DISTRIBUTION

C.K. Bedient	ARH254
F.A. Bischke	ARH254
W.H. Braunwarth	ARH254
J.B. Farr	ARH254
D.A. Henseler	ARH254
G.J. Kramer	ARH254
J.J. Krautbauer	ARH254
W.V. Mahal	ARH254
A.E. Murray	ARH254
R.A. Peterson	ARH254
T.J. Sparrow	ARH254
J.F. Steiner	ARH254
H.A. Wohlwend	ARH254
S.C. Wood	ARH254
R.B. Beeson	ARH260
R.E. Erickson	ARH260
K.M. Jacob	ARH260
R.A. Mann	ARH260
G.S. Barrett	ARH263
J.C. Bohnhoff	ARH263
A.J. Lawson	ARH263
L.E. Leskinen	ARH263
J.L. Nading	ARH263
C.G. Nelson	ARH263
G.W. Propp	ARH263
J. Sutherland	CANCDD
H. McGilton	SVL164
R. Westgaard	SVL118

Please help keep the above distribution list current. If your name should be removed from the list or another name added, contact Mike Carter at ARH260 - extension 2553.

DISTRIBUTION

P.W. Haynes	ARH254
R.H. Kingdon	ARH254
T.C. McGee	ARH254
R.M. Medin	ARH254
J.R. Ruble	ARH254
R.E. Dennis	ARH260
J.W. Graffius	ARH260
D.J. Maguire	ARH260
N.E. Meyer	ARH260
S.W. Fewer	ARH263
E. LaRowe	ARH263
D.R. Boeckling	SVL100E
A.W. Ceglia	SVL114
J.S. White	SVL102
D. Williams	SVL102

Please help keep the above distribution list current. If your name should be removed from the list or another name added, contact Mike Carter at ARH260 - extension 2553.

```
          MM   MM  EEEEEEE  MM   MM   OOOOO
    C     M M M M  E        M M M M  O     O     C
    D     M  M  M  EEEEE    M  M  M  O     O     D
    C     M     M  E        M     M  O     O     C
          M     M  EEEEEEE  M     M   OOOOO
```

DATE : SEPTEMBER 17, 1980

TO : DISTRIBUTION LOCATION :

FROM : M. D. CARTER LOCATION : ARH260

SUBJECT : UPDATED PROCEDURES NOTEBOOK DOCUMENTATION

The Build J update of the Integration Procedures Notebook is now available. Copies of this document may be obtained via the following command sequence:

```
ATTACH,IPNDOC/UN=DEV1
SES.PRINT IPNDOC
```

Change pages are not being distributed for this level (as has been suggested) due to the fact that the volume of change pages is only sligtly smaller than the document itself. Some highlights of this revision are as follows:

- Section 1 has been reduced in size as the information in this section was either inaccurate or out of date.
- Several procedure updates have been documented in Section 2, and Section 3 has been corrected or clarification added where necessary.
- The current Regression tests and test sequence has been documented in Section 4. The source of the tests as they are actually run has been replicated here, and the test sequence is exactly that used by the Integration project at this level.
- The NOS/VE Transmittal Form has been changed to provide more information to the analyst transmitting code.
- The latest version of the S2 Machine Usage Document (or "Helpful Hints") has been picked up as Appendix E. Information in this document takes precedence over that in Appendix F as it is more up to date.
- Appendix F has been reduced in size considerably, as redundant and inaccurate information has been deleted. The information in this Appendix still reflects much of the HCS syntax and conventions, some of which are still valid for NOS/VE at this level.

1.0 OVERVIEW OF INTEGRATION PROCESS

1.0 OVERVIEW OF INTEGRATION PROCESS

The Integration process begins with the transmittal of a software product, the command language procedures required to build the product, installation procedure documentation, and baseline documentation from the software development organization. Subsequent to this transmittal, the Integration project is responsible for maintaining the program library, standardizing the installation procedures, maintaining the installation procedure documentation, and preparing the software release package for the Software Manufacturing and Distribution organization. In the interim time between the initial transmittal and the release of a software product, the Integration project schedules periodic builds. The outputs from these builds are delivered to software development and test organizations and/or made part of the software release package.

1.0.1 RELATED DOCUMENTS

Document Title	Distributor
NOS/VE Procedures and Conventions	S.W. Fewer
NOS/VE User's Guide	Appendix F
NOS/VE Command Interface ERS	DCS - ARH3609
NOS/VE Program Interface ERS	DCS - ARH3610
SES User's Handbook	DCS - ARH1833
CYBER 180 System Interface Standard	DCS - S2196
Simulated NOS/VE Program Interfaces ERS	DCS - ARH3125
VEGEN ERS	DCS - ARH2591
VELINK ERS	DCS - ARH2816
CYBIL Language Specification	DCS - ARH2298
CYBER 180 CPU Assembler ERS	DCS - ARH1693
CYBER 180 Simulator ERS	DCS - ARH1729
SES Procedure Writers Guide	DCS - ARH2894
CYBER 180 Object Code Utilities ERS	DCS - ARH2922
Source Code Utility ERS	DCS - ARH1766

1.0.2 STANDARDS

In order to facilitate the Installation process, certain standards will have to be set and adhered to by all members of the Operating System and Product Set. These standards (to be defined in the Systems Interface Standard) will cover the following items:

a) All program libraries will have the same format, this will be defined by (TBD).

b) All output tapes will conform to some predetermined format in terms of numbers of files and what each file will contain. This will be defined by (TBD).

c) The above formats are intended to facilitate establishment of proceduralized installation decks. This implies that some convenient naming conventions must be observed. These conventions will be defined by (TBD).

2.0 NOS/VE OPERATING SYSTEM BUILDS (CI)

2.0 NOS/VE OPERATING SYSTEM BUILDS (CI)

2.0.1 INTRODUCTION

The command language procedures corresponding to NOS/VE builds all reside in the INT1, INT2, or DEV1 catalogs depending upon the desired level of verification the system has attained. It is assumed that the DEV1 level of verification is the minimum level of system verification required by most users, therefore the DEV1 catalog is frequently referenced in the remainder of this section. (A description of the catalog management policies employed for NOS/VE follows.) To obtain a listing of the complete set of command language procedures provided in the Integration catalog, execute the following command language sequence:

```
ACQUIRE,PROCLIB/UN=<Integration_Catalog>
SES.PRINT PROCLIB
```

Before running the build procedures as batch jobs, a check must be made to insure that the user number under which the job will run has sufficient validation limits for the job to execute. The minimum values for certain limits must be as follows:

```
CM = 2437B
NF = unlimited
MS = unlimited
DS = 4096
EC = 2008 (if simulator is to use LCM)
DB = unlimited (each library is built via batch job)
```

The current values may be obtained with the LIMITS control card. If they are not large enough, have the operations staff change them.

2.0.2 CATALOG MANAGEMENT POLICIES

The catalog management function done by the Integration project approximates the cycle concept of permanent file management. New system files begin in the INT1 catalog and move to the INT2 and DEV1 catalogs as some significant

verification milestone has been achieved. Each catalog is intended to be self-contained, in that procedures executed from either catalog will access only those files which have the same level of verification associated with them. The INT1 catalog will access the most recent compilers, SES tools, etc. while the INT2 catalog only contains selected system files from the last "stable" interim build which will deadstart and run a minimum set of test cases. The DEV1 catalog represents the "frozen" catalog for which changes are no longer being accepted (this is typically a snapshot of the last build cycle). The DEV1 catalog will change no more frequently than once for each build cycle. The INT2 catalog will change only as frequently as bug fixes or system upgrades demand, with the objective of keeping this catalog no more than one week out of synchronization with the INT1 catalog. The INT1 catalog, however, is a "working catalog" for the debug of new system fixes, new procedures, etc. The stability of this catalog cannot be predicted.

2.1 SOURCE MAINTENANCE PROCEDURES

This section is being replaced by a user's guide which is targeted for inclusion as one of the appendices to this document. The procedures being developed are based upon the prototype Source Code Utility which is being made available in SES Release 14.

2.2 FULL SYSTEM BUILDS

2.2.1 INTRODUCTION

The basic version of NOS/VE consists of 5 base object files (XLMMTR, XLJ11F, XLJ12F, XLJ13F, XLJ1FF), the CYBIL runtime routine library (CYBILIB), and several user program libraries. The general scheme used in modifying or replacing existing modules and adding new modules is to apply the changes to the appropriate base object text file and save this modified file in the current catalog. The checkpoint file build procedure NOSLINK will always attempt to obtain updated object text files from the current user's catalog before it attempts to get them from the Integration catalog, thereby allowing a user to replace these object text files with his own. The "quick link" option of NOSLINK cannot be used when modifying the base object text files as it does not use these

directly, but rather a pre-linked form of these files.

2.2.2 NOS/VE PARTITIONING

The system can be thought of as being partitioned into two sections consisting of the five object files mentioned previously. Each partition has associated with it certain protection, privilege and responsibility. The first partition consists of those routines that run in monitor mode and is known as the monitor. The second partition consists of those routines that run in job mode and is known as task services. The modules on file XLMMTR make up the monitor, and the modules on files XLJ11F, XLJ12F, XLJ13F and XLJ1FF make up task services. A single user task can perhaps be thought of as a third type of partition.

Any XDCL'd symbol within a given partition can be XREF'd by any module within the same partition. To allow other partitions to XREF these same symbols, the symbols must be gated. Gating a symbol only makes the symbol available to other partitions during the linking process, it does not necessarily mean that the XDCL'd location can actually be referenced - that is controlled by the ring brackets. In general, only selected XDCL'd symbols are gated. Refer to the MAP_offset_K file for a list of the entry points available to a user task. This list of entry points preceeds the linkage of user tasks and is entitled "INBOARD SYMBOL TABLE ENTRY POINTS FROM FILE : STSXOST". It should also be noted that a similar list exists for the Monitor entry points available to Task Services and is entitled similarly with the substitution of MTRXOST for STSXOST in the above title.

2.2.2.1 XLMMTR

The packed object text file XLMMTR contains all modules which run in monitor mode. Modules from this file execute in ring 1 of monitor and have a ring bracket of (1,1,1). They also execute with global privilege.

The following modules reside in monitor:

```
CPMTR
SYSTEM_MONITOR
SIGNAL_MONITOR
CPERR
MMM$MEMORY_MANAGER_MONITOR_MODE
CIP$COMMON_INPUT_OUTPUT
```

FAP$FILE_ALLOCATION_MANAGER
SQP$SEQUENCE_MANAGER
CM$CONFIGURATION_MANAGER
CPMCB
KPPROC
TMM$MTR_FLAG_SIGNAL_FUNCTIONS
MEMORY_LINK_MONITOR_MODE
MMMASM
OSAINX
TMM$DISPATCHER
MTM$SERVICE_ROUTINES
MSM$MCU_REQUEST_PROCESSOR

2.2.2.2 XLJ11F

The packed object text file XLJ11F contains all modules which run in ring 1 task services. Modules from this file execute in ring 1 and have a ring bracket of (1,1,F).

The following modules reside in ring 1 task services:

MISC_SERVICES_RING_1
MEMORY_MANAGER
SYSTEM_ROUTINE_JOB_MODE
CONSOLE_DISPLAY_MANAGER
FILE_MANAGER
MDUST
DMP$SYSTEM_INITIALIZATION
JOB_DEADSTART
QUEUED_FILE_MODULE
INITIALIZE_MEMORY_LINK
MEMORY_LINK_INTERFACE
KPPROC
OFM$B_DISPLAY_MANAGER
TMM$DISPOSE_OF_RING1_PREEMPTS
MLP$C170_HELPER
TMM$MANAGE_SIGNALS
OSM$INITIALIZE_TABLES
MMM$SEGMENT_SIGNAL_HANDLER
QUEUE_FILE_INTERNAL_INTERFACES
STM$MODIFY_AST_R1
STM$READ_AST_R1
DMM$KEEP_MVT
DMM$R1_MODIFY_MVT

2.2.2.3 XLJ12F

The packed object text file XLJ12F contains all modules which run in ring 2 task services. Modules from this file execute in ring 2 and have a ring bracket of (1,2,F).

The following modules reside in ring 2 task services:

- JBTBLS
- DMP$JOB_INIT
- SEGMENT_MANAGER
- PROGRAM_LOADER
- SIGNAL_HANDLING_ROUTINES
- TASK_MANAGER
- EXECUTE_PROCESSOR
- LGM$LOCAL_LOG_MANAGER
- TM2_COMMAND_INTERFACE
- LOGICAL_NAME_SPACE_MANAGER
- TMM$ALLOCATE_EXECUTION_RINGS
- TMM$DISPOSE_OF_RING2_PREEMPTS
- TMM$GET_MONITOR_FAULT
- TMM$MANAGE_PREEMPTIVE_BUFFERS
- JOB_INITIATOR
- JOB_MONITOR
- JOB_TERMINATOR
- JM_PROC_R2
- CLM$READ_INPUT_FILE
- LUM$SAVE_LINK_USER_DESCRIPTOR
- MLMDDS
- TMM$CLEAR_WAIT_INHIBITED
- PMM$DEFAULT_LOADER_PARAM_MGMT_2
- FMM$LOCAL_NAME_TABLE_MANAGER
- FMM$TABLES
- INTERACTIVE_USER
- IIM$REPORT_UNHANDLED_DATA_MSG
- IIM$TIME_WAIT
- IIM$REPORT_UNHANDLED_SUPER_MSG
- IIM$REPORT_STATUS_ERROR
- IIM$REPORT_LOGICAL_ERROR
- IIM$ASCII_170_TO_HEX
- STM$MODIFY_VST_R2
- STM$READ_VST_R2
- STM$MODIFY_JOB_ASSOC_CAT
- STM$READ_JOB_ASSOC_CAT
- STM$MODIFY_JOB_SET_TABLE
- STM$READ_JOB_SET_TABLE
- STM$SET_END_JOB
- STM$GET_MOD
- STM$RING2_CREATE_SET

STM$RING2_PURGE_SET
STM$RING2_ADD_MEMBER
STM$RING2_REMOVE_MEMBER
DMM$TEMPORARY_NON_SET_CODE
PFM$R2_REQUEST_PROCESSOR
PFA$CATALOG_SEGMENT_DEFINITION

2.2.2.4 XLJ13F

The packed object text file XLJ13F contains all task service modules which run in ring 3 task services. Modules from this file execute in ring 3 and have a ring bracket of (1,3,F).

The following modules reside in ring 3 task services:

BAM$OPEN
BUFFER_MANAGER
HPP$HEAP_MANAGER
BAM$RECORD_MANAGER
BAM$BLOCK_MANAGER
BAM$SEGMENT_POINTER
BAM$FILE_STRUCTURE_FUNCTIONS
BAM$FETCH_TABLE
PMM$MANAGE_CONDITION_STACKS
PMM$PROGRAM_CONTROL_SERVICES
TMM$DISPOSE_OF_RING3_PREEMPTS
TMM$DISPOSE_PREEMPTIVE_COMMO
TMM$MONITOR_FAULT_HANDLERS
LOM$LOADER_EXECUTIVE
LOM$LOAD_LIBRARY_MODULES
LOM$LIBRARY_LIST_MANAGEMENT
LOM$LOAD_MAP_GENERATION
LOM$MODULE_LOADER
LOM$ENTRY_EXTERNAL_MATCHING
LOM$TEXT_GENERATION
LOM$LINKAGE_GENERATION
LOM$LINKAGE_NAME_TREE_MGMT
LOM$PROGRAM_SEGMENT_MANAGEMENT
LOM$DYNAMIC_TABLE_MANAGEMENT
LOM$PROGRAM_LOAD_LIEUTENANTS
PMM$KLUDGE_LOADER_IF
DATA_PROFILE
GET_DATA_PROFILE
LOM$LOADER_STUBS
LOM$TASK_SERVICES_DEF_MATCHING
LOM$CROSS_REFERENCE_MANAGEMENT
CLM$MANAGE_STND_INP
PMM$INTERFACE_TO_LOGGING

RHM$INTERIM_SIMULATED_IO
RHM$SIMULATED_REPLACE
RHM$SIMULATED_GET
LUM$MANAGE_LINK_USER_INTERFACE
CLM$COMMAND_LIST_MANAGER
CLM$REMOVE_VARIABLE
CLM$WRITE_VARIABLE
PMM$DEBUG_STACK_MANAGERS_13F
PMM$DEBUG_TABLE_BUILDER
PMM$DEFAULT_LOADER_PARAM_MGMT
CLM$BLOCK_STACK_MANAGER
CLM$DECLARE_VARIABLE
CLM$READ_VARIABLE
CLM$INPUT_STACK_MANAGER
DEBUG_SYSTEM_LEVEL_EXECUTION
JMM$PROGRAM_LEVEL_INTERFACES
QUEUE_FILE_PROGRAM_INTERFACES
CLM$PROCESS_ASSIGNMENT
PMM$MANAGE_LOCAL_QUEUES
AMM$STORE_FAP_POINTER
JMQF_TEMPORARY_MODULE
BAM$CLOSE
BAM$CONTROL
BAM$FETCH_ART_TABLE_POINTER
BAM$PAD_RECORD
BAM$REWIND
BAM$STORE_ART_TABLE_POINTER
RMM$REQUEST_TERMINAL
IIM$TASK_PRIVATE_DATA
BAM$SEEK_DIRECT
AMM$STORE_PROCEDURE_POINTER
STM$CREATE_SET
STM$PURGE_SET
STM$ASSOCIATE_CATALOG
STM$MISC_SERVICE_ROUTINES
STM$ADD_MEMBER
STM$CHANGE_ACCESS_TO_SET
STM$SIMULATED_REST_OF_WORLD
STM$REMOVE_MEMBER
DMM$MAKE_VOLUME
PFM$PROGRAM_INTERFACE_PROCESSOR
PFM$PARAMETER_CHECKING
PFM$PARAMETER_CONVERSION

2.2.2.5 XLJ1EE

The packed object text file XLJ1FF contains all modules which run in the ring of the caller. Modules from this file have a ring bracket of (1,F,F).

