CYCLE 9 HELPFUL HINTS

This paper describes helpful hints on how to use Cycle 9 of NOS/VE. It is intended to supplement, rather than to replace, the standard NOS/VE documentation. If you have any questions or suggestions, please see Tom McGee or Bonnie Swierzbin. Appendix D lists background documents and how to obtain them.

To obtain additional copies of this document while running on SN101 at Arden Hills, please type:

```
SES,INT1.LISTHINTS  C=<number of copies>
```

To obtain a copy with revision bars against the Helpful Hints of the previous build, one can type:

```
SES,INT1.LISTHINTS  REVB C=<number of copies>
```

The C parameter is optional and defaults to one.

Update_History

Date	Changes
12/22/80	Revisions for NOS/VE Phase C
2/12/81	Additional Revisions for NOS/VE Phase C
6/09/81	Revisions for NOS/VE Build N
6/19/81	Additional Revisions for NOS/VE Build N
8/28/81	Revisions for NOS/VE Build O
11/06/81	Revisions for NOS/VE Build P
3/01/82	Revisions for NOS/VE Build Q
4/15/82	Revisions for NOS/VE Cycle 2
5/01/82	Revisions for NOS/VE Cycle 3
6/30/82	Revisions for NOS/VE Cycle 5
7/29/82	Revisions for NOS/VE Cycle 7
10/22/82	Revisions for NOS/VE Cycle 9

1.0 MAJOR CHARACTERISTICS OF THIS BUILD

- Cycle 6 and subsequent cycles of NOS/VE are built to run with a NOS 6.1 base system and the DSD displays and commands have changed considerably from the NOS 5.3 base system used with Cycle 5. (See also the next bullet.) This hints document has been updated to reflect these changes where they affect NOS/VE, for instance, using the command NVEnnnn to bring up NOS/VE instead of cp.NVEnnnn. For more information on the NOS commands see BJ Swierzbin's memo of July 23 titled 'NOS 6.1 Notes', the NOS reference manuals listed in Appendix A, or the NOS R6 Software Release Bulletin.

  WARNING (1): NOS jobs and subsystems may now be brought down by entering STOP,jsn., for example, STOP,BIO. will bring down BATCHIO. DO NOT, DO NOT, DO NOT enter STOP,NVE.

  WARNING (2): SES R2C must be used with NOS V2 systems.

  The formats of the validations and user index files (usually named VALIDUZ and VALINDZ) have changed between NOS Version 1 and NOS Version 2. The modset which was added to the NOS 5E55 system (the base for Cycle 7) changing the names of these files to NEWDUZ and NEWNDZ has <u>not</u> been added to 5F40 (the base for Cycle 9). This means that in order to run Cycle 9 the site analyst must make a copy of file NEWDUZ on file VALIDUZ and a copy of file file NEWNDZ on file VALINDZ. Then when there is no longer any need to run Cycle 7, NEWDUZ and NEWNDZ can be purged.

- 9.1 runs with NOS 5F41 and 1 Series networks, which is the same level of networks that Cycle 7 runs with. 9.2 and subsequent systems run with NOS 5G41 and 4 Series networks. The process for bringing up NAM and IAF has changed with the new level networks; see Section 4 for details.

- Substantial changes have been made to the DS procedure in Cycle 9. See Section 4 for details.

- Backup files created in Cycle 7 by the NOS/VE BACKUP_PERMANENT_FILE utility will not be able to be used by the RESTORE_PERMANENT_FILE utility in Cycles 9 and 10. Also, backup files created in Cycle 9 will not be able to be used by the RESTORE_PERMANENT_FILE utility in Cycle 10.

1.0 MAJOR CHARACTERISTICS OF THIS BUILD

o SCL Command List Changes

Management of SCL command lists has changed, particularly as it relates to asynchronous tasks. The changes were made primarily for performance and should have no impact on most (all) users. The changes are:

- An asynchronous task now has its own "job level" command list which is initialized to the "job level" command list of its parent task. The asynchronous task may alter its command list without affecting its parent task.

- The system supplied command library (osf$command_library) is now searched as part of the $system command list entry.

- Support for the following "old" command names has been removed:
  ACCEPT, CT, DECLARE_VARIABLE, DECVAR, DO, INCLUDE, REMOVE_VARIABLE and REMVAR

- Certain commands which previously were always found at the front of the command list have been moved to a list which is searched after all other command list entries have been searched. (This is to allow a user to supply his/her own version.) The affected commands are:
  ACCEPT_LINE, ACCL, CREATE_VARIABLE, CREV, DELETE_VARIABLE, DELV, INCF, INCLUDE_FILE, INCL, INCLUDE_LINE, SETCL and SET_COMMAND_LIST.

- The convention for function names (that their first character be a "$") has been made into a rule (i.e. a name that does not begin with a "$" will not be recognized as a function name.)

- The PROCedure "append_command_list" no longer exists. Its operation has been replaced by a new parameter on the set_command_list command, namely "placement" or "p" with possible values of "after", "a", "before", and "b", with "before" as the default. This parameter affects whether command list entries being added are placed before or after the current entries in the list.

- The commands documented as being operator only commands are now actually available only to the operator.

- The HCS commands are no longer supported.

1.0 MAJOR CHARACTERISTICS OF THIS BUILD

- The full mode of the display_command_list command has been upgraded so that the commands on osf$command_library are displayed as part of the $system command list entry. Also, the commands available on an object library which is a command list entry are shown in the display.

- The $file function now supports the cycle_number (cn) option.

o QUICK_DEADSTART does not work in Cycle 9. Use of this command will hang the system.

o To access NOS/VE interactively, the user must login to the application named VEIAF. For example: ,DAH,DAHX,VEIAF The previous application name was TAF.

o With Cycle 7, the operator of the NOS/VE dual state system can simulate terminal breaks. She/he may issue at any time a *BREAK at the K display for the NVE job. This will start the terminal break, which works just like the interactive break. The broken command may be continued by issuing the resume_command (resc) or may be terminated by issuing the terminate_command (terc). If the command get_file is broken into, the command will be terminated by the Remote Host. After the DS proc has been executed, a terminal break at the command level will cause the SCL task to terminate. The job monitor task will restart the SCL task after a brief delay (up to 20 seconds).

o On 2 occasions, a procedure which (among other things) executed a TERMINATE_LOG crashed a Cycle 7 system. The PSR number for this problem is not available yet and the guilty party hasn't been identified, but it is suggested that the user treat TERMINATE_LOG with a healthy suspicion.

o The user should not issue a pause or terminate break while DISPLAY_COMMAND_LIST is executing; this will CRASH the system.

o Known Remote Host problems:

The user name put on the banner for a job routed from the 170 is incorrect. This has been reported with PSR NVOD073 and will be fixed in Cycle 9.

o Miscellaneous SCL Changes:

A RING parameter has been added to the TASK/TASKEND command. This parameter may be used to switch to a new ring of execution within the user's validated minimum_ring and nominal_ring.

A $RING function has been added. It has no arguments and returns the current ring of execution.

Commands from user jobs are no longer written to the system log by default. This function is controlled by the new operator commands ACTIVATE_SYSTEM_LOGGING and DEACTIVATE_SYSTEM_LOGGING.

The abbreviation for the SKIP_TAPE command has been corrected to SKIT from SKIPT.

A new internal interface, CLP$VALIDATE_NAME has been added. This is an INLINE procedure that should be used instead of the high overhead CLP$CONVERT_STRING_TO_NAME. (See common deck CLXVN for interface details.)

- DISPLAY_FILE leaves the source file with an attribute of RT=U. This can be corrected by setting the RT attribute back to the correct value after the DISPLAY value: SETFA file RT=V

- COPY_FILE will not copy at EOI if it had been specified on the COPF command. The following sequence will accomplish copying at EOI:

```
SETFA dest_file OP=$EOI
COPF source_file dest_file
SETFA dest_file OP=$BOI
```

- The permanent file system has been modified to make files invisible when all cycles have been purged even if one or more of those cycles are still attached. This results in references to the file emitting an unknown file message as expected, rather than an unknown cycle message containing an invalid cycle number. It also prevents display catalog commands from showing the file as having zero cycles.

- The administer utility has been updated to correspond to Rev. 9 of the NOS/VE Command Interface ERS. Old command and parameter names are no longer supported.

- The TAFNVE operator command (TAF control point) is no longer required or available. The capabilities that were provided by it have been packaged within the NVE subsystem control point. The impact of this change is as follows:

~~~~~~

1.0 MAJOR CHARACTERISTICS OF THIS BUILD

~~~~~~

- The K display is assigned to the same control point during both deadstart and normal system operation.
- Output from the system core debugger will no longer appear at the NVE control point K-display. All system core debugger communication is via the MDD terminal.
- The K.*BYEVE. command is no longer available.
- The OFFSW,jsn,6. command before doing a STOP,jsn. is not required to bring NVE down.
- All capabilities are available via the NVExxxx. command. The NVE subsystem may be placed at any control point (like NAM is).

o Cycle 5 (actually Cycle 4) of NOS/VE no longer supports the "old" command names for system commands. Both "old" and "new" names have been supported since Build Q. See DAP ARH4776 for details. As part of this change the abbreviations for the COPY_FILE and PRINT_FILE commands have been corrected to COPF and PRIF (from COPYF and PRINTF).

o The EXPLAIN command was implemented in Cycle 4 of NOS/VE. Don't get too excited about this, however, because as yet there are no "explain level" message templates for any of the system conditions. EXPLAIN will simply regurgitate the regular message.

While implementing the EXPLAIN command it was discovered that the specification of the command (i.e. that it have an optional "condition" parameter) was not nearly as useful as having the first parameter be a "status" value. So the implementation deviates from the ERS in that the first parameter to EXPLAIN is MESSAGE_STATUS or MS and is of kind STATUS. The $STATUS function can be used to transform a "condition" into a "status". A DAP is being written to make this change official.

o NOS/VE multiple mass storage volumes have been implemented in Cycle 5. For more information see the section 'Configuration Management'.

o The Interstate Communication Facility, which is described in Section 9 of the NOS/VE ERS - Program Interface (Rev. 8), has been implemented. The callable subroutines described in section 9.3 reside in User Library LINKLIB in the Integration catalogs INT1, DEV1, REL1, etc. NOS libraries SYSLIB and SRVLIB are also required to complete the loading process.

1.0 MAJOR CHARACTERISTICS OF THIS BUILD

LDSET loader commands must be used to select these libraries.

- o Interactive Usage Restrictions:

  - When logging in to NOS/VE (i.e. HELLO,VEIAF etc.) do not enter a terminate break (CTRL t) or a pause break (CTRL p) before the 'welcome message' appears at the terminal. A pause or terminate break entered before the interactive NOS/VE job has completed it's initialization may crash the system.

  - A REQUEST_TERMINAL command in a batch job no longer crashes the system. !

- o Any product or utility that is placed in the $SYSTEM catalog (or any frequently loaded program) should be bound using the CREATE_MODULE subcommand of the CREATE_OBJECT_LIBRARY utility. This will minimize overhead associated with loading the product or utility.

- o Debug responds to terminal breaks when a program is being debugged. However, entering a pause or terminate break when debug is active (i.e. the DB/ prompt has appeared and the user has not issued the RUN command) will cause the task to terminate.

- o When sharing executable files via permanent files (i.e. compilers, libraries, etc.) you should make the file an object library via the CREATE_OBJECT_LIBRARY utility. By sharing object libraries instead of object files, the code is actually shared among all tasks using the library; the library is not copied to another segment but is executed directly.

1.1 NOS/VE_USAGE_EXAMPLES

1.1.1 EXECUTING PROGRAMS

PROCESS

Create an object text file by compiling a program on NOS. Then perform the following steps on NOS/VE:

- Acquire any necessary libraries (which are not quoted in text embedded directives) by either:

- o Attaching them from the system catalog, either explicitly or via prolog

  or
- o Creating the library file via the object library generator

  or
- o Staging the library file from NOS to NOS/VE using the GET_OBJECT_LIBRARY command.

- Get the file from NOS and convert the object text file from the CI data mapping to II data mapping by executing the CONVERT_OBJECT_FILE command.

- Load and execute the program via the EXECUTE_TASK command, specifying the necessary libraries with the LIBRARY parameter; alternatively SET_PROGRAM_ATTRIBUTES may be used to include the libraries in all subsequent EXECUTE_TASK commands.

- Stage the loadmap from NOS/VE to NOS for printing by using either:

  - o The REPLACE_FILE command with A6 conversion mode specified if running on the simulator.

    or
  - o The PRINT_FILE command if running on the hardware.

EXAMPLES

The following is an example command sequence for executing a program not requiring any libraries for loading:

Assumptions: all modules to be loaded are contained on the NOS permanent file 'citxtrs'.

```
CONVERT_OBJECT_FILE CITXTRS
EXECUTE_TASK CITXTRS PARAMETER='program parameters'
PRINT_FILE LOADMAP
```

The following is an example command sequence for executing a program requiring libraries for loading:

Assumptions: the NOS permanent file 'citxtrs' contains object text generated by the CYBIL CI compiler. The compiler modules reference procedures contained on the library 'mylib' and the CYBIL run-time library. These libraries have been generated on NOS/VE and saved on NOS.

```
GET_OBJECT_LIBRARY MYLIB
SET_PROGRAM_ATTRIBUTES LOAD_MAP_OPTIONS=(BLOCK,ENTRY_POINT,SEGMENT
```

1.0 MAJOR CHARACTERISTICS OF THIS BUILD
1.1.1 EXECUTING PROGRAMS

```
CONVERT_OBJECT_FILE CITXTRS
EXECUTE_TASK CITXTRS 'program parameters' LIBRARY=MYLIB
PRINT_FILE LOADMAP
```

1.1.2 CREATE OBJECT LIBRARY ON NOS/VE AND SAVE IT ON NOS

Notes:

- o CLG0170 is NOS permanent file name for file containing CI object text for modules to be included in the library.
- o IITEXT180 is NOS/VE local file name for file containing II object text for modules to be included in the library.
- o LIBRARY180 is NOS/VE local file name for the library being created.
- o ILIB170 is NOS permanent file name for file containing the library.

NOS/VE Job Commands

```
CONVERT_OBJECT_FILE IITEXT180 CLG0170
CREATE_OBJECT_LIBRARY
ADD_MODULE LIBRARY=IITEXT180
GENERATE_LIBRARY LIBRARY=LIBRARY180
QUIT
REPLACE_FILE LIBRARY180 ILIB170 DC=856
```

1.1.3 MODIFY A PREVIOUSLY SAVED OBJECT LIBRARY

Notes:

- o ILIB170 is NOS permanent file name for file containing the old library
- o LIBRARY180 is NOS/VE local file name for file containing the old library
- o CMOD170 is NOS permanent file name for file containing CI object text for the new module
- o NEWIIMODULE is NOS/VE local file name for file containing II object text for the new module

- o NEWLIBRARY is NOS/VE local file name for the library being created

- o NLIB170 is NOS local file name for new library

NOS/VE Job Commands

```
GET_OBJECT_LIBRARY LIBRARY180 ILIB170
CONVERT_OBJECT_FILE NEWIIMODULE CMOD170
CREATE_OBJECT_LIBRARY
ADD_MODULE LIBRARY=LIBRARY180
REPLACE_MODULE LIBRARY=NEWIIMODULE
GENERATE_LIBRARY LIBRARY=NEWLIBRARY
QUIT
REPLACE_FILE NEWLIBRARY NLIB170 DC=B56
```

1.1.4 ROUTE AN INPUT FILE FROM NOS TO NOS/VE

Running from an interactive terminal, enter:

```
GET,filename.
ROUTE,filename,DC=LP,FC=RH.
```

The input file which is sent to NOS/VE must be in 6/12 ASCII (or display code subset). The job file must be a single partition NOS record containing NOS/VE commands. The first statement must be a valid LOGIN command with user, password and family name specified. Multi partition input files are not supported by NOS/VE so NOS data files used by the program must be obtained through the GET_FILE command.

1.1.5 PRINT A NOS/VE FILE

At NOS/VE job termination the job log will be automatically returned to NOS. The job log will be appended to the NOS/VE output file OUTPUT. NOS/VE print files must be written by BAM as 8/8 ASCII RT=V. Print files will be converted from 8/8 ASCII RT=V to NOS 8/12 ASCII when they are sent to NOS and will be printed in upper/lower case.

All NOS/VE output files will appear in the NOS output queue (NOS Q,PR display) with the name NVExxxx as a banner. In order to print a NOS/VE file, the following command must be issued within your job or be entered from the system console via the operator facility:

```
        PRINT_FILE filename
```

2.0 COMMAND_INTERFACE_STATUS

2.1 ACCESS_TO_NOS/VE_IN_DUAL_STATE

2.1.1 LOGIN TO NOS/VE

To initially login to NOS/VE via VEIAF, you must cause the first login attempt to fail. This can be done by responding to the "FAMILY:" login prompt with something like: ",,,". This must be done because the system will try to connect the terminal to IAF on the first login attempt no matter what is typed. To access VEIAF do the following on the second "FAMILY:" prompt:

```
,user,password,VEIAF
```

You can access VEIAF from IAF by doing "HELLO,VEIAF" or by answering VEIAF to the system prompt "APPLICATION:".

