is labeled, the label (followed by the continuation ellipses) will be placed on a line by itself indented two (2) columns less than the current indent column (unless, of course, the current indent column is less than 3).

10. SCL control statement identifiers (IF, WHILE, PROC, etc.) along with certain control words (DO, EXIT, etc.) will be written to the output file in uppercase. All other names (except certain utility names) will be written in lowercase.

11. Utilities will be treated as SCL structure blocks.

12. SCL comments will be retained and written to the output file. Comments at the beginning of a statement (before the command name) will be written as separate lines and indented to the current indent column. An exception will be comments starting in column 1 of the input file. Such comments will be written starting in column 1 of the output file.

13. Spaces will be added or deleted as follows:

    a.. A comma will be written with no leading space and with a trailing space.
    b.. The colon separating the parameter name from the parameter specification in a PROC declaration will have leading spaces deleted and will be follwed by a space.
    c.. Contiguous spaces on input will be output as one space.
    d.. A space will precede and follow the equals sign of an assignment statement.
    e.. Unnecessary spaces in a parameter list will not be written. That is "command par_name = par_value" will be written as "command par_name=par_value".
    f.. No rule in this sub-section will apply to characters inside a string or comment expression.

14. If multiple statements (separated by semi-colons) exist on an input line, each statement will be formatted separately and the semi-colons discarded.

15. If a statement terminator is optional (THEN of an IF statement, for example), the formatter will generate the terminator if it does not exist on the input statement.

16. COLLECT_TEXT (COLT) is treated as a special case since the text being collected may not be SCL statements. Whenever a collect_text command is encountered, the parameter list is scanned for the value of the UNTIL (U) parameter. If the UNTIL parameter is not specified, or is specified as a literal string, formatting is turned off until the line containing the specified string (or **) is read. All collect_text lines are copied to the output file without any other processing. If the UNTIL parameter is specified as a variable (other than a literal string), formatting will not be turned off (since there is no way of knowing when it is to be re-activated) and the results may be undefined. The collect_text command will be output starting in column 1 with the command name written in uppercase.

17. Any input line beginning with the key_character specified in the call to the formmatter will be written directly to the output file without any other processing attempted.

18. If a statement formatted for output exceeds the maximum command size (currently 256 characters), any indentation present in the output line(s) of the statement will be decremented until the maximum command size is not exceeded. If deleting all indentation for the statement does not result in an acceptable command size, a comment will be issued to the output file and the input line(s) of the statement written instead of the formatted line(s).

19. If a blank line does not follow the last line of a proc delcaration, a blank line will be added. If a blank line does not appear before a PROCEND statement, a blank line will be added.

20. Certain errors (such as unbalanced parentheses) are detected by the formatter by calls to modified versions of some SCL interpreter procedures. Such errors are reported to the output file in the format of system error messages. The message mode (set by set_message_mode) will be reflected in these messages.

21. If the formatter encounters a problem with a statement (such as not being able to determine the UNTIL value of COLLECT_TEXT), then a warning message will be written to the output file as a comment.

22. For each error or warning message written to the output file, the input line(s) in which the error was detected will immediately follow the message.

23. Certain information about a line in error may be retained by the formatter. For example, if a "WHEN" statement is recognized but an error is detected in some remaining portion of the statement, a WHEN block will still be established and will require a WHENEND statement.

24. The formatter keeps track of SCL structure blocks in a somewhat different manner than the SCL interpreter. Whenever a statement specifying an end of a block (such as WHILEND) is encountered, the formatter checks that the block exists. If it does and if any block was started within the block being terminated, then a formatter error message will be generated indicating the line number of the statement beginning the block. This means that if a PROCEND statement is encountered (and a PROC block exists), then any unterminated block will be noted in an error message.

25. In checking syntax and the like, the formatter assumes no knowledge of variable types or the like. For example, the statement

```
IF 'this is a string' = 255(10) THEN
```

will be considered to be acceptable. That is, the formatter will detect only the most obvious errors.

26. A count of error and warning messages issued will be maintained and reported to the status variable at termination of the formatter. If the error count exceeds a certain threshold (currently 25), the formatter will give up, feeling that it is probably processing unknown text.

D2.5 EXAMPLE:

Input:

```
proc aaa (
input,i : file = $REQUIRED
output, o : file=$OUTPUT
status)
```

D2.0 COMMAND FORMATTER
D2.5 EXAMPLE:

```
put_this _message_out 'this is a message to be put to ..
output' o=$output
edit_file xyz
1,,a
include_file abcc
 "$ quit
while_label: while abc>def do
if hij='???' then
create_object_library
add_module library=:nve.abcdefghijklmnopqrstuvwxyz1.abcdefghi..
jklmn2
generate_library a
quit
else;exit
ifend;whilend;procend


Output:

PROC aaa (
  input, i: file = $required
  output, o: file = $output
  status)

  put_this _message_out 'this is a message to be put to output' ..
        o=$output
  EDIT_FILE xyz
    1,, a
    include_file abcc
  "$ quit
while_label: ..
  WHILE abc > def DO
    IF hij = '???' THEN
      CREATE_OBJECT_LIBRARY
        add_module library=:nve.abcdefghijklmnopqrstuvwxyz1..
.abcdefghijklmn2
        generate_library a
      QUIT
    ELSE
      EXIT
    IFEND
  WHILEND while_label

PROCEND aaa
```

E1.0 EDIT_CATALOG

E1.0 EDIT_CATALOG

E1.1 USING_EDIT_CATALOG

EDIT_CATALOG (EDIC) is a full screen utility that allows you to view and to manipulate the contents of NOS/VE files and catalogs. EDIC uses the terminal attribute terminal_model to determine the screen interface. EDIC includes the following general activities.

- Viewing a catalog or a file.
- Copying a file.
- Renaming a file within a catalog.
- Moving a file to another catalog.
- Creating a new file or catalog.
- Switching to a different catalog.
- Deleting a catalog or a file.
- Sorting a catalog display.
- Locating a file within a catalog display.
- Executing a file.
- Printing a file.
- Displaying file attributes.
- Editing a file with FSE/VE.
- Executing NOS/VE commands.

E1.2 CALLING_EDIT_CATALOG

The format for calling EDIT_CATALOG is as follows.

```
EDIT_CATALOG (EDIC)
[CATALOG=catalog]
[DISPLAY_OPTION=keyword value]
[NO_DOLLAR_FILES=boolean_value]
[STATUS VARIABLE=status variable]
```

The CATALOG (C) parameter specifies the catalog for which information is to be displayed. If you do not specify a catalog name, the system displays $CATALOG.

The DISPLAY_OPTION (DO) parameter specifies how much information that you wish to have displayed. The display options are:

BRIEF (B)

E1.0 EDIT_CATALOG
E1.2 CALLING EDIT_CATALOG

Selects a display that shows files and their cycles only.

ALL (A)

Selects a display that shows files and the following information for each file: size, processor, file type, and last changed date and time.

If the DISPLAY_OPTION parameter is omitted, brief is selected.

NO_DOLLAR_FILES (NDF)

TRUE	Selects a display in which system file names (those beginning with a dollar sign) are not displayed.
FALSE	Selects a display that shows both system file names and user file names.

A sample EDIT_CATALOG call is shown below.

```
/EDIT_CATALOG c=$user do=all ndf
```

This entry returns a screen similar to the following sample display.

MY_CATALOG

Name.cycle	Size (KB)	Processor	File Type	Changed Date	Changed Time
Command_Library.2	24	OCU	Library	840813	10:55
Cobol_Binary	9	Unknown	Object	840821	09:41
External		Catalog			

F1	F2	F3	F4	F5	F6	F7	F8
PRINT VIEW	MOVE COPY	CREATE EDIC	XECUTE DELETE	UNDO	QUIT	SORT	LOCATE ATTRIB

This display shows several columns of information about files and

E1.0 EDIT_CATALOG
E1.2 CALLING EDIT_CATALOG

subcatalogs within the catalog you are currently viewing.

1. The Name.cycle column displays the names of files within the catalog. A file name may be followed by a period and a cycle number. If the cycle number is 1, and there is only one cycle, the cycle number is not displayed.

2. The Size column shows the size of the file in kilobytes. Files less than one kilobyte show a size of 1.

3. The Processor column shows the name of the processor that can handle the file.

4. The File Type column displays the type category of the file. A file can be any one of four types: legible, object, library, or catalog.

   Legible means the file contains text that is human-readable. Object means the file contains compiler-generated object code. Library is a collection of object files. Catalog indicates a subcatalog of the currently displayed catalog.

5. The Changed Date and Time column shows when the last change was made to the file. The date is shown in yymmdd format. Time is shown in hh:mm format. If no changes have been made to a file, this column displays the date and time the file was created.

Some columns may be displayed as blank if their contents cannot be obtained. You may receive a display with some blank columns if:

1. You are not authorized to access the file.

2. The file is in use.

3. The catalog is $LOCAL.

E1.3 MOVING ABOUT WITHIN EDIT CATALOG

When using EDIT_CATALOG, function keys enable you to move from screen to screen. If there are more entries in a catalog or more information in a file than fit on a single screen, you can see the next screen of information by pressing the FWD key. To return to the previous screen, press the BKW key. When you reach the last screen of a catalog or a file, a message is displayed in the upper right corner of the screen

stating 'End of Catalog' or 'End of File'. The UP and the DOWN keys are used for cursor positioning.

If your terminal does not have function keys, there are alternate characters you can use to move from screen to screen. You enter these characters in the first column of a line in the file display. The character entered determines the activity performed. Alternate characters and the actions performed are shown below.