2.0 NOS/VE OPERATING SYSTEM BUILDS (CI)
2.2.2.5 XLJ1FF

The following modules reside in the any ring task services:

MCPJTP
OCALL
SRM$CONVERSION_SERVICE_MANAGER
MISC_SERVICES_ANY_RING
CMD_PROCESSOR_UTILITIES
COMMAND_INTERFACE
FILE_MGR_COMMANDS
DEBUG_USER_LEVEL_EXECUTION
PMM$RUNANYWHERE_PRG_SERVICES
MEMORY_MANAGER_REQUEST_PROCS
CPMCB
PFM$INTERIM_COMMAND_PROCESSOR
AMM$FILE_STRUCTURE_FUNCTIONS
PMM$CONDITION_STACK_PROCESSOR
PMM$DISPOSE_OF_CONDITIONS
PMM$DISPOSE_OF_TRAPS
TMM$DISPOSE_OF_MONITOR_FAULT
CLM$ANALYZE_TOKEN
CLM$ANALYZE_VALUE
CLM$SCAN_EXPRESSION
CLM$CONVERT_INTEGER_TO_STRING
CLM$LEXICAL_PROCESSORS
CLM$CONVERT_VALUE_TO_STRING
OSM$FORMAT_MESSAGE
OSM$SET_STATUS_ABNORMAL
OSM$TRANSLATION_TABLES
OSAINX
MMM$SEGMENT_FAULT_HANDLER
CLM$ACCEPT
CLM$COLLECT_TEXT_COMMAND
CLM$INCLUDE
CLM$PROCESS_COMMANDS
CLM$SCAN_PARAMETER_LIST
CLM$SET_OBJECT_LIST
CLM$SET_PROGRAM_OPTIONS
OSM$LOCK_MANAGER
MAINT_COMMANDS
OSM$GENERATE_MESSAGE
CLM$DISPLAY_VALUE_COMMAND
CLM$ACCESS_BLOCK_STACK
CLM$ACCESS_PARAMETERS
CLM$FILE_REFERENCE_MANAGER
CLM$CONVERT_STATUS
PMM$DEBUG_STACK_MANAGERS_1FF
PMM$PROGRAM_SERVICES
PMM$SYSTEM_TIME_DECLARATIONS

PMM$STATUS_QUEUES_DEFINED
REC_MGR_COMMANDS
AMM$ADVANCED_ACCESS_METHODS
AMM$CLOSE
AMM$FETCH_ACCESS_INFORMATION
AMM$FETCH
AMM$FLUSH
AMM$GET_NEXT
AMM$GET_SEGMENT_POINTER
AMM$OPEN
AMM$PUT_PARTIAL
AMM$PUT_NEXT
AMM$REWIND
AMM$SKIP
AMM$SET_SEGMENT_EOI
AMM$STORE
AMM$CHECK_RECORD
AMM$DELETE_KEY
AMM$GET_KEY
AMM$GET_NEXT_KEY
AMM$PUT_KEY
AMM$PUTREP
AMM$REPLACE_KEY
AMM$START
AMM$SET_SEGMENT_POSITION
AMM$GET_PARTIAL
AMM$SEEK_DIRECT
AMM$GET_DIRECT
AMM$PUT_DIRECT
AMM$GET_PARTIAL_DIRECT
AMM$PUT_PARTIAL_DIRECT
CLM$PERMANENT_FILE_COMMANDS
CLM$DEBUG_COMMANDS

2.2.2.6 Data Residency/Lifetime Based on Partition

The following rules apply to static data defined by modules in monitor or task services:

1) Only modules within monitor may declare static data that is mainframe wired.

2) Only modules within ring 1 task services may declare static data that is mainframe paged.

3) Only modules within ring 2 task services may declare static data that is in the job fixed segment. No modules may declare data in job pagable until the new system generator

is available.

4) Only modules within ring 3 task services may declare static data that is in the task private segment.

5) Rules 1 through 5 also mean that all static data for a module in a given partition will reside as specified.

6) Static data for modules that run in the ring of the caller (XLJ1FF) must be read only when executing above ring 3.

2.2.3 MANIPULATION OF NOS/VE PARTITIONS AND LIBRARIES

When building a system, monitor must be linked first. All gated symbols within monitor then become available to task services, which is linked second. Although some monitor symbols can be referenced by task services, the only way to execute monitor code is via the exchange jump - i.e., the CALL/RETURN mechanism is not valid for use between monitor and job modes. User tasks are linked last and can reference gated symbols defined in task services. It is important to note that although the linker will allow a reference to a given symbol, the ability to actually reference the location is determined by the ring brackets on both ends of the reference.

2.2.4 SYSTEM BUILD PROCEDURE DESCRIPTIONS

In order to understand the procedure descriptions which follow, something should be said as to the sequence in which these procedures are used to generate systems. The following is an attempt to accomplish this:

2.2.4.0.1 BACKGROUND INFORMATION

Invoking_the_Procedures

The procedures described below are documented as "SES.<Procedure_Name>". In actuality, to invoke the procedures in this manner assumes that there is a file named 'PROFILE' in the current catalog which names the Integration catalog to search for the procedure (via the 'SEARCH' directive). The alternative mechanism for invoking these procedures is to code the procedure call as: "SES,<Integration_Catalog>.<Procedure_Name>". Many of the procedures use the 'PRCUNAM' value for substitutable user

names, meaning that the catalog in which the procedure is found is the catalog which is searched for files. This is as it should be, since each of the Integration catalogs contains a different version of the system.

All of the procedures described in this document have "HELP" documentation associated with them. Use the SES,<Integration_Catalog>,HELP.<Procedure_Name> call to have procedure documentation printed at your terminal.

Current Packaging of NOS/VE Source

There are two execution modes of NOS/VE which are referred to as the "standalone" mode and the "dual-state" mode. All of the NOS/VE source modules which execute in the CY180 Virtual State are contained on a program library named 'NOSVEPL'. The program interfaces to the Virtual State system, those described in the NOS/VE Program Interface ERS, exist as common decks on a program library named 'OSLPI'. The content of these two program libraries is referred to as the standalone system. A deadstart tape can be produced of the standalone system for execution on the hardware, or the output of the Virtual Environment generator can be executed directly on the Hardware System Simulator. The I/O support of this standalone system when running on the simulator is defined in a separate set of common decks on a program library named 'CYBICMN'. Refer to the Simulated I/O ERS for documentation of these I/O interfaces.

The dual-state execution of NOS/VE, in conjunction with the NOS operating system, requires NOS system modifications and the presence of a set of NOS utilities and procedure files. The software which supports this dual-state environment from the 'NOS' side of the hardware is contained on a program library named 'VE170PL'. Included in this package of NOS/VE support programs is a software application called the Interim Remote Host Facility which supplies job-to-job communication between the Virtual State and NOS portions of the CY180 machine. The Interim Remote Host Facility build procedures are not documented here at this time.

2.2.4.0.2 THE BUILD SEQUENCE

Update the Source Libraries

The Integration project typically updates the base source libraries prior to starting any recompilation or assembly of the system. In order for a user of these procedures to modify the source of a system routine he/she can use the SES 'GETMODS' procedure to extract the source being modified, or

create the source in some other manner. If GETMODS was used to extract the source, then REPMODS can be used to put this changed source on a MADIFY program library in the user's catalog. Then the filename containing this program library must be specified as the value of the 'AB' parameter of the NOSBILD procedure. (Refer to the Source Maintenance Section of the SES User's Handbook if you have questions about source maintenance.)

Compile/Assemble from Source

One must first determine how much of the system is to be compiled or assembled. If only a few modules (less than 5) are being compiled, then the NOSBILD procedure should be used with the 'M' option to update the appropriate files. If a larger number of modules (5 or more in the same system library) are to be recompiled, then it is "cheaper" to use the 'L' option and rebuild the entire system library from scratch. If more than two libraries are to be recompiled, then the NOSBILF procedure should be used to submit a separate execution of the NOSBILD procedure for each library to be rebuilt. This latter method is the one used for rebuilding all of the NOS/VE Virtual System from scratch.

The general philosopy behind the NOSBILD procedure is to extract the latest source of a module from a program library, compile or assemble the source to produce the appropriate object text, replace/add the updated object text to the appropriate system library, and save this library in the catalog in which the procedure is executed. The final result of the execution of the NOSBILD procedure should be an updated system library in the current user's catalog which is ready for the 'LINKER' phase of the build. Jobs which run in "user mode", that is the interface to the system is only through use of the program interfaces (OSLPI), are saved merely as object text files in the user's catalog and LINKER-LOADER directive modifications are required to include these files as part of the system. This latter capability will gradually be replaced by the Virtual State LOADER and Library Generator features as they become available.

Begin The LINKER-LOADER phase

The LINKER (SES.VELINK) and LOADER (SES.VEGEN) are packaged together in the Integration procedure NOSLINK. This is for convenience purposes, in that most LINKER changes to the system require a corresponding LOADER directive change, and the intermediate results from the LINKER execution are not the primary output used for system checkout. Prior to starting the LINKER-LOADER phase of system builds, some decisions need

to be made as to the target execution environment for the resultant output.

If the target execution environment is standalone NOS/VE, then the default NOSLINK options should be used to produce the file named 'LGBOK'. This file can be run on the simulator using the NOSSIM procedure, or can be used to create a deadstart tape using the 'VSN' parameter of the NOSSYS procedure.

If the target execution environment is a dual-state environment, then the OFFSET parameter must be used to specify how large the NOS system will be. The systems produced for the Arden Hills S2 use OFFSET=256 to produce a LGB256K file. The LGB256K file is used by the NOSSYS procedure (when OFFSET=256 is specified) to produce a deadstart tape image on disk named 'TP256K'. This deadstart tape image must then replace the file named TPXXXK, which the dual-state deadstart procedure UPMYVE will then find. Refer to Appendix E for Dual-State and standalone deadstart procedures.

Generate the Deadstart File

In order to generate a deadstart tape for standalone NOS/VE, it is only necessary to run the NOSSYS procedure and specify the VSN of the tape to be written. Prior to generating a dual-state deadstart file, however, it is necessary to verify that the utilities necessary to support the dual-state deadstart have been rebuilt via the DSBILD and BLDEI procedures. There are two portions of the dual-state EI, the A170 portion is built using the BLDEI procedure, and the C180 portion is rebuild using the NOSBILD procedure (deck named MLAEI).

2.2.4.1 NOSBILD Procedure Description

The procedure NOSBILD is used to add or replace modules on a base object text file. NOSBILD retrieves the source module from a program library, using the following search order:

1) an alternate base optionally specified by the user (looking first in the current catalog, and then in the <Integration> catalog)
2) OSLPI (from the <Integration> catalog)
3) NOSVEPL (from the <Integration> catalog)

This module is then compiled or assembled, and the resulting object text is either added to or replaced on a base file. A new version of the base file will be created in the current catalog, along with the direct access file NOSLIST

which contains the compilation or assembly listing(s) of the module(s) compiled or assembled (one listing per record, headed by the matching MADIFY module name, listed via the LISTNVE procedure described in this document). If there are any compilation errors, the error listing(s) will be put on the direct access file RPTERR2 (which has the same format as NOSLIST) in the current catalog. The indirect access file RPTERR1 will contain the message "COMPILATION ERROR IN DECK <module>." for each module which had compilation errors. Any GENCOMP errors will cause a similar message to be written to the indirect access file CMPERR1.

If a specified module is to be replaced (i.e. it is already part of the existing system), NOSBILD will by default use the same compilation options and will replace it on the same base object text file as when it was first added to the system. These options may be overridden by specifying the corresponding parameters described below.

If a specified module is new to the system, the compilation and base object text file options may be directly specified using the parameters described below. If a list of new modules is specified, the compilation and base file options must also be specified as lists, and NOSBILD will match everything up positionally. If these parameters are not specified, and NOSBILD is executing in LOCAL mode, a warning message will be issued telling the user that the module is not in the current system. The user will then be prompted for the necessary information. If these parameters are not specified and NOSBILD is executing in BATCH mode, the compilation and base file options default as specified below.

If the 'I' parameter is specified , each module's object text will be copied to a temporary 'z' file. The old library file will then be purged, and the 'z' file will be renamed as the new library file. All compilation listings will be LIBEDIT'ed onto NOSLIST from a temporary listing file at the end of the procedure.

When an entire library is being rebuilt via the 'I' parameter, the module names and their corresponding compilation options are obtained from a file which contains all this information for each library. NOSBILD searches for this file first in the current catalog, and then in the <Integration> catalog. The name of this file must be the name of the library minus its first character (e.g. 'LJ13F' for the library 'XLJ13F'), and the first line of this file must be the file name. To make additional entries or change existing entries in this file, the following procedure should be

followed:

1) EXTRACT,<libdeks>/UN=<Integration>.
   (where <libdeks> is the name of the compilation information file for the library as described above, and <Integration> is the Integration catalog)
2) Edit <libdeks> to add or change entries. The format and spacing of each entry is important and must be as follows:
   (<m>,<c>,<xref>,<l1>)
   where
   - m = a 7-character left-justified module name
   - c = a 1-character compilation option ('0','1','2','3', or '4'; see description of 'c' parameter below)
   - xref = a 3-character left-justified cross reference option (either 'YES' or 'NO ')
   - l1 = a 7-character left-justified destination library name.

   the entries currently in these files all follow this format so that any additional entries may be lined up quite easily with them.
3) SAVE,<libdeks>.

NOSBILD (with the 'l' parameter specified) may then be used to rebuild the library using these new/modified compilation options.

The format of the NOSBILD is as follows:

```
SES.NOSBILD [ m=(<module name>..<module name>) ]
            [ l=< library name > ]
            [ c=(<compilation option>..<compilation option>) ]
            [ xref=(< xref option>..<xref option>) ]
            [ l1 = (<base file>..<base file>) ]
            [ ab = <alternate base> ]
            [ omit = (<module name>..<module name>) ]
            [ link ; link = <offset value> ]
            [ test = <test file name> ]
            [ print ]
            [ batch ]
```

m : The module name, or range of modules, or list of module names.

l : The library name. Only one library at a time may

be selected. To select more than one, use the NOSBILF procedure.

c : c = 0 to assemble a module
c = 1 to compile a CYBIL module (DEFAULT)
c = 2 to compile a CYBIL module without range checking
c = 3 to compile a CYBIL module using CYBICMN type declarations
c = 4 to compile a CYBIL module using CYBICMN type declarations, and with range checking turned off

xref : 'YES' to run a cross reference (CYBREF)
'NO' to not run a cross reference (DEFAULT)

ll : The base file (list of base files) onto which the module (list of modules) is to be added or replaced. If a new module, the default is 'XLJ1FF'.

ab : The user's alternate base program library containing new and modified modules. The default is 'NEWDKPL'.

omit : Used when running a full build, a module name or list of module names to omit from the build. The default is none.

link : Option to link the newly built or modified system using the procedure NOSLINK. Specifying simply the keyword 'LINK' or 'LINK = 0' links a NOS/VE stand-alone system. To link a dual state system, specify 'LINK = <offset>' where <offset> may be given the values 256, 128, or 64. The default is to not link the system.

test : The name of the file containing the NOS/VE test commands to be input to the simulator, which will be executed after the system has been linked (using procedure NOSSIM). This parameter is invalid if the 'LINK' option has not also been specified. The default is to not run the simulator test.

print : Option to print the link map following the linking of the system. The default is not to print the link map.

batch : Run NOSBILD in BATCH mode. The default is to run it locally.

NOTE : One of 'm' or 'l' parameters must be specified.

2.2.4.2 NOSBILF Procedure Description

NOSBILF is an SES procedure file which submits one batch procedure execution of NOSBILD for each system library, each with the 'l' parameter specified for the library to be built.

The format of the NOSBILF is as follows:

```
SES.NOSBILF [ l = <library name >]
            [ batch ]
```

l : The 'l' parameter specifies the library to be built. It can be one library or a list of libraries. The default is to rebuild the entire system.

batch : Run NOSBILF in BATCH mode. The default is to run it locally.

Note : To rebuild one library, the following are identical:
1) SES.NOSBILF l=< library name >
2) SES.NOSBILD l=< library name > batch

2.2.4.3 LISINVE Procedure Description

LISTNVE is an SES procedure file which extracts the compilation listings of the modules specified by the 'M' parameter (module names correspond to the MADIFY deck name given the module) from a text library file and writes them to the file specified by the 'F' parameter in a printable format. The 'M' parameter may select a single module, a list of modules, and/or a range of modules on the library file.

The library file which contains the listings may be selected via the 'I' parameter, and defaults to NOSLIST. LISTNVE will search for this file in the current catalog first, and if it is not there it will go to the catalog specified by the 'UN' parameter.

When LISTNVE has completed, the output file selected by the 'O' parameter will be a local file. It is not automatically printed unless either the 'PRINT' or 'BATCH' option is selected.