2.1.2 TERMINAL USAGE

1) The slant (/) is the prompt to enter a NOS/VE command. Any normal NOS/VE command can now be entered (continuation lines are prompted with ../). The full ASCII character set, lower or upper case and all special characters, can be used.

2) A LOGOUT command will cause the NOS/VE Interactive Job to terminate. A new NOS/VE Interactive Job can then be started by responding to the 'APPLICATION:' prompt with VEIAF.

3) Terminal breaks (control-t and control-p) can be used to terminate a task or command and suspend a task and enter a new task to process SCL commands. Control-t causes a terminate break and control-p causes a pause break. Terminate break will terminate a command or the most recently executed task. A pause break will suspend execution and allow commands to be entered. When a

terminal is in pause break state, two additional commands are available:

RESUME_COMMAND - resume execution at the point of interruption.

TERMINATE_COMMAND - cause a terminate break condition as a terminate break had been entered.

Both terminate break and pause break are available to programs as conditions via the program management condition mechanism.

2.1.3 NOS/VE PROGRAM ACCESS TO THE TERMINAL

1) Interactive NOS/VE jobs are able to obtain terminal input through the AMP$GET_NEXT or AMP$GET_PARTIAL program interface which can be used by both task services and user ring programs. Interactive programs which use this interface should be able to handle both upper and lower case input in order to make them more convenient to use in both 64 and 96 character set modes.

2.2 COMMAND_AND_PARAMETER_NAMES

During the next few months a command supported by the system may not be in sync with your command interface document. The parameter descriptor table gives an accurate, concise description of the command interface as currently supported.

PDT_Reader's_Guide

The definition of a command's parameter list is enclosed in parenthesis with a parameter description per line. Each description has the general form:

PARAMETER NAME: ALLOWED PARAMETER VALUES = PARAMETER DEFAULT VALUE

Parameter Names - describes the parameter name and any abbreviations.

ALLOWED PARAMETER VALUES - describes the kind of value allowed and whether a list of values is possible. The value kind can be further qualified. In some cases, the actual values allowed are described using the KEY notation. The value kinds include INTEGER, STRING, NAME, FILE, STATUS.

PARAMETER DEFAULT VALUES - describes the defaulting rules and/or values for the parameter. $REQUIRED and $OPTIONAL are obvious. Other values in this position will be treated as if they were entered by the user on command invocation.

See the PROC command in the Command Interface ERS for more details.

The PDTs for the commands currently in the system can be displayed using the DISPLAY_COMMAND_INFORMATION command. This is documented in the nonstandard command section of this document.

2.3 COMMAND FUNCTIONS

Function	Status
$MOD	unchanged
$CHAR	unchanged
$CLOCK	unchanged
$DATE	unchanged
$FILE	unchanged
$FNAME	unchanged
$INTEGER	unchanged
$NAME	unchanged
$ORD	unchanged
$REAL	unchanged
$STRING	unchanged
$STRLEN	unchanged
$STRREP	unchanged
$SUBSTR	unchanged
$UNIQUE	unchanged
$TIME	unchanged
$VAR	unchanged
$SPECIFIED	unchanged
$SET_COUNT	unchanged
$VALUE_COUNT	unchanged
$RANGE	unchanged
$PARAMETER_LIST	unchanged
$PARAMETER	unchanged
$STATUS	unchanged
$CONDITION	unchanged
$SEVERITY	unchanged
$PROCESSOR	unchanged

$JOB	unchanged
$PROGRAM	unchanged
$RING	new

2.4 SYSTEM_ACCESS_COMMANDS

Commands	Status
SET_LINK_ATTRIBUTES	unchanged
LOGIN	unchanged - *1
LOGOUT	unchanged
SET_PASSWORD	unchanged

*1 The family name of the job doing the submit will be used as the default family name on batch jobs. The default for jobs submitted from NOS will be family $SYSTEM. This effectively means that whenever NOS/VE jobs are submitted from NOS the family parameter is required.

2.5 RESOURCE_MANAGEMENT

Command	Status
REQUEST_TERMINAL	unchanged

2.6 FILE_MANAGEMENT

Command	Status
SET_FILE_ATTRIBUTES	unchanged
COPY_FILE	unchanged
DISPLAY_FILE	unchanged
COMPARE_FILE	unchanged
DISPLAY_FILE_ATTRIBUTES	unchanged
SKIP_TAPE	unchanged

2.7 PERMANENT_FILE_MANAGEMENT

Command	Status
GET_FILE	unchanged
REPLACE_FILE	unchanged

CREATE_FILE	unchanged
ATTACH_FILE	unchanged
DELETE_FILE	unchanged
CHANGE_CATALOG_ENTRY	unchanged
CREATE_FILE_PERMIT	unchanged
DELETE_FILE_PERMIT	unchanged
CREATE_CATALOG	unchanged
DELETE_CATALOG	unchanged
DELETE_CATALOG_PERMIT	unchanged
CREATE_PERMIT_CATALOG	unchanged
DISPLAY_CATALOG	unchanged
DISPLAY_CATALOG_ENTRY	unchanged
SET_WORKING_CATALOG	unchanged

2.8 SCL STATEMENTS AND PROCEDURES

Command	Status
PROC/PROCEND	unchanged
SET_COMMAND_LIST	unchanged
DISPLAY_COMMAND_LIST	unchanged
REPEAT/UNTIL	unchanged
WHILE/WHILEND	unchanged
CREATE_VARIABLE	unchanged
DELETE_VARIABLE	unchanged
BLOCK/BLOCKEND	unchanged
LOOP/LOOPEND	unchanged
FOR/FOREND	unchanged
IF/ELSEIF/ELSE/IFEND	unchanged
CYCLE	unchanged
EXIT	unchanged
WHEN/WHENEND	unchanged
CONTINUE	unchanged
CANCEL	unchanged
INCLUDE_FILE	unchanged
COLLECT_TEXT	unchanged
DISPLAY_VALUE	unchanged
EXIT_PROC	unchanged
ACCEPT_LINE	unchanged
INCLUDE_LINE	unchanged
CREATE_FILE_CONNECTION	unchanged
DELETE_FILE_CONNECTION	unchanged
DISPLAY_FILE_CONNECTION	unchanged
change HCS variable	unchanged
display HCS variable	unchanged

2.9 INTERACTIVE_COMMANDS

Command	Status
RESUME_COMMAND	unchanged
TERMINATE_COMMAND	unchanged
SET_TERMINAL_ATTRIBUTES	unchanged
DISPLAY_TERMINAL_ATTRIBUTES	unchanged
esc-e	new - 1*
esc-l	new - 1*
esc-j	new - 1*
esc-t	new - 1*
esc-x	new - 1*

*1 These commands are entered with the 3-key sequence: escape_key, character, carriage_return. The characters have the following meanings:

e	perform "display_job_status" command
l	perform "display_log 10" command
j	perform "display_job_status all" command
t	discard all unprocessed, typed-ahead input
x	terminate job, but do not disconnect

2.10 OBJECT_CODE_MAINTENANCE

Command	Status
CREATE_OBJECT_LIBRARY	unchanged
DISPLAY_NEW_LIBRARY	unchanged
SELECT_DISPLAY_OPTION	unchanged
ADD_MODULE	unchanged
REPLACE_MODULE	unchanged
COMBINE_MODULE	unchanged
CREATE_MODULE	unchanged
BIND_MODULE	unchanged
CREATE_PROGRAM_DESCRIPTION	unchanged
DELETE_MODULE	unchanged
CHANGE_MODULE_ATTRIBUTE	unchanged
SATISFY_EXTERNAL_REFERENCES	unchanged
REORDER_MODULE	unchanged
GENERATE_LIBRARY	unchanged
DISPLAY_OBJECT_LIBRARY	unchanged
COMPARE_OBJECT_LIBRARY	unchanged
QUIT	unchanged
CI to II Conversion	unchanged

2.11 USER_SERVICES

Command	Status
DISPLAY_LOG	unchanged
DISPLAY_MESSAGE	unchanged

2.12 FILE_ROUTING

Command	Status
HCS JMROUTE	removed

2.13 PROGRAM_EXECUTION

Command	Status
SET_PROGRAM_ATTRIBUTES	unchanged
DISPLAY_PROGRAM	unchanged
EXECUTE	unchanged
"name call"	unchanged - *1
TASK/TASKEND	unchanged
TERMINATE_TASK	unchanged
WAIT	unchanged
SET_DEBUG_RING	unchanged
DISPLAY_ACTIVE_TASKS	unchanged

*1 Warning - "name call" works only for SCL procedures unless a SETFA command has been issued to specify that the FILE_CONTENTS are OBJECT and the FILE_ORGANIZATION is DATA or LIBRARY. The SETFA command must be reissued every time the file is brought over from NOS. The CONVERT_OBJECT_FILE, GET_OBJECT_FILE, and GET_OBJECT_LIBRARY nonstandard commands issue the appropriate SET_FILE_ATTRIBUTES command and are therefore recommended.

2.14 JOB_MANAGEMENT

Command	Status
SUBMIT_JOB	unchanged
DISPLAY_JOB_STATUS	unchanged
TERMINATE_JOB	unchanged
PRINT_FILE	unchanged
TERMINATE_PRINT	unchanged
DISPLAY_PRINT_STATUS	unchanged

2.15 NON_STANDARD_COMMANDS

The following commands provide a nonstandard means of performing various frequently performed functions. They may be superceded in subsequent builds by standard commands and capabilities.

2.15.1 DELETE_CATALOG_CONTENTS : DELCC

The purpose of this command is to delete all entries from the specified catalog. This includes subcatalogs and the files they contain.

```
delete_catalog_contents [catalog=<catalog>]
                         [status=<status variable>]
```

catalog!c: This parameter specifies from which catalog all files are to be deleted. Omission will cause the current working catalog to be used.

status: See ERROR HANDLING.

2.15.2 DISPLAY_ACTIVE_TASK : DISAT

The purpose of this command is to display task statistics for all currently active tasks in a job. The following information is displayed.

task name
execution time use
number of page faults

```
display_active_task [output=<file>]
                     [status=<status variable>]
```

output!o: This parameter specifies the file to which the task statistics is displayed. Omission will cause $OUTPUT to be used.

2.15.3 DISPLAY_SYSTEM_DATA ! DISSD

The purpose of this command is to display system page fault statistics and system monitor request statistics.

```
display_system_data [display_option=page_faults!pf
                      !monitor_requests!mr!all]
                    [display_format=incremental!i!total!t]
                    [output=<file>]
                    [status=<status variable>]
```

display_option!do: This parameter specifies which statistics are to be displayed. The following options are allowed :

page_faults	- display the page fault statistics.
monitor_requests	- display the system monitor request statistics.
Omission	will cause ALL to be used.

display_format!df: This parameter specifies whether a display of the all statistics recorded so far (total) or only those statistics recorded since the last display_system_data command (incremental) should be displayed. Omission will cause incremental to be used.

output!o: This parameter specifies the file to which the system data will be displayed. Omission will cause $OUTPUT to be used.

status: See ERROR HANDLING.

2.15.4 DISPLAY_JOB_DATA ! DISJD

The purpose of this command is to display the following job related statistics:

time in job mode
time in monitor mode
count of page in operations
reclaimed pages
new pages assigned
working set size
count of ready tasks

```
display_job_data [display_option=job_data]
                 [display_format=incremental!i!total!t]
                 [output=<file>]
                 [status=<status variable>]
```

display_option!do: This parameter specifies which statistics are to be displayed. The following options are allowed:

job_data - display job related data.

Omission will cause job_data to be used.

display_format!df: This parameter specifies whether a display of the all statistics recorded so far (total) or only those statistics recorded since the last display_job_data command (incremental) should be displayed. Omission will cause incremental to be used.

output!o: This parameter specifies the file to which the job data will be displayed. Omission will cause $OUTPUT to be used.

status: See ERROR HANDLING.

2.15.5 DISPLAY_COMMAND_INFORMATION ! DISCI

The purpose of this command is to display current information about a NOS/VE command. The parameter names, abbreviations, allowed values and known problems for a command, as supported in the current system, can be determined. This is a nonstandard command and will be replaced by the help utility sometime in the future.

```
display_command_information command_name=<name>!all
        [utility_name=create_object_library!
```

```
                     col!source_code_utility!scu!system]
         [display_option=parameter_description_table!
                      pdt!notes!names!help]
         [output=<file reference>]
         [status=<status variable>]
```

command_name!cn: This parameter specifies the name of the command about which information is to be displayed.

utility_name!un: This parameter specifies which utility the command belongs to. Omission will cause SYSTEM to be used.

display_option!do: This parameter specifies the type of display being requested. The options are:

parameter_description_table!pdt - selects a display of the parameter description table used by the command when executed.

notes - selects a display of any known problems with the command.

names - selects a display of the command names for a utility.

help - selects a display of the command interface description of the command.

Omission will cause PDT to be used.

output!o: This parameter specifies the file to which information will be displayed. Omission will cause $OUTPUT to be used.

status: See ERROR HANDLING.

2.15.6 CONVERT_OBJECT_FILE ! CONOF

The purpose of this command is to get a NOS/VE object file produced on NOS and to convert it to an object file suitable for processing by the NOS/VE loader or object code maintenance commands.

```
convert_object_file to=<file reference>
```

```
                    [from=<name>]
                    [user=<name>]
                    [status=<status variable>]
```

to:t: This parameter specifies the NOS/VE file name on which the converted object file is to be written.

from:f: This parameter specifies the name of the NOS file to be converted. This is the permanent file name as defined in the NOS file system and can be up to seven characters in length.

Omission will cause the permanent file name of the TO parameter to be used.

user:u: This parameter specifies the NOS user identification of the owner of the file. This parameter is only neccessary if the file is in a catalog other than the user who was specified by the most recently issued SET_LINK_ATTRIBUTES command.

status: See ERROR HANDLING.

2.15.7 GET_OBJECT_FILE : GETOF

The purpose of this command is to get a previously converted NOS/VE object file from the NOS side and sets the appropriate file attributes that will allow the object file to be used by NOS/VE.

```
get_object_file to=<file reference>
                [from=<name>]
                [user=<name>]
                [status=<status variable>]
```

to:t: This parameter specifies the NOS/VE file name of the object file.

from:f: This parameter specifies the NOS file name of the object file. This is the permanent file name as defined in NOS and can be up to seven characters in length.

Omission will cause the permanent file name of the TO parameter to be used.

user:u: This parameter specifies the NOS user identification of the owner of the file. This parameter is only necessary if the file is in a catalog other than the user who was specified by the most recently issued SET_LINK_ATTRIBUTES command.

status: See ERROR HANDLING.

2.15.8 GET_OBJECT_LIBRARY ! GETOL

The purpose of this command is to get a previously created NOS/VE object library from the NOS side and set the appropriate file attributes that will allow the object library to be used on NOS/VE.

```
get_object_library to=<file reference>
                   [from=<name>]
                   [user=<name>]
                   [status=<status variable>]
```

to!t: This parameter specifies the NOS/VE file name of the object library.

from!f: This parameter specifies the NOS file name of the object file. This is the permanent file name as defined in NOS and can be up to seven characters in length.

Omission will cause the permanent file name of the TO parameter to be used.

user!u: This parameter specifies the NOS user identification of the owner of the file. This parameter is only necessary if the file is in a catalog other than the user who was specified on the most recently issued SET_LINK_ATTRIBUTES command.

status: See ERROR HANDLING.

2.15.9 DISPLAY_OBJECT_TEXT ! DISOT

The purpose of this command is to produce a formatted display of the object text contained in an object file or object library produced on NOS/VE.

```
display_object_text file=<file>
```

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

file!f: This parameter specifies the object file or object library containing the object text to be listed.

output!o: This parameter specifies the file to which the display is to be written.

Omission will cause the file $OUTPUT to be used.

status: See ERROR HANDLING.

2.15.10 GET_SOURCE_LIBRARY : GETSL

The purpose of this command is to get a previously created SCU source library from the NOS side and set the appropriate file attributes that will allow the source library to be used on NOS/VE.

```
get_source_library to=<file reference>
                   [from=<name>]
                   [user=<name>]
                   [status=<status variable>]
```

to!t: This parameter specifies the NOS/VE file name of the source library.

from!f: This parameter specifies the NOS file name of the source library. This is the permanent file name as defined in NOS and can be up to seven characters in length.

Omission will cause the permanent file name of the TO parameter to be used.

user!u: This parameter specifies the NOS user identification of the owner of the file. This parameter is only necessary if the file is in a catalog other than the user who was specified on the most recently issued SET_LINK_ATTRIBUTES command.

status: See ERROR HANDLING.

2.15.11 EDIT_FILE ! EDIF

The purpose of EDIT_FILE is to initiate the execution of the SCU editor on a text file. (For details see ARH3883.)

edit_file ! edif - edit lines on a source file. (procedure file not necessarily in its final form)

parameters	defaults
file=file(source)	$REQUIRED
[result=file(source)]	$VALUE(FILE)
[input=file reference]	$COMMAND
[output=file reference]	$OUTPUT
[status]	--

2.15.12 JEDIT

The purpose of this command is to initiate execution of the JEDIT editor built by Jack Bohnhoff. Anyone wanting information about the editor should contact Jack.

```
jedit from=<file>
      [status=<status variable>]
```

from!f:	This parameter specifies the file to be edited. This file is rewritten after the editor terminates. !
status:	See ERROR HANDLING in the NOS/VE Command Interface.