CHARACTER	ACTION	CHARACTER	ACTION
+	FWD	H	HELP
-	BKW	L	LOCATE
/	DOWN	M	MOVE
*	UP	N	CREATE
<	BACK	P	PRINT
A	ATTRIBUTE	Q	QUIT
C	COPY	S	SORT
D	DELETE	U	UNDO
E	EDIT	V	VIEW
G	EDIC	X	EXECUTE

If you are using these alternate characters, key prompts similar to those below will be displayed instead of the function key prompts shown earlier.

MY_CATALOG

Name.cycle	Size (KB)	Processor	File Type	Changed Date	Changed Time
Command_Library.2	224	OCU	Library	840813	10:55
Cobol_Binary	9	Unknown	Object	840821	09:41
External		Catalog			

+=FWD -=BKW *=UP /=DOWN <=BACK A=Attrib C=Copy D=Delete E=Edit G=Goto
H=HELP L=Locate N=New P=Print Q=Quit S=Sort U=Undo V=View X=Execute

---

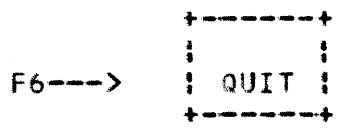
E1.4 INFORMATIONAL_MESSAGES_AND_ERROR_DISPLAYS

EDIC displays informational messages and error messages in different ways.

1. If a message is an informational or a non-error type prompting message generated by EDIC, the message is displayed on the message line which is one line below the home line when the home line is at the top of the screen and one above the home line when the home line is at the bottom of the screen. (There is one exception to this. When deleting a catalog, the informational message stating a catalog DELETE cannot be UNDOne appears in a window in the center of the display with two function keys labeled OK and CANCEL.)

2. If the message results from an error reported by a part of the system that performs user-requested functions, EDIC opens a window in the middle of the catalog display and shows the message there.

E1.5 EXITING_EDIT_CATALOG



Pressing the QUIT key takes you out of EDIT_CATALOG at any time. Your working catalog is restored to what it was when you entered EDIC, regardless of which catalog you were viewing when you pressed QUIT.

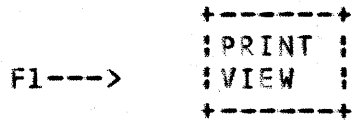
Deleted files that have not been UNDOne are deleted at this time. (See Undoing File Deletions later in this chapter.) If there is a problem with the deletion of a file, you will be informed of the error. The problem file is not deleted and the deletion process continues with subsequent files in the delete list.

E1.6 EDIT_CATALOG_FEATURES

EDIT_CATALOG simplifies many common procedures used in file and catalog management. Using this utility reduces the number of keystrokes required and speeds processing.

E1.0 EDIT_CATALOG
E1.6.1 VIEWING A CATALOG OR FILE

E1.6.1 VIEWING A CATALOG OR FILE

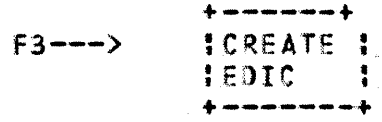


When it is called, EDIC initially displays the catalog specified on the first parameter entered. For example, if you enter EDIC $USER, your $USER catalog is displayed. The display contains several columns of information about the files and the subcatalogs within the catalog specified. If you omit the first parameter, you get a display of your current working catalog.

If you wish to see the contents of a particular entry on the display, use the arrow keys to position the cursor on the line of the entry and press the VIEW key. If the entry selected is a file, the first screen of the file will be displayed on a separate screen. If the entry selected is a subcatalog, entries in the subcatalog will be displayed on the screen.

The screens displaying the selected file or subcatalog provide key prompts instructing you how to move about within the file or subcatalog. Pressing the BACK key returns you to the previous catalog you were viewing or to the catalog immediately above the currently displayed catalog.

E1.6.2 CREATING A NEW CATALOG OR FILE



You can create a new file or new catalog within the currently displayed catalog by pressing the CREATE key. After pressing the CREATE key, the following prompt is displayed:

```
--Enter file name to create a file; name. to create a catalog
```

To create a new file, enter the file name on the home line and press RETURN. You will be taken to the Full Screen Editor to create the contents of the new file.

E1.0 EDIT_CATALOG
E1.6.2 CREATING A NEW CATALOG OR FILE

If you decide not to create a new file, press the BACK key to cancel the CREATE. EDIC will then display the following message:

--Creation not done

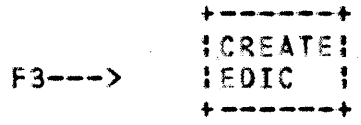
To create a new catalog, enter the catalog name followed by a period on the home line and press RETURN. The following message is displayed:

--Catalog created

After you have created a new catalog, you can move or copy files to it.

If you have created a new catalog and you then decide you do not want the catalog, follow the instructions given in the Deleting a Catalog or File heading later in this chapter.

E1.6.3 SWITCHING CATALOGS



Your working catalog is the catalog you currently are viewing. You can switch and edit a different catalog by pressing the EDIC key. You then see the following prompt:

--Enter catalog name to switch to

Enter the name of the catalog you want to edit and press RETURN.

The catalog name you enter is treated according to the NOS/VE rules for catalog path names. If the name starts with a period it is an absolute path. Thus, the entry .YOUR_NAME takes you to the path YOUR_NAME.

If you enter a name without a leading period, you are referencing a subcatlog of the current catalog. For example, if you are viewing $SYSTEM and you enter MANUALS, you will be switched to $SYSTEM.MANUALS.

There is, however, an easier way to access a subcatalog of the current catalog. Simply position the cursor on the line that contains the subcatalog name and press the VIEW key.

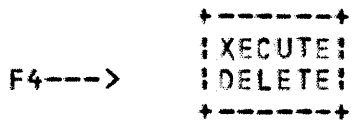
If you decide not to switch catalogs, press the BACK key before entering

E1.0 EDIT_CATALOG
E1.6.3 SWITCHING CATALOGS

a new catalog name. The system displays the following message:

--Catalog not changed

E1.6.4 DELETING A CATALOG OR FILE



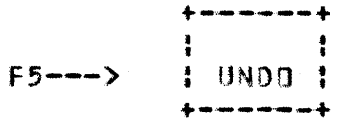
You can delete a catalog or file by positioning the cursor on the line of the entry you want to delete and pressing the DELETE key. The entry is highlighted and then erased from the display. However, the entry is not fully deleted until you exit EDIT_CATALOG.

If you delete an entry by mistake, you may be able to correct the error. Some files can be restored by pressing the UNDO key. Multiple file restores are done in the reverse order in which they were deleted. Each time you press the UNDO key, the cursor position shifts to the newly restored file.

Deletion of catalogs, files in $LOCAL, and files with more than one cycle cannot be UNDOne. When you make these types of deletions you will receive the following warning:

--Delete of a catalog cannot be UNDOne

E1.6.5 UNDOING FILE DELETIONS

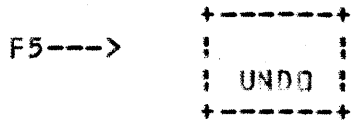


File restores are done in the reverse order in which they were deleted. Each UNDO positions the cursor to the newly restored file (possibly changing catalogs to do so). Files are fully deleted only when you leave EDIT_CATALOG.

If no files have been deleted prior to performing an UNDO, you will receive the following message:

--No deletes to UNDO

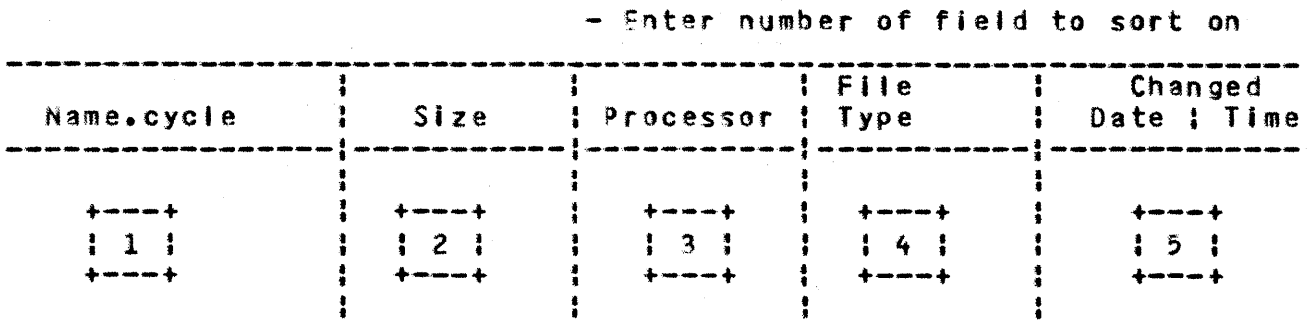
E1.6.6 SORTING THE CATALOG DISPLAY



You can sort entries in any one of the five columns. You can sort entries by:

- Name
- Size
- Processor
- File Type
- Changed Date/Time

To sort entries, press the SORT key. Each column is then numbered with a numeric overlay on top of the text and a prompt is displayed in the upper-right corner of the display.



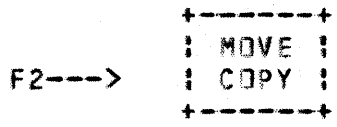
Enter the number of the column by which you want to sort the display and press RETURN. The screen is erased and redisplayed with the entries sorted according to the field you chose.

Press BACK if you decide not to sort. The system redisplays the screen without the numeric overlays and the following message is displayed:

--Sort not done

E1.0 EDIT_CATALOG
E1.6.7 COPYING A FILE

E1.6.7 COPYING A FILE



You can copy a file by positioning the cursor over the name of the file you want to copy and pressing the COPY key. The file entry will be highlighted and you will see the following prompt:

```
--Enter name to copy to
```

Enter the name of the destination file to which you want the original file copied.

The destination file name you enter can be a file name or a path name. If you enter a file name (or a file name.cycle number) the copy will be placed in a destination file within the catalog you are viewing. To copy a file to a destination file in a different catalog, you must enter a path name. For example, to copy a file to a destination file called NEW in subcatalog MINE, enter $USER.MINE.NEW as the destination file name.