The format of the LISTNVE is as follows:

```
SES.LISTNVE        [ m = ( <module name>..<module name> ) ]
                   [ i = <file name> ]
                   [ o = <print file name> ]
                   [ un = <user name> ]
                   [ print ]
                   [ batch ]
```

m :	The module name(s) and / or range of module names which are to be extracted for printing. The default is to extract and format all of the modules.
i ; f ; from :	The name of the text library file from which the compilation listings are to be extracted. The default is NOSLIST.
o ; to ; upon :	The name of the file which will receive the formatted listings to be printed. The default is LISTING.
un :	The name of the catalog to search for the library file should it not be found in the current catalog. The default is the <Integration> catalog.
print :	Option to print the listing file after it is formatted. The default is to not print the listing file.
batch :	Run LISTNVE in BATCH mode. The default is to run it locally.

2.2.5 NOSLINK PROCEDURE DESCRIPTION

NOSLINK is an SES command language procedure file which will call both the VE Linker and VE Generator (using the standard SES procedures VELINK and VEGEN) to produce a checkpoint file and link map file. In order to do this it will link monitor and task service routines from their object text files. It will search all files that it requires 1) from local files 2) from the current catalog, 3) from area user's catalog (if the area parameter is specified), 4) from the <Integration> catalog.

NOSLINK will link either a stand-alone or a dual state

system. The choice is made via the OFFSET parameter (see description below). NOSLINK will put the checkpoint file on the direct access file LGB<offset>K, and the link map will be placed on the direct access file MAP<offset>K, where the value given the OFFSET parameter is substituted into each name for <offset>.

To link additional user jobs into the system, create a file in the current catalog containing the commands needed to obtain all the necessary files as well as a call to VELINK for each user job to be linked. Specify this file via the ADD parameter, and NOSLINK will pick it up and physically insert it into procedure command stream immediately following the last call to VELINK. The first line in this file MUST be the file name!!

The format of the NOSLINK is as follows:

```
SES.NOSLINK [ offset = < load offset > ]
            [ save ]
            [ quick ]
            [ uj = ( < user job >,...,< user job > ) ]
            [ add = < additional links file > ]
            [ print ]
            [ test = < test file name > ]
            [ dump ]
            [ area = < user name > ]
            [ batch ]
```

offset : The load offset, used to determine whether to link a stand-alone or a dual state system.
OFFSET = 0 links a stand-alone system (DEFAULT)
OFFEST = 256 links a dual state system (to use on the S2)
OFFSET = 128 links a 128K dual state system
OFFSET = 64 links a 64K dual state system

save : Option to save the monitor and task services segment files created during the link for subsequent "quick link" use by this procedure. The default is not to save these segment files.

quick : Option to do a "quick link". A more complete description of this option can be found in the section entitled "Quick Link Option of NOSLINK Procedure". The default is not to do a quick link.

uj : File or list of files containing the existing

user job binaries that are to be linked into the system. The default is to link in all existing user jobs.

add : The name of the file containing the commands needed to link additional user jobs into the system. The default is to not link in any additional user jobs.

print : Option to print the link map. The default is not to print the link map.

test : The name of the file containing the NOS/VE test commands to be input to the simulator, which will be executed after the system has been linked (using procedure NOSSIM). The default is to not run the simulator test.

dump : Option to print a memory dump of the system. Default is no memory dump.

area : Option to obtain the object files or linker parameter files from another user's catalog (other than the current catalog in which the procedure is executing). The default is for no area user catalog to be searched.

batch : Run NOSLINK in BATCH mode. The default is to run it locally.

2.2.5.1 LPF File Description

The LINK commands used in the NOSLINK procedure do not specify enough information to totally define the requirements of the linking operation. Many additional parameters are supplied to the linker through additional data files. This includes information such as:

- Ring Numbers
- Segment Numbers
- Segment Attributes
- Execution Privilege

Currently this information is supplied to the linker via the SES Linker Parameter File (LPF) file. The linkage between the linker and the LPF file is activated by the LPF=LIBLCB parameter on the LINK commands. For the monitor linkage this information is on LPF file MTRLCB, task services linkage

information is on LPF file STSLCB, and EI/EIE linkage information is on EILCB/EIELCB respectively.

2.2.5.2 VELDCM / LDR File Description

The VELDCM file used by the procedure NOSLINK contains directives to the CPF Generator which allow it to produce a checkpoint file from the segment files produced by the VE Linker. These directives set up the physical environment into which NOS/VE is placed, and include such things as the definition of the page size, job and monitor exchange package addresses, page table address and length, preallocated segment array definitions, etc.

VELDCM is a "skeleton" file which is dynamically edited during the execution of the NOSLINK procedure, depending upon the specification of the OFFSET parameter. The edited file is then put on a direct access file named LDR<offset>K (where <offset> is replaced by the value given to OFFSET when the procedure was called) in the user's catalog. It contains the directives to the CPF Generator which set up the physical environment for that particular link. This file must remain permanent in the user's catalog after NOSLINK has been executed, as the procedure NOSSYS uses this file in building a deadstart tape.

2.3 ADDING USER TASKS TO NOS/VE

2.3.1 INTRODUCTION

A user task can be defined as a group of modules linked together that will execute in the 'user ring' of NOS/VE, currently ring 11. This task may make calls to any gated entries within task services (rings 1 through 3) if the call bracket will allow the call. Data defined within task services may not be referenced from rings 4-15.

2.3.2 QUICK LINK OPTION OF NOSLINK PROCEDURE

To do a "quick link", specify the QUICK option on the call to NOSLINK (see the section entitled "NOSLINK Procedure Description). In this case NOSLINK does not link monitor and task services from their object text files each time. Instead, it uses already linked monitor and task service environments, and just links the specified user object files.

The QUICK option causes NOSLINK to run faster and requires less resources than the full link, and therefore should be used instead of the full link when only user tasks are being added to NOS/VE.

NOSLINK, with the QUICK option specified, should not require any modification to execute in different user task configurations. The user can merely name the user object text file produced by an assembly or compilation as 'XUUSER1'. To link additional user object files into the system, either specify the existing files to be linked via the UJ parameter, or specify the necessary command file via the ADD parameter. The contents of this file, its use, and the function of the UJ and ADD parameters are described in greater detail in the section entitled "NOSLINK Procedure Description".

2.4 NOS/VE SIMULATION

2.4.1 RUNNING A SIMULATOR TEST (NOSSIM PROCEDURE)

NOSSIM is an SES procedure file which will run either a batch mode or an interactive simulation of NOS/VE. This option is selected via the 'TEST' parameter. If 'TEST' is not specified, then the simulation will be run interactively. If a batch mode simulation is desired, then 'TEST' is used to specify the name of the file containing the NOS/VE test commands that are to be input to the simulator. The 'BATCH' keyword must also then be specified. If the user wants to use his/her own simulator directives file, the 'CMDS' parameter must be specified.

NOSSIM also allows the selection of the checkpoint file to be used for the start of simulation. A checkpoint file may also be optionally saved at the end of the test. The C180 memory size may be changed via the 'MEM' parameter.

The NOSSIM procedure will create several permanent files in the user's catalog if not run interactively. These are itemized as follows:

1) IOUIPUI. This direct access file contains all of the output of the NOSSIM procedure, including
   - a copy of the command file used as input to the simulator ('TEST' parameter)
   - the output produced by the system

- the SESLOG file
- a reformatted keypoint listing
- DEBUG output (if 'SIMDBG' was specified on the NOSSIM call)
- a summary of all (paging) disk I/O (HIOLOG file)
- the load map produced by the CITOII conversion and execution of XUUTL (SIMLOAD file)
- (optionally) a hex dump of the checkpoint file at the end of simulation
- the job dayfile.

This file is automatically sent to the line printer.

2) SESSMKE. This direct access file contains the keypoint data produced by the simulator. It is reformatted by the procedure NOSKEY before being written to the file TOUTPUT.

3) IDAYE. The dayfile of the NOSSIM job will be written to this direct access file should it terminate abnormally.

Additionally, if the 'NCPF' parameter is specified, NOSSIM will create 4 direct access files which together contain the NOS/VE environment at the end of simulation. The file specified by the 'NCPF' parameter will contain the current NOS/VE checkpoint file. The other 3 files (formed by adding the characters 0, 1, and 2 to the 'NCPF' file name - which must therefore be six or less characters long) are used for NOS/VE memory paging.

The format of the NOSSIM is as follows:

```
SES.NOSSIM  [ test = < command file > ]
            [ cpf = < checkpoint file > ]
            [ ncpf = < new checkpoint file > ]
            [ mem = < memory size in hex > ]
            [ cmds = < simulator directives file > ]
            [ nods ]
            [ simdbg ]
            [ dump ]
            [ batch ]
```

test : The file containing the NOS/VE test commands. The default is to run interactively.

cpf : The checkpoint file used for the start of simulation. The default is "LGBOK".

ncpf : The checkpoint file to be saved at the end of

simulation. The default is not to save a checkpoint file.

mem : The C180 machine memory size, in hex, needed to run the simulation. The default is "100000(16)".

cmds : Simulator directives file which should be supplied by the user. The default is to use the one created by the NOSSIM procedure.

nods : Option to use the version of the checkpoint file from the <Integration> catalog which has already been deadstarted. The default is to use a checkpoint file which has not been deadstarted.

simdbg : Option to turn DEBUG on for the current simulator run. The default is to run with DEBUG off.

dump : Option to include the dump of the checkpoint file at the end of simulation as part of the NOSSIM output. The default is not to dump the checkpoint file.

batch : Run NOSSIM in batch mode. The default is to run it locally.

2.4.2 NOSKEY PROCEDURE DESCRIPTION

NOSKEY is an SES procedure file which creates a simulator generated keypoint trace file. The output of this procedure is the local file 'KEYFILE'.

The format of the NOSKEY is as follows:

SES.NOSKEY [kpf = < keypoint file >]

kpf : The keypoint file generated by the simulator which is used as input to XXM7KEY. The default is 'SESSMKF'.

2.4.3 DUMPING A SIMULATOR CHECKPOINT FILE (NOSDUMP PROCEDURE)

NOSDUMP is an SES procedure file which makes a DSDI dump of a simulator checkpoint file.

2.0 NOS/VE OPERATING SYSTEM BUILDS (CI)
2.4.3 DUMPING A SIMULATOR CHECKPOINT FILE (NOSDUMP PROCEDURE)

The format of the NOSDUMP is as follows:

```
SES.NOSDUMP [ cpf = < checkpoint file > ]
            [ l = < output file > ]
            [ dump = STND ; ALL ]
            [ print ]
            [ batch ]
```

cpf : The checkpoint file which is to be dumped. The default is "CKPT".

l : The file which is to receive the dump output. This file will be a local file after the procedure has finished execution. It is not automatically printed. The default is "DSDIOUT".

dump : Option to either dump the environment according to ASID (DUMP=STND) or dump the entire environment (DUMP=ALL). If "DUMP=STND" is chosen, then the DSDI directives are taken from the file DSDIX, which the procedure will search for first in the current catalog and then in the <Integration> catalog. The default is "DUMP=STND".

print : Option to print the DSDI dump output. The default is not to print the dump.

batch : Run NOSDUMP in BATCH mode. The default is to run it locally.

2.5 BUILDING A DEADSTART FILE

2.5.1 INTRODUCTION

2.5.2 CREATING THE FILE (NOSSYS PROCEDURE)

The SES procedure NOSSYS builds a deadstart file from the checkpoint file created by the linking of the system. The 'OFFSET' parameter allows the option of building either a stand-alone or a dual state deadstart file. If the parameter 'VSN' is specified, then the deadstart file will be written to tape; otherwise it is written to the file TP<offset>K where

the value of the 'OFFSET' parameter is substituted for <offset> in the name of the file.

NOSSYS requires additional object files for inclusion on the deadstart file. These object files contain PP object code for the following functions:

1) Deadstart (file XIDST)
2) 844 driver (file XIDSK)
3) Console/Printer drivers (file XIDSP)
4) PP helper (file XIHLP)
5) PP Resident program (file XIRES)
6) Others to be defined later

If these files are not present in the current user catalog, they will be obtained from the appropriate catalog. (ie. SES,INT2. prefixed procedure calls access INT2 level system files only, while SES,INT1. prefixed procedure calls may access files from either INT2 or INT1 catalogs as is appropriate for the system being built.)

A copy of the linkmap file (linker/loader output) will also be included on the stand-alone deadstart file. A copy of the loader directives will be included on the dual state deadstart file. These are the files MAP<offset>K and LDR<offset>K (descriptions of these files are included in previous sections), and must be in the same catalog as the file specified by the 'CPF' parameter.

The format of the NOSSYS is as follows:

```
SES.NOSSYS  [ vsn = < tape vsn > ]
            [ offset = < load offset > ]
            [ cpf = < checkpoint file > ]
            [ cyblink ]
            [ batch ]
```

vsn : The VSN of the tape to be written. This file must be available to the operator. The default is to write the file to a tape as specified above.

offset : The load offset, used to determine whether to build a stand-alone or a dual state deadstart file.
OFFSET = 0 builds a stand-alone deadstart file (DEFAULT)
OFFSET = 256 builds a dual state deadstart file (to use on the S2)

OFFSET = 128 builds a dual state deadstart file for a 128K system
OFFSET = 64 builds a dual state deadstart file for a 64K system

cpf : The checkpoint file used in creating the deadstart file. If the file does not exist in the current catalog, it will be obtained from the <Integration> catalog. The default is LGBOK.

cyblink : Option to create a tape for CYBERLINKING to CANCCD. The default is to build a hardware deadstart file.

batch : Run NOSSYS in BATCH mode. The default is to run it locally.

2.5.3 COMPILING 180 PP CODE (CPP180 PROCEDURE)

CPP180 is an SES procedure file which compiles 180 PP code. The source for the PP code may be retrieved from a source file ("SF" parameter) or a program library ("M" parameter). If the "AB" parameter is specified, CPP180 will search this PL first before searching NOSVEPL to satisfy externals. The "UN" parameter specifies the catalog in which "AB" resides. NOSVEPL comes from the <Integration> catalog.

The format of the CPP180 is as follows:

```
SES.CPP180  [ m = < module name > ]
            [ sf = < source file name > ]
            [ ab = < alternate base > ]
            [ un = < user name > ]
            [ print ]
            [ batch ]
```

m : The module name of the PP program to be compiled. If "M" is not specified then "SF" must be specified.

sf : The source file residing in the current catalog which contains the 180 PP source code for the PP program to be compiled. If "SF" is not specified then "M" must be specified.

ab : The alternate base searched by CPP180 to satisfy

externals before searching NOSVEPL. The default is to search only NOSVEPL.

un : The user name of the catalog containing the alternate base specified by the "AB" parameter. The default is the current catalog.

print : Option to print the assembly listing. The default is to not print the listing.

batch : Run CPP180 in BATCH mode. The default is to run it locally.

2.6 DUAL STATE PROCEDURES

2.6.1 BLDEI PROCEDURE DESCRIPTION

BLDEI is an SES procedure file which builds the absolute file for dual state EI. The I parameter may be specified if a file containing the dual state EI source exists in the current catalog; otherwise BLDEI retrieves EI from NOSVEPL in the <Integration> catalog.

BLDEI uses the linker parameter file EILCB to link EI. If this file does not exist in the current catalog, it is obtained from the <Integration> catalog by the procedure.

The outputs of BLDEI include the direct access absolute file 'EI' and the direct access file 'EILIST' which contains the assembly listing and the link map for EI.

The format of the BLDEI is as follows:

SES.BLDEI [i = < EI source file >]

i : The file in the current catalog which contains the dual state EI source from which EI is to be built. The default is to get the EI source from NOSVEPL in the <Integration> catalog.

2.6.2 CPUMBLD PROCEDURE DESCRIPTION

CPUMBLD is an SES procedure file which builds the dual state binaries for CPUMTR, which will be put on the direct access file XCPUMTR. The 'I' parameter may be specified if

the compile file for CPUMTR exists in the current catalog; otherwise CPUMBLD retrieves CPUMTR from VE170PL, searching first in the current catalog and then in the <Integration> catalog. The assembly listing will be written to the direct access text library file DSLIST, with the heading CPUMTR.

The format of the CPUMBLD is as follows:

```
SES.CPUMBLD [ i = < CPUMTR compile file > ]
            [ batch ]
```

i : The file in the current catalog which contains the dual state CPUMTR compile file from which the CPUMTR binaries are to be built. The default is to get CPUMTR from VE170PL in the <Integration> catalog.

batch : Run CPUMBLD in BATCH mode. The default is to run it locally.

2.6.3 CTSBILD PROCEDURE DESCRIPTION

CTSBILD is an SES procedure file which builds the dual state binaries for CTS and MXS and puts them on the direct access file XCTS. The assembly listings for CTS and MXS are written as one record to the direct access text library file DSLIST, with the heading "CTS".

The format of the CTSBILD is as follows:

```
SES.CTSBILD [ batch ]
```

batch : Run CTSBILD in BATCH mode. The default is to run it locally.

2.6.4 DSBILD PROCEDURE DESCRIPTION

DSBILD is an SES procedure file which builds the dual state binaries XDSTVE, XRUNVE, and XTRMVE. All assembly and ISWL compilation listings are put on the direct access text library file DSLIST (one listing per record, each headed by the corresponding MADIFY deckname) and the three load maps are saved on the direct access file DSMAP.

The format of the DSBILD is as follows:

```
SES.DSBILD  [ batch ]
```

batch : Run DSBILD in BATCH mode. The default is to run it locally.

2.6.5 PPBILD PROCEDURE DESCRIPTION

PPBILD is an SES procedure file which builds the dual state PP binaries for MMCPP and PPHELP, which will be put on the direct access files XMMCPP and XPPHELP respectively. Both assembly listings are put on the direct access text library file DSLIST (one listing per record, each headed by the corresponding MADIFY deck name).

The format of the PPBILD is as follows:

SES.PPBILD [batch]

batch : Run PPBILD in BATCH mode. The default is to run it locally.