2.15.13 DEBUG

The prototype R1 NOS/VE debugger is now available. Details on how to use the debugger can be found in the "CYBER 180 INTERACTIVE DEBUG External Reference Specification and User's Guide", Sunnyvale DCS number S4028.

2.15.14 SET_LINK_ATTRIBUTES ! SETLA

The SET_LINK_ATTRIBUTES command is the same as documented in the NOS/VE command interface with the exception that the CHARGE and PROJECT parameters are optional (and in fact not useful in the current environment since we disable that feature on the NOS side).

3.0 PROGRAM_INTERFACE_STATUS

The 'status' column indicates whether the procedure is unchanged from the previous build, modified from the previous build or not available in this build. Footnotes are numbered within each section.

3.1 COMMAND_PROCESSING

Procedure	Status
CLP$SCAN_PARAM_LIST	unchanged
CLP$TEST_PARAMETER	unchanged
CLP$GET_KEYWORD	unchanged
CLP$GET_SET_COUNT	unchanged
CLP$GET_VALUE_COUNT	unchanged
CLP$TEST_RANGE	unchanged
CLP$GET_VALUE	unchanged
CLP$CREATE_VARIABLE	unchanged
CLP$DELETE_VARIABLE	unchanged
CLP$READ_VARIABLE	unchanged
CLP$WRITE_VARIABLE	unchanged
CLP$SCAN_COMMAND_FILE	unchanged
CLP$END_SCAN_COMMAND_FILE	unchanged
CLP$SCAN_COMMAND_LINE	unchanged
CLP$CREATE_FILE_CONNECTION	unchanged
CLP$DELETE_FILE_CONNECTION	unchanged
CLP$PUSH/POP_UTILITY	unchanged
CLP$GET_COMMAND_ORIGIN	unchanged
CLP$GET_DATA_LINE	unchanged
CLP$SCAN_PROC_DECLARATION	unchanged

3.2 MESSAGE_GENERATOR

Procedure	Status
OSP$FORMAT_MESSAGE	unchanged
OSP$SET_STATUS_ABNORMAL	unchanged
OSP$APPEND_STATUS_PARAMETER	unchanged
OSP$APPEND_STATUS_INTEGER	unchanged

3.3 RESOURCE_MANAGEMENT

Procedure	Status
RMP$REQUEST_MASS_STORAGE	unchanged
RMP$REQUEST_TERMINAL	unchanged

All terminal attributes can be specified on the RMP$REQUEST_TERMINAL call but only the following are operational:

- o auto_input
- o transparent_mode
- o prompt_file
- o prompt_string

Files assigned to a terminal device can be accessed via the following BAM requests:

- o AMP$OPEN
- o AMP$GET_NEXT
- o AMP$GET_DIRECT
- o AMP$GET_PARTIAL
- o AMP$PUT_NEXT
- o AMP$PUT_DIRECT
- o AMP$PUT_PARTIAL
- o AMP$CLOSE
- o AMP$REWIND
- o AMP$SKIP
- o AMP$SEEK_DIRECT

3.4 PROGRAM_EXECUTION

Procedure	Status
PMP$EXIT	unchanged
PMP$EXECUTE	unchanged
PMP$TERMINATE	unchanged
PMP$AWAIT_TASK_TERMINATION	unchanged
PMP$MODULE_TABLE_ADDRESS	unchanged
PMP$ENTRY_POINT_TABLE_ADDRESS	unchanged
PMP$PUSH_TASK_DEBUG_MODE	unchanged
PMP$SET_TASK_DEBUG_MODE	unchanged
PMP$TASK_DEBUG_MODE_ON	unchanged
PMP$SET_DEBUG_RING	unchanged
PMP$DEBUG_RING	unchanged
PMP$CHANGE_DEBUG_LIBRARY_LIST	unchanged
PMP$POP_TASK_DEBUG_MODE	unchanged

3.5 PROGRAM_COMMUNICATION

Procedure	Status
OSP$AWAIT_ACTIVITY_COMPLETION	unchanged
PMP$DEFINE_QUEUE	unchanged
PMP$REMOVE_QUEUE	unchanged
PMP$CONNECT_QUEUE	unchanged
PMP$DISCONNECT_QUEUE	unchanged
PMP$SEND_TO_QUEUE	unchanged
PMP$RECEIVE_FROM_QUEUE	unchanged
PMP$STATUS_QUEUE	unchanged
PMP$STATUS_QUEUES_DEFINED	unchanged
PMP$GET_QUEUE_LIMITS	unchanged

3.6 CONDITION_PROCESSING

Procedure	Status
PMP$ESTABLISH_CONDITION_HANDLER	Added support of detected uncorrected error
PMP$DISESTABLISH_COND_HANDLER	unchanged
PMP$CAUSE_CONDITION	unchanged
PMP$CONTINUE_TO_CAUSE	unchanged
PMP$TEST_CONDITION_HANDLER	unchanged
PMP$VALIDATE_PREVIOUS_SAVE_AREA	unchanged
PMP$ESTABLISH_DEBUG_OFF	unchanged
OSP$SET_STATUS_FROM_CONDITION	unchanged

3.7 PROGRAM_SERVICES

Procedure	Status
PMP$GENERATE_UNIQUE_NAME	unchanged
PMP$GET_TIME	unchanged
PMP$GET_MICROSECOND_CLOCK	unchanged
PMP$GET_TASK_CP_TIME	unchanged
PMP$GET_DATE	unchanged
PMP$GET_USER_IDENTIFICATON	unchanged
PMP$GET_ACCOUNT_PROJECT	unchanged
PMP$GET_JOB_NAMES	unchanged
PMP$GET_JOB_ID	unchanged
PMP$GET_JOB_MODE	unchanged
PMP$GET_PROGRAM	unchanged
PMP$GET_TASK_ID	unchanged
PMP$MANAGE_SENSE_SWITCHES	unchanged
PMP$GET_OS_VERSION	unchanged

PMP$GET_PROCESSOR_ATTRIBUTES	unchanged
PMP$DEFINE_DEBUG_ENTRY	unchanged
PMP$GET_DEBUG_ENTRY	unchanged
PMP$MODIFY_DEBUG_ENTRY	unchanged
PMP$REMOVE_DEBUG_ENTRY	unchanged

3.8 LOGGING

Procedure	Status
PMP$LOG	unchanged
PMP$LOG_ASCII	unchanged

3.9 FILE_MANAGEMENT

Procedure	Status
Sequential Access	unchanged
Byte_Addressable Access	unchanged
Record Access	unchanged
Segment Access	unchanged - *1
V_System Specified	unchanged
V_User Specified	unchanged
U_System Specified	unchanged
U_User Specified	unchanged
F_System Specified	unchanged
F_User Specified	unchanged
AMP$DESCRIBE_NEW_FILE	deleted
AMP$FILE	unchanged
AMP$GET_FILE_ATTRIBUTES	unchanged
AMP$FETCH	unchanged
AMP$STORE	unchanged
AMP$COPY_FILE	unchanged
AMP$RENAME	unchanged
AMP$RETURN_FILE	new name
AMP$OPEN	unchanged
AMP$CLOSE	unchanged
AMP$FETCH_ACCESS_INFORMATION	unchanged
AMP$SKIP	unchanged
AMP$REWIND	*2
AMP$WRITE_END_PARTITION	unchanged
AMP$GET_NEXT	unchanged
AMP$GET_DIRECT	unchanged
AMP$GET_PARTIAL	unchanged
AMP$PUT_NEXT	unchanged
AMP$PUT_DIRECT	unchanged
AMP$PUT_PARTIAL	unchanged - *3

AMP$SEEK_DIRECT	unchanged
AMP$GET_SEGMENT_POINTER	unchanged
AMP$SET_SEGMENT_EOI	unchanged
AMP$SET_SEGMENT_POSITION	unchanged
AMP$SET_LOCAL_NAME_ABNORMAL	unchanged
AMP$SET_FILE_INSTANCE_ABNORMAL	unchanged
AMP$ACCESS_METHOD	unchanged
AMP$FETCH_FAP_POINTER	unchanged
AMP$STORE_FAP_POINTER	unchanged

*1 Segment access If a segment access file is written and an AMP$SET_SEGMENT_EOI is not issued to record the EOI, EOI remains zero. The highest page referenced is not yet used as the default EOI. This particularly affects those who wish to make heaps permanent because EOI is always zero for a heap.

*2 AMP$REWIND The WAIT parameter on the procedure call is not supported.

*3 AMP$PUT_PARTIAL PUT_PARTIAL with the TERM_OPTION = AMC$TERMINATE does <u>not</u> act as a put_next if a preceding START was not issued.

3.10 <u>PERMANENT_FILE_MANAGEMENT</u>

<u>Procedure</u>	<u>Status</u>
PFP$DEFINE	unchanged
PFP$ATTACH	unchanged
PFP$PURGE	unchanged
PFP$CHANGE	unchanged
PFP$PERMIT	unchanged
PFP$DELETE_PERMIT	unchanged
PFP$DEFINE_CATALOG	unchanged
PFP$PURGE_CATALOG	unchanged
PFP$PERMIT_CATALOG	unchanged
PFP$DELETE_CATALOG_PERMIT	unchanged

3.11 <u>MEMORY_MANAGEMENT</u>

MMP$ADVISE_IN	unchanged
MMP$ADVISE_OUT	unchanged
MMP$ADVISE_OUT_IN	unchanged
MMP$WRITE_MODIFIED_PAGES	unchanged
MMP$CREATE_SEGMENT	unchanged
MMP$DELETE_SEGMENT	unchanged
MMP$STORE_SEGMENT_ATTRIBUTES	unchanged
MMP$FETCH_SEGMENT_ATTRIBUTES	unchanged

MMP$VERIFY_ACCESS	unchanged
MMP$FREE	unchanged
MMP$LOCK_PAGES	number of locked pages per
MMP$UNLOCK_PAGES	segment restricted to 32
MMP$FETCH_PVA_UNWRITTEN_PAGES	unchanged

3.12 STATISTICS_FACILITY

SFP$ESTABLISH_STATISTIC	unchanged
SFP$ENABLE_STATISTIC	unchanged
SFP$DISABLE_STATISTIC	unchanged
SFP$DISESTABLISH_STATISTIC	unchanged
SFP$EMIT_STATISTIC	unchanged
SFP$EMIT_SYSTEM_STATISTIC	unchanged

3.13 INTERACTIVE_FACILITY

IFP$TERMINAL	unchanged
IFP$FETCH_TERMINAL	unchanged
IFP$STORE_TERMINAL	unchanged
IFP$GET_DEFLT_TERMINAL_ATTRIBUTES	unchanged
IFP$GET_TERMINAL_ATTRIBUTES	unchanged
IFP$ADVANCE	new - *1

*1 Only the option IFC$ADVANCE_ALL_QUEUED_OUTPUT is supported.

3.14 NOS/VE_EXCEPTIONS

The following summarizes the exception code ranges currently assigned to NOS/VE. These code ranges represent a finer breakdown than the one specified in the SIS for internal NOS/VE development purposes. However, it is important to remember that only the product identifiers documented in the SIS may appear in error messages.

Common Modules	9,000 - 9,999
Common Code Generator	8,000 - 8,999

3.0 PROGRAM INTERFACE STATUS
3.14 NOS/VE EXCEPTIONS

Exception Code	Product Identifier	Product Name	
1 - 158,999	Reserved		
159,000 - 159,999	SY	System Core	
160,000 - 169,999	AM	Basic Access Methods	
160,000 - 163,999	BA	Basic Access	
164,000 - 164,999	LN	Local Name Mgr	
165,000 - 165,999	JF	Job File Mgr	
166,000 - 166,999	SR	Conversion Services	
170,000 - 179,999	CL	Command Language	
180,000 - 189,999	JM	Job Management	
190,000 - 199,999	LL	Loader	
200,000 - 209,999	MM	Memory Management	
200,000 - 204,999	MM	Monitor Level	
205,000 - 205,999	MM	Task Level	
210,000 - 219,999	OS	Operating System	
210,000 - 210,999	OS	OS	
211,000 - 211,999	MT	EXEC	
212,000 - 212,999	IO	MS I/O	
213,000 - 213,999	IO	Tape I/O	
214,000 - 214,999	DM	Device Management	
215,000 - 215,999	ML	Memory Link	
216,000 - 216,999	IF	Interactive	
217,000 - 217,999	TM	TM Monitor	
218,000 - 218,999	TM	TM Task	
219,000 - 219,999	JS	Job Swappers	
220,000 - 229,999	PF	Permanent File Management	
221,000 - 221,999	ST	Set Management	
222,000 - 222,999	PU	Permanent File Utilities	
230,000 - 239,999	PM	Program Management	
240,000 - 249,999	RM	Resource Management	
250,000 - 259,999	OF	Operator Facility	
260,000 - 269,999	AV	User Administrator	
270,000 - 279,999	IC	Interstate Communication	
280,000 - 289,999	RH	Remote Host Facility	
290,000 - 299,999	OC	Object Code Utilities	
300,000 - 309,999	DS	Deadstart/Recovery	!
310,000 - 319,999	MS	Maintenance Services	
320,000 - 329,999	IF	Interactive Facility	!
330,000 - 339,999	US	User Errors	
340,000 - 349,999	SF	Statistics Fac.	
350,000 - 359,999	CM	Configuration Management	!
360,000 - 369,999	HU	Help Utilities	!
370,000 - 379,999	NA	Network Access Method	!
500,000 - 509,999	AA	Advanced Access Method	
510,000 - 519,999	AG	ALGOL	
520,000 - 529,999	AL	Assembly Language	

3.0 PROGRAM INTERFACE STATUS
3.14 NOS/VE EXCEPTIONS

530,000 - 539,999	AP	APL
540,000 - 549,999	BA	BASIC
550,000 - 559,999	CA	Conversion Aids System
560,000 - 569,999	CB	COBOL
570,000 - 579,999	CY	CYBIL
580,000 - 589,999	FT	FORTRAN
590,000 - 599,999	PA	PASCAL (Wirth)
600,000 - 609,999	P1	PL/1
610,000 - 619,999	SM	Sort Merge
620,000 - 629,999	SC	Source Code Utility
640,000 - 649,999	DB	Debug

4.0 DUAL_STATE_DEADSTART_AND_OPERATION

4.1 CURRENT_DUAL_STATE_CONFIGURATION

The Arden Hills S2 development lab contains 12 FMD units. |

o FMD Unit 43 (Ch. 1, 21, 22) |

This unit contains NOS permanent files. |

o FMD Unit 41 (Ch. 1, 21, 22) |

This unit contains the following:

- Files required to deadstart dual state Cycle 5; A170 NOS (5.3 plus changes necessary for Cycle 5), CTI, MSL, EI binaries, and NOS Deadstart files.
- It is also used as a temp device.

o FMD Unit 42 (Ch. 1, 21, 22) |

This unit contains NOS permanent files. |

o FMD Unit 44 (Ch. 1) |

This is another NOS PF device.

o FMD Unit 45 (Ch. 1) |

This unit contains files required to deadstart dual state Cycle 7; NOS 5E55 deadstart file, CTI143PR, MSL143PR, and EI. It is also a NOS temp device. |

The following devices are available for hands-on use: channel 1, 21, 22: unit 40; channel 2, 22, 31: units 44 and 45; channel 2: units 40, 41, 42, and 43. |

4.2 USER_NAMES_AND_PERMANENT_FILES

1) The convention used for creating user names on NOS/VE is as follows:
   - o Your user name will be your initials.
   - o Your password will be these 3 letters followed by the letter 'x'.
   - o You must see COMSOURCE (R.K. Cooper - x3092) to be assigned a user index

2) PF dumping and loading

   You may use "SES.DUMPPF" on SN/101 to dump your permanent files to tape, and then load them onto your user name on A170 NOS using "SES.LOADPF". Documentation on how to use these SES procedures and what their parameters are is included in the SES User's Guide, or they can be obtained by typing:

   SES,HELP.DUMPPF and SES,HELP.LOADPF.

4.3 CONVERTING_VERSION_1_VALIDATION_FILES_TO_VERSION_2

The basic reference for this section is the NOS V2 System Maintenance Manual (60459300), Section 5. The best specific reference is Example 13 on page 5-37 in that manual.