A shorthand notation is provided for copying between catalogs. Type the path name up to and including the period preceding the name of the file in the destination catalog and press RETURN. EDIC assumes you want the file to have the same name in the destination catalog as it has in the current catalog.

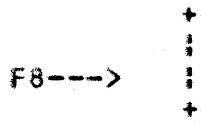
If you copy a file to a non-existent file, you get an exact byte-for-byte copy with all the original file attributes preserved (except ring attributes).

If the target of the copy exists and has different file attributes from the source file, a record-by-record copy will be made (providing the attributes are compatible).

If you decide not to copy a file, press the BACK key. The file entry is no longer highlighted and the following message is displayed:

```
--Copy not made
```

E1.6.8 LOCATING A FILE



You can locate a particular file within a catalog using the LOCATE key. When you press this key you see the following prompt:

```
--Enter name to locate
```

Enter all or part of the file name you wish to locate and press RETURN. If the file name is found, the cursor will move to the name of that file. If the file is not located, the following message appears on your screen:

```
--NOT FOUND
```

A feature called "wild cards" is provided to help you search for file names. The simplest form of a wild card is the question mark (?) character. The question mark can represent any single character in a file name. For example, to find all your files whose names begin with a character followed by BC; you would search for ?BC. The question mark can be used in any character position.

Another type of wild card is the asterisk (*) character. Like the question mark, the asterisk can represent any character. However, the asterisk can represent any number of characters, not just one. For example, you enter SV* to find all your files that begin with SV. The SV matches the first two characters of the file name and the asterisk matches the rest of the name. To find all the files with SOURCE as part of the name, you would enter *SOURCE*.

For example, if you entered the following:

```
?e                --Enter name to locate
```

a display similar to the following would be returned.

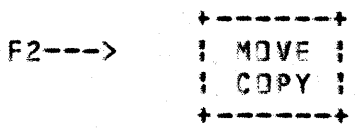
```
          --Select entry with cursor and press RETURN or press RETURN
- Displaying located files
```

E1.0 EDIT_CATALOG
E1.6.8 LOCATING A FILE

Name.cycle	Size	Processor	File Type	Changed Date	Time
$ECHO	1	SCL	LIST		
$ERRORS	1	UNKNOWN	LIST		

The list of located files appears on a separate display. You can choose a particular file by placing the cursor on the line where the file name appears and pressing RETURN. If you decide not to locate any files, press the BACK key. No message is displayed.

E1.6.9 MOVING OR RENAMING A FILE



You can move a file to a new catalog or rename a file within the same catalog by positioning the cursor to the line where the file name is displayed and pressing the MOVE key. The file entry will be highlighted and you will see the following prompt:

```
--Enter name to move
```

If you want to move a file to another catalog, enter the path name and press RETURN. The system displays the following message:

```
--File moved
```

A shorthand notation is provided for moving a file between catalogs. Type the path name up to and including the period preceding the name of the file in the destination catalog and press RETURN. EDIC assumes you want the file to have the same name in the destination catalog as in the current one.

If you want to rename a file in the the current catalog, enter the new file name and press RETURN. The following message is returned:

```
--File renamed
```

E1.0 EDIT_CATALOG
E1.6.9 MOVING OR RENAMING A FILE

If you decide not to move or rename a file, press the BACK key to cancel the operation. The system responds with the following message:

```
--File not moved
```

E1.6.10 EXECUTING A FILE

You can execute an SCL or object file using the XECUTE key. Position the cursor on the line of the file you want to execute and press the XECUTE key.

The file entry is highlighted and you see the following prompt:

```
--Enter parameters or press RETURN if none
```

Enter any parameters you wish to add and press RETURN. Or, if you prefer not to supply any parameters, simply press RETURN. The program will then execute. During execution the screen is blank except for the following message:

```
--Executing command
```

The blank screen allows any program output to be clearly visible.

When execution is complete, processing pauses. This pause gives you time to read any interactive output produced by your program before the catalog display overwrites it. When you have read the output, press RETURN to go back to the catalog display.

If you decide not to execute a file, press the BACK key. Highlighting disappears, but no message is displayed.

E1.6.11 EDITING A FILE

You can edit a file using the NOS/VE Full Screen Editor (FSE) by placing the cursor on the line that displays the name of the file you wish to edit and pressing the EDIT key. You are taken automatically into the Full Screen Editor.

E1.0 EDIT_CATALOG
E1.6.11 EDITING A FILE

When you are done editing the file, leave the editor in the same manner you normally use to end an editing session. You are returned to EDIC exactly where you left it.

E1.6.12 PRINTING A FILE

```
                    +-------+
          F1--->    !PRINT !
                    ! VIEW !
                    +-------+
```

You can print a file using the PRINT key. Place the cursor on the line of the file you want to print and press the PRINT key. The entry is highlighted and you are asked:

```
--What is your bin number?
```

Enter your bin number and press RETURN. The system returns the following message:

```
--File sent to printer
```

The file is sent to the central site line printer.

E1.6.13 DISPLAYING FILE ATTRIBUTES

```
                    +-------+
                    !LOCATE!
          F8--->    !ATTRIB!
                    +-------+
```

You can display the file attributes for a file by positioning the cursor on the file name line and pressing the ATTRIB key. You will then see a separate display similar to the one shown below. This display shows the most frequently referenced file attributes.

```
--File attributes of $MYFILE
----------------------------

File_organization: SEQUENTIAL                Size: 1312
Block_type: SYSTEM_SPECIFIED                 Record_type: VARIABLE
Maximum_record_length: 256                   Maximum_record_length: 0
Access_mode: READ APPEND                     Maximum_block_length: 18
Open_position: $BOI
```

E1.0 EDIT_CATALOG
E1.6.13 DISPLAYING FILE ATTRIBUTES

When the new set of function keys appears at the bottom of the screen, press the SEEALL key if you wish to see a pageable display of all of the file's attributes. Pressing the BACK key returns you to the main catalog display. Pressing QUIT takes you out of EDIC.

E1.6.14 EXECUTING NOS/VE COMMANDS

You can execute NOS/VE commands by moving the cursor to the home line and typing the command preceded by a slash. For example, to display the current working catalog, enter the following command:

```
/DISC $CATALOG
```

After typing the command information, press the RETURN key. During execution the screen is blank except for the following message:

```
--Executing command
```

The blank screen allows any system response to the command to be clearly visible.

When execution of the command is complete, press RETURN to restore the current catalog display.

F1.0 NOS/VE DUMP ANALYZER

F1.0 NOS/VE_DUMP_ANALYZER

The NOS/VE Dump Analyzer is a NOS/VE utility provided to aid in the analysis of NOS/VE system failures and Cyber 180 hardware failures. The Dump Analyzer accesses a tape created by the Express Deadstart Dump (EDD) deadstart utility. The NVE subsystem may also produce a dump tape in the event of a NOS/VE failure. This tape is compatible with tapes produced by EDD and is therefore acceptable as input to the dump analyzer.

You may use the dump analyzer either interactively or run from a procedure or an INCLUDE_FILE as part of a batch or interactive job. The dump analyzer is implemented as a command utility. Subcommands are available to display the following information:

1. Central memory, displayed in numeric and ascii format, accessed in virtual or real address mode.

2. Peripheral processor memory.

3. Maintenance registers for the IOU, memory and processors.

4. Formatted display of exchange packages.

5. Process information resulting from an analysis of stack segments.

F1.1 ANAD (ANALYZE_DUMP)

Purpose: The ANALYZE_DUMP command invokes the dump analyzer utility.

Format:
```
ANALYZE_DUMP
dump_file = file reference
restart_file = file reference
debug_table = file reference or keyword value
title = string
output = file reference
status = status variable
```

Parameters: dump_file (df)

Specifies the dump to be analyzed. There is no default for this parameter. If omitted, it is assumed that the restart

F1.0 NOS/VE DUMP ANALYZER
F1.1 ANAD (ANALYZE_DUMP)

file contains the dump information. If dump_file is omitted and the restart file is empty or non-existent, an error status is returned.

restart_file (rf)

The restart file is either an input file or an output file, depending on whether the dump_file parameter is specified. If dump_file is specified, a restart file is written to this file.This file is accessed during the execution of the dump analyzer. If you do not specify a dump_file parameter, the restart file indicates a file created from a dump by a previous execution of the analyze_dump utility. If you do not specify this parameter, the value $local.restart_file to be used.

debug_table (dt)

Generates symbolic names from addresses and addresses from symbolic names. This table is built when the system is generated. It is included in the generated system and is also saved on a file. If the dump is being analyzed on the same system as the dumped system then the debug table in the running system may be used. Otherwise, specify the file name of the file saved from the system generation. Specify the keyword "none" if no debug table is to be accessed. If you do not specify this parameter, the running system debug table is used.

title (t)

A string of 1 to 25 characters may be specified and is included in the page headers if these headers are generated. If you do not specify this parameter, the string 'ANALYZE_DUMP Version 1.0' is used.

output (o)

Establishes the default output file for all subcommands. This default may be overriden with the output parameter on

F1.0 NOS/VE DUMP ANALYZER
F1.1 ANAD (ANALYZE_DUMP)

each subcommand. If you do not specify this parameter, the value $output is used.

status

Optional status variable.

Remarks: Analyze dump does not request a tape. You must supply the REQUEST_MAGNETIC_TAPE command prior to invoking ANALYZE_DUMP.