3.0 DUAL STATE INSTALLATION SEQUENCE

This section describes how to install all of the files needed to run NOS/VE in Dual State mode. To do this from "scratch" the following materials are necessary:

1 CMSE tape
1 CTI tape
1 EI tape
1 Dual State NOS Deadstart tape
- The LOADPF tape(s) which contain the NOS/VE environment

If CMSE, CTI, and EI are already present and correct, then it is only necessary to install a new deadstart sector on disk or to load a new NOS/VE environment. If A170 NOS has been run on this machine prior to the installation of NOS/VE, then chances are excellent that the proper versions of CMSE and CTI were used. There are incompatibilities between the A170 EI/NOS system and the Dual State environment which preclude the usage of these same materials, however.

3.1 CLEAR POINTERS AND INSTALL CTI

To clear the pointers, deadstart from the CTI tape which is MT, D=800, F=SI, LB=KU, and enter:

```
*U*--->R (Release)
Channel=xx
Eq=xx
Unit=xx
```

Deadstart again and then enter:

```
*U*--->I (Install D/S module on disk)
Channel=xx
Eq=xx
Unit=xx
```

3.2 INSTALL_CMSE

To install CMSE, deadstart from the CMSE tape which is MT, D=800, F=SI, and then enter:

```
DT=2
CHANNEL=xx (CMSE tape)
TDX
     'C' option (Build directory in CM and copy to disk)
     Device type=x
     Channel=xx
     Eq=xx
     Unit=xx
     MT type=3
     MT channel=xx
     Eq=xx
     Unit=xx
     User type=02 (Shared O/S)
     Install options=02
     Options=01 (Build without O/S sector since it was          :
                 already created by CTI.)                       :
```

3.3 DELETE_OLD_EI

To delete the old Error Interface (EI) file, deadstart from the unit where CMSE is installed, then enter:

```
*M*
```

Then to display the list of binary files and check if EI is there, enter:

```
AF.

*DP EI (will delete EI if it is there)

opt: *WK INSTRD 0 C00 (to write microcode from P2 to CMSE
                       disk - takes a long time)
```

3.4 INSTALL_EI

To install EI mount the EI tape which is MT, D=800, F=SI, LB=KU, and deadstart from the unit where CMSE is installed. Then enter:

```
*M*
TDX
```

```
        {CR} (A directory for CMSE already exists, so this
              will update the directory.)
    Disk Type=xx
    Channel=xx
    Eq=xx
    Device Type=(EI tape)
    MT type=xx
    Channel=xx
    Eq=xx
    Unit=xx
    User type=02
    User Options=02
    Options=02 (Add a new program)
```

When END appears, EI is installed.

3.5 INSTALL SYSTEM

Deadstart from the NOS Dual State deadstart tape, using the deadstart tape which is to be installed. Choose the 'O' option on the first display, for operator intervention. Then choose the 'H' option on the next display to see the hardware parameters. Enter CM=10000. Optionally, the 'P' display may be selected to choose a CMRDECK. (CMRDC14 contains the CANCDD S2 configuration, CMRDCK6 contains the ARHOPS S2 configuration.) After the system is deadstarted, enter the following commands:

```
X.DIS.
  COMMON,SYSTEM.
  INSTALL,SYSTEM,EQxx.
```

NOTE: xx is the EST ordinal of the disk where the deadstart sector is to be installed; this is the same disk where CMSE was installed previously.

3.6 LOADPF FILES

The LOADPF tapes, which are NT, D=PE, F=SI, and LB=KL, contain the NOS/VE operating system source and binaries, tools to assemble and link the operating system, and various other files. Included in these files is the DSINSTF file which contains the Dual State execution environment.

Deadstart from the disk upon which the NOS Dual State system was installed, LOADPF the files to the desired user number, and install the Dual State execution environment in the following manner:

```
ATTACH,DSINSTF.
BEGIN,DSINSTF,DSINST,SCAT=cat,INST=FULL.
```

where: cat is the user number of the catalog in which the execution environment is being installed. (In ARHOPS this is the DEV1 catalog.)

Check the indirect access file CMDS1 to make sure it reflects the hardware configuration:

```
*RELOADCH=x.  x should be an empty channel.
*DISKCH=y.  NOS  cannot use channel y with Dual State active.
*SYST,DISKUz=ON.  z can only be 0 through 3.
                    (Unit z on channel y cannot appear
                      in the NOS CMRDECK or EST.)
```

3.7 BRING UP DUAL STATE

After the NOS Dual State system has been deadstarted, and the Dual State execution environment has been installed, NOS/VE is brought up by entering the following console command:

```
X.UPMYVE(CAT=c)
```

where: c is the user number in which the execution environment was installed. Enter K,n. (n = the control point number of the UPMYVE job) to see the NOS/VE display. K.*BYEVE. will terminate NOS/VE. For further information regarding the operation and execution of this environment refer to the S2 Machine Usage Document.

4.0 NOS/VE HARDWARE REGRESSION TESTING

4.0 NOS/VE HARDWARE REGRESSION TESTING

4.1 INTRODUCTION

The verification currently performed on NOS/VE systems consists of the following:

1) running the simulator test input contained on file VQLTST3 until visual examination of the output from this test warrants that further testing should be performed, and
2) running the S2 Regression Test Sequence, as outlined in the following sections, on the hardware.

4.2 S2 REGRESSION TESTS

4.2.1 REGTEST

REGTEST is a file containing a sequence of HCS commands. It runs a series of user test programs on the hardware, and puts quite a heavy load on the system. REGTEST takes approximately 20 minutes to run. The command sequence follows:

```
EX UUTL *BULK,2*
PFSTATS
EX UUTL *CALLER,UUTL,10,**TESTMEM,1000000***
EX SORT *2000,100*
EX UUTL *CALLER,SORT,5,*1000,50***
EX UUTL *TESTMOVE,1000000*
EX UUTL *A170,10*
EX UUTL *LOOP,30000*
TMTERM UUTL
EX UUTL *CALLER,UUTL,10,**CYCLE,30000***
TMTERM UUTL
EX UUTL *CALLER,UUTL,5,**LOOP,1800***
EX UUTL *CALLER,UUTL,5,**TIMEOUT,100,1800***
/SSET QUANTUM 5000
/SSET PITVAL 1000
EX UUTL *CALLER,UUTL,5,**BULKNTC,2***
```

4.0 NOS/VE HARDWARE REGRESSION TESTING
4.2.1 REGTEST

```
EX UUTL *CALLER,SORT,5,**1000,50***
EX UUTL *TESTMOVE,1000000*
TSTATUS
JMEXIT
```

4.2.2 TESTBAM

TESTBAM is a file containing the statements necessary to execute all of the BAM test cases supplied by W. V. Mahal. These procedures exercise various portions of the basic access method, and are used to show some level of confidence that BAM works as well as it has previously. The command sequence follows:

```
EX BAMTEST
TES1
TES2
TES3
TES4
TES5
TES6
TES7
TES8
TES9
TES10
TES11
TES13
TES14
TES15
TES16
TES20
BAMSTOP
JMEXIT
```

4.2.3 JOB1

JOB1 is a file containing the NOS/VE commands which stage a CI object file from the 170 side to the 180 side, convert this file to an II library file, and replace the II library on the 170 side. It tests the following NOS/VE features:

- LINK_USER command
- GETPF B60
- CITOII conversion
- Object Library Generator
- Display Library Information
- REPLACE B56

```
        JMEXIT
```

The command sequence follows:

```
        EX ZDIS,,A
        SCL
        LIU,USER=(INT1,NVE),PA=INT1X,A=NOTUSED,PR=NOTUSED
        HCS
        GET,CITEXT180,CYBILGO,,,NVE,B60
        EX CITOII *CITEXT180,IITEXT180*
        EX COL
        ADD,OF=IITEXT180
        DILIB,ON=A
        GEN,LIBRARY=LIBRARY180
        END
        REPLACE,LIBRARY180,CYBIILB,,,NVE,B56
        JMEXIT
```

4.2.4 JOB2

JOB2 is a file containing the NOS/VE commands which stage an II library and a CI user job object file from the 170 side to the 180 side, convert the CI user job object file to an II object file, and then load and execute this user job with the library. It then stages the LOADMAP back to the 170 side to be printed. JOB2 tests the following NOS/VE features:

- LINK_USER command
- SET_OBJECT_LIBRARY (SOL) command
- SET_PROGRAM_OPTIONS (SPO) command
- GETPF B56
- GETPF B60
- CITOII conversion
- Load/Execute User Program + Library
- JMROUTE C180 print file
- JMEXIT

The command sequence follows:

```
        EX ZDIS,,A
        SCL
        LIU,USER=(INT1,NVE),PA=INT1X,A=NOTUSED,PR=NOTUSED
        SOL,ADD=NEWLIBRARY
        SPO,MO=(B,E,X,S),EA=FATAL
        HCS
        GET,NEWLIBRARY,CYBIILB,,,NVE,B56
        GET,XUSORT,XUSORT,,,NVE,B60
```

```
EX CITOII *XUSORT,LGO*
EX LGO
JMROUTE,NOTUSED,LOADMAP,PR,REMOTE
SCL
SOL,DELETE=ALL
HCS
GET,CYBILIB,CYBIILB,,,NVE,B56
EX LGO
JMROUTE,NOTUSED,LOADMAP,PR,REMOTE
JMEXIT
```

4.3 S2 REGRESSION TEST SEQUENCE

1) Mount S2 System Scratch pack on 844 Unit 1.

2) Initialize memory:
   - Set the deadstart panel to disk deadstart from:
     CH=1
     UNIT=43
     WORD 13=0106
     WORD 16=0000
   - Push deadstart button.
   - Hit carriage return.
     System should go through a very lengthy "CHECK COMPUTER MEMORY" step and then stop and display the CMR deck.
   - Reset panel to desired setting and re-deadstart.

3) Deadstart A170 NOS.

4) If necessary, update the INT2 catalog and load the latest system files into the INT1 catalog:
   - Mount the INT1 catalog DUMPPF tape on Unit 0.
   - X.DIS.
     USER,INT1,INT1X.
     CALL,UPCATS.
     DROP.

   The UPCATS procedure performs the following functions:
   a) Updates the INT2 catalog with the following files from the INT1 catalog:
      - the NOS/VE deadstart file (TPXXXK)
      - the "fast files" for NOS/VE deadstart (CMIMAGE, PPIMAGE, RGIMAGE)
      - the command files needed for NOS/VE deadstart (CMDS1, CMDS2, CMDS3)
      - the Remote Host binaries to be SYSEDIT*ed into the system at the current level (RHLQEP, RHLQSO, RHLQOQ, RHLPFP, RHLPSO)

```
        - the procedure to SYSEDIT the Remote Host
          binaries (RHPROCS)
        - the current CYBIL compiler (CYBILI)
        - the CYBIL II object library (CYBIILB)
        - the CI object file used to create CYBIILB
          (CYBILGO)
        - OSLPI
     b) LOADPF's the latest system into the INT1 catalog
        from the DUMPPF tape mounted on Unit 0
        (NT,D=PE,F=SI,LB=KL).
     c) Purges the old "fast files" from the last INT1
        system.
     d) SYSEDIT's the newest level of Remote Host into the
        system.

5)   Bring up dual state:
        X.UPMYVE(CAT=INT1)
        K,n.  (where "n" is the UPMYVE control point)

6)   Test if paging I/O is working:
        K.DECLARE P POINTER.
        K.SMOPEN P.
        K.CM P *1234*.
        K.MMWMP P.
        -> Disk unit light should flash on 844 Unit 1.
        K.DM P 100.
        -> If system is hung at this point then paging is
        not working.
        K.SMCLOSE P.

7)   Bring up A170 Remote Host:
        X.IRHF170.

8)   Bring up C180 Remote Host:
        K.EX RHINPUT,,A.
        K.EX RHOUTQ8,,A.

9)   Test input file route / job exit / output file route:
        X.DIS.
        USER,INT1,INT1X.
        GET,REGTEST,TESTBAM,JOB1,JOB2.
        ROUTE,REGTEST,DC=LP,FC=RH.
        ROUTE,TESTBAM,DC=LP,FC=RH.
        ROUTE,JOB1,DC=LP,FC=RH.

     NOTE: JOB2 uses the output of JOB1 in its execution;
     hence JOB2 cannot be ROUTE'd until JOB1 finishes. To
     determine when JOB1 is finished:
       - Hit "*" key to return to K-display.
```

The JOB1 dayfile will be displayed as the job is running. When the system has executed the JMEXIT statement, JOB1 has finished. The JOB1 dayfile will then be staged back to the 170 side to be printed. To print files, do:

- Make sure the printer is on (i.e. the START light is lit).
- FORM32,TM.
  ON32.

(The dayfiles of the REGTEST, TESTBAM, and JOB2 jobs will also be printed when these jobs finish.)
Now JOB2 can be submitted:

- Hit "*" key to return to DIS.
- ROUTE,JOB2,DC=LP,FC=RH.
  DROP.

The JOB2 dayfile will be displayed as it is running. When JOB2 finishes (JMEXIT has been executed) do a

```
K.EX ZDIS,,A.
```

to return to the system job. When all jobs have finished executing and their dayfiles have printed - i.e. the commands

```
K.JMSTATUS REMOTE EX.  (lists jobs executing)
K.JMSTATUS REMOTE PR.  (lists jobs in print queue)
```

return the status "NO ENTRIES FOUND" - then NOS/VE and Remote Host may be terminated by doing the following:

```
n.DROP.  (where "n" is the IRHF170 control point)
K.*BYEVE.
```

10) Bring down A170 NOS:

```
AB.
CHECKPOINT SYSTEM.
E,M.  (make sure that all checkpoints complete)
STEP.
```

(1) NOS/VE Transmittal Form

Originator_____ DATE ___/___/___ Target Build _____

Code Location: (PACKed Modsets) FN=_____ UN=_____

(Decks in "GROUP" format) FN=_____ UN=_____

(2) Description File: FN=_____ UN=_____

Code Destination (if not NOSVEPL): PL= _____

New Feature [____] Resubmittal [____] Corrective Code [____]

NOS/VE [____] A170 NOS [____] Dual State [____]

PSRs Answered_____

COMMENTS TO INTEGRATION

(3) Installation procedure changes required? [____]
(3) Dependent upon other feature, fix, or tool? [____]
(3) Documentation changes required? [____]
(3) OSLPI or Internal Interface changes required? [____]

Module(s) to be recompiled _____

Has this code been tested? yes _____ (3) no _____

Notes regarding code submittal:

(1) Use right margin of form if more space needed. More forms are on FN = XMIT10 UN = DEV1.
(2) Attach copy of description file to form (both 14 7/8 by 11).
Format is: #MODSET_IDENTIFIER (or NEW_DECK_NAME) (upper case)
Descriptive text which describes code content
**DECK_MODIFIED (or NEW_DECK_NAME) (upper case)
(3) If any of the above are checked, then explain below.

Code Reviewer _____ Approval _____

B1

09/17/80

Request Number _____

SOFTWARE CHANGE REQUEST

Requestor_____ DATE ___/___/___ PSR Number _____

Software Needing Change

S/N 101 NOS System	☐	Integration Catalog	☐	Stand-alone NOS/VE	☐
Installation Procedures	☐	Product Set	☐	S2 Prototype Closed Shop	☐
SES Tools	☐	A170 NOS System	☐	Dual State System	☐

TYPE OF PROBLEM BEING FIXED

Critical Fix	☐	Major PSR	☐	Other	☐

DESCRIPTION OF CHANGE

Modset Identifier(s)_____

Deck(s) Modified (include all common decks) _____

DESCRIPTION OF PROBLEM BEING FIXED

Approved By _____

INSTRUCTIONS

The purpose of this form is to initiate a mini-build of the requested changes into the current build level. The scope of the changes requested and the impacts of making these changes must be adequately described. This is the only way to make changes to a system after the feature code cutoff for that build has occurred.

Requestor: Name of the person submitting the change.
Date: Date of the request.
PSR Number: Number of PSR being fixed.

Check those which apply
S/N 101 System: Change affects the closed shop NOS system.
Integration Catalog: A correction is necessary to the INT1, INT2, DEV1, or A170INT catalog(s).
Stand-alone NOS/VE: Change affects only the Virtual State execution of NOS/VE.
Installation Procedures: A correction to the existing Installation Procedures is necessary (ie. such as moving a module to a different library)
Product Set: Change affects an assembler or compiler.
S2 Prototype Closed Shop: Change is necessary to the system(s) provided in the S2 lab.
SES Tools: Some change is necessary to the SES tools used to generate systems (ie. Linker, Loader, Simulator, etc.).
A170 NOS System: A change is required to the A170 Version of the NOS operating system.
Dual State System: Change affects the Dual State software.

Critical Fix: Fixes a problem which cannot be avoided either operationally, or programmed around.
Major PSR: A serious problem for which a PSR exists and is a considerable nuisance to system users.
Other: Fixes to nuisance, or time wasting problems.

Modset Identifiers: Name of the modset(s) which need to be added to the affected software.

Decks Modified: Name(s) of module(s) which require recompilation or assembly as a result of this change.

Description of Problem: A description of the severity of the problem being fixed, and the scope of the changes caused by integrating this change.

Approval: Authorization signature for the change, currently requires T.C. McGee approval for NOS/VE, and J.M. Graffius approval for A170 NOS.