Using NOS 5.3, the analyst should do the following:

```
X.DIS.
SUI,377777.
PURGE(SOURCE/NA)
DEFINE(SOURCE)                                          !
MODVAL(OP=S,S=SOURCE,FA)                                !
DROP.
```

Next, the file SOURCE should be XEDITed with the following directives:

```
                                                        !
D/AB...=/*                                              !
D/NF...=/*                                              !
D/OF...=/*                                              !
C/STANDARD/NORMAL/*
C/CMLI/CNVE/*
```

Finally, after bringing up a NOS V2 system, do the following: (Note: An idle family situation must exist when the ISF commands are given.) |

```
X.DIS.
SUI,377777.
PURGE,NEWDUZ,NEWNDZ.
DEFINE,NEWDUZ,NEWNDZ.
ATTACH,SOURCE.                                                  |
MODVAL(OP=C,I=SOURCE,SI,N=NEWDUZ,U=NEWNDZ)                      |
RETURN(NEWDUZ,NEWNDZ)                                           |
ISF,R=NEWDUZ.    (will release NEWDUZ if it is fast-attach)     |
ISF,E=NEWDUZ.
DROP.
```

4.4 TO_RELOAD_CONTROLWARE_FOR_THE_NOS/VE_DISK_DRIVER

At deadstart time NOS will automatically load 7155-1x disk controlware on one channel with controller type=FM (LBC CMRdeck entry), and will automatically load 7155-4x disk controlware on any channel with controller type=HT (LBC CMRdeck entry). NOS/VE supports both of those types of controllers. NOTE: It is not possible to use 844 half-track controlware in this environment.

4.5 A170_NOS_DEADSTART

4.5.1 CTI AND CHECKING CENTRAL MEMORY

Deadstarting A170 NOS assumes some knowledge of CTI. CTI stands for Common Test and Initialization. It is software that places an 800 series machine in a state such that it is possible to deadstart an operating system. CTI is used somewhat ambiguously in the software community to imply CTI and MSL (Maintenance Software Library). The MSL is a collection of programs and data that includes such things as a subset of CMSE (Cyber Maintenance Software Executive) that enables one to load controlware to controllers, look at CYBER 180 maintenance registers, look at microcode, etc. The MSL also contains microcode that can be loaded by CTI. The MSL is actually an operating system that runs independently of NOS. An important element of CTI/MSL is HIVS (Hardware Verification Sequence), which is a program that loads microcode, clears and checks central memory and tests all 170 opcodes. If you are not sure what the machine was used for (particularly the first hands on user each morning) then the HIVS program should be run. This is accomplished by:

1) deadstart to NOS/VE (unit 40 for S3, unit 45 for S2) |

2) Enter O (operator intervention)

3) Enter P (deadstart panel,make sure level 0 deadstart)

4) <BKSP>

5) Enter H (assure yourself that CS=YES to reload microcode)

6) <BKSP>

7) Enter V (verification sequence)

8) Hit <CR> at 'parameter display' to test CM & CP

When you see text that tells you that verification is complete and a deadstart is required, you are now ready to deadstart NOS.

4.5.2 NOS DEADSTART

See Section 3.3 of the Integration Procedures Notebook for important NOS CMRDECK changes.

- o Set the D/S panel to deadstart from the primary system disk. This is 844 pack PD345 for 9.1 and 844 pack PD367 for 9.2 and subsequent systems. |
- o Push D/S button
- o Enter (CR)
- o Enter date/time

Wait for deadstart to complete.

Note: The deadstart tapes DU91A and DU92A for 9.1 and 9.2 respectively are found in the area in the northeast corner of the S2 lab where the tape cabinet is found. |

4.6 NOS/VE DEADSTART AND INSTALLATION

o Enter DOWN,CH2. so NOS/VE can use the channel.

o Enter DOWN,CH32. so NOS/VE can use the channel.

o The following file must be available in your catalog on the S2:

TPXXXK contains a NOS/VE deadstart image. This must be a copy of the dual state deadstart images available from the link procedures.

o If you've never deadstarted NOS/VE from the user number from which you want to run or if you wish to change the current parameter settings for your particular user number, then do a SETVE. SETVE assumes the file TPXXXK is in your user number; you do not have to do another SETVE if TPXXXK has changed since the last time you ran. The general form of SETVE is:

```
        X.SETVE(PN=ffff,UN=un,C=6)
```

where ffff is an identifier of up to 4 characters and un is the user number to search first for files. 6 is the number of the system core command deck for the Arden Hills S2 configuration. Warning: Specifying C=6 on the Arden Hills S3 will destroy the closed shop permanent file base. In general ffff and un will be the same, e.g. X.SETVE(DAH,UN=DAH,C=6)

Only ONE SETVE should be done for each user number and a SETVE should NOT be done for ANY Integration user number except by the Integration project.

SEE SECTION 5.1 FOR MORE DETAILS.

o Bring up dual state:

NVEffff.

where ffff is the identifier specified in SETVE, e.g. NVEINT1.

o Bring up the Operator Facility

Enter K,NVE.

NOS/VE is currently generated and initialized on both NOS and NOS/VE. All source and object libraries that make up the

4.0 DUAL STATE DEADSTART AND OPERATION
4.6 NOS/VE DEADSTART AND INSTALLATION

NOS/VE system are produced on NOS and therefore must be converted from their CI to II counterparts. Other parts of installing and initializing the system (e.g. building the $SYSTEM catalog) are performed by command language procedures on NOS/VE. Since the same system will many times in a closed shop environment, it is advantageous to only perform the conversion from CI to II a single time; save the results in the NOS file system and then simply bring the files back during deadstart.

The actual files that get installed and loaded on each deadstart are determined by a command language procedure (the system profile) interpreted on NOS/VE. This procedure can be modified by each site to initialize their NOS/VE environment in the most suitable fashion. The process of building the system profile and of performing the CI to II conversions is referred to as an installation deadstart and the process of executing the system profile and of fetching previously converted files from NOS and making them available in the NOS/VE file system is referred to as a deadstart. A single command is available to perform both an installation deadstart and a deadstart.

4.6.1 THE DS PROCEDURE

The purpose of this command is to perform an installation, normal or recovery deadstart of NOS/VE. The defaults for parameters are those most convenient for "closed shop" deadstarts.

The procedure "brings up" the job log display on the left screen where the progress of the procedure may be watched, and the control point display on the right screen. Just before the procedure completes it changes the left screen to display the system log and writes to that log the message:

```
'---- Deadstart Completed ----'
```

at which point the operator may enter commands.

```
ds [kind=install ! normal ! recover]
   [get_products=<boolean>]
   [echo=<boolean>]
   [alternate_user=<NOS_user_name>]
   [save_install_files=<boolean>]
   [validate_users=<boolean>]
   [quick_validate=list of <name>]
```

[status=<status variable>]

kind ! k: This parameter specifies what kind of deadstart is to be performed. Valid specifications are:

install ! i - installation deadstart to be performed. The system libraries are built from CI object files.

normal ! n - normal deadstart to be performed. The system libraries are obtained from the results of a previous installation deadstart.

recover ! r - recovery deadstart. Just initiates system tasks. Permanent files are "recovered" from a previous run of the system.

Omission will cause a recovery deadstart to be performed.

get_products ! gp: This parameter specifies whether the object libraries defining the current product set and SCU libraries defining the source libraries are to be installed. Valid specifications are:

true ! yes ! on - the products are to be installed

false ! no ! off - the products are not to be installed

The list of products installed on the Arden Hills closed shop S3 system when get_products=yes is specified is given in the INSTALL_PRODUCTS section.

Omission will cause the product set to be installed.

echo ! e: This parameter specifies whether the commands should be echoed to the console during execution. Valid specifications are:

true ! yes ! on - echo commands

false ! no ! off - do not echo commands

Omission will cause commands not to be echoed.

alternate_user ! au: This parameter specifies what NOS user to check if the default NVE user does not have the needed file. Any NOS user name is allowed.

Omission will cause INT1 to be used.

validate_users ! vu: This parameter specifies whether to run the job that validates NOS/VE users. This parameter is ignored for a recovery deadstart. Valid specifications are:

true ! yes ! on - run the validation job

false ! no ! off - do not run the validation job

Omission will cause the validation job to be run.

quick_validate ! qv: This parameter determines which users will be validated by the validation job if it is run. When specified, this parameter gives a list of user names to be validated in addition to the users: INT1, INT2, DEV1, DEV2, REL1, EVAL and RKC.

Omission will cause all users to be validated.

status: See ERROR HANDLING in the NOS/VE ERS.
NOTES

The get_source_libraries parameter has been removed. Source libraries are now treated as products and their installation is now controlled by the get_products parameter.

The debug parameter has also been removed. Later in this document there is an explanation of how DS now handles abnormal situations.

Summary of DS Processing

Kind of Installation	Processes Performed
I, N, R	A condition handler is established for any fault. If something goes wrong during the DS procedure, it will get control, display the status, and let the operator interact with the DS procedure. The operator can enter commands to investigate and/or correct the problem, resume execution of DS by entering CONTINUE, or abort DS by entering EXIT_PROC.

I, N, R	The procedure "brings up" the job log display (on the left screen) where the progress of the DS procedure may be watched, and the control point display (on the right screen).
I only	System files are created from 170 CI binaries for CYF$RUN_TIME_LIBRARY OSF$OPERATOR_LIBRARY OSF$SYSTEM_LIBRARY OSF$OPERATOR_COMMAND_LIBRARY OSF$COMMAND_LIBRARY If save_install_files=yes, the II binaries of the above files are replaced on the 170.
N only	System files are accessed from 170, these are the replaced II binaries mentioned above.
I, N, R	Remote Host Output is started.
I, N	System and User Prologs and Epilogs are built.
I, N	The validation job is submitted and system statistics are output.
I, N	If get_products=yes, the product set and source libraries are installed; failed product set or source library installations will print a job log.
I, N	Wait for validation job to complete.
I, N, R	RHINPUT, IFEXEC, DUMP_BROKEN_JOB tasks are started.
I, N	Feedback capability set up.
I, N, R	SCL task started. Job log printed. The following message is output: -- Deadstart Completed --

The biggest change to the DS procedure is in the product set installation (get_products=yes). A procedure called INSTALL_PRODUCTS is used to install the entire product set (product set and source libraries). In this procedure, a job is submitted for each product, in each job a GET_PRODUCT is executed. The products are installed in the $SYSTEM.LIBRARY catalog under the product name. The submitted jobs can be viewed on a right K display, the left K display shows the processing of the DS procedure.

Upon job completion, a success or failure message with the job name (reflecting the product being installed, i.e. cyl$compiler for product cyf$compiler - the CYBIL II Compiler) is written to a status file (dsf$product_job_status). The contents of this file is used to make sure all the product jobs have completed before continuing with the DS procedure, as well as the contents being displayed to the job log which can be viewed by the operator.

If a product installation fails, several things happen:

- A failure message with job name is written to the status file, which will subsequently be displayed to the operator.

- The job log for the failed product installation will be printed, indicating the error.

- At the end of the INSTALL_PRODUCTS procedure, if any products have failed, the condition handler established at the start of the DS procedure gets control. A status message of "n product installations failed" is displayed and a prompt for operator interaction occurs. The operator at this point can attempt reinstallation of failed products through the use of INSTALL_PRODUCTS or just continue with the DS procedure.

The INSTALL_PRODUCTS procedure can be used as an independent command to do an installation/reinstallation at the operator's discretion.

If get_products=yes, all of the following will be installed.

C170 File	Product Name (C180_File)	File_Contents
cyfilc	CYF$COMPILER	CYBIL II Compiler
alfol	ALF$OBJECT_LIBRARY	180 Assembler
scfol	SCF$OBJECT_LIBRARY	Source Code Utility
scfcl	SCF$COMMAND_LIBRARY	Source Code Utility "stand-alone" command library
iffedit	IFF$EDITOR	Jack Bohnhoff's Editor
psfcl	PSF$COMMAND_LIBRARY	Product Set Commands
aaf44d	AAF$44D_LIBRARY	AAM Library
aaf4dd	AAF$4DD_LIBRARY	AAM-Fortran Interface
cbf7dd	CBF$7DD_LIBRARY	Cobol Compiler
cbf4dd	CBF$4DD_LIBRARY	Cobol Run-Time Library
ccflib	CCF$LIBRARY	Common Compiler Modules
cgflib	CGF$LIBRARY	Common Code Generator
dbflib	DBF$LIBRARY	Debug Library
fcflib	FCF$LIBRARY	Fortran Compiler
flflib	FLF$LIBRARY	Fortran Run-Time
fmflib	FMF$LIBRARY	File Management Utility
mlflib	MLF$LIBRARY	Math Library
smflib	SMF$LIBRARY	Sort-Merge
ttflib	TTF$TEST_TOOL_LIBRARY	Test Tool Library - ACR & Examine
osfpil	OSF$PROGRAM_INTERFACE_LIBRARY	Operating System Program Interface
osfsl	OSF$SOURCE_LIBRARY	Subset of Operating System Source Library

INSTALL_PRODUCTS_Procedure

```
install_products  [products=list of <name> ! all]
                  [status=status_variable]
```

products ! product ! p: the names of one or products to be installed.

Omission will cause all products to be installed.

status: See ERROR HANDLING in the NOS/VE Command Interface ERS

GET_PRODUCT Procedure

```
get_product  product=<name>
             file=<170_file_name>
             user=<170_user_name>
             [catalog=<180_catalog_reference>]
             [file_contents=<name>]
             [file_structure=<name>]
             [file_processor=<name>]
             [ring_attributes=list 3..3 of <integer>]
             [status=status_variable]
```

product ! p: name of the product

file ! f: name of the 170 file that contains the 856 form of the product.

user ! u: name of the 170 user that owns "file".

catalog ! c: name of the catalog in which the product is to be installed.

Omission will cause $user to be used.

file_contents ! file_cntent ! fc: the value for the product file's file_contents attribute.

Omission will cause object to be used.

file_structure ! fs: the value for the product file's file_structure attribute.

Omission will cause library to be used.

file_processor ! fp: the value for the product file's file_processor attribute.

Omission will cause unknown to be used.

ring_attributes ! ra: the values for the product file's ring_attributes attribute.

Omission will cause (11,11,11) to be used.

status: See ERROR HANDLING in the NOS/VE Command Interface ERS

If you change any of the following decks you MUST use the installation deadstart from your own catalog (with files CYBILGO, XLJOCM, XLJOSL and XLJLIB), or you must use the alternate_user parameter to specify a NOS catalog containing the files (e.g. DEV1).

```
AVMUTIL CLMDP   DMMDISA ICMCLOS ICMFAI  ICMFAPC ICMFLSH ICMGET
  ICMOPEN ICMPUT  ICMWEOP IFMEXEC IIMA72H IIMDC2S IIMRLE
IIMRSE  IIMRUM  IIMRUSM IIMTDEL OCMADD  OCMBIM  OCMBIM  OCMCOL
  OCMCOM  OCMCPY  OCMCRM  OCMDEF  OCMDEL  OCMDLB  OCMDNL
OCMDOL  OCMEND  OCMGEN  OCMLCH  OCMLMG  OCMLP   OCMMUR
OCMNP   OCMOBJ  OCMOFH  OCMOMS  OCMRCH  OCMRED  OCMREP  OCMRMB
  OCMSAT  OCMSOL  OCMVEL  OCMVLU  OCMVOL  PFMDC   PFMTALL
PUMBCAT PUMBCYC PUMBFIL PUMBFO  PUMBLST PUMBPF  PUMBSET
PUMCOMN PUMCRAK PUMPURG PUMCRAK PUMIOBF PUMLIST PUMMISC
PUMPURG PUMRALL PUMRCAT PUMREC  PUMREF  PUMRFIL PUMRPF
PUMSTUB RHMLML  RHMQAT  RHMQOP  RHMQRE  RHMSFM  USORT   UTMDUR
  UTMPC1  UTMPC2  UTMPC3  UTMPC4  UTMPC5  UTMTSA  UUSER1
```

4.6.2 EXAMPLE OF NOS/VE INSTALLATION DEADSTART

Type

```
K,NVE.
K.SETLA (your_un,NVE) your_password
K.GETF DS U=scat                                                 :
K.DS INSTALL  GP=NO AU=scat                                      :
```

4.6.3 EXAMPLE OF NOS/VE "NORMAL" DEADSTART

The Integration system has had the installation deadstart run on it. Also the files produced by the installation deadstart have been made semi-private and are found on the catalog used in the NVExxxx call.

Type (where DEV1 is the same as the xxxx in the NVExxxx call):

```
K,NVE.
K.SETLA (DEV1,NVE) DEV1X
K.GETF DS
K.DS NORMAL
```

4.6.4 EXAMPLE OF NOS/VE RECOVERY DEADSTART

This is the kind of deadstart that should most frequently be done in a "closed shop" environment and consequently is the one for which all the parameter defaults are oriented. It presupposes that permanent file recovery has been successful.

Type (where DEV1 is the same as the xxxx in the NVExxxx call):

```
K,NVE.
K.SETLA (DEV1,NVE) DEV1X
K.GETF DS
K.DS
```

4.6.5 EXAMPLE OF MINIMAL NOS/VE DEADSTART

The minimal deadstart shown below may be useful to OS developers who need to get the system up quickly and do not need the product set or all validated users.

Type

```
K,NVE.
K.SETLA (your_un,NVE) your_password
K.GETF DS U=scat
K.DS NORMAL  GP=NO QV=your_un AU=scat
```

4.6.6 USE OF THE QUICK_DEADSTART COMMAND

This command is intended as a development tool to facilitate 'fast' deadstarts where recovery is not needed; indeed, if this command is entered <u>recovery will not be performed</u> when the system is brought down for whatever reason. Specifying this command will cause an installation deadstart to take place. If the INITDD command is not specified then a default value of 'VSN001' is used for the system deadstart device. Use of INITDD will allow setting the deadstart devices identifier to any value. THIS COMMAND WILL NOT BE ACCEPTED FROM A DEADSTART COMMAND FILE, SO 'D=T' MUST BE SPECIFIED ON THE SETVE PROCEDURE TO PERMIT ENTRY OF THE QUICKDS COMMAND.