Example:

```
/request_magnetic_tape $local.dump type=mt9$6250 ..
   evsn='dmp001'
/analyze_dump dump_file=$local.dump restart_file=..
   $user.dmp001
ad/quit
/analyze_dump rf=$user.dmp001
ad/
```

F1.2 DUMP_ANALYZER_SUBCOMMANDS

The following subcommands are described in this section:

- CHANGE_DEFAULT (CHAD)
- CHANGE_PROCESSOR_REGISTER (CHAPR)
- COPY_MEMORY (COPM)
- COPY_PP_MEMORY (COPPM)
- DISPLAY_CALL (DISC)
- DISPLAY_EXCHANGE_PACKAGE (DISEP)
- DISPLAY_MAINTENANCE_REGISTERS (DISMR)
- DISPLAY_MEMORY (DIS,)
- DISPLAY_PP_MEMORY (DISPM)
- QUIT (QUI)

F1.0 NOS/VE DUMP ANALYZER
F1.2.1 (CHAD) CHANGE_DEFAULT

F1.2.1 (CHAD) CHANGE_DEFAULT

Purpose: This subcommand allows you to change the default setting of certain parameters on dump analyzer subcommands.

Format:

```
CHANGE_DEFAULT
exchange = integer or keyword value
processor = integer
bytes = integer
address_mode = keyword value
words = integer
status = status variable
```

Parameters:

exchange (e)

If you specify this parameter, the value supplied is used as the default setting for all subsequent subcommands and functions which accept the exchange parameter. Allowable values are the keywords monitor, m, job, j or an integer value. If this parameter is not specified, the default is not changed.

processor (p)

If you specify this parameter, the value supplied is used as the default setting for all subsequent subcommands and functions which accept the processor parameter. Allowable values are integers between 0 and 3. If this parameter is not specified, the default is not changed.

bytes (b)

If you specify this parameter, the value supplied is used as the default setting for the bytes parameter on the DISPLAY_MEMORY subcommand. Allowable values are integers between 0 and 33554432. If you do not specify this parameter, the default is not changed.

F1.0 NOS/VE DUMP ANALYZER
F1.2.1 (CHAD) CHANGE_DEFAULT

address_mode (am)

If you specify this parameter, the value supplied is used as the default setting for all subsequent subcommands and functions which accept the address_mode parameter. Allowable values are the keywords process_virtual_address, pva, system_virtual_address, sva, real_memory_address and rma. If you do not specify this parameter, the default is not changed.

words (w)

If you specify this parameter, the value supplied is used as the default setting for the words parameter on the DISPLAY_PP_MEMORY subcommand. Allowable values are integers between 0 and 4096. If you do not specify this parameter, the default is not changed.

Status

Optional status variable.

Example: ad/change_default e=monitor

F1.0 NOS/VE DUMP ANALYZER
F1.2.2 CHAPR (CHANGE_PROCESSOR_REGISTER)

F1.2.2 CHAPR (CHANGE_PROCESSOR_REGISTER)

Purpose: This subcommand allows you to change the setting of the processor registers which are used for virtual address translation. These registers are initialized in the dump analyzer from their settings in the processor maintenance registers.

Format:

```
CHANGE_PROCESSOR_REGISTER
job_process_state = integer
monitor_process_state = integer
page_size_mask = integer
page_table_address = integer
page_table_length = integer
processor = integer
status = status variable
```

Parameters:

job_process_state (jps)

Subcommands and functions which use the exchange parameter can obtain a real memory address from the maintenance registers to use as the location of the exchange package. This occurs if the exchange parameter is equated to either job or monitor. When equated to job, the value of the jps register is used as the exchange address.

The jps parameter allows you to change the value that is used when the exchange parameter is equated to job.

monitor_process_state (mps)

The mps parameter allows you to change the value that is used when the exchange parameter is equated to monitor.

page_size_mask (psm)

The page size mask correlates directly to a page size. The following table shows the allowable values for page size mask and the resulting page size:

F1.0 NOS/VE DUMP ANALYZER
F1.2.2 CHAPR (CHANGE_PROCESSOR_REGISTER)

page size mask	page size (bytes)
7f(16)	512(10)
7e(16)	1024(10)
7c(16)	2048(10)
78(16)	4096(10)
70(16)	8192(10)
60(16)	16384(10)
40(16)	32768(10)
0	65536(10)

For this subcommand, enter the page size mask setting, not the page size.

page_table_address (pta)

The page table address is specified as the actual real memory starting address of the page table.

page_table_length (ptl)

The following table shows the allowable values for page table length register and the resulting page table length:

page table length register	page table length (bytes)
0 0	4096(10)
1 0	8192(10)
3 3	16384(10)
7 7	32768(10)
0f(16)	65536(10)
1f(16)	131072(10)
3f(16)	262144(10)
7f(16)	524288(10)
0ff(16)	1048576(10)

For this subcommand, enter the setting of the page table length register, not the actual page table length.

processor (p)

F1.0 NOS/VE DUMP ANALYZER
F1.2.2 CHAPR (CHANGE_PROCESSOR_REGISTER)

The processor parameter specifies which processor is to be selected for the subcommand in a multi-processor environment.

Allowable values are 0 through 3. Omission causes 0 to be used.

Status

Optional status variable.

Example: ad/change_processor_register jps=4d4420(16)

F1.0 NOS/VE DUMP ANALYZER
F1.2.3 COPM (COPY_MEMORY)

F1.2.3 COPM (COPY_MEMORY)

Purpose: Copies central memory to a file. The address of the memory to be copied can be specified as a process virtual address (pva), a system virtual address (sva) or as a real memory address(rma). The memory is copied exactly as it exists in the dump.

Format:

```
COPY_MEMORY
address = integer
file = file reference
byte_count = integer
exchange = integer or keyword value
processor = integer
address_mode = keyword value
status = status variable
```

Parameters:

address (a)

Specifies the address of the memory to be copied. Its format depends on the value supplied for the address_mode parameter.

This parameter is required.

file (f)

Specifies the file to which the memory is copied.

This parameter is required.

byte_count (bc)

Indicates the number of bytes to be copied.

If you do not specify this parameter, a default value of 100000(16) bytes is used.

exchange (e)

Designates the exchange package to be used to define the virtual address. An integer value is interpreted as the starting real memory address of an exchange package.

The following keywords may also be used:

Active (a)

Indicates that the contents of the CPU at the time of the dump are to be used.

Monitor (m)

Indicates that the processor maintenance registers are to be accessed and the value in the MPS register be used as the real memory address of the exchange package.

Job (j) indicates that the processor maintenance registers are to be accessed and the value in the JPS register used as the real memory address of the exchange package.

If you do not specify COPY_MEMORY, the active exchange package is used.

processor (p)

Specifies which processor is to be selected for the subcommand in a multi-processor environment. The processor determines which set of maintenance registers are to be used for virtual addressing parameters page size mask, page table address and page size mask. In addition if job or monitor is specified for the exchange parameter, the jps or mps register for that processor will be accessed. If active is specified for exchange, the processor parameter determines which processor's active exchange package is to be used.

Allowable values are 0 through 3. If you do not specify a parameter, a default value of 0 is used.

address_mode (am)

Determines the interpretation of the address parameter. Possible settings are:

process_virtual_address (pva) causes interpretation as follows:

sssnnnnnnnnn(16)	with s being the segment number and n the offset.

system_virtual_address (sva) causes interpretation as follows:

aaaannnnnnnn(16)	with a being the assigned segment identifier (asid) and n is the offset.

real_memory_address (rma) specifies the starting real memory byte address to be copied.

If you do not specify this parameter, a default of process_virtual_address is used.

Status

Optional status variable.

Remarks: COPY_MEMORY is useful for determining which portion of virtual memory is paged in by copying the memory to file $null.

Example:

```
ad/copy_memory a=300000000(16) f=$user.segment_3 ..
  bc=10000(16)
ad/copm 900000000(16) $null 20000(16)
--WARNING--  Virtual memory from offset 0(16) to 0FFF(16)
is paged out.
--WARNING--  Virtual memory from offset 2000(16) to
1FFFF(16) is paged out.
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.3 COPM (COPY_MEMORY)

WARNING-- Not all memory requested was copied: a page fault was encountered.

F1.0 NOS/VE DUMP ANALYZER
F1.2.4 COPPM (COPY_PP_MEMORY)

F1.2.4 COPPM (COPY_PP_MEMORY)

Purpose: copies peripheral processor memory to a file.

Format:

```
Copy_pp_memory
pp_number = integer
file = file reference
address = integer
word_count = integer
status = status variable
```

Parameters:

pp_number (pn)

Specifies the number of the peripheral processor whose memory is to be copied to a file.

This parameter is required.

file (f)

Specifies the file to which the memory is copied.

This parameter is required.

address (a)

Specifies the starting address in pp memory of the copy.

If you do not specify a value for this parameter, a default value of 0 is used.

word_count (wc)

Indicates the number of words to be copied.

If you do not specify a value for this parameter, a default value of 4096 words (the entire pp) is copied.

status

F1.0 NOS/VE DUMP ANALYZER
F1.2.4 COPPM (COPY_PP_MEMORY)

Optional status variable.

Example: ad/copy_pp_memory 4 f=$user.pp_4

F1.0 NOS/VE DUMP ANALYZER
F1.2.5 DISC (DISPLAY_CALL)

F1.2.5 DISC (DISPLAY_CALL)

Purpose: The DISPLAY_CALL subcommand produces a formatted display of the dynamic call chain for a selected task.

Format:

```
DISPLAY_CALL
exchange = integer or keyword value
processor = integer
count = integer or keyword value
start = integer or keyword value
display_option = list of keyword value
title = string
output = file reference
status = status variable
```

Parameters: exchange (e)

Designates the exchange package to be used for the analysis. The exchange package defines the process whose call chain is being displayed. An integer value is interpreted as the starting real memory address of an exchange package.

The following keywords may also be used:

active (a) indicates that the contents of the processor at the time of the dump are to be used.

monitor (m) indicates that the processor maintenance registers are to be accessed, and the value in the MPS register used as the real memory address of the exchange package.

job (j) indicates that the processor maintenance registers are to be accessed and the value in the JPS register used as the real memory address of the exchange package.

If you do not specify this parameter, the active exchange package is used.

processor (p)

Specifies which processor is to be selected for the subcommand in a multi-processor environment. The processor determines which set of maintenance registers are to be used for virtual addressing parameters, page table address, and page size mask. In addition, if job or monitor is specified for DISPLAY_CALL (disc), the exchange parameter (the jps or mps register for that processor) will be accessed. If active is specified for the exchange, the processor parameter determines which processor's active exchange package is to be used.

Allowable values are 0 through 3. If you do not specify a value for this parameter, a default value of 0 is used.

count (c)

Number of stack frames to be displayed or the keyword, "all". If the value given is greater than the number of existing frames, all are displayed.

If you do not specify this parameter, the entire call chain is displayed.

start (s)

The start parameter indicates which frame in the call chain will begin the display. Frame one represents the most recent call or trap. Start may also specify the keyword EXCHANGE_PACKAGE. This begins the display with a formatted display of the exchange package, and then displays stack frames starting with one.

If you do not specify this parameter, the keyword "exchange_package" is used.

display_options (display_option, do)

The following display options are provided:

brief (b) - provides a minimum display.

full (f) - provides a more detailed display.

save (s) - displays the contents of the stack frame save area in hexadecimal interpretation.

You can use a list of keywords, but full and brief cannot be used together.

If you do not specify an option, a display_option default of brief is used.

Title (t)

Specifies a string of 1 to 31 characters which is included in the page headers if these headers are generated.

If you do not specify a title, the string 'display_call' is used.

Output (o)

File receiving the display. If you do not specify an out put file, the value specified on the ANALYZE_DUMP command output parameter is used.

Status

Optional status variable.

Example:

```
ad/display_call e=job
Exchange address = 5C54A0(16)

Exchange package: PROCESS_COMMAND_IN_LIST + A0(16) in
CLM$PROCESS_COMMANDS
monitor conditions: page table search without find

    Frame 1:  CLP$PROCESS_COMMAND + 682(16) in CLM$PROCESS_COMMANDS
    Frame 2:  PROCESS_COMMAND + 100(16) in CLM$INCLUDE
              The on condition flag is set for this frame.
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.5 DISC (DISPLAY_CALL)

```
    Frame 3:  PROCESS_COMMAND_LINE + 550(16) in CLM$INCLUDE
    Frame 4:  CLP$SCAN_COMMAND_LINE + AE(16) in CLM$INCLUDE
    Frame 5:  P =  B  42     6B42C
    Frame 6:  P =  B  42     6CE82
    Frame 7:  P =  B  42     6DB40
                   The critical frame flag is set for this frame.
                   The on condition flag is set for this frame.
    Frame 8:  P =  B  42     5E3F4
                   The critical frame flag is set for this frame.
                   The on condition flag is set for this frame.
    Frame 9:  PMP$ORIGINAL_CALLER + 186(16) in PMM$OUTWARD_CALL
                   The critical frame flag is set for this frame.
                   The on condition flag is set for this frame.
                   Display terminated: end of call chain encountered in
                   frame 9
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.6 DISEP (DISPLAY_EXCHANGE_PACKAGE)

F1.2.6 DISEP (DISPLAY_EXCHANGE_PACKAGE)

Purpose: The purpose of the DISPLAY_EXCHANGE_PACKAGE subcommand is to produce a formatted display the contents of a selected exchange package.

Format:
```
Display_exchange_package
exchange = integer or keyword value
processor = integer
title = string
output = file reference
status = status variable
```

Parameters: exchange (e)

The exchange parameter designates the exchange package to be interpreted. An integer value is interpreted as the starting real memory address of an exchange package.

The following keywords may also be used:

Active (a) indicates that the contents of the CPU at the time of the dump are to be displayed.

Monitor (m) indicates that the processor maintenance registers are to be accessed and the value in the MPS register used as the real memory address of the exchange package.

Job (j) indicates that the processor maintenance registers are to be accessed and the value in the JPS register used as the real memory address of the exchange package.

If you do not specify this parameter, the active exchange package is displayed.

processor (p)

The processor parameter specifies which processor is to be

selected for the subcommand in a multi-processor environment. The processor determines which set of maintenance registers are to be used for virtual addressing parameters page size mask, page table address and page size mask. In addition if job or monitor is specified for the exchange parameter, the jps or mps register for that processor will be accessed. If active is specified for exchange, the processor parameter determines which processor's active exchange package is to be used.

Allowable values are 0 through 3. If you do not specify this parameter, a default value of 0 is used.

Title (t)

A string of 1 to 31 characters may be specified and is included in the page headers if these headers are generated.

If you do not specify this parameter, the string 'display_exchange_package' is used.

Output (o)

Indicates the file receiving the display. If you do not specify this parameter, the value specified on the ANALYZE_DUMP_COMMAND output parameter is used.

Status

Optional status variable.

Example:

```
ad/display_exchange_package e=job
Exchange address = 5C54A0(16)

Exchange package: PROCESS_COMMAND_IN_LIST + A0(16) in
CLM$PROCESS_COMMANDS
monitor conditions: page table search without find

                  0000      P     B     1D        5F450
vmid/uvmid        0000      A0    B     4          3890
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.6 DISEP (DISPLAY_EXCHANGE_PACKAGE)

```
flags/te        0002        A     B     4E          3970
user mask       F77         A2    B     4E          3908
monitor mask    FFFC        A3    B     18          5858
ucr             0000        A4    B     4E          3890
mcr             0040        A5    B     4E          39F8
kypt cl/lpid    0000        A6    B     5          18520
kypt mask       0000        A7    F     FFF  -80000000
kypt code       0000        A8    B     5          18520
                0000        A9    B     5          18520
pit             7FF1        AA    B     4E          2C50
                52AF        AB    B     4E          3A68
base const      0000        AC    B     4E          39F7
                54A0        AD    B     4E          2C50
md flags        0000        AE    B      F             0
stl             0064        AF    B     4E          3F17

                X0 0000 0000 0000 0003
                X1 0000 0000 0000 0000
                X2 0000 0000 0000 0001
                X3 0000 0000 0000 0000
                X4 0000 0000 0000 0000
                X5 0000 0000 0000 0000
                X6 0000 B04E 0000 2C50
                X7 0000 0000 0000 0000
                X8 0000 0000 0000 001B
                X9 0000 0000 0000 0000
                XA 0000 0000 0000 0064
                XB 0000 0000 0000 076C
                XC 0000 0000 0000 0054
                XD 0000 0000 0000 00FA
                XE 0000 0000 0000 0001
                XF 0000 0000 0000 0008

md word            0000 0000 0000 0000

sta           005C  utp      1 FFF  7FFFFFFF
              5880  trap     1   E         0
db ind/mask   0000  db list F FFF -80000000

max ring      000F  tos  1  1   F          0
              0000  tos  2  2  10          0
              0000  tos  3  3  11          0
              0000  tos  4  0   0          0
              0000  tos  5  0   0          0
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.6 DISEP (DISPLAY_EXCHANGE_PACKAGE)

```
0000  tos  6   0   0          0
0000  tos  7   0   0          0
0000  tos  8   0   0          0
0000  tos  9   0   0          0
0000  tos 10   0   0          0
0000  tos 11   8   4E      3970
0000  tos 12   0   0          0
0000  tos 13   0   0          0
0000  tos 14   0   0          0
0000  tos 15   0   0          0
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.7 DISMR (DISPLAY_MAINTENANCE_REGISTERS)

F1.2.7 DISMR (DISPLAY_MAINTENANCE_REGISTERS)

Purpose: Displays the contents of the processor maintenance registers, iou maintenance registers or memory maintenance registers.

Format:
```
DISPLAY_MAINTENANCE_REGISTERS or
DISPLAY_MAINTENANCE REGISTER
element = list of keyword value
title = string
output = file reference
status = status variable
```

Parameters: element (elements, e)

Specifies the elements whose maintenance registers are to be displayed. Possible values are:

Processor (p)

input_output_unit (iou)

memory (m)

all (a)

If you do not specify an element for this parameter, the registers from all elements are displayed.

Title (t) Specifies a string of 1 to 31 characters which is included in the page headers if these headers are generated.

If you do not specify a title, the string "display_maintenance_registers" is used

Output (o) Specifies the file receiving the display. If you do not specify an output file, the value specified on the ANALYZE_DUMP_COMMAND output parameter is used.

F1.0 NOS/VE DUMP ANALYZER
F1.2.7 DISMR (DISPLAY_MAINTENANCE_REGISTERS)

Status

Optional status variable.