C1

Advanced Systems Integration Build Activity Matrix

09/17/80

NOS/VE Build	NOS A170 Build	S/Bld Cy; Freeze Interfaces	Start Build	Feature Code Cutoff	PSR Code Cutoff	Complete Build	Xmit to SVLOPS, TTOFAC	Return from SVLOPS	Complete BCR
			(O/S)			(O/S)			
	4	–	4/25	–	4/25	5/1	5/8	6/2	6/4
I		6/3	6/10	6/24	7/4	7/9	7/14	–	7/14
			(O/S)			(O/S)			
	5	–	6/16	–	6/16	6/20	6/23	7/10	7/14
			(O/S)			(O/S)			
	6	–	7/14	–	7/14	7/18	7/21	8/7	8/12
J		7/16	7/23	8/6	8/18	8/21	8/26	–	8/26
			(O/S)			(O/S)			
	6a	–	8/18	–	8/18	8/22	–	–	8/27
K		8/28	9/4	9/18	9/29	10/2	10/7	–	10/7
L		10/9	10/16	10/30	11/3	11/5	11/11	–	11/11
M		11/12	11/19	12/2	12/12	12/16	12/19	–	12/19

Files maintained by Integration

09/17/80

Source Files

USER NUMBERS	FILENAME(S)	FUNCTION	VERSION/FREQUENCY OF UPDATE
INT1 DEV1	NOSVEPL	MADIFY program library of Virtual State code	Matches the level of system binaries contained in same catalog. Updated on periodic scheduled basis.
INT1 DEV1	OSLPI	MADIFY program library of NOS/VE Program Interface decks.	Matches the level of system binaries contained in same catalog. Updated once for each build cycle.
INT1 DEV1	VE170PL	MADIFY program library of NOS code which supports NOS/VE.	Matches the level of system binaries contained in same catalog. Updated on periodic scheduled basis.
A170INT LIBRARY	A170OPL OPL	MODIFY program library which matches NOS system level for S2. (Installed on FMD unit 43).	Updated on a scheduled basis. (CPUMTR which supports NOS/VE is on VE170PL and not on this PL).
INT2/INT1 DEV1	PROCLIB	Command Language Procedure Library (Documented in Integration Proced- ures Notebook).	Matches the level of system binaries contained in the same catalog, and accesses the appropriate build tool versions. Scheduled updates.
INT1 DEV1	NOSLIST	Contains compilation/ assembly listings of all Virtual State code. Accessed via LISTNVE procedure.	Matches the level of system binaries contained in the same catalog.
INT1 DEV1	MTRLCB,EIELCB, EILCB,STSLCB LIBLCB	Linker directives files for monitor, error interface, task services, and user modules respectively.	Matches the level of system binaries contained in the same catalog. EI is built using the BLDEI procedure, while NOSBILD is used for EIE.
INT1 DEV1	NEWDKPL	Meaningless Madify program library which users may substitute for as an alternate	Never, disappears when SCU conversion is complete.

		base when using Integration compilation procedures.	
INT2/INT1 DEV1	MAP_offset_K	Contains link map of Dual State system created, where MAPOK is a standalone NOS/VE system and MAP256K is a 256K NOS Dual State system.	Each Velink of a Dual State system.
INT1 DEV1	VQLTST3	Contains simulator commands for a batch mode test of the NOS/VE system.	As required by system content changes.
INT1 DEV1	LDR_offset_K	Contains VE generator directives for Dual State offset loads.	As required by system content or structure changes.
DEV1	KEYDESC	Contains Keypoint descriptions for the Keypoint report program XXM7KEY.	Non-standard, updated upon development's request.

Files maintained by Integration

09/17/80

Object Text Files

INT1 DEV1	XLMMTR	Object text file of modules which execute in monitor mode.	Each recompilation of a monitor mode module.
INT1 DEV1	XLJ11F,XLJ12F XLJ13F,XLJ1FF	Object text files of task services modules which run in job mode with ring attributes 11F,12F,13F and 1FF respectively.	Each recompilation of a task services module within these libraries.
INT1 DEV1	XOLG,XSCL, XLLMCII	Object text files of the Object Library Generator, System Command Language, and Object Text converter.	Each recompilation of these utilities.
INT1 DEV1	XUCNTL,XUTEST, XUSORT,XUUXER1, XUUTL,XUVLEX, XUVLEX2	Object text files of user test programs added to the system.	Each recompilation of these tests
INT1 DEV1	MTRXHDR, STSXHDR, EIEXHDR,	Header files which name the monitor, task services, and error interface segment files, produced by VELINK.	Each Velink of the system.
INT1 DEV1	MTRXOST, STSXOST, EIEXOST	Outboard symbol table files for monitor, task services, and error interface produced by VELINK.	Each Velink of the system.
INT1 DEV1	STSX101 thru STSX118	The task services segment files produced by VELINK.	Each Velink of the system.
INT1 DEV1	MTRX101 thru MTRX105	The monitor segment files produced by	Each Velink of the system.

Files maintained by Integration

Object Text Files

		VELINK.	
INT1 DEV1	EIEX101 thru EIEX102	The error interface segment files produced by VELINK.	Each Velink of a Dual State system.
INT1 DEV1	XCPUMTR	A170 NOS CPU Monitor module binaries.	Each recompilation due to modset corrections or changes to the base A170 NOS level system.
INT2/INT1 DEV1	LGB_offset_K	The Virtual Environment file produced by VEGEN.	Each VELINK/VEGEN of the system.
INT2/INT1 DEV1	CKPT	The checkpoint file from the last simulation run as a result of running the VQLTST3 test commands.	Each batch mode simulation of NOS/VE.
INT2/INT1 DEV1	CKPT0 thru CKPT2	The simulated disk files produced during the batch mode simulation run.	Each batch mode simulation of NOS/VE.
A170INT	NOSTEXT	A170 NOS system text for current NOS version.	Each A170 NOS update.
DEV1	XXM7KEY	Program to report NOS/VE Keypoints encountered during a simulation run.	Non-standard ISWL utility.
DEV1	XXM7DSI	Standalone version of NOS/VE deadstart file generator.	Non-standard, unsupported.
DEV1	XXDSGEN	Dual State deadstart file generator.	Upon demand.
DEV1	XIDST,XIDSK, XIDSP,XIHLP, XIRES	CYBER 180 PPU programs.	Upon demand.
INT2/INT1	TP_offset_K	Dual State	Each time a new

Object Text Files

DEV1		deadstart file created by the NOSSYS procedure.	deadstart file is generated (upon demand).

This paper represents the beginning of a 'helpful hints on how to do your job's document. Some of the areas discussed are still incomplete. I will be periodically adding information to the document. If you have any qustions or suggestions, please see Tom McGee. Appendix A lists background documents and how to obtain them.

Update History

Date	Changes
2/8/80	Section 3.0 revised
2/12/80	Appendix A Section 3.3 corrected
2/13/80	Section 1.0 revised
9/11/80	Section 3.5.2.8 revised, Section 3.5.2.8.4 added

E1.0 DOCUMENTATION FOR RUNNING ON S2

E1.0 DOCUMENTATION FOR RUNNING ON S2

E1.1 STANDALONE NOS/VE

Topic	Document
1. Standalone NOS/VE HW/SW Configuration	On bulletin board in S2 area
2. Integration Procedures Notebook Build Procedure	NOS/VE User's Guide - Appendix A
3. Deadstart Panel Settings (Real Panel, CMSE Emulated Panel)	On deadstart panel
4. Standalone Deadstart Procedure	JFS Paper - Section 2.0
5. Standalone CC545 Operator Console Interface	NOS/VE User's Guide - Appendix A
6. Debug Mode Interface	NOS/VE User's Guide Appendix A
7. Standalone Dump to NOS/VE Printer	NOS/VE User's Guide Appendix A
8. How to Analyze a Standalone NOS/VE Dump	
9. What To Do If:	
o NOS/VE abort	
o NOS/VE hung	
o Can't deadstart	JFS Paper - Section 2.0
o HW errors	JFS Paper - Section 2.0
o Unrecovered disk errors	JFS Paper - Section 2.0
etc.	
10. Detailed Documentation for EC Register (Machine Dependent)	
11. Patching Memory	

E1.2 DUAL STATE NOS AND NOS/VE

1. Dual State NOS and NOS/VE HW/SW Configuration	On bulletin board in S2 area
2. Dual State NOS/VE Deadstart File Building Procedure	Integration Procedures Notebook
3. Deadstart Panel Settings (Tape and Disk)	On deadstart panel
4. NOS A170 Deadstart Procedures	KMJ Paper - Section 3.1

E1.0 DOCUMENTATION FOR RUNNING ON S2
E1.2 DUAL STATE NOS AND NOS/VE

5.	NOS Operator Command & Display Interface	NOS Operator's Guide - Appendix A
6.	NOS DIS Interface	NOS Operator's Guide Appendix A
7.	NOS 026 Interface	NOS Operator's Guide Appendix A
8.	CTS Interface (Simulated TELEX ASCII from Operator Console)	Terminal
9.	Dual State Initialization (UPVE), Operation (RUNVE) and Termination (TRMVE)	WHB Paper - Section 3.3, 3.5
10.	UPVE Command File Format (Configuration, Memory Patches, NOS/VE Commands)	WHB Paper
11.	MCU (DKD) Commands and Displays	WHB Paper
12.	K Display Operation of NOS/VE	WHB Paper - Section 3.4
13.	NOS Error Messages	NOS Diagnostic Handbook Appendix A
14.	LOADPF/DUMPF to Move PF's Between SN101 and S2 (Non standard)	KMJ Paper-Section 3.1
15.	How to Analyze a Dual State Dump	
16.	What To Do If: o NOS abort o NOS hung o NOS unrecovered disk errors o NOS detected HW errors o Screens blank o NOS/VE abort o Can't deadstart NOS	
17.	CMSE/CTI Disk Pack Build Procedures	
18.	Installation Procedures for New EI	Integration Procedures Notebook
19.	Installation Procedures for New NOS Dual State Stuff (SYSEDIT UPVE and Friends)	Integration Procedures Notebook
20.	Debug Mode Interface	NOS/VE User's Guide - Appendix A
21.	MODVAL - Add New Users or Change Validation	KMJ Paper - Section 3.1

E1.3 BOTH STANDALONE AND DUAL STATE

1.	General Use of CMSE (ODDD, WK, DP, etc.)	
2.	EDD (CTI) Dead Dump Procedure	DAH Paper - Section 3.6
3.	DSDI Commands and Procedures for EDD Dump Interpretation	DAH Paper - Section 3.6
4.	Reload/Restart Microcode Procedure	

E1.0 DOCUMENTATION FOR RUNNING ON S2
E1.3 BOTH STANDALONE AND DUAL STATE

5.	NOS/VE Error Message List	NOS/VE User's Guide Appendix A
6.	Keypoint Usage and Trace Analysis Simulator keypoints	NOS/VE User's Guide Appendix A Integration Procedures Notebook
7.	Dead Dump to Printer Procedure Using ODDD	
8.	Use of Trace Information Currently Kept by NOS/VE CPMTR	
9.	Error Halt Addresses	

E1.4 OTHER

1.	Things "not to do"	
2.	Phone #s on machine, CE's office, etc.	
3.	Door lock combination #253	
4.	Location of tapes and disks	MDC Paper-Section 4.0
5.	Who to call for assistance	

E2.0 STANDALONE NOS/VE DEADSTART

E2.0 STANDALONE NOS/VE DEADSTART

E2.1 GENERAL PROCEDURE FOR DEADSTART (OVERVIEW)

1. Mount deadstart tape
* 2. make sure microcode (firmware) is loaded into the P2, tape controller and disk controller.
3. Mount scratch pack(s) on HCS disk drive(s).
4. Press deadstart button.
** 5. Make sure P2 DEC register and M2 BR (bounds register) are set correctly.
6. Start CPU running
7a. Verify proccessor is running

OR

7b. Take dump of memory.

* usually not necessary
** not necessary except on 1st deadstart

E2.2 DETAILED PROCEDURE FOR STAND-ALONE DEADSTART

1. Mount deadstart tape
   - o mount on UNIT 0 (other units can be used if location 5 of the deadstart program is set appropriately. See step 5).

2. Set up the deadstart panel for CTI deadstart from disk._
   - o use the 'DISK deadstart' switch setting. Switch settings are pasted to deadstart panel.
   - o the channel-Eq-Unit setting of the deadstart program must reflect the location of the CE pack referred to as 'ARH Regression' disk. The disk should be mounted on Unit 4.

3. Mount the scratch pack(s) on the NOS-VE disk drive(s).
   - o currently the OS is set up to run with 1-double density disk on unit 1. This can be changed during deadstart
   - o use the pack labeled 'HCS scratch'

4. Make sure microcode is loaded in the P2.
   - o (DS) Press deadstart button. You should see a display that

E2.0 STANDALONE NOS/VE DEADSTART
E2.2 DETAILED PROCEDURE FOR STAND-ALONE DEADSTART

says:

```
+----------------------+
| (CR) - OS load       | If not, check deadstart
| D - Deadstart auto   | panel, disk unit.  Try
| U - Util             | long D.S.  seq.  Reconf
| M - Maint            | PP's.  Load disk cntrl
+----------------------+ cntrlware.
```

- Type 'M'. This causes another display to appear
- Type '(CR)'
- At this point it is sometimes wise to preset memory. Type KC 0,100000,eeeeeeeeeeeeeee (CR) and wait until lower left of display is static - takes 10 to 20 seconds.
- Type 'GO INSTRN (CR)'. Wait until lower left of display is static. Takes 5-10 seconds.

5. Now you are ready to deadstart NOS/VE. Subsequent deadstarts will start with this step.
   - (DS) Press deadstart button
   - Type 'M'
   - Type 'DST(CR)' deadstart panel settings appear
   - Type '5=12u'(CR). ('u' is the unit number of the deadstart tape.)
   - Type '(BKSP)' 3 times to clear the screen
   - Type '(CR)' Tape should start moving. If not,
     - check unit
     - go back to step 4
     - try another tape
     - call for help

6. Message 'PROCEED' should appear on the screen. At this point all NOSVE PP code has been loaded, processor registers are loaded and central memory is loaded.
   - If tape moves a lot but 'PROCEED' doesn't appear, call for help.
   - Type 'DR,P2(CR)'. If the value of the SIT is changing, all is well. If SIT is not changing, repeat steps 3, 4, 5. If it still isn't changing, call for help.

7. If this is the first deadstart since a non-NOSVE user used the machine, make sure the P2 EC register and the M2 BR register are set correctly.
   - Type 'DR,P2(CR)'. This brings up a display of P2 registers
   - Type 'CR,P2,EC=08FF 0000 4502 6000 (CR)'. You should see the value being displayed for the EC register change to the value specified above.

E2.0 STANDALONE NOS/VE DEADSTART
E2.2 DETAILED PROCEDURE FOR STAND-ALONE DEADSTART

- Type 'DR,M2(CR)'. This brings up a displayof M2 registers
- If any register has a non-zero value in it, set it to all zeros.

8. At this point hex patches can be entered from the console or card reader to fix software problems or to change the values of system constants. If you are lucky you can skip this step. Patches which may be necessary are:
   - CF, 2028=X000 0000 0000 0006 where X specifies the unit number of the disk(s). (X=8 for unit 0, X=4 for unit 1, X=2 for unit 2, X=1 for unit 3, X=3 for units 2 and 3, etc.)
   - CF,2020=0000 00XX 00mm 0000 where mm = amount of memory to use in hex megabytes. Default is 2. The value of the XX field must not be changed from what is initially there.

9. Type 'SS(CR)' to start the CP running

10. Type 'DD(CR)' to look at the dayfile for the system job. Values in the header should be changing.

    If the display header is changing, all is well. Go to step 11. Otherwise either hdw or sw is broken.
    Type 'DR,P2 (CCR)'.
    - If PFS, CEL0, CEL1 are non-zero, its probably a hardware problem
    - The value of P will indicate where you got to in software
    - Type 'DR,0 (CR)'. Location 0 of CM contains the XP that was executing when the processor halted. The XP dumped to loc 0 does not contain P, i.e., word 0 is A0. Look at the MCR/UCR.
    - Type 'DR,M2(CR). If UC01 or UC02 are non-zero, its probably a memory problem.

11. You are in the idle loop. Type some commands. (Same commands you type if on the simulator. Additional commands processed by the PP are also available.)

E2.3 DISK ERRORS

The system cannot handle all disk errors. If the CPU halts, you can determine if the halt was caused by a disk error by doing the following:

- display location 2008

E2.0 STANDALONE NOS/VE DEADSTART
E2.3 DISK ERRORS

- The left half of this word contains the RMA of the mtr halt msg that describes the reasons for a voluntary monitor halt. If monitor did not voluntarily halt, word 2008 will contain 000A00FF07D0000. (approx)

- Display the location pointed to by location 2008 (left).
- If the location contains
  xx 73713031 ... (ASCII for ppSQ01 - 3, where p is a descriptor for the string)
  then you halted because of a fatal disk error.
  - The cylinder, track, sector of the bad spot is in the 2nd word of the message pointed to by loc 2008. Word 2, 3 contains
    xx00 0u00 cc00 tt00
    ss00 0000 0000 0000
    where
    u = unit number tt - track
    cc = cylinder ss - sector
  - To 'down' the bad spot, after each deadstart, TYPE
    FMDOWNAU u c t 0 s
    u = unit +1 from halt msg
    c,t,s = decimal representation from halt msg. of cyl, track, sector.

E3.0 DUAL STATE NOS/VE DEADSTART

E3.0 DUAL STATE NOS/VE DEADSTART

E3.1 A170 NOS DEADSTART

1) The system is configured to run with three FMD units (41, 42, and 43). No 844 drives are needed.

2) Turn off all 844 disk drives. Leave the FMD devices on.