Format: QUICKDS or QUICK_DEADSTART

Values: The default is false. Executing this command causes this initial value to be toggled, thus executing this command twice will cause the final value to be false.

Note: QUICK_TEMPLATE_LOAD does not exist now.

4.7 NOS/VE_INTERACTIVE_FACILITY_OPERATION

4.7.1 OPERATOR INITIATION

To bring up the NOS/VE interactive facility do the following:

1) Bring up NOS/VE.

2) Bring up NAM

At the system console enter:

```
For 9.1/5F:                  For 9.2/5G:
FCN,5,7700.                  ENABLE,NAM,2.
NAMV2.                       NAM.
```

With the 5G networks, a flashing request for a dump tape will appear, enabling the operator to dump a previous network crash *after* bringing NAM & friends up again. If you don't want a dump enter ASSIGN,jsn,77. for the appropriate job.

3) If IAF is not up at control point 1, enter:

```
For 9.1/5F:                  For 9.2/5G:
IAFV2.                       IAF.
```

4.7.2 OPERATOR TERMINATION

4.7.2.1 Termination_of_1_Series_Networks_-_9.1/5F

To terminate NOS/VE interactive any of the following may be done:

- CFO,NAM.DI,AP=VEIAF.

  This is the preferred method. To bring NOS/VE interactive back up, you must first do a CFO,NAM.EN,AP=VEIAF.

- CFO,NAM.DI,NE.

This terminates the entire network including IAF, RBF, etc.

4.7.2.2 Termination of 4 Series Networks - 9.2/5G

To terminate NOS/VE interactive either of the following may be done.

- Preferred method:

```
K,NAM.
K.AP=NVF.
K.DI,AP=VEIAF.
```

  To bring VEIAF back up, you must do:

```
K,NAM.
K.AP=NVF.
K.EN,AP=VEIAF.

X.DIS.
IIPPAS.
DROP.
```

- Second method:

```
IDLE,NAM.
IDLE,IAF.    (Don't idle IAF when IAFEX2 is executing
              or a PP will hang.)
```

This terminates the entire network including IAF, RBF, etc. and should be used only if you have no intention of bringing VEIAF back up because the 170 and 180 sides of VEIAF get out of synch when the network is brought down this way.

4.7.3 OTHER OPERATOR CAPABILITIES

- To logically turn the printer on, under DSD enter:

```
ON,33.
FORM,33,TM,.
```

- To send a "shutdown warning" to all terminals logged on to VEIAF do:

```
For 9.1/5F:                    For 9.2/5G:

CFO,NAM.ID,AP=VEIAF.           K,NAM.
                               K.AP=NVF.
                               K.ID,AP=VEIAF.
```

- To send a message to all terminals do:

```
For 9.1/5F:                     For 9.2/5G:
CFO,NAM.MSG,ALL,message.        K,NAM.
                                K.AP=CS.
                                K.NPU=npuname,message.
                                  npuname=SN1147 on the S2,
                                         =SN1322 on the S3.
```

- PASSON has the ability to record various types of diagnostic information. This capability is controlled via the sense switches at the PASSON control point. To turn a sense switch on (off) at job jsn do:

  ```
  ONSW,jsn,x.  (OFFSW,jsn,x.)
  ```

  where x is the desired sense switch (1 to 6). The PASSON default is all sense switches off. It will take a short period of time before PASSON detects a change in a sense switch and reacts to it. The sense switches currently used by PASSON are:

switch_#	use
1	Network Trace
2	PASSON Logic Trace To Dayfile
3	Memory Link Trace To Dayfile

4.8 NOS/VE_OPERATOR_FACILITY_AND_OPERATOR_COMMANDS

With the release of the Operator Facility Phase 1 several changes to the NOS/VE Operating System will occur that will effect the users of the NOS/VE Operators Console. The Operator Facility runs as part of the NVE control point. When the request for K display appears on the NOS B Display, assign the K Display to the NVE control point. The Operator Facility is capable of displaying both a left and a right screen area at the same time. If the operator wants both screens then type in KK. The contents of these displays are determined by the commands entered by the operator.

The left screen is divided into four different areas. The top most area is the system header which contains the current date and time, memory statistics, and an operator action message if one is posted. The operator action will include the job sequence number of the owner of the message and 'message cancelled' if the message is cancelled because the task has terminated. The next line contains the first 64 characters of the operator action message (60 characters in stand alone).

The next area of the screen is the main output area. This area has the file name of OUTPUT. Any display command can have its output directed to this area as well as any system command.

The third area is towards the bottom of the screen. This area is two lines long and contains the response area. This will contain error messages from system commands. The area is cleared when the next operator typein is entered at the operator's console and received by NOS/VE.

The fourth area is the prompt area. This will contain the status of the keyboard. If NOS/VE is processing a command, then the keyboard is locked and all typeins will be ignored. When the keyboard is locked, the message 'data received by 180 - keyboard locked' will appear at the screen's bottom. When the keyboard is unlocked then any data in the keyboard buffer will be sent to NOS/VE. The bottom line is the last line that was processed by Operator Facility.

The right K Display has the file name OUTPUT_RIGHT. There is only one area on the right screen therefore the main display area is 10 lines longer than the left screen area. If a dayfile display or CP display is shown on the right screen you will get more lines of information than on the left screen.

There are no default displays that come up automatically on either output display area. It is up to the operator to decide the display the operator wishes to see. The only parts of the display that come up automatically is the system header display and the prompt area for keyboard status.

The page width of the screen is 60 characters for standalone and 64 characters for dual state. The character set translation code is the same as that for the current NVE Subsystem control point. The escape code sequence for the special characters to be typed has not changed. There are a few differences in the processing of data by the Operator Facility and NVE Subsystem.

1) Do NOT end commands with a period. Periods are sent to NOS/VE.

2) The NVE subsystem commands that begin with an asterisk will not be supported from the Operator Facility control point. If these commands are entered from the Operator facility they will be passed on to SCL where an illegal command will be issued.

3) Routing of console job data to a specific job by the 'n=command' protocol will not be supported in dual state. This feature should work in standalone but will not be supported.

4) No type ahead - commands cannot be entered until the prompt area shows that they are requested.

There is one new command to replace the current display commands. The entry points for Zdis, Zdisb, and Sdis have been deleted. The new command is VEDISPLAY and has two parameters. The options are listed below. The values in paranthesis are the abbreviations. Note that this command does not begin with an asterisk (*). This command will be processed by SCL and create a new system control point task to display it's data. The user can have the same display type on each of the display areas, if the user so desires.

Command Name	Display Type	Screen Area
	Parameter Name	Parameter Name
	DISPLAY_OPTIONS DISPLAY_OPTION (DO)	OUTPUT (O)

	Parameter Values	Parameter Values
VEDISPLAY (VED)	DISPLAY_SYSTEM_LOG JOB_LOG (JL) CONTROL_POINT (CP)	OUTPUT OUTPUT_RIGHT (OR)

The default file name for all displays is OUTPUT.

The following is a brief list of commands to bring up NOSVE with the Operator Facility installed.

```
NVEffff.              to bring up NOS/VE.   (See Section 5)

K,NVE.

KK.                   to bring up the K display on both  screens.

K.VED JL              to bring up the job log.
or
K.VED DISPLAY_OPTIONS=JL OUTPUT=OUTPUT
                      to bring up the job log  using  key  word
                                                identifiers.

K.VED CP OUTPUT_RIGHT
                      to bring up the control  point  display  on
                                                the right screen.
or
K.VED DISPLAY_OPTIONS=CONTROL_POINT OUTPUT=OUTPUT_RIGHT
                      to bring up the control point display using
                                                key          word
                                                identifiers.

K.xxx                 send any command to NOS/VE.
.
.
.

K.TERMINATE_SYSTEM                              to terminate NOS/VE.
```

Note: After the DS procedure has completed execution the command to enter to bring down the system is TERMINATE_SYSTEM, not TERMINATE_SYSTEM_JOB. Yes, this is the OPPOSITE of what it used to be.

4.8.1 DELETE_JOB_QUEUE ! DELETE_JOB_QUEUES ! DELJQ

The purpose of this command is to delete all files from the job input subcatalog, the print subcatalog or both. This command is only allowed from jobs with operator and or system privileges.

```
delete_job_queue [queue_name=input!output!all]
                [status=<status variable>]
```

queue_name ! qn: This parameter specifies from which subcatalogs files are to be deleted. Specifying INPUT will cause all files to be deleted from the job swap subcatalog and the job input subcatalog. Specifying OUTPUT will cause all files to be deleted from the job output subcatalog. Omission will cause all to be used.

status: See ERROR HANDLING.

4.8.2 REBUILD_INPUT_QUEUE ! REBIQ

The purpose of this command is to rebuild an entry in the Known Job List (KJL) from information in the System Label of the file representing the job being processed. This command is to be used during the process of recovering the input queues during recovery deadstart.

```
rebuild_input_queue [name=<name>]
                   [status=<status variable>]
```

name ! n: This parameter specifies the file name of the file representing the job. An attempt is made to process the specified file within the catalog where job input queues are known to reside.

status: See ERROR HANDLING.

4.8.3 REBUILD_OUTPUT_QUEUE ! REBOQ

The purpose of this command is to rebuild an entry in the Known Output List (KOL) from information retained in the System Label of the file representing the output being processed. This command is to be used during the process of recovering the output queues during a recovery deadstart.

```
rebuild_output_queue [name=<name>]
                     [status=<status variable>]
```

name: This parameter specifies the file name of the file representing the output. An attempt is made to process the specified file within the catalog path of where job output queues are known to reside.

status: See ERROR HANDLING.

4.9 ROUTE_AN_INPUT_FILE_FROM_C170_TO_C180

Through the system console, enter:

Type

```
X.DIS.
USER,A,B.
GET,filename.
where filename identifies the input file to be routed.
ROUTE,filename,DC=LP,FC=RH.
```

4.10 CONFIGURATION_MANAGEMENT

In order to run NOS/VE in a dual state environment, certain hardware configuration rules must be in effect. All storage devices and channels that NOS/VE will request must be in a 'down' state in the NOS EST. For more information on the NOS EST, see the NOS Version 2 Operator/Analyst Handbook, 60459310, Section 4. Refer to the E display.

To run S2 S/N 104 in the standard configuration, channels 2 and 32 must be downed as shown below.

```
DOWN,CH2.
DOWN,CH32.
```

4.10.1 SYSTEM CORE COMMANDS

Configuration Management (CM) system core commands define the system mass storage device. The commands are described in detail in the NOS/VE Installation Command Interface ERS, Section 3. The CM system core commands define the hardware configuration of the system device and the controller accessing the system device. An example for S2 S/N 104 is shown here.

```
SETDCT $7155_1
SETDD $885_11 40(8)
```

4.10.2 JOB TEMPLATE COMMANDS

During an installation deadstart, the Physical Configuration Utility (PCU) and the Logical Configuration Utility (LCU) must be run. The PCU is run to describe the physical configuration that NOS/VE may access. The LCU will logically install all or part of the physical configuration and allow NOS/VE to access the equipment. Commands for both utilities are described in the NOS/VE Installation Command Interface ERS, Section 2.

The system device which was defined by the system core commands must the defined via the PCU, using the same controller and channel number. Additional devices may be defined as long as the device is in a down state, and all channels defined are also down. An example is listed here.

```
EXECUTE_TASK SP=MANAGE_PHYSICAL_CONFIGURATION
   SET_MAINFRAME_DEFINITION MAINFRAME=S2
   SET_DATA_CHANNEL_DEFINITION CHANNEL_NUMBER=2
   SET_DATA_CHANNEL_DEFINITION CHANNEL_NUMBER=18
   SET_CONTROLLER_DEFINITION ELEMENT=CT1 ..
      PRODUCT_IDENTIFICATION=$7155_1 ..
      SERIAL_NUMBER=2 CHANNEL_CONNECTION=CHANNEL2
   SET_CONTROLLER_DEFINITION ELEMENT=CT2 ..
      PRODUCT_IDENTIFICATION=$7155_1 ..
      SERIAL_NUMBER=3 CHANNEL_CONNECTION=CHANNEL18
   SET_STORAGE_DEVICE_DEFINITION ELEMENT=FMD40_A ..
      PRODUCT_IDENTIFICATION=$885_11 SERIAL_NUMBER=1234 ..
      UNIT_NUMBER=40(8) CONTROLLER_CONNECTION=CT1
   SET_STORAGE_DEVICE_DEFINITION ELEMENT=FMD40_B ..
      PRODUCT_IDENTIFICATION=$855_11 SERIAL_NUMBER=3456 ..
```

```
         UNIT_NUMBER=40(8) CONTROLLER_CONNECTION=CT2
      INSTALL_PHYSICAL_CONFIGURATION MAINFRAME=S2
   QUIT

   EXECUTE_TASK SP=MANAGE_LOGICAL_CONFIGURATION
      INSTALL_LOGICAL_CONFIGURATION INCLUDE=ALL
   QUIT
```

4.10.3 MULTIPLE VOLUME CONSIDERATIONS

If a configuration is used defining multiple mass storage volumes, the execution of the PCU and LCU will only permit access to the system device. In order to bring additional volumes online, the LCU must be run again. The user may want to initialize a fresh volume. To do this, use the LCU subcommand INITMV. To add a volume to the NOS/VE set, i.e. to allow NOS/VE use of the volume, use the LCU subcommand ADDMTS. An example is shown here.

```
EXECUTE_TASK TP=MANAGE_LOGICAL_CONFIGUATION
   INITIALIZE_MS_VOLUME ELEMENT=FMD40_B RECORDED_VSN='VSN002'
   ADD_MEMBER_TO_SET MEMBER_VSN='VSN002'
QUIT
```

During a noninstallation deadstart, an attempt will be made to bring all logically configured volumes online and reactivate them. This means the above sequence need only occur when a new volume is being brought online the first time.

Reference has been made to the NOS/VE Installation Command Interface ERS. To get a copy of this document on S/N 101, enter SES,MAD.LISTNIH.

4.11 K_DISPLAY_ASCII

Support of 6-12 ASCII from the console (K display) causes the following changes:

INPUT	TRANSLATED_TO	INPUT	TRANSLATED_TO
/1		/(	[
/2	"	/)	]

/3	#	/+	>
/4	$	/-	<
/5	(reversed /)	/=	.
/6	;	/*	' (single quote)
/7	?	//	/
/8	{	/,	:
/9	}	/A to /Z	a - z (lower case)
/0	_ (underscore)		

4.12 EDD AND DSDI INFORMATION

Most of the steps listed here are used to gather information for recovery problems, etc. If you do not wish to do this, execute steps D and E only (EDD dump and NOS level 3 deadstart).

A. Enter DR I at the MDD display.

B. Enter HP at the MDD display.

C. If IOU.FS1 NOT = 0 OR IOU.FS2 NOT = 0 THEN do NOT enter DU at the MDD display ELSE do enter DU at the MDD display.

D. Press deadstart button and perform EDD:

1) Mount scratch tape (ring in) on a 9-track drive.

2) Push D/S button.

3) Select U (utilities) display.

4) Select E (EDD) display.

5) Set channel (S2=13).

6) Set ECUU (S2=01uu)

E = equipment

C = 1 for 67X drives, 2 for 66X drives

uu = unit number of the tape drive to be used.

7) Answer "dump number" with a CR.

8) Answer "non zero inhibits rewind" with a CR.

9) Answer "channel controlware" with a CR. Warning: if this step is omitted, DSDI canot process the dump tape.

E. Perform a level 3 NOS deadstart (See Section 4.14). |

F. If you are running on the S3, enter the following at the console:

```
X.DIS.
USER,INT1,INT1X.
GET,DMPDFS/UN=JLG.
BEGIN,,DMPDFS.
•
•
DROP.
```

This is a procedure to dump several dayfiles associated with the NVExxxx job and are needed along with the EDD tape to adequately debug a NOS/VE hang.

Some of the crashes that NOS/VE encounters will cause NOS/VE to terminate such that the operator will notice the K-display which requests that a K.*RUN. be entered. Before a K.*RUN. is entered the EDD should be performed.

To create a listing of the EDD tape, type SES,INT1.DSDI See the procedure's HELP documentation for parameter descriptions.

C170 DSDI information can be found in Chapter 10 of the NOS SYSTEM MAINTENANCE Manual.

A170 DSDI info can be found in document ARH3060 -- GID for A170 NOS/S2.

4.13 NOS/VE_TERMINATION

o Bringing down dual state:

K,NVE.

K.TERMINATE_SYSTEM

o If not a normal termination

```
K,NVE.

K.*RUN.
K.*ENDLST.
K.*ENDRUN.
```

4.14 A170_NOS_SHUTDOWN

Before leaving the machine, it is necessary to bring NOS down. If NOS has crashed, a level 3 deadstart must be attempted even if the only reason is to bring NOS down. To do a level 3 deadstart:

1) Push D/S button

2) Select "O" display

3) Select "P" display

4) Enter I=3

5) Enter (CR)

6) Enter date/time

If a dump is desired but a crash has not occurred STEP. should always be entered before pushing the deadstart button. After the dump has been taken a level 3 deadstart should be performed.