Example:

```
ad/display_maintenance_registers

    PROCESSOR   00

00  0202 0202 0202 0202        status summary
                                  bit 62 : corrected error
10  0000 0000 0030 0109        element id
                                  element: PROCESSOR
                                  model  : 855
                                  s / n  : 0109
11  0000 0000 0000 0000        processor id
12  0000 0000 0000 0012        options installed
13  0000 0000 0000 C000        vm capability list
30  0000 0000 0FC0 01F8        dependent env ctrl
40  0000 1003 0007 D44C        program addr
41  0000 0000 0026 9000        mtr process state ptr
44  0000 0000 0000 0000        untranslatable pointer
45  0000 0000 0000 0006        segment table length
46  0000 0000 00FF A2F8        segment table addr
48  0000 0000 0020 0000        page table address
49  0000 0000 0000 001F        page table length
4A  0000 0000 0000 0078        page size mask
61  0000 0000 0022 0780        job process state ptr
62  0000 0000 E17C 1080        sys interval timer
80  4000 0000 0000 0000        proc fault status 0
                                  bit 01 : correctable /
                                  soft / bypass error ,
                                           MAC operation PDM
81  0800 0000 0000 0000        proc fault status 1
                                  bit 04 : A stream stage 2
                                  data register parity error
82  0000 0000 0000 0000        proc fault status 2
83  0000 0000 0000 0000        proc fault status 3
84  0000 0000 0041 0000        proc fault status 4
                                  bit 41 : successful retry
                                  bit 47 : rank 50 before
                                  PONR PDM
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.7 DISMR (DISPLAY_MAINTENANCE_REGISTERS)

```
85  0000 0000 0000 0000        proc fault status 5
86  0000 0000 0000 0000        proc fault status 6
87  0000 0000 0000 0000        proc fault status 7
88  0000 0000 0000 0000        proc fault status 8
89  0000 0000 0000 0000        proc fault status 9

    MEMORY

00  0000 0000 0000 0000        status summary
10  0000 0000 0131 0002        element id
                                  element: MEMORY
                                  model  : 850
                                  s / n  : 0002

12  0808 0000 0000 0000        options installed
20  0000 0000 3200 0000        environment control
21  4000 0000 0FF6 0000        bounds register
A0  0000 0000 0000 0000        corr err log
A4  0000 0000 0000 0000        uncorr err log 1
A8  0000 0000 0000 0000        uncorr err log 2

INPUT/OUTPUT UNIT

00  1010 1010 1010 1010        status summary
                                  bit 59 : summary status
10  0000 0000 0220 0633           element id
                               element: IOU
                               model  : 835
                                  s / n  : 0633
12  0000 0FFF AFFF 0F0F        options installed
18  0000 0000 0000 0000        fault status mask
21  0718 1F1F 0000 0100        os bounds
30  0000 0000 0000 0009        dependent env control
40  0000 0000 0009 FF00        status reg
80  0000 0000 0000 0000        fault status 1
81  0000 0000 0000 0000        fault status 2
A0  0000 0000 0000 0011        test mode
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.8 DISPLAY_MEMORY (DISM)

F1.2.8 DISPLAY_MEMORY (DISM)

Purpose: Displays central memory in numeric or ascii format. The address of the memory to be displayed can be specified as a process virtual address (pva), a system virtual address (sva) or as a real memory address (rma).

Format:
```
Display_memory
address = integer
bytes = integer
exchange = integer or keyword value
processor = integer
address_mode = keyword value
display_option = list of keyword value
title = string
radix = integer
output = file reference
status = status variable
```

Parameters:

Address (a)

Specifies the address of the memory to be displayed. Its format depends on the value supplied for the ADDRESS_MODE parameter.

This parameter is required.

Bytes (b)

This parameter indicates the number of bytes to be displayed. If you do not specify a value for this parameter, eight is used.

Exchange (e)

Designates the exchange package to be used to define the virtual address. An integer value is interpreted as the starting real memory address of an exchange package.

The following keywords may also be used:

- Active (a) - indicates that the contents of the CPU at the time of the dump are to be used.
- Monitor (m) - indicates that the processor maintenance registers are to be accessed and the value in the MPS register be used as the real memory address of the exchange package.
- Job (j) - indicates that the processor maintenance registers are to be accessed and the value in the JPS register be used as the real memory address of the exchange package.

If you do not specify an exchange package, the active exchange package is used.

processor (p)

Specifies which processor is to be selected for the subcommand in a multi-processor environment. The processor determines which set of maintenance registers are to be used for virtual addressing parameters page size mask, page table address and page size mask. In addition, if * job or monitor is specified for the exchange parameter, the jps or mps register for that processor will be accessed. If active is specified for exchange, the processor parameter determines which processor's active exchange package is to be used.

Allowable values are 0 through 3. If you do not specify a value for this parameter, 0 is used.

address_mode (am)

Determines the interpretation of the address parameter. Possible settings are:

- process_virtual_address (pva) - causes interpretation as follows:

  sssnnnnnnnn(16)

with s being the segment number and n the offset.

- system_virtual_address (sva) -

  aaaannnnnnnn(16)

  a is the assigned segment identifier (asid) and n is the offset.

- real_memory_address (rma) - Address specifies the starting real memory byte address to be displayed.

If you do not specify a value for this parameter, process_virtual_address is used.

Display_option (do)

Allowable values:

- numeric (n) - displays the numeric representation of the requested memory. The interpretation of the memory depends on the radix parameter.
- Ascii (a) - displays the ascii characters represented.

If you do not specify a value for this parameter, numeric and ascii are displayed together.

Title (t)

Specifies a string of 1 to 31 characters which is included in the page headers if these headers are generated.

If you do not specify a title, the string 'display_memory' is used.

Radix (r)

Determines how the numeric data will be interpreted and displayed. Any integer between 8 and 16 may be specified. If the radix is 16, each byte of memory is interpreted as

an 8 bit unsigned integer. If the radix is 8, the byte count is rounded up to the nearest number of words (8 bytes) and the address parameter is adjusted down to the nearest word boundary. Each word is then displayed as an unsigned integer. For all other radixes, the bytes parameter is adjusted to display an integral number of 8 byte entities. Each 8 bytes is displayed as a signed 64 bit integer.

If you do not specify a value for this parameter, 16 is used.

Output (o)

File receiving the display. If you do not specify an output file, the value specified on the output parameter on the analyze_dump command is used.

Status

Optional status variable.

Example:

```
ad/display_memory a=30000000(16) 20(16) e=j
00000000  4142 4147  2454 5253   5F35 2020  2020 2020   ABAG$TRS_5
00000010  2020 2020  2020 2020   2020 2020  2020 2020
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.9 DISPLAY_PP_MEMORY (DISPM)

F1.2.9 DISPLAY_PP_MEMORY (DISPM)

Purpose: This subcommand displays the contents of peripheral processor memory.

Format:

```
DISPLAY_PP_MEMORY
pp_number = integer
address = integer
words = integer
display_option = list of keyword value
radix = integer
display_relocation_register = boolean
title = string
output = file reference
status = status variable
```

Parameters: pp_number (pn)

Specifies pp whose contents are to be displayed. Allowable values are integers from 0 to 25. An error status is returned if the pp specified is not present in the dump.

This parameter is required.

address (a) Specifies the starting address of pp memory to be displayed. Values of 0 through 4095 are allowed.

If you do not specify a value for this parameter, a value of 0 isused.

words (w)

This parameter specifies the number of words of pp memory to be displayed. Values of 0 through 4096 are allowed.

If you do not specify a value for this parameter, a value of 4096 is used.

display_option (do)

Specifies either numeric (n) display, ascii (a) display or both.

F1.0 NOS/VE DUMP ANALYZER
F1.2.9 DISPLAY_PP_MEMORY (DISPM)

If you do not specify a display option both numberic and ascii are displayed.

radix (r)

Determines how pp memory is displayed. If the radix is eight, and the most significant 4 bits of a word are zero, the word will be displayed with 4 octal digits. If these bits are non-zero, 6 octal digits will be used. With any radix, a word of all zeros will be displayed with hyphens (-). The radix used to display the memory will also be the radix used to disolay the address of the memory being displayed.

display_relocation_register (drr)

Indicates whether or not to display the contents of the pp's relocation register along with the memory display. The parameter is tvpe boolean the default is to display the register if the entire contents of the the pp's memory is also being displayed. If only a portion of the pp's memory is being displayed then the default is to not display the relocation register.

Title (t)

Specifies a string of 1 to 31 characters which is included in the page headers if these headers are generated.

If you do not specify a title, the string 'display_pp_ memory pp_number=', concatenated with the pp number being displayed, is used

Output (o)

File receiving the display. If you do not spescify an output file, the value specified on the output parameter on the analyze_dump command is used.

Status

F1.0 NOS/VE DUMP ANALYZER
F1.2.9 DISPLAY_PP_MEMORY (DISPM)

Optional status variable.

Example:

```
ad/DISPLAY_PP_MEMORY pp_number=6 address=0 words=100(8)
     drr=yes do=n
R register = 0(8)
0000    2463   ----   0001   0004   6445   0054   ----   ----
0010    ----   ----   ----   0025   0035   0144 020460   ----
0020    ----   ----   ----   ----   1730   0020   0002   0004
0030    0001   0001   ----   0025   ----   0240   0240   0002
0040    0004   0010   ----   0004   6446   0046   0001   ----
0050    0013   0400   ----   0400   ----   0002   0004   ----
0060    ----   ----   0002   ----   0001   ----   ----   0010
0070    0014   ----   ----   0570   ----   1440   ----   ----
```

F1.0 NOS/VE DUMP ANALYZER
F1.2.9 DISPLAY_PP_MEMORY (DISPM)

QUIT (qui)

Purpose: Terminates the analyze dump utility.

Format: quit

status = status variable

Parameters: Status

Optional status variable.

F1.0 NOS/VE DUMP ANALYZER
F1.3 DUMP ANALYZER FUNCTIONS

F1.3 DUMP_ANALYZER_FUNCTIONS

The following functions are provided:

- $MAINTENANCE_REGISTER ($MR)
- $MEMORY ($MEM)
- $MEMORY_STRING ($MS)
- $MODULE
- $OFFSET ($OFF)
- $PP_MEMORY ($PM)
- $PROCESS_REGISTER ($PR)
- $REAL_MEMORY ($RM)
- $REAL_MEMORY_ADDRESS ($RMA)
- $RING
- $SECTION ($SEC)
- $SEGMENT ($SEG)
- $SYMBOL_ADDRESS ($SA)

F1.0 NOS/VE DUMP ANALYZER
F1.3.1 $MAINTENANCE_REGISTER ($MR)

F1.3.1 $MAINTENANCE_REGISTER ($MR)

Purpose: Returns the contents of the specified maintenance register. The result is type integer.

Format: $MAINTENANCE_REGISTER(register, element, processor)

Parameters: register

Specifies the register whose contents are to be returned as the function result. It may be either a keyword or an integer. Allowable keywords are MONITOR_PROCESS_STATE (MPS) and JOB_PROCESS_STATE (JPS). Allowable integers are any valid register number.

This parameters is required.

element

Specifes which set of maintenance registers are to be accessed. Allowable values are processor (p), INPUT_OUTPUT_UNIT (iou), and memory (m).

If you do not specify a value for this parameter, processor is used.

processor

Specifies the number of the processor to be used for processor maintenance register access.

If you do not specify a value for this parameter, a value of 0 is used.

Remarks: An error status is returned if the register is not contained in the dump.

Example:

```
ad/disv $MAINTENANCE_REGISTER(41(16))
269000(16)
```

F1.3.2 $MEMORY ($MEM)

Purpose: Returns the contents of central memory. The result is type integer.

Format: $MEMORY(address, byte_count, exchange, processor, address_mode)

Parameters: address

Specifies the address whose location is to be returned as the result of the function.

This parameters is required.

byte_count

Specifies the number of consecutive bytes which constitues the value returned by $memory. The minimum is one byte and maximum is eight.

If you do not specify a value for this parameter, a value of six is used.

exchange

Designates the exchange package to be used to define a process virtual address. It may be specified as an integer or as a keyword. An integer value is interpreted as the starting real memory address of an exchange package.

The following are allowable keywords:

- Active (a) - indicates that the contents of the CPU at the time of the dump are to be used.
- Monitor (m) - indicates that the processor maintenance registers are to be accessed and the value in the MPS register be used as the real memory address of the exchange package.
- Job (j) - indicates that the processor maintenance

registers are to be accessed and the value in the JPS register be used as the real memory address of the exchange package.

If you do not specify a value for this parameter, the active exchange package is used.

processor

Specifies the number of the processor to be used for maintenance register access, to obtain an exchange package address or to locate an active exchange package.

address_mode

This parameter determines the interpretation of the address argument. Possible settings are:

- PROCESS_VIRTUAL_ADDRESS (PVA) - causes interpretation as follows:

  sssnnnnnnnn(16)

  with s being the segment number and n the offset.

- SYSTEM_VIRTUAL_ADDRESS (SVA) -

  aaaannnnnnnn(16)

  a is the assigned segment identifier (asid) and n is the offset.

- REAL_MEMORY_ADDRESS (RMA) - Adress specifies the starting real memory byte address to be displayed.

If you do not specify a value for this parameter, process_virtual_address is used.

Example:

```
ad/disv $memory(8, 8, m, 0, pva)
4003200064007CF0(16)
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.3 $MEMORY_STRING ($MS)

F1.3.3 $MEMORY_STRING ($MS)

Purpose: Returns the contents of central memory. The result is type string.

Format: $MEMORY_STRING(address, byte_count, exchange, processor, address_mode)

Parameters: address

Specifies the starting address of central memory to be returned as the result of the function.

This parameter is required.

byte_count

Specifies the number of consecutive bytes which constitues the value returned by $MEMORY_STRING. The minimum is zero bytes and maximum is 256.

If you do not specify the value of this parameter, a value of one is used.

exchange

Designates the exchange package to be used to define a process virtual address. It may be specified as an integer or as a keyword. An integer value is interpreted as the starting real memory address of an exchange package.

The following are allowable keywords:

- Active (a) - indicates that the contents of the CPU at the time of the dump are to be used.
- Monitor (m) - indicates that the processor maintenance registers are to be accessed and the value in the MPS register used as the real memory address of the exchange package.

- Job (j) - indicates that the processor maintenance registers are to be accessed and the value in the JPS register used as the real memory address of the exchange package.

If you do not specify a value for this parameter, the active exchange package is used.

processor

Specifies the number of the processor to be used for maintenance register access, to obtain an exchange package address or to locate an active exchange package.

address_mode

Determines the interpretation of the address argument. Possible settings are:

- PROCESS_VIRTUAL_ADDRESS (PVA) - causes interpretation as follows:

  sssnnnnnnnn(16)

with s being the segment number and n the offset.

- SYSTEM_VIRTUAL_ADDRESS (SVA) -

  aaaannnnnnnn(16)

a is the assigned segment identifier (asid) and n is the offset.

- REAL_MEMORY_ADDRESS (RMA) - Adress specifies the starting real memory byte address to be displayed.

If you do not specify a value for this parameter, process_virtual_address is used.

Example: ad/scb = $SYMBOL_ADDRESS(mtv$scb)

F1.0 NOS/VE DUMP ANALYZER
F1.3.3 $MEMORY_STRING ($MS)

```
ad/msga = scb + 3a(16)
ad/msgl = $memory(scb+38(16) 1 monitor)
ad/disv $MEMORY_STRING(msga msgl monitor)
HR- unable to flush file
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.4 $MODULE

F1.3.4 $MODULE

Purpose: Returns the module name corresponding to a given address by searching the debug table. The result is type string.

Format: $MODULE(address)

Parameters: address

Specifies the process virtual address to be used to search the debug table. An error status will be returned if no module can be found that corresponds to the given address.

Example:
```
ad/disv $module(301c0003e17e(16))
BAM$SYS_BLK_VARIABLE_REC_FAP
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.5 $NIL_PVA ($NP)

F1.3.5 $NIL_PVA ($NP)

Purpose: Returns a value of true or false based upon the given address. A nil pva is an integer of the form:

rsssnnnnnnnn(16)

Where r = 0f(16), sss = 0fff(16) and the offset portion is negative (bit 2 ** 31 is set). The result is type boolean.

Format: $NIL_PVA(address)

Parameters: address

Specifies the process virtual address to be tested.

Example:

```
ad/disv $NIL_PVA($memory(30000000(16)))
FALSE
ad/disv $NIL_PVA(0ffffffffffff(16))
TRUE
ad/disv $NIL_PVA(0ffff80000000(16))
TRUE
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.6 $OFFSET ($OFF)

F1.3.6 $OFFSET ($OFF)

Purpose: Returns the segment offset portion of a given address. The result is type integer.

Format: $OFFSET(address)

Parameters: address

Specifies the address that is to be interpreted.

This argument is required.

Example:
```
ad/disv $OFFSET(301c0003e17e(16))
3E17E(16)
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.7 $PP_MEMORY ($PM)

F1.3.7 $PP_MEMORY ($PM)

Purpose: Returns the contents of a word of pp memory. The result is type integer.

Format: $PP_MEMORY(pp_number, address)

Parameters: pp_number

Specifies the number of the perpheral processor. Values of 0 to 25 are allowed. If the specified pp is not included in the dump, an error status is returned.

This parameter is required.

address

Specifies the word in pp memory to be returned. Values of 0 to 4095 are allowed.

This parameter is required.

Example:

```
ad/disv $PP_MEMORY(6,140(8))
6(8)
```

F1.3.8 $PROCESS_REGISTER ($PR)

Purpose: Returns the contents of a process register as defined by an exchange package. The exchange package can be one active at the time of the dump or it can be an exchange package in memory. The result is type integer.

Format: $PROCESS_REGISTER(register_kind, register_number, exchange, processor)

Parameters: register_kind

Specifies the kind of register. The following keywords are allowed:

- p
- a
- x
- monitor_condition_register (mcr)
- user_condition_register (ucr)
- segment_table_address (sta)
- top_of_stack (tos)

This argument is required.

register_number

Defines the a, x or top_of_stack register. It is ignored for other register_kinds. Allowable values are 0 through 15 for a and x, 1 through 15 for top_of_stack.

If you do not specify a value for this parameter, a value of 0 is used.

exchange

F1.0 NOS/VE DUMP ANALYZER
F1.3.8 $PROCESS_REGISTER ($PR)

Specifies the exchange package location. It may be given as an integer or as a keyword. An integer value is interpreted as the starting real memory address of an exchange package.

The following are allowable keywords:

- Active (a) - indicates that the contents of the CPU at the time of the dump are to be used.
- Monitor (m) - indicates that the processor maintenance registers are to be accessed and the value in the MPS register be used as the real memory address of the exchange package.
- Job (j) - indicates that the processor maintenance registers are to be accessed and the value in the JPS register used as the real memory address of the exchange package.

If you do not specify a value for this parameter, the active exchange package is used.

processor

Specifies the number of the processor to be used for maintenance register access, to obtain an exchange package address or to locate an active exchange package.

Example:

```
ad/disv $PROCESS_REGISTER(a 2)
100F00000CB0(16)
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.9 $REAL_MEMORY_ADDRESS ($RMA)

F1.3.9 $REAL_MEMORY_ADDRESS ($RMA)

Purpose: Translates a process virtual address or a system virtual address into a real memory address. It can also be used to test for a valid real memory address. The result is type integer.

Format: $REAL_MEMORY_ADDRESS(address, exchange, processor, address_mode)

Parameters: address

Specifies the address to be converted.

exchange

Designates the exchange package to be used to define the virtual address. It may be specified as an integer or as a keyword. An integer value is interpreted as the starting real memory address of an exchange package.

The following are allowable keywords:

- Active (a) - indicates that the contents of the CPU at the time of the dump are to be used.
- Monitor (m) - indicates that the processor maintenance registers are to be accessed and the value in the MPS register be used as the real memory address of the exchange package.
- Job (j) - indicates that the processor maintenance registers are to be accessed and the value in the JPS register be used as the real memory address of the exchange package.

If you do not specify a value for this parameter, the active exchange package is used.

processor

F1.0 NOS/VE DUMP ANALYZER
F1.3.9 $REAL_MEMORY_ADDRESS ($RMA)

Specifies the number of the processor to be used for maintenance register access, to obtain an exchange package address or to locate an active exchange package.

address_mode

$real_memory_address ($rma)

This argument determines the interpretation of the address parameter. Possible settings are:

- PROCESS_VIRTUAL_ADDRESS (pva) - causes interpretation as follows:

  sssnnnnnnnn(16)

  with s being the segment number and n the offset.

- SYSTEM_VIRTUAL_ADDRESS (sva) -

  aaaannnnnnnn(16)

  a is the assigned segment identifier (asid) and n is the offset.

- REAL_MEMORY_ADDRESS (rma) - Adress specifies a real memory address. An error status will be returned if the given address is not on the dump. Otherwise, the address argument is returned as the function result.

If you do not specify a value for this parameter, PROCESS_VIRTUAL_ADDRESS is used.

Example:

```
ad/disv $RMA(100F00000CB0(16))
588CB0(16)
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.10 $RING

F1.3.10 $RING

Purpose: Returns the ring portion of a given process virtual address. The result is type integer.