3) Set the D/S panel to deadstart from the primary system disk (CH=1, Unit=41 or 43. See console or bulletin board for a description of which h disk contains the primary system.
   Set the D/S panel word 14 to RPXX
   where R = D/S level (0 or 3)
   P = 1 if you wish to see the CMRDECK, P = 0 if not.
   XX = CMRDECK number
   For the first deadstart in a session use "0006"
   Set word 16 to 00001

4) Push D/S button

5) Select "O" display and then select "H" display

6) Set "CS = NO" only if you don't want microcode to be loaded. After the initial deadstart, loading microcode isn't necessary.

7) Enter a "backspace"

8) Select "P" display
   Here you may set "I=3" for recovery level deadstart
   (Use I=0 for first deadstart)
   Set "D=Y" to see CMRDECK. This may be needed if you wish to see or to change the current system configuration.

9) Enter "(CR)" and the system should deadstart. You should then enter the date and time when the system displays these requests.

E3.0 DUAL STATE NOS/VE DEADSTART
E3.2 CURRENT DUAL STATE CONFIGURATION

E3.2 CURRENT DUAL STATE CONFIGURATION

o FMD Unit 43

This unit contains the following:

- A170 NOS, CTI, CMSE, EI binaries (NOS deadstart files)
- Files associated with user number LIBRARY
- Files associated with user number SES

o FMD Unit 42

This is a scratch unit.

o FMD Unit 41

This unit contains the following:

- A170 NOS, CTI, CMSE, EI binaries (NOS deadstart file)
- Files associated with user number DEV1
- NOS/VE Development Area PL's and Member PL's
- NOS/VE Deadstart Files to be tested (saved in individual user's catalogs)

E3.3 DUAL STATE, NOS OPERATION

1) The convention used for creating user numbers on NOS/VE is as follows:

o Your user number will be your initials.
o Your password will be these 3 letters followed by the letter 'x'.
o You must see Integration to be assigned a user index

User numbers are created by executing the program "MODVAL" as follows:

o Type "X.MODVAL".
o Type "K.m." where m is the MODVAL control point
o Type "K.C,uuu." where uuu = your user number. Note that the "K." stays on the screen.
o Type "K.PW=uuuX, FUI=n." where n = your user index
o Set all other parameters to their maximum values. Do a "+" to see next page - there are 3 pages associated with a user

E3.0 DUAL STATE NOS/VE DEADSTART
E3.3 DUAL STATE, NOS OPERATION

number.
- o Type "K.END." to end creation of that user number. Another user number may now be created.
- o Type "K.END." again to exit MODVAL.

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 number on A170 NOS using "SES.LOADPF". Documentation on how to use these SES procedure 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".

E3.4 NOS/VE DEADSTART

1) 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.

CMIMAGE, PPIMAGE, RGIMAGE are "fast" files, which are built from TPXXXK the first time you deadstart NOS/VE. These files are then used on subsequent deadstart attempts. Before a new TPXXXK can be used, these "fast" files must be purged off your user number.

2) Place a SCRATCH disk on 844 drive number 1.

3) Type X.UPMYVE(CAT=uuu) where uuu is your user number.

4) The UPMYVE job will display the following:

REQUEST *K* DISPLAY on the B display

Type K,n. where n is the control point number of the UPMYVE job.

5) Type K.*PPLOAD=TRUE. if you want to load and use disk drivers. Default is no driver. This is a reminder to perform step 2. This must be entered if you wish to buid the fast files mentioned in step 1.

E3.0 DUAL STATE NOS/VE DEADSTART
E3.4 NOS/VE DEADSTART

6) Type K.*RUN. Note that the deadstart that creates the fast file will take 1-3 minutes to complete. After the fast files are built, it usually takes about 30 seconds.

E3.5 NOS/VE OPERATION

E3.5.1 COMMUNICATION WITH A NOS/VE JOB

E3.5.1.1 Job Dayfile Display

Type

K=EX ZDISB,,A.

where n is the number of the job.

To bring up the NOS/VE "B" display, enter:

K.*VEDISPLAY=CP.

To return to the NOS/VE dayfile display, enter:

K.*VEDISPLAY=DAYFILE.

E3.5.1.2 Sending Commands

Type

K.n=XXX.

where n is the job number and XXX is the NOS/VE command. NOTE: XXX cannot contain periods.

E3.5.2 USING THE REMOTE HOST

The 180 side of the remote host is included in Build G of NOS/VE and subsequent builds. To use it, NOS/VE must be up and running.

To initiate the current NOS/VE build, type X.UPMYVE(CAT=NVE).

E3.0 DUAL STATE NOS/VE DEADSTART
E3.5.2 USING THE REMOTE HOST

The 170 side of the remote host is included in the NOS 170 system at deadstart time.

E3.5.2.1 Bring Up C170 Remote Host

Type

X.IHRF170.

E3.5.2.2 Bring Up C180 Remote Host

Type

K,n. where n is the UPMYVE control point number.
K.EX RHINPUT,,A.
K.EX RHOUTQ8,,A.
RHINPUT is the 180 task which receives job input files from the 170.
RHOUTQ8 is the 180 task which sends print files to the 170.

E3.5.2.3 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.

If you are running from an interactive terminal, enter:

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

The input file which is sent to the 180 must be in Display Code (64 character set - upper case only). The job file must be a single partition 170 record contaning NOS/VE commands. Multi partition input files are not yet supported by NOS/VE so 170 data files used by the program must be obtained through the NOS/VE permanent file GET command. The last command of the job file must

be JMEXIT in order to cause the job to terminate properly. Support for ASCII job files (6/12 ASCII) will be added at a later build. The asterisk character should be used in place of the quote character to delimit parameters for the NOS/VE EX command.

E3.5.2.4 Route A Print File From C180 to C170

At NOS/VE job termination the job log (dayfile) will be automatically returned to the 170. Data written to the NOS/VE output file $OUTPUT will be overwritten by the job log so all program print output must be written to alternate files which are explicitly routed to the 170 with the JMROUTE command. NOS/VE print files must be written by BAM as 8/8 ASCII RT=W. Print files will be converted from 8/8 ASCII RT=W to Display Code (64 character set - upper case only) when they are sent to the 170. Support for ASCII print files (8/12 ASCII) will be added at a later build. All NOS/VE output files will appear in the 170 output queue (NOS H,O display) with the name IRHFxxx as a banner. In order to route a NOS/VE print file to the 170, the following command must be contained in the 180 job file or be entered from the system console via the K display:

JMROUTE,jobname,filename,PR,REMOTE

jobname - name that the print file will have in the 180 output queue.

filename - name of the local 180 file created by BAM that is to be printed.

PR - specifies that the file is a print file (must always be PR).

REMOTE - name of the 180 family for the print file (must always be REMOTE). The NOS command language must be in HCS mode (HCS command) in order to enter the JMROUTE command.

Example of JMROUTE command:

JMROUTE,LISTING,LINKMAP,PR,REMOTE.

On the C170 side, the printer must be physically and logically on. To logically turn the printer on, under DSD enter:
ON32.

FORM32,TM. (in honor of our illustrious leader, Tom McGee)

E3.5.2.5 LINK_USER Command

In order to access C170 permanent files from a C180 job, the NOS/VE job must issue a LINK_USER command. The LINK_USER command specifies the user identification on the 170 under which C170 permanent files will be accessed.

The following must be done to enter the LINK_USER command:

```
SCL.
LIU,US=(user,family), PA=password, A=account, PR=project.
HCS.
```

The SCL command puts the NOS/VE command processor into SCL mode. The HCS command puts the NOS/VE command processor into HCS mod (HCS mode is required for the GET and REPLACE commands).

The LIU (or LINK_USER) command specifies the NOS user, family, password, account and project parameters which are used by IRHF170 to create 170 jobs which access C170 permanent files. A NOS/VE job can issue only one LINK_USER command per 170 family. All LINK_USER parameters are specified with keywords and all are requied.

US (or USER) - This parameter specifies the NOS user number (catalog) and family in which the 170 permanent files reside. 'user' will be the first parameter on the 170 job's USER card and 'family' will be the second parameter. Currently the only family on the S2 Dual State system is called NVE.

PA (or PASSWORD) This parameter specifies the password that is used to login to the user number. 'password' will be the third parameter on the 170 job's USER card.

A or (ACCOUNT) This parameter specifies the account number (charge number) for the 170 job. It will be the first parameter on the 170 job's CHARGE card.

PR or (PROJECT) This parameter specifies the project number for the 170 job. It will be the second parameter on the 170 job's CHARGE card.

Note: When running on the Simulator, the LINK_USER command is not required to use the GET and REPLACE commands.

E3.0 DUAL STATE NOS/VE DEADSTART
E3.5.2.5 LINK_USER Command

Example of LINK_USER command:

```
SCL.
LIU,US=(FAB,NVE),PA=FABX,A=7136,PR=73E08802.
HCS.
```

E3.5.2.6 Get_A_170_Permanent_File_From_180

The GET command obtains a copy of a permanent file residing on the 170. The 170 permanent file can be either a direct or an indirect access permanent file. The NOS/VE command processor must be in HCS mode (HCS command issued) in order to use the GET command. All parameters on the GET command are positional. Only the 'lfn' parameter is required. A LINK_USER command must be issued (for the 170 family on which the permanent file resides) prior to issuing the GET ccmmand. The format of the GET command is:

GET,lfn,pfn,pw,un,fm,cd.

lfn (local file name) - This is the name of the local NOS/VE file to which the 170 permanent file will be transferred.

pfn (permanent file name) - This is the name of the 170 permanent file that is to be accessed. If this parameter is omitted then 'lfn' will be used for the 170 permanent file name.

pw (password) - This is the password that will be used to access the 170 permanent file if a password is required to access the file on the 170.

un (user name) - This is the user name (alternate catalog) on which the 170 permanent file resides.

fm (family) - This is the family on which the 170 permanent file resides. Currently the only 170 family on the S2 Dual State system is NVE.

ca (conversion alternatives) - This parameter specifies the type of conversion that is performed by the IRHF on files transferred from 170 to 180. If this parameter is omitted then a default of B60 will be assumed. Values for this parameter are:

B60: Basic Biary

E3.0 DUAL STATE NOS/VE DEADSTART
E3.5.2.6 Get A 170 Permanent File From 180

The full 60 bits of each 170 word are transferred to the lower 60 bits of each 64 bit 180 word. The upper 4 bits of each 64 bit 180 word are set to 0. The file is written to 180 using BAM with Block Type = System (Unblocked) and Record Type = Undefined (RT=U) so no control information is inserted in the file. The 170 logical record structure is dropped (i.e., EORs are deleted causing the logical records to be packed together.

B56: C180 Binary

The lower 56 bits (7 8 bit bytes) of each 170 word are packed into contiguous 8 bit bytes on the 180 (i.e., 7 8 bit bytes from the first 170 word and 1 8 bit byte from the second 170 word go into the first 180 word etc.). The 170 logical record structure (EORs) are dropped. The way that the 180 file which was transferred from 170 is accessed should correspond to the method used to create it on the 180 originally (assuming that the file originated on the 180).

A6: 6/12 ASCII

A170 6/12 ASCII character files (used by XEDIT and most SES utilities) are converted to 180 8/8 ASCII with Block Type = System (Unblocked) and Record Type = Variable (RT=W). The 170 logical record structure EORs are dropped.

A8: 8/12 ASCII

170 8/12 ASCII character files are converted to 180 8/8 ASCII with Block Type = System (Unblocked) and Record Type = Variable (RT=W). The 170 logical record structure (EORs) are dropped.

D64: Display Code 64 Character Set

170 Display Code character files are converted to upper case 180 8/8 ASCII with Block Type = System (Unblocked) and Record Type = Variable (RT=W). The 170 logical record structure EORs are dropped.

Example of GET command:

```
SCL.
LIU,US=(FAB,NVE),PA=FABX,A=7136,PR=73E08802.
HCS.
```

E3.0 DUAL STATE NOS/VE DEADSTART
E3.5.2.6 Get A 170 Permanent File From 180

GET,TEXT,TEXT612,,,NVE,A6.

Note: When the GET command is used on the Simulator, the file specified by the 'pfn' parameter must be a 170 file which is local to the simulator job.

E3.5.2.7 REPLACE A 170 Permanent File From 180

The REPLACE command transfers a copy of a 180 local file to a permanent file on the 170. If a permanent file of the same name does not exist for the specified user (catalog), a direct acccess permanent file is created. If a direct access permanent file of the same name already exists in the catalog and the file can be attached with write mode then the existing direct access file is overwritten with the file from the 180. If an indirect access permanent file of the same name already exists in the catalog then the indirect access file is replaced by the file from the 180. An existing indirect access file will not be changed to a direct access file if the user's indirect access file size limit is exceeded. The NOS/VE command processor must be in HCS mode (HCS command issued) in order to use the REPLACE command. All parameters on the REPLACE command are positional. Only the 'lfn' parameter is required. A LINK_USER command must be issued (for the 170 family on which the permanent file resides) prior to issuing the REPLACE command. The format of the REPLACE command is:

REPLACE,lfn,pfn,pw,un,fm,ca.

lfn (local file name) - This is the name of the local NOS/VE file which will be transferred to a permanent file on the 170.

pfn (permanent file name) - This is the name of the 170 permanent file that is to be created or replaced. If this parameter is omitted then 'lfn' will be used for the 170 permanent file name.

pw (password) - This is the password that will be associated with a newly created direct access file or which is used to gain access to an already existing direct or indirect access permanent file.

un (user name) - This is the user name (catalog) on which an existing 170 direct or indirect access file resides. This parameter is illegal if the file does not exist.

fm (family) - This is the family on which the 170 permanent

file is to reside. Currently the only 170 family on the S2 Dual State system is NVE.

ca (converion alternatives) - This parameter specifies the type of conversion that is performed by the IRHF on files transferred from 180 to 170. If this parameter is omitted then a default of B60 will be assumed. Values for this parameter are:

B60: Basic Binary

The lower 60 bits of each 64 bit 180 word are transferred to the full 60 bits of each 170 word. The upper 4 bits of each 64 bit 180 word are discarded. The 180 file which is to be transferred should be written by BAM with Block Type = System (Unblocked) and Record Type = Undefind (RT=U). The file is transferred to the 170 as a single logical record (i.e. files with multiple EORs cannot be created on the 170 from the 180).

B56: C180 Binary

Groups of 7 contiguous 8 bit bytes from the 180 will be transferred to the lower 56 bits of each 170 word (i.e. the first 7 8 bit bytes from the first 180 word go to the lower 56 bits of the first 170 word, the 8th 8 bit byte of the first 180 word and the first 6 8 bit bytes from the second 180 word go to the lower 56 bits of the second 170 word etc.). The way that the 180 file to be transferred is created does not matter because the entire structure of the 180 file is preserved on the 170. The file is transferred to the 170 as a single logical record.

A6: 6/12 ASCII

A 180 8/8 ASCII character file with Block Type = System (Unblocked) and Record Type = Variable (RT=W) is converted to a 170 6/12 ASCII file (used by XEDIT and most SES utilities). The file is transferred to the 170 as a single logical record.

A8: 8/12 ASCII

A 180 8/8 ASCII character file with Block Type = System (Unblocked) and Record Type = Variable (RT=W) is converted to a 170 8/12 ASCII file. The 170 file can be routed directly to the printer with the 170 ROUTE command with the EC=A9 parameter. The file is transferred to the 170

as a single logical record.

D64: Display Code 64 Character Set

A 180 8/8 ASCII character file with Block Type = System (Unblocked) and Record Type = Variable (RT=W) is converted to a 170 Display Code file with lower case characters mapped to upper case. ASCII special characters that do not have a Display Code equivalent are converted to Display Code blanks. The file is transferred to the 170 as a single logical record.

Example of REPLACE command:

```
SCL.
LIU,US=(FAB,NVE),PA=FABX,A=7136,PR=73E08802.
HCS.
REPLACE,MYFILE,FILEB56,,,NVE,B56.
```

Note: When the REPLACE command is used on the Simulator, the file specified by the 'pfn' parameter will become a 170 file which is local to the simulator job.

E3.5.2.8 Example Jobs Representing Phase A/B Library Creation/Modi

General Notes

- HCS command causes NOS/VE to switch to HCS command language interpreter
- SCL command causes NOS/VE to switch to SCL command language interpreter
- Comment capability within command line is not currently available
- HCS comment line format (starts with single quote) not SCL compatible
- The B60 conversion parameter on the GET command is utilized to indicate that the A170 file is to be transferred to NOS/VE without conversion
- The NOS/VE job command stream must use display code
- The B56 conversion parameter on the REPLACE command is utilized to indicate that the 64 bit oriented NOS/VE file is to be transferred to the A170 without truncation of data. 56 C180 data bits are stored in each 60 bit A170 word.
- The B56 conversion parameter on the GET command is utilized to indicate that an A170 file with 56 C180 data bits per 60 bit A170 word are to be transferred to NOS/VE as a 64 bit oriented NOS/VE file.

E3.0 DUAL STATE NOS/VE DEADSTART
E3.5.2.8 Example Jobs Representing Phase A/B Library Creation/Modi

- REPLACE cannot be used to replace an existing A170 file
- GET and REPLACE do not allow specification of alternate user names. Use of multiple LIU commands with different families for each user catalog is a way to circumvent this restriction.
- An asterisk should be temporarily used in place of the quote mark.
- An @ character should be temporarily used in place of the underline.

E3.5.2.8.1 CREATE OBJECT LIBRARY ON NOS/VE AND SAVE IT ON A170 NOS

Notes

- CLG0170 is A170 permanent file name for file containing object text (in CI data mapping) for modules to be included in the library.
- CITEXT180 is NOS/VE local file name for file containing object text (in CI data mapping) for modules to be included in the library.
- IITEXT180 is NOS/VE local file name for file containing object text (in II data mapping) for modules to be included in the library.
- LIBRARY180 is NOS/VE local file name for the library being created.
- ILIB170 is A170 permanent file name for file containing the library.