To bring NOS down, do the following:

1) Enter:

   CHE
   The screen will display:
   CHECK POINT SYSTEM.
   Enter: carriage return

2) Make sure no mass storage device has a checkpoint rquested. To do this, enter: E,M. If the display shows there are no "C"s in the status field, then all devices are checkpointed and you may continue.

3) Enter:

STEP.

4) Push deadstart button.

5.0 RECOVERY_OF_NOS/VE_PERMANENT_FILES

5.1 SETVE_FORMAT

The general format of the SETVE command is

```
X.SETVE(PN=ffff,UN=un,VSN=vsn,D=d,P=p,B=b,C=c,CH=ch)
```

ffff is a string of no more than four characters. SETVE appends ffff to 'NVE' to construct the name of a procedure file which, when invoked, will deadstart NOS/VE. The default is TST.

un specifies the user number from which TPXXXK is attached. Un is the first catalog searched for other files used in deadstarting and terminating NOS/VE. The default is INT1.

c specifies the deadstart command deck to be used when deadstarting NOS/VE. The function served by the deadstart command deck is analogous to the function served by the CMRDECK of NOS. Currently supported values for c and their respective uses are:

1	Arden Hills S3 S/N 02 open shop/hands-on time
3	Arden Hills S3 S/N 02 closed shop
6	Arden Hills S2 S/N 104 open shop/hands-on time
10	Sunnyvale S2 closed shop installation
40	Sunnyvale S2 closed shop continuation

The default is set in the file CMDS1. Currently the default is 6. Warning:_Specifying__C=6__on__the Arden__Hills__S3__will__destroy_the_NOS/VE_permanent file_base._

b specifies an alternate catalog to be searched for the various files used in deadstarting and terminating NOS/VE. The default is INT1.

d is used to indicate that the system core command processor should accept commands from the console. Specifying D=T in the SETVE command allows the operator to enter commands from the console after processing the deadstart command deck. If the operator wishes to initialize the system device and/or install a new version of NOS/VE, D=T must be specified. The default is set in the file CMDS1. Currently the default is D=F.

p	specifies the password for the catalog indicated by the un parameter. If this parameter is omitted the password will be generated by appending an 'X' to the UN parameter.
ch	specifies the octal channel to be used for NOS/VE disk I/O. The default is set in the file CMDS1. Currently the default is 2.
vsn	specifies the vsn of a deadstart tape. If this parameter is used then NOS/VE will be deadstarted from the tape specified. If it is omitted, then NOS/VE will be deadstarted from the permanent file TPXXXK.

5.2 SETVE_USAGE

Earlier versions of NOS/VE required that two SETVE commands be issued if the system was to be installed and subsequently recovered. The current system does not require this. The only reason for issuing two SETVE commands is to provide a deadstart procedure that does not require/permit operator intervention.

It should be noted that once the SETVE command has been issued, it need not be issued again unless...

1) There is a need to change one or more of the parameters specified in SETVE.

2) The command file, NVEffff, generated by the SETVE command is purged from the system.

Two examples of SETVE usage and subsequent NOS/VE deadstart are given below. The first example shows a "hands on" user working with recovery. The second illustrates these concepts for a typical NOS/VE closed shop. This writer hopes that the reader will find both examples useful and illuminating.

I. A "hands on" user

The command file NVERSD is built and installed in NOS by typing

```
X.SETVE(RSD,UN=RSD,VSN=TAPE,B=DEV1,C=6,D=T)
```

Several of the choices of parameters are worth noting.

1) By specifying the VSN parameter the user has built a command file that will deadstart the system directly from the tape produced by NVESYS.

2) The user in this example has specified deadstart command deck 6 (C parameter) and has allowed the NOS/VE disk I/O channel (ch parameter) to default to 2. One concludes that the user is running on the Arden Hills S2.

3) The user has specified D=T. This is important. The deadstart command which triggers installation of a recoverable system cannot be read from a deadstart command deck. It must be entered from the console at deadstart time. Specifying D=T allows the operator to enter commands from the console.

NOS/VE is deadstarted by typing

```
NVERSD.
```

at the console.

The user brings up the K display by typing

```
K,NVE.
```

Presently, the deadstart command deck is displayed and the user is prompted for input. The deadstart command deck used in this example looks like this:

```
USECP    S2CFIG
USEIP    EMPTY
SETDCT   $7155_1
SETDD    $885_12 32
```

The user types

```
K.INITDD VSN001.
```

```
K.GO.
```

The system accepts the commands and installs and deadstarts NOS/VE.

After the system comes down, via either controlled termination or a crash, the system can be recovered (if necessary) and redeadstarted by typing

```
NVERSD.
```

When the deadstart command deck is displayed, the user types

```
K.GO.
```

This will cause NOS/VE to be deadstarted without initializing the system device.

II. A typical "closed shop"

Two command files, NVECLSH and NVEINST, are created by typing

```
X.SETVE(CLSH,UN=CLSH,B=DEV1,C=40,CH=1)
X.SETVE(INST,UN=CLSH,B=DEV1,C=10,CH=1,D=T)
```

One notes that

1) Closed shop is deadstarted from a TPXXXK file in the CLSH catalog.

2) Specifying D=T, for NVEINST, causes deadstart to pause for operator intervention.

3) Using DCF deck 40 for continuation suppresses redundant (and possibly damaging) reexecution of the configuration prolog.

A normal deadstart is used when bringing up NOS/VE at the beginning of closed shop or following a system failure. The operator, in this example, types

```
NVECLSH.
```

NOS/VE will recover (if necessary) and deadstart. When recovery runs, the operator must respond to a request for input from the MANLC utility. See note 4 at the end of this section for more information.

If it is necessary to reinitialize the system device, or if the installation is upgrading to a new version of NOS/VE, the operator keys

```
NVEINST.
```

This causes deadstart to pause and wait for operator input when deadstart commands are being processed. There are two cases requiring discussion here. The first is the case of upgrading to a new and compatible version of NOS/VE. The second case is used only when it is necessary to reinitialize the system device.

The first case involves the installation of a new version of NOS/VE. In order to install a new version of the system, the old system must have been idled in an orderly way. A new system cannot be installed if, following a crash, the system being superceded was not recovered. Assuming everything in the old system is tidy, and the file systems are compatible, the operator keys

```
K.USECP EMPTY.
K.SETSA INSTALL_JOB_TEMPLATES 1.
K.GO.
```

when the system displays the deadstart command deck and prompts for input. The new system is installed, the file system is preserved, and deadstart proceeds.

The second case amounts to an installation deadstart. This should be used only with full knowledge that any files which may have existed on mass storage prior to this deadstart, will be blasted into oblivion by it.

An installation deadstart will be required if

1) This is the initial installation of a recoverable version of NOS/VE.

2) If the file systems of the system being installed and the system being superceded are not compatible.

3) If the file system has been damaged beyond the possibility of recovery.

An installation deadstart is effected by typing

```
K.INITDD VSN001.
K.GO.
```

when the deadstart command deck is displayed and the operator is prompted for input.

NOTES:

1. The CMDS1 file used to deadstart NOS/VE must have the DEBUG2 flag set to TRUE. When NOS/VE is deadstarted, the catalog specified by the UN parameter is first in the search order for CMDS1, followed by the catalog specified by the B parameter.

2. See Section 3.3 of the Integration Procedures Notebook for other information about the CMDS1 file.

3. If NOS/VE crashes and a dump is desired (in the context of our second example)

   i. Type du at the MDD console. The message "WRITING IMAGE FILE" should appear immediately. The message "IMAGE FILE COMPLETE" should appear a few moments later.

   ii. Push the deadstart button.

   iii. Take the EDD dump.

   iv. Do a level 3 NOS deadstart.

   Alternatively the operator can skip step i if she/he is sure to redeadstart NOS/VE after step iv. In this event the system will detect that the image file was never created, will create one, and will recover from it.

4. A recovery deadstart which is under the control of a SETVE command in which D=T was specified will pause in the recovery system with the message "OPERATOR INTERVENTION BEFORE RECOVERY BEGINS". The system is executing the logical configuration utility at this point. In order to exit this utility and allow recovery to proceed the operator types K.QUIT (no period). For more details on the use of the LCU see Section 4.9 of this document

and the NOS/VE Installation Command Interface ERS.

6.0 SYSTEM_CORE_DEBUGGER

The System Core debugger provides a set of capabilities intended to assist in debugging the operating system. Services provided by the debugger are task oriented: selection of the tasks to be debugged must be made via debugger subcommands. No tasks will be under control of the debugger unless they are selected. The selection capability allows any number of tasks to be debugged simultaneously; from one task to all tasks in the system. Obviously a capability this powerful must be used with some care. The System Core debugger uses the debug hardware to provide these capabilities.

6.1 SYSDEBUG

The purpose of this command is to initiate execution of the system core debugger. This command can be issued from the deadstart command file or as a command in any job.

```
sysdebug
```

This command has no parameters; all information the debugger requires is provided via subcommands.

The system core debugger can also be invoked from the MDD console. The format of the command is:

```
DO n.sysdebug
```

where n is the job ordinal of the desired job. The debugger is brought up in the job monitor task of the job. All system core debugger subcommands are available, but must be prefixed by the MDD command DO.

The system core debugger can also be brought up (from the MDD console) by specifying a global task id. The format of the command is:

```
DO n.tdebug gggggg
```

The value of n is ignored, and the value gggggg specifies the

6.0 SYSTEM CORE DEBUGGER
6.1 SYSDEBUG

NOS/VE global task id (3 hex bytes) of the task to bring the debugger up in. If the task id is invalid, then the command will be ignored.

6.2 SUBCOMMAND_PARAMETER_DEFINITIONS

```
<name> ::= 1-8 character breakpoint name
<condition> ::= READ!WRITE!RNI!BRANCH!CALL!DIVFLT!ARLOS!
                AROVFL!EXOVFL!EXUNFL!FPLOS!FPINDEF!INVBDP
<base> ::= process virtual address
<offset> ::= integer
<length> ::= integer
<frame> ::= 1..100
<count> ::= 1..10000
<regid> ::= X!A!P
<regno> ::= 0..15!0..0F(16)
<value> ::= integer
<time> ::= 1..(2**31)-1
<vstring> ::= 'charstring'
<datatype> ::= HEX!ASCII!ASC!DEC
<change_count> ::= 1..8
<selector> ::= FULL!AUTO!SAVE
```

6.3 SYSTEM_CORE_DEBUGGER_SUBCOMMANDS

Within the descriptions which follow, optional parameters are enclosed in brackets. Default values for optional parameters are also defined.

6.3.1 SELECT

The purpose of this subcommand is to select the tasks in which the system core debugger is to be active. When the debugger is first called, it is not active in any task. To use the debugger therefore, it is necessary to select the tasks in which it is to beactive.

```
select <selection option> [<ring number> ! <active job list
      ordinal>]
```

selection_option: This parameter specifies one of a series of selection options used to control the tasks in which the debugger will be active and some other debug options. The selections are remain in effect until they are explicitly changed with subsequent SELECT subcommands. Valid selection options are:

<right!left> - This selects the screen for the debug display. The display stays active when the screen is switched.

<jobmonitor!nojobmonitor> - This selects whether or not to debug job monitor tasks.

<user!nouser> - This selects whether or not to debug user tasks (i.e. those that are not job monitors).

<highring> - This specifies the highest ring in which debug traps will be recognized. Traps occurring in rings above this selection will be ignored.

<job!nojob> - This enables or disables debugging for the jot at the specified active job list ordinal. The system job has an active job list ordinal of zero.

<alljobs!nojobs> - This activates or deactivates debugging in all jobs.

The initial selections are: RIGHT, NOSTEP, NOJOBMONITOR, NOUSER, HIGHRING=3, NOJOBS.

6.3.2 BREAKPOINT ! B

The purpose of this subcommand is to select a program interrupt which is to take place upon occurrence of a specified condition within a specified virtual address range.

```
breakpoint <name> <condition> [<base>] [<offset>] [<length>]
```

The <name> is any user supplied name for identifying the breakpoint. A maximum of thirty two breakpoints can be selected. When a trap occurs, the <name> of the breakpoint which caused the trap is displayed.

The base parameter is required when specifying a new breakpoint name; offset and length specifications are optional in this case. When adding a new condition selection to an existing breakpoint, base, offset, and length parameters may not be specified.

Base, offset, and length parameters define the desired virtual address range: <base> + <offset> yields a first-byte-address; first-byte-address + <length> -1 yields a last byte address.

Default parameter values:

```
<offset>: 0
<length>: 1
```

6.3.3 REMOVE_BREAKPOINT : RB

The purpose of this subcommand is to deselect a previously selected program inte

```
remove_breakpoint <name> [<condition>]
```

If only the name parameter is specified, all conditions associated with the breakpoint are deselected and all evidence of the breakpoint is removed. If the condition parameter is specified, only that condition is deselected; however, if the specified condition is the only condition selected, all evidence of the named breakpoint is removed.

6.3.4 LIST_BREAKPOINT : LB

The purpose of this subcommand is to provide a list of currently selected breakp and associated conditions.

```
list_breakpoint [<name>]
```

If the name parameter is specified, information is displayed for the named breakpoint only. If the name parameter is not specified, information is displayed for all currently defined breakpoints.

6.3.5 CHANGE_BREAKPOINT ; CB

The purpose of this subcommand is to change the virtual address range of a previ specified breakpoint.

```
change_breakpoint <name> <base> [<offset>] [<length>]
```

Base, offset, and length parameters define the desired virtual address range: <base> + <offset> yields a first-byte-address; first-byte-address + <length> -1 yields a last byte address.

Default parameter values:

```
    <offset>: 0
    <length>: 1
```

6.3.6 TRACE_BACK ; TB

The purpose of this subcommand is to provide information relevant to stack frame associated with an interrupted procedure and its predecessor procedures. Validation of PVA's is now performed.

Information displayed for each selected stack frame consists of:

- Stack frame number;
- Current P-address of the associated procedure;
- Virtual address of the start of the stack frame;
- Virtual address of the stack frame save area.

```
trace_back [<frame>] [<count>] [FULL;SHORT]
```

The frame parameter specifies the number of the first stack frame for which information is to be displayed. Stack frame number one is associated with the interrupted procedure, stack frame two is associated with the interrupted procedure's predecessor, etc.

The module name provided on the traceback is usually correct but not guaranteed.

The count parameter specifies the total number of stack frames for which information is to be displayed.

Default parameter values:

```
<frame>: 1
<count>: 1
```

6.3.7 DISPLAY_STACK_FRAME ! DSF

The purpose of this subcommand is to display selected information from a specifi stack frame.

```
display_stack_frame [<frame>] [<selector>]
```

The frame parameter specifies the number of the stack frame for which information is to be displayed. (Stack frame number one is associated with the interrupted procedure, stack frame two is associated with the interrupted procedure's predecessor, etc.)

The selector parameter identifies a region of the specified stack frame:

AUTO: Causes the automatic region of the stack frame to be displayed.

SAVE: Causes the save area of the stack frame to be displayed.

FULL: Causes both the automatic and save areas of the stack frame to be displayed.

Default parameter values:

```
<frame>: 1
<selector>: FULL
```

6.3.8 DISPLAY_REGISTER ! DR

The purpose of this subcommand is to display the contents of a specified registe interrupted procedure.

```
display_register <regid> [<regno>] [<datatype>]
```

Default parameter values:

```
<regno>: 0
```

<datatype>: HEX

6.3.9 DISPLAY_MEMORY ! DM

The purpose of this subcommand is to display the contents of a specified area of virtual memory. Validation of PVA's is now performed.

```
display_memory <base> [<length>]
```

Default parameter values:

<length>: 8

6.3.10 CHANGE_MEMORY ! CM

The purpose of this subcommand is to set a specified value into a specified loca of virtual memory for a specified number of bytes. Validation of PVA's is now performed.

```
change_memory <base> <value> <change_count>
```

Default parameter values:

<change_count>: 1

6.3.11 RUN

The purpose of this subcommand is to invoke program execution after a selected p interrupt has occurred.

```
run
```

6.3.12 SUPER_CHANGE_MEMORY ! SCM

The purpose of this subcommand is the same as the change memory subcommand, that is, to change the contents of virtual memory. It differs from change memory, however, in that it will change the attributes of the segment to allow memory to be written, and then change the attributes back to their original values.

The command format is the same as the change memory subcommand.

6.3.13 FORMAT ! FMT

The purpose of this subcommand is to set the system core debugger into a mode where all subcommand output is sent to a permanent file. This is done by having the task running the debugger communicate with another task running in the system job. It is this other task that actually creates and writes the permanent file. The entry point of this task is OSP$BROKEN_JOB_DUMP_TASK. It will normally be initiated by the DS procedure. If it is not running, a diagnostic will be issued. This task will create successive cycles of the permanent file 'DUMP' in the $SYSTEM catalog. These files contain ASCII text data written in BAM variable records. The parameter to this command is a string which will be output as the first line of the file.

```
format string
```

6.3.14 UNFORMAT ! UNFMT

The purpose of this subcommand is to leave the output mode established by the FORMAT command. Output will again be sent to the operator console. At this point the permanent file will be flushed to mass storage.

```
unformat
```

6.3.15 DISPLAY_MONITOR_FAULT ! DISMF

The purpose of this subcommand is to display any monitor faults present in this task. See the section titled 'NOS/VE Processing of Job Mode Software Errors' for more information.