Format: $RING(address)

Parameters: address

Specifies the address that is to be interpreted.

This argument is required.

Example:

```
ad/disv $RING(301c0003e17e(16))
3(16)
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.11 $SECTION ($SEC)

F1.3.11 $SECTION ($SEC)

Purpose: Returns the section name and offset into the section given a process virtual address. This function uses the debug table. The result is type string.

Format: $SECTION(address)

Parameters: address

Specifies the process virtual address to be used to search the debug table. An error status will be returned if no entry can be found that corresponds to the given address. This parameter is required.

Example:
```
ad/disv $SECTION(301c0003e17e(16))
GET_NEXT + 9AE(16)
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.12 $SEGMENT ($SEG)

F1.3.12 $SEGMENT ($SEG)

Purpose: Returns the segment portion of a given process virtual address. The result is type integer.

Format: $SEGMENT(address)

Parameters: address

Specifies the address that is to be interpreted.

This argument is required.

Example:

```
ad/disv $SEGMENT(301c0003e17e(16))
1C(16)
```

F1.0 NOS/VE DUMP ANALYZER
F1.3.13 $SYMBOL_ADDRESS ($SA)

F1.3.13 $SYMBOL_ADDRESS ($SA)

Purpose: Converts a name into a process virtual address by accessing the debug table. It allows you to find the address of a variable, procedure or module. The result is type integer.

Format:

```
$SYMBOL_ADDRESS(symbol_name)
```

Parameters:

symbol_name

Specifies the name of the variable, procedure or module.

This argument is required.

Example:

```
ad/disv $SYMBOL_ADDRESS(mtv$scb)
100000238(16)
```

F1.0 NOS/VE DUMP ANALYZER
F1.4 QUICK REFERENCE

F1.4 QUICK_REFERENCE

F1.4.1 ANALYZE_DUMP COMMAND

```
analyze_dump (anad)
  dump_file, df     : file = $optional
  restart_file, rf  : file = $local.restart_file
  debug_table, dt   : file or key running_system, rs, none, n = running_system
  title, t          : string 1..25 = $optional
  output, o         : file = $output
  status            : var of status = $optional
```

F1.4.2 SUBCOMMANDS

```
change_default (chad)
 exchange, e       : integer 0..$max_integer or key monitor, m, job, j, ..
                        active, a = $optional
 processor, p      : integer 0..3 = $optional
 bytes, b          : integer 0..33554432 = $optional
 address_mode, am  : key process_virtual_address, pva, ..
                        system_virtual_address, sva, real_memory_address, ..
                        rma = $optional
 words, w          : integer 0 .. 4096 = $optional
 status            : var of status = $optional


change_processor_register (change_processor_registers,  chapr)
 job_process_state, jps      : integer 0..$max_integer = $optional
 monitor_process_state, mps  : integer 0..$max_integer = $optional
 page_size_mask, psm         : integer 0..127 = $optional
 page_table_address, pta     : integer 0..$max_integer = $optional
 page_table_length, ptl      : integer 0..255 = $optional
 processor, p                : integer 0..3 or key all, a = 0
 status                      : var of status = $optional



copy_memory (copm)
 address, a          : integer = $required
 file, f             : file = $required
 byte_count, bc      : integer 0..$max_integer = 100000(16)
 exchange, e         : integer 0..$max_integer or key active, a, ..
                          monitor, m, job, j = active
 processor, p        : integer 0..3 = 0
```

F1.0 NOS/VE DUMP ANALYZER
F1.4.2 SUBCOMMANDS

```
 address_mode, am    : key process_virtual_address, pva, ..
                           system_virtual_address, sva, ..
                           real_memory_address, rma = ..
                           process_virtual_address
 status              : var of status = $optional


copy_pp_memory (coppm)
 pp_number, pn   : integer 0..25 = $required
 file, f         : file = $required
 address, a      : integer 0..4095 = 0
 word_count, wc  : integer 0..4096 = 4096
 status          : var of status = $optional


display_call (display_calls, disc)
 exchange, e         : integer 0..$max_integer or key monitor, m, job, j, ..
                           active, a = active
 processor, p        : integer 0..3 = 0
 count, c            : integer 1..10000 or key all = all
 start, s            : integer 0..10000 or key exchange_package, ep ..
                           = exchange_package
 display_option, do  : list 1..2 of key full, f, brief, b, save, s = brief
 title, t            : string 1..31 = 'display_call'
 output, o           : file = $output
 status              : var of status = $optional


display_exchange_package (disep)
 exchange, e   : integer 0..$max_integer or key monitor, m, job, j, ..
                     active, a = active
 processor, p  : integer 0..3 = 0
 title, t      : string 1..31 = 'display_exchange_package'
 output, o     : file = $output
 status        : var of status = $optional


display_maintenance_registers (display_maintenance_register, dismr)
 element, e          : key processor, p, input_output_unit, iou, memory, m, ..
                           all = all
 processor, p        : integer 0..3 = 0
 display_option, do  : key full, f, brief, b = full
 title, t            : string 1..31 = 'display_maintenance_registers'
 output, o           : file = $output
```

F1.0 NOS/VE DUMP ANALYZER
F1.4.2 SUBCOMMANDS

```
 status                : var of status = $optional


display_memory (dism)
 address, a            : integer = $required
 bytes, b              : integer 0..33554432 = 8
 exchange, e           : integer 0..$max_integer or key active, a, ..
                             monitor, m, job, j = active
 processor, p          : integer 0..3 = 0
 address_mode, am      : key process_virtual_address, pva, ..
                             system_virtual_address, sva, ..
                             real_memory_address, rma = ..
                             process_virtual_address
 display_option, do    : list 1..2 of key numeric, n, ascii, a ..
                             = (numeric,     ascii)
 title, t              : string 1..31 = 'display_memory'
 output, o             : file = $optional
 status                : var of status = $optional


display_pp_memory (dispm)
 pp_number, pn                      : integer 0..25 = $required
 address, a                         : integer 0..4095 = 0
 words, w                           : integer 0..4096 = 4096
 display_option, do                 : list 1..2 of key numeric, n, ..
                                          ascii, a = (numeric, ascii)
 radix, r                           : integer 8..16 = 8
 display_relocation_register, drr   : boolean = $optional
 title, t                           : string 1..31 = $optional
 output, o                          : file = $optional
 status                             : var of status = $optional


quit (qui)
 status : var of status = $optional
```

F1.4.3 FUNCTIONS

```
$maintenance_register: integer ($mr)
 register  : integer 0 .. 0ff(16) or key monitor_process_state, mps,
                job_process_state, jps = $required
 element   : key processor, p, input_output_unit, iou, memory, m
                = processor
```

F1.0 NOS/VE DUMP ANALYZER
F1.4.3 FUNCTIONS

```
 processor : integer 0..3 = 0



$memory: integer ($mem)
 address           : integer = $required
 byte_count        : integer 1..8 = 6
 exchange          : integer 0..$max_integer or key active, a, monitor, m, ..
                         job, j = active
 processor         : integer 0..3 = 0
 address_mode, am  : key process_virtual_address, pva, ..
                         system_virtual_address, sva, real_memory_address, ..
                         rma = process_virtual_address


$memory_string: string ($ms)
 address           : integer = $required
 byte_count        : integer 0..osc$max_string_size = 1
 exchange          : integer 0..$max_integer or key active, a, monitor, m, ..
                         job, j = active
 processor         : integer 0..3 = 0
 address_mode, am  : key process_virtual_address, pva, ..
                         system_virtual_address, sva, real_memory_address, ..
                         rma = process_virtual_address


$module: string
 address: integer = $required


$nil_pva: boolean ($np)
 address: integer = $required


$offset: integer ($off)
 address: integer = $required


$pp_memory: integer ($pm)
 pp_number: integer 0 .. 25 = $required
 address  : integer 0 .. 4095 = $required
```

F1.0 NOS/VE DUMP ANALYZER
F1.4.3 FUNCTIONS

```
$process_register: integer ($pr)
 register_type   : key p, a, x, monitor_condition_register, mcr
                       user_condition_register, ucr
                       segment_table_address, sta, top_of_stack, tos
                       = $required
 register_number : integer 0..0f(16) = 0
 exchange        : integer 0..$max_integer or key active, a, monitor, m, ..
                     job, j = active
 processor       : integer 0..3 = 0



$real_memory_address: integer ($rma)
 address      : integer = $required
 exchange     : integer 0..$max_integer or key active, a, monitor, m, ..
                    job, j = active
 processor    : integer 0..3 = 0
 address_mode : key process_virtual_address, pva, ..
                       system_virtual_address, sva, real_memory_address, ..
                       rma = process_virtual_address


$ring: integer
 address : integer = $required


$section: string ($sec)
 address : integer = $required


$segment: integer ($seg)
 address : integer = $required


$symbol_address: integer ($sa)
 symbol_name : name = $required
```

Table of Contents