NOS/VE Job Commands

```
SCL
LIU, USER=(jln,NVE),..
     PASSWORD=jlnx,..
     ACCOUNT=notused,..
     PROJECT=notused
HCS
GET,citext180,clg0170,,,NVE,B60
EX,CITOII,'citext180,iitext180'
EX,COL
ADD,OBJECT_FILE=iitext180
GENERATE,LIBRARY=library180
END
REPLACE,library180,ilib170,,,NVE,B56
JMEXIT
```

E3.5.2.8.2 MODIFY A PREVIOUSLY SAVED OBJECT LIBRARY

Notes

E3.0 DUAL STATE NOS/VE DEADSTART
E3.5.2.8.2 MODIFY A PREVIOUSLY SAVED OBJECT LIBRARY

- o ILIB170 is A170 permanent file name for file containg the old library
- o LIBRARY180 is NOS/VE local file name for file containing the old library
- o CMOD170 is A170 permanent file name for file containing CI object text for the new module
- o NEWCIMODULE is NOS/VE local 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

NLIB170 is A170 local file name for new library

NOS/VE Job Commands

```
SCL
LIU,USR=(jin,NVE),..
     PASSWORD=jinx,..
     ACCOUNT=notused,..
     PROJECT=notused,..
HCS
GET,library180,ilib170,,,NVE,B56
GET,newcimodule,cmod170,,,NVE,B60
EX,CITOII,'newcimodule,newiimodule'
EX,COL
ADD,library=library180
REPLACE,OBJECT_FILE=newiimodule
GENERATE,LIBRARY=newlibrary
END
REPLACE,newlibrary,nlib170,,,NVE,B56
JMEXIT
```

E3.5.2.8.3 USAGE OF NOS/VE LOADER

LIMITATIONS

The NOS/VE loader is activated by the HCS version of the EXECUTE command. This causes some limitations to loader usage since the HCS version of EXECUTE does not support all of the parameters (loader options in particular) that the NOS/VE version supports. These limitations are detailed below:

- Exactly one file must be specified in the object list. This file must be an object text file (as opposed to a library file).

- Modules are loaded from libraries only to satisfy externals.

E3.0 DUAL STATE NOS/VE DEADSTART
E3.5.2.8.3 USAGE OF NOS/VE LOADER

Libraries to be used for this purpose may be specified by either the (SCL) command SET_OBJECT_LIST or directives embedded in object text being loaded. The library of task_services entry points is always used to satisfy externals.

- At least one module to be loaded must contain a CYBIL PROGRAM or the equivalent (i.e., a transfer symbol must be specified).

- Load map options B, E and S are selected by default. These selections may be changed via the (SCL) command SET_PROGRAM_OPTIONS.

- The default error action is ERROR. This selection may be changed via the (SCL) command SET_PROGRAM_OPTIONS.

- The load map is always generated on the NOS/VE file named 'loadmap'.

- The program options STACK_SIZE AND PRESET are not supported.

PROCESS

Create an object text file by compiling a program in real state. Then perform the following steps in virtual state:

- Use the (SCL) command SET_OBJECT_LIST to specify necessary libraries which are not quoted in text embedded directives.

- Acquire any necessary libraries by either:

  o Creating the library file via the object library generator
  or
  o Staging the library file from real state to vitual state using the (HCS) command GET (with B56 conversion mode specified).

- Stage the object text file from real state to virtual state using the (HCS) command GET (with B60 conversion mode specified).

- Convert the object text file from the CI data mapping to II data mapping by executing the HCS program CITOII.

- Load and execute the program via the (HCS) command EXECUTE.

- Stage the loadmap from virtual state to real state (for printing) by using either:

E3.0 DUAL STATE NOS/VE DEADSTART
E3.5.2.8.3 USAGE OF NOS/VE LOADER

- The (HCS) command REPLACE (with A6 conversion mode specified) if running on the simulator.

or

- The (HCS) command JMROUTE 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 (real state) permanent file 'citxtrs'.

```
SCL
LINK_USER,USER=(jin,NVE),PASSWORD=jinx,..
          ACCOUNT=notused,PROJECT=notused
HCS
GET,citxtvs,citxtrs,,,NVE,B60
EXECUTE,CITOII,'citxtvs,iitext'
EXECUTE,iitext,'program parameters'
JMROUTE,notused,LOADMAP,PR,REMOTE
```

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

Assumptions: the (real state) permanent file 'citxtrs' contains object text generated by the CYBIL CI compiler. The compiler modules reference procedures contained on the user library 'mylib' and the CYBIL run-time library. These libraries have been generated in virtual state and saved in real state.

```
SCL
LINK_USER,USER=(DEV1,DEV1F),PASSWORD=DEV1X..
          ACCOUNT=notused,PROJECT=notused
LINK_USER,USER=(jin,NVE),PASSWORD=jinx,..
          ACCOUNT=notused,PROJECT=notused
SET_OBJECT_LIST,ADD=mylib
SET_PROGRAM_OPTIONS,MAP_OPTIONS=(B,E,X,S)
HCS
GET,CYBILIB,CYBIILB,,,DEV1F,B56
GET,mlib,mylib,,,NVE,B56
GEt,citxtvs,citxtrs,,,NVE,B60
EXECUTE,CITOII,'citxtvs,iitext'
EXECUTE,iitext,'program parameters'
JMROUTE,notused,LOADMAP,PR,REMOTE
```

E3.0 DUAL STATE NOS/VE DEADSTART
E3.5.2.8.4 CYBIL RUNTIME

E3.5.2.8.4 CYBIL RUNTIME

The CYBIL runtime procedure can be obtained via the following:

```
GET,CYBILIB,CYBILIB,,DEV1,NVE,B56
```

E3.5.2.8.5 PERMANENT FILE PROGRAM INTERFACE BUILD J DEFICIENCIES

1) Since no validation facility exists at build J, there is no official way for a 180 user's master catalog to be created. Without a master catalog, it is impossible to use NOS/VE permanent file requests aimed at "180 side" permanent files. It is, however, possible for a 180 job to access 170 permanent files. If anyone requires access to "180 side" permanent files, a job can be provided that will create the required master catalog.

2) Permanent files on the "180 side" are only permanent until a NOS/VE deadstart.

3) Usage - selections and share-requiements may be specified on the ATTACH request but they will not be honored. Files will never appear busy and hence the hold parameter has no affect on the ATTACH or GET requests.

4) GET, SAVE and REPLACE are not completely implemented at build J. In their partial implementation, GET acts like an ATTACH (i.e., no file copy occurs); SAVE acts like a DEFINE but requires that the local file already exist; REPLACE acts like a DEFINE but requies that the local file already exist and will replace the specified cycle if it already exists. This means that a file that is already a permanent file (via DEFINE, SAVE or REPLACE) may not be saved or replaced.

5) Until build K it will not be possible to purge or replace a "180 side" permanent fie if it is attached by any job, including the one doing the purge or replace.

6) Permanent files are never purged because a retention period has expired.

7) Only cycle permission is required to replace a "180 side" permanent file.

E3.0 DUAL STATE NOS/VE DEADSTART
E3.6 NOS/VE TERMINATION

E3.6 NOS/VE TERMINATION

Type

K.*BYEVE.

E3.7 DSDI INFORMATION

To create an Express Deadstart Dump (EDD) tape:

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=12).

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 "non zero inhibits rewind" with a CR.

8) Answer "dump number" with a CR.

9) Answer "CM/(MB)" with the size of memory you want to dump (in megabytes).
   Note: This display may not appear, depending on the version of EDD being used.

10) Answer "dump controlware" with a CR.

To create a listing of the EDD tape:

1) REQUEST,DUMP,NT,D=PE,F=S,LB=KU,PO=R,VSN=your choice.

2) GET,DSDI/UN=BRH. (On S/N 101.)

E3.0 DUAL STATE NOS/VE DEADSTART
E3.7 DSDI INFORMATION

or

GET,DSDI/UN=LIBRARY. (On S2.)

3) Create DSDI directives file:

A DSDI directive file should include the following:

IOUMR.
PROMR.
MEMMR.
PRORF.
W,first_byte_address,last_byte_adress,asid. (where the first_byte_address and last_byte_address are hex byte addresses and asid is the asid of the segment to be dumped)

4) Execute DSDI:

RFL,60000.
DSDI,M,D,I="input directives file".

5) To run (after the first time):

DSDI,I=n.

(Does not read tape again.)

6) To run interactively:

Same as above, except to do W command must first do:

OUTPUT,LISTFIL.

7) 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.

E3.8 NAMIAE_INEORMAIION

Bringing NAM up.

1) To run NAMIAF, the 2550 in the northeast corner of the fishbowl is used. CLA's 10, 11, 12 and 13 in the middle cabinet must be

E3.0 DUAL STATE NOS/VE DEADSTART
E3.8 NAMIAF INFORMATION

turned off, and the CLA's in the right hand cabinet must be on. The teletype beside the 2550 must be ON and turned to LINE.

2) At the system console enter:
FNC5,7700.
2.NAM.

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

4) To send messages to all terminals enter:
2.CFO.MSG,ALL,message.

Bringing NAM down.

1) At the system console, enter:
2.CFO.DI,NE.

2) Turn the teletype by the 2550 off.

3) To bring IAF down enter:
1.STOP.

E3.9 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 (see Section 3.1.8) even if the only reason is to bring NOS down. To bring NOS down, do the following:

1) Enter:

CHE
The screen will display:
CHECKPOINT 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.

E3.0 DUAL STATE NOS/VE DEADSTART
E3.9 A170 NOS SHUTDOWN

4) Push deadstart button.

E3.10 INTERIM MEMORY LINK STORAGE MOVE CONSIDERATIONS

The following precautions must be taken when running dual state with the Interim Memory Link storage move fix (NOS/VE Build J6 and associated NOS/A170 System).

1) Drop IRHF170, PASSON and all permanent file partner jobs before doing *BYEVE.

2) Do not rollout IRHF170, PASSON or permanent file partner jobs at any time.

3) Before doing a CHECKPOINT SYSTEM, drop IRHF170, PASSON and all permanent file partner jobs.

4) If the system crashes a NOS/A170 level 3 deadstart is the preferred action. If for some reason you must do an MCU recovery (REC command) do the following:

- Clear word 17(8) via:
  99.
  17,0.
- Enter REC on the MCU console.
- Finish up in A170 only mode (i.e., do not do an UPMYVE), then do a level 3 deadstart. If you bring up NOS/VE again without doing a level 3 deadstart, the results are unpredictable.

E3.11 NOS/VE INTERACTIVE FACILITY OPERATION

E3.11.1 OPERATOR INITIATION

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

1) Bring up NOS/VE (build J7 or later).

2) Bring up NOS/A170 networks:

- FNC5,7700. (may sometimes be skipped)

E3.0 DUAL STATE NOS/VE DEADSTART
E3.11.1 OPERATOR INITIATION

- 2.NAM.

3) Bring up A170 part of interactive:

- X.PASSON (CAT=nnn)

  Where nnn is (usually) the same catalog that was specified on the X.UPMYVE in step 1.

4) When NOS/VE is up, bring up the C180 part of interactive:

- K.EX IFEXEC,,A.

5) Bring up the remote host (IRHF170,RHINPUT,RHOUT48) if desired.

E3.11.2 OPERATOR TERMINATION

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

- 2.CFO.DI,AP=TAF. (2 is the NAM control point number)

  This is the preferred method. To bring NOS/VE interactive back up, you must first do a 2.CFO.EN,AP=TAF.

- 2.CFO.DI,NE. (2 is the NAM control point number)

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

- N.DROP. (and) K.TMTERM IFEXEC. (N is the PASSON control point number)

E3.11.3 OTHER OPERATOR CAPABILITIES

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

  2.CFO.ID,AP=TAF. (2 is the NAM control point number)

- To send a message to all terminals do:

  2.CFO.MSG,ALL,mesage. (2 is the NAM control point number)

E3.0 DUAL STATE NOS/VE DEADSTART
E3.11.3 OTHER OPERATOR CAPABILITIES

- 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 control point N do:

  N.ONSWX. (N.OFFSWX.)

  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

E3.11.4 INTERACTIVE TERMINAL OPERATION

E3.11.4.1 Validation To Access NOS/VE

To access NOS/VE via the interactive facility, your user number must be validated to logon to the application named TAF. If you get the message 'ILLEGAL APPLICATION' when trying to logon, you are probably not validated to access TAF. To correct this do the following from the console:

```
X.MODVAL.

K,N.  (N is the MODVAL control point number)
K.U,ccc.  (ccc is your user number)
K.AP=ALL.
K.END.
K.END.
```

If you get the message 'CONNECTION PROHIBITED' it means that only one terminal per user number can be logged on to TAF at one time. This indicates that someone has changed the network configuration - try a different user number until the network configuration is changed.

E3.0 DUAL STATE NOS/VE DEADSTART
E3.11.4.2 Login To NOS/VE

E3.11.4.2 Login_To_NOS/VE

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

```
,user,password,TAF
```

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

When the terminal has been successfully logged in to NOS/VE, the following message will be displayed at the terminal:

```
Welcome to NOS/VE Interactive On The S2.
```

E3.11.4.3 Terminal_Usage

1) The slash (/) is a prompt to enter a NOS/VE command. Any normal NOS/VE command can now be entered. The full ASCII character set (lower or upper case and all special characters can be used). Commands do not need to be ended with a period.

2) A JMEXIT command will cause the NOS/VE Interactive Job to terminate and it's dayfile (job log) will be returned to the A170 for printing. A new NOS/VE Interactive Job can then be started by responding to the 'APPLICATION:' prompt with TAF.

3) A Terminal Break (Control T) can be used to discard output from a NOS/VE command. A Terminal Break will not terminate a NOS/VE user task (i.e., a task initiated with the EX command) or cause it's output to be discarded. An asynchronous (A parameter on EX command) NOS/VE user task can be terminated with a TMTERM XXX command (XXX is the user task name as specified on the EX command). A synchronous NOS/VE task or any NOS/VE task waiting for input from the terminal cannot be terminated either by a Terminal Break or a TMTERM command.

E3.11.4.4 NOS/VE_Program_Access_To_The_Terminal

1) Interactive NOS/VE jobs are able to obtain terminal input through the CLP$GET_STND_INP program interface which can be

E3.0 DUAL STATE NOS/VE DEADSTART
E3.11.4.4 NOS/VE Program Access To The Terminal

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) Interactive NOS/VE jobs can send output to a terminal through the CLP$PUT_STND_OUT program interface which can be used by both task services and user ring programs. This interface will cause the terminal to operate in 'no format effector' mode (i.e. pre and post print format effectors should not be used - each CLP$PUT_STND_OUT call will start a new line).

E4.0 S2 DEVELOPMENT LAB SUPPORT BY INTEGRATION

E4.0 S2 DEVELOPMENT LAB SUPPORT BY INTEGRATION

What we have established in the lab so far is the following:

- A 600 tape capacity tape rack for general use (located in near proximity to the 67X tape drives). If your project would like to reserve a section of this tape rack, contact Tim or myself.

- A tape and disk cabinet for storage of system support materials which this project will manage and keep up to date. (If you have been using this cabinet for unauthorized storage - beware. We have the key to the lock!) More will be published about the contents of this cabinet later, and a cabinet index will be posted in the lab to help locate where things are supposed to be placed within the cabinet. This cabinet is currently located against th East wall of the lab, is 6 ft. 8 in. tall, gray in color and with sliding door.

- A two-level documentation rack for system documentation listings. This contains the current build compilation listing interface deck compilation listing (from module named QLNOS), listing of the NOS/VE PPU routines, system link map, and various assorted PVE listings. This rack is next to the tape and disk cabinet at this time.

- A desk documentation rack for reference manuals and Tom McGee's collection of "how to" goodies. The objective is to have this reference information at arm's length of the console, but it is currently on top of the two-level unit by the East wall.

- At or near the console is a small notebook containing the NOS System Programmer's Instant, NOS Application Programmer's Instant, and the 180 Instruction codes.

Feel free to examine and use all of the above materials while in the lab. Do not remove or abuse any of these materials. Please notify Tim McGibbon or Mike Carter of any problems or deficiencies of these materials. Leave a note if we are not available.

E4.0 S2 DEVELOPMENT LAB SUPPORT BY INTEGRATION

APPENDIX A 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 Reference Manual - Vol 1, 60435400 - Vol 2, 60445300
- 2.4 NOS Instant
- 2.5 NOS Operators Guide - 60435600
- 2.6 NOS Diagnostic Handbook
- 2.7 NOS A170 ERS
- 2.8 NOS A170 GID - ARH3060

3.0 NOS/VE Reference Documents

- 3.1 Program Interface ERS - obtained from Karen Rubey
- 3.2 Command Interface ERS - obtained from Karen Rubey
- 3.3 NOS/VE User's Guide - obtained by the following:
  ATTACH,HUG/UN=DAH
  SES.FORMAT I=HUG,L=LIST,LOCAL,TXTFORM
  SES.PRINT LIST
- 3.4 NOS/VE Procedures and Conventions - obtained by
  SES,MAD.LISTPC

E4.0 S2 DEVELOPMENT LAB SUPPORT BY INTEGRATION

3.5 JFS Deadstart/180 Operating Procedures

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

3.7 Listing of all NOS/VE type declarations - obtained by doing, SES,DEV1.LISTNVE M=QLNOS.

3.8 NOS/VE Code Transmittal/PL Maintenance Procedures

See Integration Procedures Notebook.