All monitor fault buffers are displayed in the hope they will show some task history. If a given fault buffer is invalid the message "following fault is not present" is displayed.

```
display_monitor_fault
```

6.3.16 DISPLAY_XCB : DISXCB

The purpose of this subcommand is to display all of the fields of the current task's (i.e., the task running the debugger) execution control block.

6.3.17 DISPLAY_TASK_ENVIRONMENT : DISTE

The purpose of this subcommand is to display the XCB of all tasks running within the current job (i.e., the job with the task running the debugger). If the command is entered while the debugger is in format mode, then a full XCB is displayed, otherwise just the task name, XCB address and global task id are displayed.

7.0 NOS/VE PROCESSING OF JOB MODE SOFTWARE ERRORS

7.1 INTRODUCTION

Tasks running in job mode will occasionally cause an error which is detected either by the hardware or NOS/VE monitor. The action taken when an error like this occurs is controlled by various system attributes. The purpose of this section is to discuss the types of errors and the effect a given system attribute will have upon the handling of the error.

7.2 TYPES OF ERRORS

1) BROKEN TASK: A broken task is a task in which the trap mechanism is not able to function correctly. NOS/VE monitor will attempt to repair the trap mechanism and send a broken task fault to the task. The specific cases of a broken task are:

system error	job mode software has declared the task to be broken. (This is a special case of broken task.)
monitor fault buffer full	job mode errors are occuring but are not being processed by job mode.
traps disabled	a job mode error has occurred while traps were disabled.
invalid A0	the task's A0 register was invalid.
UCR/MCR traps disabled	UCR/MCR error occurred with traps disabled.

2) MCR FAULT: This error signifies that job mode caused a hardware detected MCR fault. This may be caused by

software or hardware detected uncorrectable error.

3) UNKNOWN SYSTEM REQUEST: This error signifies that job mode issued a monitor request that is either invalid or cannot be issued from the ring it was issued from.

4) SEGMENT ACCESS FAULTS: These errors signify that job mode encountered or caused one of the following errors:

- page fault for an address greater than EOI on a read-only file (segment)

- disk read error

These errors either originate in NOS/VE monitor or cause the hardware to exchange to monitor. Depending on the values of certain system attributes, monitor will halt or reflect the error back to job mode as a monitor default.

It is at this point that the system core debugger can be activated. (See the definition of SYSTEM_DEBUG_RING in the next section.)

The normal job mode OS actions for these faults are:

broken task	exit
MCR fault	cause condition
invalid system request	exit
segment access	cause condition

7.3 SYSTEM ATTRIBUTES FOR ERROR PROCESSING

The following system attributes can be set or displayed by the SETSA and DISSA commands.

7.3.1 HALTRING

If a broken task or MCR fault occurs at or below the value of HALTRING (P register ring number), NOS/VE monitor will halt the system. Broken tasks occurring above HALTRING will cause a monitor fault to be sent back to job mode.

7.3.2 SYSTEM_ERROR_HANG_COUNT

This is the number of broken task errors allowed to occur in any given task before that task is considered a hung task.

7.3.3 HALT_ON_HUNG_TASK

If this attribute is true, then an occurence of a hung task will cause NOS/VE monitor to halt the system. If the attribute is false, the task will be sent a signal to 'hang' itself, i.e. to go into an infinite wait doing nothing. Jobs with hung tasks will have a *H in the status field on the operator CP display.

A hung task will also occur if any error happens in job mode ring 1.

7.3.4 SYSTEM_DEBUG_RING

If an error (broken task, MCR fault, unknown system request, or segment access fault) occurs at or below the value of this attribute (P register ring number), the system core debugger will be invoked within the task. At that point in time the task environment can be examined by using system core debugger commands.

If the RUN command is issued to the debugger, the system will take its normal action for the specific fault.

7.3.5 DUMP_WHEN_DEBUG

When the system core debugger is invoked by a fault at or below SYSTEM_DEBUG_RING and the DUMP_WHEN_DEBUG attribute is true, the debugger will automatically create a dump of the task (see system core debugger command FORMAT). When the dump is complete, normal fault action will take place. The following system core debugger commands are executed during an automatic dump:

```
FORMAT automatic dump data
TB 1 1000
DISMF
```

```
DISXCB
DM 00300000000 100C0(16)
DM 00400000000 100C0(16)
DM 00500000000 100C0(16)
DM 00600000000 100C0(16)
DM 00F00000000 100C0(16)
DM 01000000000 100C0(16)
DM 01100000000 100C0(16)
DISTE
UNFORMAT
```

8.0 STAND_ALONE_DEADSTART

A standalone deadstart tape is a 9-track, unlabeled, I format, 1600 BPI density tape (produced by the NVESYS procedure). To deadstart in standalone mode, perform the following steps:

1) Set the deadstart panel for standard disk deadstart.

2) Perform an alternate deadstart to the NVE deadstart tape using CTI. The CTI parameters on the 'P' display should be set as follows

```
 L = 0
 C = <'C' parameter of the SETVE procedure>
 D = <'D' parameter of the SETVE procedure
       D = Y when display is true
       D = N when no display>
 W14 = <'CH' parameter of the SETVE procedure>
```

3) The tape should move and eventually the message "PROCEED" will appear on the lower left on the console.

4) Enter CR,Mx,BR=0000000000000000 (Editor's note: 16 zeros!)

```
where Mx is M1 for S1
            M2 for S2
            M3 for S3
```

This clears the memory bounds register which is set by th deadstart.

5) Enter DR,P2 This displays the CPU registers. The SIT should be changing; if not, give up.

6) At this point the system core is loaded and can be patched. The commands to display and change memory are documented in IPNDOC.

7) To start the system enter SS The CPU registers should spin; if they ever stop the CPU has halted.

8) Enter DD to display the system dayfile on the right screen.

8.0 STAND ALONE DEADSTART

9) Enter NVE deadstart commands when the system requests them. ('D = Y' must be specified on the 'P' display for this to happen.)

   Note: Standalone deadstart is always a 'quick' deadstart, so no recovery of permanent files is possible. 'QUICKDS' mode is set internally and need not be specified.

10) Enter DJ to display the system job dayfile when the system requests it. The 'DJ' display is on the left screen.

11) Now the system is operational and commands can be entered.

9.0 INTERACTIVE_PROJECT_DUMP_ANALYSIS_PROCEDURES

The following procedures were developed by the interactive project to assist them in interpreting dumps. They guarantee the procedures work if your user name is IFP; otherwise caveat emptor. For more information about these procedures, contact Fred Bischke.

The following dump analysis procedures are available in the IFP catalog:

9.1

EDDSIM

This is a CCL procedure which brings an EDD dump tape on a specified

VSN into the simulator. The procedure can be accessed from the IFP

catalog as follows:

```
 get,eddsim/un=ifp

 begin,,eddsim,vsn ( vsn is the vsn of the EDD dump tape )
```

9.2

ANALEXC

This is a Simulator INCLUDE file which does a preliminary analysis of

the current simulator exchange package (when the system crashes in task services, this will normally be JPS). A qr exc=mon or qr exc=rma can be

used to get into another exchange package before doing the include.

The include file is ANALEXC/UN=IFP. It can be called from the simulator

as follows:

```
'get,analexc/un=ifp' ; include analexc
```

(carriage return) a lone carriage return must be entered after an INCLUDE
in order to start it up

9.3

SEGDUMP

This is a CCL procedure which calls DSDIV to dump a specified segment to
a list file which can then be examined with an editor or printed.

The procedure can be accessed from within the Simulator as follows:

```
'get,segdump/un=ifp' ; 'begin,,segdump,seg,len,file,exc,cpf' <*
```

The segdump parameters are:

seg - segment number in hex (default is 1)

length - number of bytes to dump in hex (default is 10000)

list - name of the list file (default is LIST)

exc - reference exchange package (default is JPS)

cpf - name of checkpoint file (default is CPF)

In most cases of task services debugging, only the seg parameter is

needed.

9.4

ANALJOB

This is a CCL procedure which uses DSDIV, XEDIT and the Simulator to

perform an analysis of all tasks in a specified job. The procedure can

be accessed from within the simulator as follows:

```
'get,analjob/un=lfp' ; 'begin,,analjob,seg,cpf'
```

The analjob parameters are:

seg - the monitor segment which contains the exchange packages of the job (14 is the system job, 15 is job 1 etc.) (default is 14)

cpf - the name of the simulator file (default is CPF)

After the procedure has completed, a list of the RMA's of the

job's exchange packages can be obtained by doing the following:

```
include tplist
```

(carriage return)

A traceback of all tasks in the job can be obtained by doing the following:

```
include tblist

(carriage return)

include tbrun

(carriage return)
```

NOS/VE BACKGROUND DOCUMENTS

1.0 Hardware Overview

1.1 An introduction to CYBER 180

1.2 C180 Instant

1.3 Model Independent General Design Specification - ARH1700

2.0 NOS Reference Manuals

2.1 XEDIT V3.0 - 60455730

2.2 IAF V1.0 User's Guide - 60455260

2.3 NOS Version 2 Reference Set - Vol 3, 60459680 - Vol 4, 60459690

2.4 NOS Systems Programmer's Instant - 60459370

2.5 NOS Version 2 Operator/Analyst Handbook - 60459310

2.6 NOS Version 2 Diagnostic Index - 60459390

2.7 NOS A170 ERS

2.8 NOS A170 GID - ARH3060

3.0 NOS/VE Reference Documents

3.1 Program Interface ERS - ARH3610 - obtained from Karen Rubey (482-3966) or via SES.TOOLDOC

3.2 Command Interface ERS - ARH3609 - obtained from Karen Rubey (482-3966) or via SES.TOOLDOC

3.3 NOS/VE Procedures and Conventions - SESD010 - obtained by SES.TOOLDOC

3.4 Listing of all NOS/VE Modules - obtained by SES,DEV1.LISTNVE. See Integration Procedures Notebook for details.

3.5 NOS/VE Internal Interface Maintenance Procedures Memo available from S.C. Wood.

3.6 Integration Procedures Notebook Obtained by: Acquire,IPNDCC/UN=DEV1. SES.PRINT,IPNDOC.

4.0 Tools Reference Documents

4.1 CYBIL Interactive Debugger - ARH3142

4.2 SES User's Guide - ARH1833

4.3 CYBIL Specification - ARH2298

4.4 C180 Assembler ERS - ARH1693

4.5 Simulator ERS - ARH1729

4.6 VEGEN ERS - ARH2591

4.7 VELINK ERS - ARH2816

4.8 Simulated I/C ERS - ARH3125

4.9 Object Code Utilities ERS - ARH2922

4.10 CYBIL Implementation Dependent Handbook - ARH3078

4.11 CYBER 180 INTERACTIVE DEBUG External Reference Specification and Users Guide - S4028

4.12 CYBER 180 II Assembler ERS - ARH3945

4.13 ERS for Source Code Utility - ARH3883

D1.0 KEYPOINTS

Keypoints are used to give an execution time trace of program flow by showing that a given function is being performed (that is, that a given procedure is being executed) . Keypoints may also be used to display request parameters, status and error conditions.

D1.1 ISSUING_KEYPOINTS_FROM_CYBIL_CODE

The general form of the keypoint instruction is:

```
  #INLINE ('keypoint', keypoint_class, osk$m * data, keypoint_id);
```

D1.1.1 KEYPOINT CLASSES

A keypoint is identified by both class, and identifier.
The following deck explains the partitioning of the keypoint classes.

```
OSDKEYS
COMMON

  CONST

{  Keypoint Classes :
{
{       The 16 keypoint classes supported by the hardware are
{ partitioned between the System, Product Set and User as follows.

    osk$system_class = 0  {  0 ..  5 },
    osk$product_set_class = 6 {  6 .. 10 },
    osk$user_class = 11   { 11 .. 14 },
    osk$pmf_control = 15;
```

D1.0 KEYPOINTS
D1.1.1 KEYPOINT CLASSES

```
{  Keypoint Multiplier:
{
{    By convention,
{ the 32 bit keypoint code supported by the hardware
{ is split into two fields.  The right field contains a keypoint
{ identifier which is used to identify a function within a
{ keypoint class.   For example, if a particular keypoint class
{ represents exit from a procedure,
{ then the keypoint identifier might identify exit from
{ procedure A versus exit from procedure B.
{    The left field is used as a data parameter appropriate to the
{ function identified by the keypoint identifier.  In the
{ procedure exit example above,
{ the data parameter field might be used to indicate the
{ status of the procedure call.
{    The keypoint multiplier is used to partition the keypoint
{ code into the two fields.  The data parameter should be
{ multiplied by the  keypoint multiplier to prevent it from
{ overlapping the keypoint  identifier field.

  CONST
    osk$m = 4096;
```

D1.1.2 NOS/VE KEYPOINT CLASSES

Five keypoint classes named ENTRY, EXIT, UNUSUAL, DEBUG, and DATA are defined, taking five of the available sixteen classes by the hardware.

ENTRY - Every gated procedure plus all major internal procedures (those shared across functional areas) should contain a keypoint of this class. These keypoints should be placed as close as possible to the entry to the procedure.

EXIT - Every gated procedure plus all major internal procedures (those shared accross functional areas) should contain a keypoint of this class. These keypoints should be placed as closed as possible to the exit to the procedure.

UNUSUAL - Every situation which is unexpected or quite unusual should contain a keypoint of this class. It is intended that these keypoints would be enabled at all times. The frequency of encountering these keypoints SHOULD BE very low. The DATA keypoint class is not allowed in conjunction with a keypoint of class unusual.

D1.0 KEYPOINTS
D1.1.2 NOS/VE KEYPOINT CLASSES

DEBUG - These keypoints are for providing additional trace information as an assist in debugging hardware or software problems. DEBUG class keypoints would be most useful in the more complex areas of the system.

DATA - This keypoint class can be used with ENTRY, EXIT, and DEBUG keypoints for the gathering of extra data. All DATA keypoints encountered are supplying additional data which will be associated with the last ENTRY, EXIT, or DEBUG keypoint.
DATA keypoints should be used with care since the PMF hardware can only buffer up 16 keypoints, keypoint cluster can cause lost keypoints.

The following deck defines the NOS/VE OS class constants.

```
OSDKEYC
COMMON
{Define KEYPOINT CLASS Codes.

  CONST
    osk$data = osk$system_class + 0, { OS - DATA keypoint}
    osk$unusual = osk$system_class + 1, {U OS - Unusual keypoint.}
    osk$entry = osk$system_class + 2, {E OS - Entry keypoint)}
    osk$exit = osk$system_class + 3, {X OS - Exit keypoint}
    osk$debug = osk$system_class + 4; {D OS - Debug keypoint.}

{*callc,osdkeys
```

D1.1.3 KEYPOINT DATA AND IDENTIFICATION

Upon successful execution each keypoint instrucion will provide a total of 32 bits of information. Our convention uses 12 bits of this for keypoint identification and the remaining 20 bits as user supplied data. Try to use this 20 bits to supply meaningful information (taskid, segment number, file identifier, queue length, page number, time, etc.). The keypoint identification codes are defined in the attached common deck. On DATA class keypoints the data belongs to the previous keypoint and the full 32 bits is available for additional user data.

D1.1.4 EXAMPLE ISSUING KEYPOINTS

ENTRY keypoint with data:

```
  #INLINE('keypoint', osk$entry, osk$m*taskid.index,
    tmk$exit_task);
```

UNUSUAL keypoint with no data:

```
  #INLINE ('keypoint', osk$unusual, 0, mmk$no_memory);
```

ENTRY keypoint with extra data:

```
  #INLINE ('keypoint', osk$entry, osk$m * segment_number,
    mmk$page_fault);
  #INLINE ('keypoint', osk#data, offset, 0);
```

D1.1.5 KEYPOINT IDENTIFIERS

Each area of the operating system has been given a range of identifiers to use for keypoints. The base for each area is defined on common deck OSDKEYD. Each area should have a deck xxDKEY (where xx is the product identifier) where the areas keypoint constants are defined(e.g.tmk$exit_task). Please reference the section on keypoint description decks, for an example of one of these decks.

```
OSDKEYD
COMMON
{This deck defines constants for use with KEYPOINTS.


{Define base keypoint procedure identifiers for each area of the
{OS.

  CONST
    amk$base = 100, {100 - 149}
    bak$base = 200, {200 - 249}
    clk$base = 250, {250 - 299}
    cmk$base = 300, {300 - 349}
    dbk$base = 350, {350 - 399}
    dmk$base = 400, {400 - 549}
    fmk$base = 550, {550 - 599}
```

```
      ick$base = 600, {600 - 649}
      ifk$base = 650, {650 - 699}
      iik$base = 700, {700 - 749}
      ink$base = 750, {750 - 799}
      jmk$base = 800, {800 - 849}
      lgk$base = 850, {850 - 899}
      llk$base = 900, {900 - 949}
      lok$base = 950, {950 - 999}
      luk$base = 1000, {1000 - 1049}
      mlk$base = 1050, {1050 - 1099}
      mmk$monitor_base = 1100, {1100 - 1149}
      mmk$job_base = 1150, {1150 - 1199}
      msk$base = 1200, {1200 - 1249}
      mtk$base = 1250, {1250 - 1299}
      ock$base = 1300, {1300 - 1349}
      ofk$base = 1350, {1350 - 1399}
      osk$base = 1400, {1400 - 1449}
      pfk$base = 1500, {1500 - 1549}
      pmk$base = 1600, {1600 - 1699}
      rhk$base = 1750, {1750 - 1799}
      srk$base = 1800, {1800 - 1819}
      stk$base = 1850, {1850 - 1899}
      tmk$monitor_base = 1900, {1900 - 1949}
      tmk$job_base = 1950, {1950 - 1999}
      jsk$monitor_base = 2000, {2000 - 2049}
      jsk$job_base = 2050, {2050 - 2099}
      avk$base = 2100, {2100 - 2149}
      sfk$base = 2150, {2150 - 2199}
      iok$base = 2200, {2200 - 2249}
      rmk$base = 2250, {2250 - 2300}
      mtk$assembly_language_base = 4000; {4000 - 4095}
   {   OS assembly language      4000 - 4095}
   {*callc,osdkeyc
```

D1.2 <u>COLLECTING KEYPOINTS</u>

D1.2.1 ON THE SIMULATOR

When executing on the simulator all keypoint instructions cause an entry to be added to the local file SESSMKF.

D1.2.2 ON THE HARDWARE

Software keypoint collection is available for collecting system and job keypoints. System keypoints are those keypoints in the entire system and job keypoints are only those dealing with a particular job. Only one system keypoint collector can be active at one time, but each job may have an active job keypoint collector. Software keypoints are collected on a file local to the job in which the keypoint collection task is running. After keypoint collection is terminated this file can saved on the 170 side and analyzed by the keypoint analyzer.

Three commands are supplied to utilize the keypoint feature: ACTK, DEAK, and EMIK.

D1.2.2.1 ACTK_command

The ACTK command initiates keypoint recording and collecting. It has the form of:

```
ACTK,mode,environment,monitor_mask,job_mask,
   start_class,stop_class,keypoint_file,
   collector_buffer_size,collector_delay

mode or m = 'software', 's',                        parameter is
        'hardware', or 'h'                          required

environment or e = 'job','j',                       parameter is
        'system', or 's'                            required

monitor_mask or mm = a list of integers             default is all
        ranging from 0 to 15
        example:  mm=(2,6,9,12)

job_mask or jm = a list of integers                 default is all
        ranging from 0 to 15
        example:  jm=(0,3,5)

start_class or start = an integer in                default is 15
        the range from 0 to 15
        This specifies that keypoint collection
        should not start until a keypoint of this
        class is encountered.

   stop_class or stop = an integer in           default is 15
        the range from 0 to 15
        This specifies that keypoint collection
        should stop when a keypoint of this
        class is encountered.

   keypoint_file or kf = file name              default is
        This specifies the file on which        $LOCAL.KEYFILE
        keypoints are to be saved.
        This parameter is used
        only with software keypoints.

   collector_buffer_size or cbs = an integer    default is 10,000
        in the range from 1000 to 100,000
        This parameter is used only with
        software keypoints.
   collector_delay or cd = an integer           default is 50
        in the range from 10 to 100,000
        The value specified is the delay
        period in milliseconds.  This
        parameter is used only with
        software keypoints.
```

D1.2.2.2 DEAK command

The DEAK command terminates keypoint collection.

```
    DEAK,environment

   environment or e = 'job','j',                default is job
        'system', or 's'
```

D1.2.2.3 EMIK command

The EMIK command is used to issue keypoints.

```
   EMIK,class,code

   class = an integer in the range              default is 15
        from 0 to 15

   code = an integer in the range               default is 0
```

from 0 to OFFFFFFF(16)

After keypoint collection is terminated the keypoint file, can be saved on the 170 by a REPLACE_FILE with B56 conversion. For example.

```
   REPLACE_FILE,keyfile,keyfile,b56
```

On the 170 side this can be analyzed by using NVEKEY, format = HDW.

D1.3 KEYPOINT_ANALYZER_UTILITY

D1.3.1 NVEKEY

The SESSMKF file produced on the simulator, or the KEYFILE produced on the hardware can be reformatted into a readable listing by executing the following procedure.

```
SES.NVEKEY [KPF= ] [FORMAT= ] [KD= ] [AREA= ]
```

NVEKEY creates a simulator generated keypoint trace file. The "kpf" parameter is the keypoint file used as input. The "kd" parameter is a file or list of files which define(s) the keypoint descriptions.

PARAMETER	DEFAULT	ALLOWABLE VALUES
kpf	'SESSMKF'	file name
	'KEYFILE'	if format=HDW
format	'SIM'	sim,hdw
kd	'KEYDESC'	file name(s)
area	&USER&	user name

If run interactively, when the procedure terminates the reformatted listing is on local file KEYFILE.

D1.3.2 KEYPOINT DESCRIPTION FILE

The keypoint descriptions are used by the keypoint analyzer utility to direct the reformatting of the keypoint information.

D1.3.2.1 keypoint_decks

Each area has a keypoint constant deck xxDKEY (where xx is the product id). The keypoint descriptions are now included in this deck immediately following the keypoint constants (simillar to the message templates).

Each description has the following format.
Note: each element (if given) is positionally dependant.

```
{CLASS [SUB_ID_FIELD]  KEYPOINT_LABEL [DATA_LABEL] [DATA_FIELD]

CLASS of keypoint - required
  E Entry
  X eXit
  U Unusual
  D Debug

SUB_ID_FIELD - optional - (described later)

KEYPOINT_LABEL - required -  This is a string that
  describes the purpose of the keypoint.

DATA_LABEL - optional -  This is a string of up to 8
  characters describing the data portion of the keypoint.

DATA_FIELD_DESCRIPTOR - optional - This consists of data
  format and length.
  data_format
    H  Hex
    I  Integer (decimal)
    A  ASCII
  Concatenated to this is the length of the data portion of
  the keypoint, in decimal bits.
  For example: I20
```

D1.3.2.1.1 EXAMPLE KEYPOINT DECK

```
STDKEY
COMMON

{ PURPOSE:
{   This deck contains all of the set manager keypoint constants.

  CONST
    stk$create_set = stk$base + 1,
      {E  'stp$create_set' 'ring    ' H }
      {X  'stp$create_set' 'status  ' I20 }

     stk$purge_set = stk$base + 2,
       {E 'stp$purge_set' }
       {X 'stp$purge_set' 'status  ' I20 }

    stk$cant_dm_store_set_ord = stk$base,
      {U  'cant dmp$store_avt_set_ordinal' 'avtindx ' H20 }

    stk$pf_root_size = stk$base + 5;
      {D  'pf_root_size' 'rootsiz ' H20 }

?? PUSH (LIST := OFF) ??
{*callc osdkeyd
?? POP ??
```

D1.3.2.1.2 SUB_ID_FIELD

This optional field allows a means of subdividing a single keypoint into several descriptors. The particular descriptor is chosen on the basis of a selectable number of bits of the data field. This field has the following format:

```
SUB_ID_LENGTH.SUB_ID_MATCH
```

SUB_ID_LENGTH - This specifies the number of bits (right most) of the data field to use, to determine which descriptor to choose.

SUB_ID_MATCH - This specifies the integer identifier used to match the data portion.

Example:

```
  mmk$page_fault = mmk$monitor_base + 6,
    {E 'page fault processor' }
    {E 4.1 'Page found in avail queue' 'pfti    ' H16}
    {E 4.2 'Page found in avail modified queue ' 'pfti   ' H16}
```

If this keypoint was issued and produced data of 2, the descriptor with the sub_id_match field of 2 would be used ('page found in avail modified queue').
These keypoints were issued with a sub_id_length = 4, thus the 4.x. For example:

```
#INLINE ('keypoint', osk$entry, osk$m *
  (pfti * 16 {2 ** sub_id_length} + 2{sub_id_match}),
  mmk$page_fault);
```

D1.3.2.2 generating_the_descriptor_file

The keypoint descriptions are kept on a file called KEYDESC on the integration catalog. This file may be produced by:

```
 SES.GENCOMP M=OSMKEYS AB=((NOSVEPL,OSLPI,INT2)) CF=KEYDESC
```

The user may add keypoints to her xxDKEY deck locally, and the KEYDESC file may be produced as above, specifying the additional local bases. The KEYDESC file may then be saved on her catalog.

If new keypoint decks are added, *callc 's to these new decks chould be added to the deck OSMKEYS, and the appropriate base constants added to deck OSDKEYD.

When transmitting changes to keypoint decks, be sure to inform integration, via the transittal form, to recreate the file KEYDESC.

D1.3.2.3 osmkeys_format

This section will only be useful to those desiring to add additional keypoint classes, keypoint class base constants, or new keypoint description decks.
The classes, identifiers, and descriptions are each buffered by a comment. For example, to add another keypoint class:

```
{$$$ START KEYPOINT CLASSES $$$}
CONST
  psk$entry = osk$product_set_class + 1; {E PS - entry keypoint}
{$$$ END KEYPOINT CLASSES $$$ }
```

note: The E follwoing the "{" will be used in the description.

D1.0 KEYPOINTS
D1.3.2.3 osmkeys format

This new section should be appended to the end of the KEYDESC file. Readers desiring more information should reference the attached BNF, and the attached decks OSMKEYS.

The following represents a sample of how to set up the description module.
Note: Comment put around *call for sake of documentation only.

```
OSMKEYS
?? LEFT := 1, RIGHT := 110 ??
MODULE keypoint_description_file;
{*callc,osdkeys
{$$$ START KEYPOINT CLASSES $$$}
{*callc,osdkeyc
{$$$ END KEYPOINT CLASSES $$$}
{$$$ START KEYPOINT IDENTIFIER BASES $$$}
{*callc,osdkeyd
{$$$ END KEYPOINT IDENTIFIER BASES $$$}
{$$$ START KEYPOINT DESCRIPTIONS $$$}
{*callc,amdkey
{*callc,badkey
{*callc,cldkey
{*callc,cmdkey
{*callc,dbdkey
{*callc,dmdkey
{*callc,fmdkey
{*callc,icdkey
{*callc,ifdkey
{*callc,iidkey
{*callc,indkey
{*callc,jmdkey
{*callc,lgdkey
{*callc,lldkey
{*callc,lodkey
{*callc,ludkey
{*callc,mldkey
{*callc,mmdmkey
{*callc,mmdjkey
{*callc,msdkey
{*callc,mtdkey
{*callc,ocdkey
{*callc,ofdkey
{*callc,osdkey
{*callc,pfdkey
{*callc,pmdkey
{*callc,rhdkey
{*callc,srdkey
{*callc,stdkey
{*callc,tmdmkey
```

```
{*callc,tmdjkey
{*callc,jsdmkey
{*callc,jsdjkey
{*callc,avdkey
{*callc,sfdkey
{*callc,iodkey
{*callc,rmdkey
{$$$ END KEYPOINT DESCRIPTIONS $$$}
MODEND keypoint_description_file;
```

D1.4 REFORMATTED_FILE_DESCRIPTION

The output from procedure NVEKEY is a file called KEYFILE. This reformatted listing contains two sections. The first section is a listing of all the keypoints in the order they were issued. The second section is a summary of the number of times each keypoint occured.

Each line in the first section has the following format;

```
RT TSL DATA DATA_LABEL S TN AREA_ID KP_LABEL
```

The RT field designates the value of the free running microsecond clock (time since deadstart) when the keypoint was executed. On the simulator the clock is incremented by 1 for each instruction executed.

The TSL field designates the time (microseconds) since the last keypoint instruction was executed.

The DATA field specifies the value of the data portion of the keypoint in the format described in the keypoint description file for this keypoint.

The DATA_LABEL field is the data label field from the keypoint description file for this keypoint. This identifies the data being displayed.

The S field specifies the state of the machine when the keypoint was issued and is one of the following:

M - Monitor mode
J - Job mode

An * preceding the S field indicates that trap processing is active, that is the trap handler has been entered, but not exited.

The TN field gives the global task id of the task that was executing when the keypoint was issued. The system is task 1.

The AREA_ID field is the area identifier for the area issuing the keypoint.

The KP_LABEL is the keypoint label field from the keypoint description file. This describes the keypoint.

NOTE: For an undefined keypoint, that is, one which has no descriptor entry, the area_id field contains the integer for the keypoint class, the class field on the output is specified as "UND", and the KP_LABEL becomes the id_number of the keypoint.

D1.5 BNF_KEYPOINT_DESCRIPTION

```
<analyzer_descriptor_input> ::= <keypoint_class_allocation_deck>
                                [ <definition_deck> ... ]

<keypoint_class_allocation_deck> ::= <cybil code and/or comments>
                                     [ <class_base_definitions> ... ]
                                       <cybil code and/or comments>

<class_base_definitions> ::= <class_base_id> <spc> = <spc> <base>

<class_base_id> ::= osk$system_class ! osk$product_set_class !
                    osk$user_class ! osk$pmf_control

<spc> ::= [ <space> ... ]

<base> ::= <integer>

<definition_deck> ::= <class_definition_deck> !
                      <base_definition_deck> !
                      <keypoint_definition_deck>

<class_definition_deck> ::= {$$$ START KEYPOINT CLASSES $$$}
                            <cybil code and/or comments>
                            [ <class_definitions> ... ]
```

```
                              <cybil code and/or comments>
                              {$$$ END KEYPOINT CLASSES $$$}

   <class_definitions> ::= <keypoint_class> <spc> = <spc>
                           <class_base_id>  <offset>
                           { <keypoint_class_id> <cybil comment>

   <keypoint_class> ::= <identifier>

   <offset> ::= + <spc> <integer> <delimiter>

   <delimiter> ::= , ! ;

   <keypoint_class_id> ::= <character>

   <base_definition_deck> ::=
                    {$$ START KEYPOINT IDENTIFIER BASES $$$}
                             <cybil code and/or comments>
                             [ <range_base_definitions> ... ]
                             <cybil code and/or comments>
                             {$$$ END KEYPOINT IDENTIFIER BASES $$$}

   <range_base_definitions> ::= <keypoint_base> <delimiter>
                                <base_range>

   <keypoint_base> ::= <spc> <base_id> <spc> = <spc> <base>

   <base_id> ::= <identifier>

   <base_range> ::= <spc> { <low_base> <sp> - <high_base> [ } ]

   <low_base> ::= <integer>

   <high_base> ::= <integer>

   <keypoint_definition_deck> ::=
                          {$$$ START KEYPOINT DESCRIPTIONS $$$}
                               <cybil code and/or comments>
                               [ <xxdkey_deck> ... ]
                               <cybil code and or comments>
                               {$$$ END KEYPOINT DESCRIPTIONS $$$}

   <xxdkey_deck> ::= [ <cybil code and/or comments> ]
                     [ <keypoint_info> ... ]

   <keypoint_info> ::= <keypoint_constant_line> <delimiter> <eol>
                              [ <keypoint_descriptor> ... ]
                              [ <blank lines> ]
   <keypoint_constant_line>  ::= <keypoint_constant> <spc> = <spc>
                               <keypoint_base> <spc> [ <offset> ] <spc>

   <keypoint_constant> ::= <identifier>

   <keypoint_base> ::= <identifier>

   <keypoint_descriptor> ::= { <keypoint_descriptor_list> <spc> [ } ]
                             <eol>

   <keypoint_descriptor_list> ::= <keypoint_class_id> <spc>
                              [ <special_case_code> ] <spc>
                              [ <sub_id_field> ] <spc> <keypoint_label>
                              <spc> [ <data_field> ]

   <special_case_code> ::= M ! N ! S ! T
           (M = Mtr, N = Nos, S = task Switch, and T = Trap)

   <sub_id_field> ::= <sub_id_length> . <sub_id_match>

   <sub_id_length> ::= <field_length>

   <field_length> ::= 0..52                                    (in bits)

   <sub_id_match> ::= <small_integer>

   <small_integer> ::= 0..0 FFFFFFFFFFFFF(16)

   <keypoint_label> ::= <label>

   <label> ::= ' <character_string> '

   <character_string> ::= any visible characters except '

   <data_field> ::= <data_label> <spc> [ <data_field_descriptor> ]

   <data_label> ::= <label>

   <data_field_descriptor> ::= <data_format> [ <data_field_length> ]

   <data_format> ::= A ! H ! I
           (A = Alphanumeric, H = Hex, I = Integer)

   <data_field_length> ::= <field_length>

   (NOTE: <sub_id_length> + <data_field_length> must be <= 52 bits )
   (NOTE: operating system <keypoint_class_id> = {D,E,U,X})
   (NOTE: <keypoint_class_id> for any keypoint used for
   additional information to previous keypoints
```

must be a space)
(NOTE: a <definition_deck> remains in effect until
superceeded by a deck which redefines the
area to which it pertains)

Table of Contents