3.9 NOS/VE Internal Interface Maintenance Procedures

Memo available from S.C. Wood.

3.10 Integration Procedures Notebook

Obtained by:

Acquire,IPNDOC2/UN=MDC. SES.PRINT,IPNDOC2.

4.0 Tools Reference Documents

4.1 PASCAL-X Interactive Debugger - ARH3142

4.2 SES User's Guide - ARH1833

4.3 PASCAL-X 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/O ERS - ARH3125

4.9 Object Code Utilities ERS - ARH2922

4.10 CYBIL Implementation Dependent Handbook - ARH3078

E4.0 S2 DEVELOPMENT LAB SUPPORT BY INTEGRATION

5.0 Dual State Cookbook

To acquire additional copies of this document enter:

```
ACQUIRE DOCUMEN/UN=SKT512
SES.FORMAT I=DOCUMEN TXTFORM.
```

F1.0 INTRODUCTION

F1.0 INTRODUCTION

F1.1 PURPOSE

The purpose of this document is to give an overview of the NOS/VE system (formerly called HCS) from the following perspectives:

- Adding user tasks (tests)
- Modifying NOS/VE components
- Adding new NOS/VE components
- System usage - hardware and simulator
- Hints

F2.0 ADDING USER TASKS TO NOS/VE

F2.0 ADDING USER TASKS TO NOS/VE

F2.0.1 INTRODUCTION

A user task can be defined as a group of modules linked together that will execute in the 'user ring' of NOS/VE, currently ring 11. This task may make calls to any gated entries within task services (rings 1 through 3) if the call bracket will allow the call. Data defined within task services may not be referenced from rings 4-15.

F2.0.2 USING THE VE LINKER

The general format of the LINK command is:

```
SES.VELINK LFL=CYBILIB LPF=LIBLCB OFL=LGO NS=LIBX
```

The LPF parameter specifies the file containing Virtual Environment Linker variables that control the linkage. If the LPF parameter is not specified, these variables default to values provided by the VELINK procedure. The values for both the LFL and OFL parameters may be anything the user requires - it is these parameter that define the makeup of a given user task. The VE Linker ERS should be consulted for a detailed description of the available parameters.

Every LINK command creates one unique user task. The value for the NS parameter must be unique among all user tasks in a given virtual environment build. The value given must be 4 characters and cannot be either MTRX or STSX, as these values are used for monitor and task services.

The following example should help to clarify how to make the modifications. Suppose we want to create two user tasks. The first requires object files A and B from the current users catalog and file CC from catalog INT2. The second task requires object files D and E from the current catalog and library file L1. The necessary commands are:

```
ACQUIRE,CC/UN=INT2
```

F2.0 ADDING USER TASKS TO NOS/VE
F2.0.2 USING THE VE LINKER

```
SES.VELINK LFL=CYBILIB LPF=LIBLCB OFL=(A,B,CC) NS=LIBX
SES.VELINK LFL=(L1,CYBILIB) LPF=LIBLCB OFL=(D,E) ..
NS=LIBZ
```

There are five things to note about this example:

1) The use of multiple values with the LFL and OFL parameters (up to 10).
2) The fact that local files are referenced first by the linker before they are searched for in the user's catalog.
3) The use of a continuation (..) card.
4) The unique NS values LIBX and LIBZ.
5) The assumption that CYBILIB exists in the current catalog, or is already local to the job. If neither of these cases is true, then CYBILIB must be ACQUIRE'd from the catalog which contains the desired version. The same assumption holds for LIBLCB.

The changes to be to the VELDCM file are described below. Immediately after the directives:

```
LOADJOB STSX
```

directives of the format:

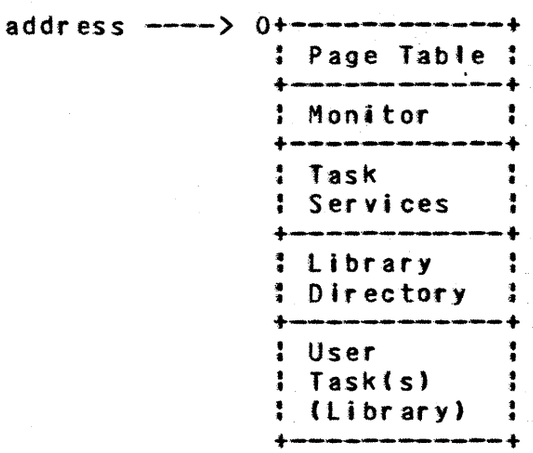
```
LOADLIB NS PNAME
```

should be placed. There should be one directive per user task (i.e., one per user task LINK command in the VELDCM file). The NS parameter value must be the same as the value specified for the NS parameter on the LINK command. The value for PNAME may be any one to eight character name and is the name of this 'program' when it resides on the NOS/VE 'library'.

It is important to note that all code and data must fit into real memory at the time of loading and deadstart. The simulator imposes a 500K (16M with next release) byte restriction on maximum size and the hardware is restricted to 2M bytes. If the memory required exceeds the default maximum of 7A000 (16) bytes, then the VELDCM file must be changed to reflect this. The size of the page table must be increased so that it has 2 to 4 times as many entries as the number of real memory pages. The page table size is changed in the VELDCM file in three different places, however, it is not just a simple substitution. Assistance should be obtained when any VELDCM file modifications are required. The following diagram shows the virtual environment after loading:

memory

F2.0 ADDING USER TASKS TO NOS/VE
F2.0.2 USING THE VE LINKER



Using the example from above, the two directives to be added to the VELDCM file are:

```
LOADLIB LIBX PROGA
LOADLIB LIBZ PROGB
```

The names PROGA and PROGB can be whatever is desired, but must be unique within a single NOS/VE build.

To execute PROGA, the following NOS/VE command is used:

```
EXEC PROGA 'string'
```

One final note about the VELDCM file. One of the last commands is a 'DM ALL' command which produces a hex dump of the virtual environment file. This command may be removed if the dump is not wanted.

When using the VE Linker specifically to produce NOS/VE systems it is recommended that the procedure NOSLINK, as described in the Advanced Systems Integration Procedures Notebook, be used to produce these systems. Use of any other procedures may lead to erroneous versions of interrelated software components.

F3.0 EXECUTION

F3.0 EXECUTION

F3.1 INTRODUCTION

NOS/VE will run on either the C180 hardware or the simulator. Any differences between the two are resolved by NOS/VE itself or by the procedures used to run it.

NOS/VE provides three different types of commands. The first type allows access to most of the software facilities within the system, such as:

- Execution Management
- Logical Name Management
- Task Management
- File Management
- Segment Management
- Memory Management
- Heap Management
- Signal Management

The second type provides a debugging capability to be used within an executing task. The features available are:

- Breakpoint
- Trace Back
- Register Manipulation
- Memory Manipulation

Both of the first two types are available on both the hardware and the simulator. The third type of command is available only on the hardware. These commands are processed by the PPU console driver, and offer the following features:

- Memory Manipulation
- Register Manipulation
- Print Memory
- OS Displays (Dayfile)

NOS/VE currently supports a single job and multiple tasks within that job. A task may be executed synchronously or

asynchronously with other tasks.

F3.2 NOS/VE COMMANDS

NOS/VE commands allow the user access to a large number of functions provided by the system. In general, any parameter to one of these commands may be either an explicit value or a logical name space (LNS) variable. One exception to this is the use of task status blocks or signal control blocks, which must always be LNS variables. A logical name space is associated with a job, a fact which must be considered when running multiple tasks.

The following types of LNS variables and parameters are available:

INTEGER
CHARACTER
NAME
BOOLEAN
VSTRING
POINTER
SIGNAL CONTROL BLOCK (temporary for HCS only)
TASK STATUS BLOCKS

Within the descriptions which follow, optional parameters are enclosed in square brackets ([]).

F3.2.1 DECLARE

This command is used to create variables within the logical name space of the current job.

Syntax: DECLARE NAME TYPE

NAME - LNS variable name, 1 to 31 characters.

TYPE - Variable type, must be one of the following:

INTEGER	-	A 64 bit integer.
BOOLEAN	-	The value TRUE or FALSE.
POINTER	-	A pointer to cell.
SCB	-	A signal control block.
VSTRING	-	A STRING (*) variable.

CHARACTER - A single character.
TSB - A task status block.

F3.2.2 REMOVE

This command is used to remove a variable definition from the logical name space of the current job.

Syntax: REMOVE NAME

NAME - LNS variable name, 1 to 31 characters.

F3.2.3 PFSTATS

This command is used to display the following page fault statistics:

avail q - Number of times a page was found in the available queue.
avail mod q - Number of times a page was found in the available/modified queue.
valid in pt - Number of times the page was found in the page table.
no memory - Number of times a page fault could not be satisfied because no real memory was available.
locked - Number of times a page fault could not be satisfied because the page frame was locked (IO was active).
on disk - Number of times a page was found on a disk.
pt full - Number of times a page fault could not be satisfied because an empty entry could not be found in the page table.
cio reject - Number of times a page fault could not be satisfied because of an I/O error.
new page - Number of times that a new page was created.

syntax: PFSTATS

F3.2.4 TSTATUS

This command is used to display the status of all currently active tasks. The following information is displayed:

- Task Name
- Execution Time Used
- Number of page faults

syntax: TSTATUS

F3.2.5 TMCYCLE

This command causes a task to give up its turn for execution until all other ready tasks have had at least one chance to execute.

syntax: TMCYCLE

F3.2.6 TMDELAY

This command causes a task to be kept from executing for a specified number of milliseconds.

syntax: TMDELAY MS

MS - Number of milliseconds to delay.

F3.2.7 TMABORT

This command causes the current task to be aborted.

syntax: TMABORT 'MES'

MES - A string to be displayed when the task is aborted.

F3.2.8 TMEXIT

This command causes the current task to terminate normally.

syntax: TMEXIT

F3.2.9 EXEC/EX

This command causes a new task to be created and executed subordinate to the current task.

syntax: EXEC PROGRAM [PARAM] [TSB]
-or-
EX PROGRAM [PARAM] [TSB]

PROGRAM - The name of the program on the system 'library' to be executed.

PARAM - A string that is passed to the program via the program header.

TSB - One of the following:

DEBUG	- Specifies that the task is to be executed by the debug processor. The task is run synchronously.
A	- Specifies that the task should be executed asynchronously, but without any task status block being used.
Other non-blank	- Specifies that a task status block variable of the name given is to be created in the current job's LNS and the new task is to be run asynchronously with the current task. The task name in this case will be the value given for this parameter. The user can determine the status of a task by printing the value of the task status block.

If the parameter is omitted, the task will be run synchronously.

F3.2.10 TMTERM

This command is used to terminate a specific task and all callees of that task.

syntax: TMTERM TASKNAME

TASKNAME - The name of the task to terminate.

F3.2.11 SMOPEN

This command causes a segment access open to be performed on a local file, or causes a transient segment to be created.

syntax: SMOPEN PVA [NAME] [SEGNUM] [R1] [R2] [ATTR]

PVA - The name of an LNS pointer variable to receive the segment pointer.

NAME - The name of the local file (1 to 8 characters) to open as a segment. If this parameter is omitted, a transient segment is created.

SEGNUM- The segment number to be assigned to the segment. If this parameter is omitted, an unused segment number will be chosen.

R1 - The R1 value for the segment. If this parameter is omitted, 11 is used.

R1 - The R2 value for the segment. If this parameter is omitted, 11 is used.

ATTR - The attributes of the segment. A legal value is any valid combination of the following letters:

R - Read
W - Write
X - Execute
B - Binding
L - Execute Local Privilege
G - Execute Global Privilege
I - Wired
K - Stack
C - Cache Bypass

S - Shared
Q - Sequential Access

The default is RW.

F3.2.12 SMCLOSE

This command causes a segment of the current task to be removed from that task's address space.

syntax: SMCLOSE PVA

PVA - A pointer to a cell. The segment represented by this pointer will be closed.

F3.2.13 SMCHANGE

This command allows some of a segments attributes to be changed after it has been created.

syntax: SMCHANGE PVA [R1] [R2] [ATTR]

PVA - Same definition as SMOPEN

R1 - Same definition as SMOPEN

R2 - Same definition as SMOPEN

ATTR - Same definition as SMOPEN.

F3.2.14 MMADVI

This command causes the system to be notified that the specified range of virtual memory should be paged in as soon as possible.

syntax: MMADVI [PVA] [LEN]

PVA - A pointer to the first byte of virtual memory to be paged in. The default is NIL.

LEN - The number of bytes to page in. The default is 1.

F3.2.15 MMADVO

This command notifies the system that the specified range of virtual memory may be paged out (removed from the working set).

syntax: MMADVO [PVA] [LEN]

PVA - Same as in MMADVI, except that memory is paged out.

LEN - Same as in MMADVI, except that memory is paged out.

F3.2.16 MMADVOI

This command performs the functions of both the MMADVO and MMADVI commands. The page out is performed first.

syntax: MMADVO [PVAO] [LENO] [PVAI] [LENI]

PVAO - Same as in MMADVO.

LENO - Same as in MMADVO.

PVAI - Same as in MMADVI.

LENI - Same as in MMADVI.

F3.2.17 MMWMP

This command is similar to the MMADVO command except that the pages are written to disk immediatly.

syntax: MMWMP [PVA] [LEN] [WAIT]

PVA - Same as MMADVO.

LEN - Same as MMADVO.

WAIT - The value TRUE if the user desires to wait until all paging I/O is complete, otherwise FALSE. The default is TRUE.

F3.2.18 MMFREE

This command causes the pages representing the specified range of virtual memory to be released.

syntax: MMFREE PVA [LEN]

PVA - same as in MMADVI.

LEN - same as in MMADVI.

F3.2.19 CONPVA

This command converts a process virtual address (PVA) to a real memory address (RMA).

syntax: CONPVA PVA [MODE]

PVA - The pointer to be converted to an RMA.

MODE - One of the following values:

DIRECT - The specified PVA is to be converted. This is the default value.

INDIR - The specified PVA is a pointer to the PVA to be converted.

F3.2.20 HPINIT

This command causes a heap to be created and initialized.

syntax: HPINIT HEAPP LEN

HEAPP - The name of an LNS pointer variable. It will be set to point to the heap.

LEN - The length of the heap in bytes.

F3.2.21 HPALLOC

This command allocates space within a previously created heap.

syntax: HPALLOC PTR LEN [PAGECROS] [ZERO] HEAPP

PTR - The name of an LNS pointer variable. It will be set to point to the allocated area.

LEN - The number of bytes to allocate.

PAGECROS- This parameter has no affect. It is included for compatibility.

ZERO - The value TRUE if the allocated area is to be preset to the value zero or FALSE if it is to be left as is. The default is FALSE.

HEAPP - A pointer to the heap in which the space is to be allocated.

F3.2.22 HPFREE

This command frees a block of space previously allocated from a heap.

syntax: HPFREE PTR HEAPP

PTR - The name of an LNS pointer variable which points to the block to be freed.

HEAPP- A pointer to the heap in which the block is allocated.

F3.2.23 SHINIT

This command is used to initialize a signal control block.

syntax: SHINIT SCB [TYPE] [VSTR]

SCB - The name of an LNS signal control block variable to be initialized.

TYPE - The signal type to be associated with the SCB. Must be one of the following:

EVENT
SEME
IORESP
MESSAGE

VSTR - If TYPE is message, then this parameter specifies a string variable whose size is the maximum message size allowed when using this SCB, and whose location is where the data will be placed.

F3.2.24 SHSEND

This command causes a send signal operation to be performed on the specified signal control block.

syntax: SHSEND SCB [INT] ['STR']

SCB - The LNS signal control block variable to which the signal is sent.

INT - Integer value to be sent with the signal

STR - A string (in quotes) to be sent as data along with the signal. If both INT and STR are specified, STR takes precedence.

F3.2.25 SHWAIT

This command causes the current task to wait until a specified amount of time has passed, or until any of up to three signals are sent.

syntax: SHWAIT [ITIME] [SCB1] [SCB2] [SCB3]

ITIME- The number of milliseconds to wait. If this parameter is omitted, infinity is used.

SCBi - Up to three LNS signal control block variables to wait for a signal on.

F3.2.26 CHANGE LNS VALUE

This command allows the value of an LNS variable to be changed. Signal control block and task status block variables cannot be changed.

syntax: LNSN = NV

LNSN - The name of the LNS variable to be changed.

NV - The new value.

F3.2.27 PRINT LNS VALUE

This command causes the value of an LNS variable to be displayed.

syntax: LNSN

LNSN - The name of the LNS variable to be displayed.

F3.2.28 ECHOINP

This command causes all command input to NOS/VE to be echoed back to the output device. This command is useful only when used as the first command to a batch mode simulation.

syntax: ECHOINP

F3.2.29 STOPSIM

This command causes NOS/VE to stop execution via a CPU halt when running on the simulator.

syntax: STOPSIM

F3.2.30 SSET

This command allows some of the NOS/VE control parameters to be changed dynamically.

syntax: SSET CPN [NV]

CPN – The name of the control parameter being changed. The value must be one of the following (entries followed by an * are not intended for general use):

- QUANTUM – Basic task time slice (microseconds) for all tasks created after the execution of this command.
- MAXIDLE* – Maximum amount of time spent in monitor idle loop before looking for lost interrupts.
- TICKTIME* – Used for paging control.
- DFDELAY* – Minimum amount of time between the issuing of dayfile messages. Used to slow down the scrolling action of the console dayfile display.
- KEYMAX – Maximum value of the id field from a keypoint that will be placed in the keypoint buffer. Any keypoint with an id field greater than this value will be ignored.
- STEPCNT* – Maximum number of monitor requests allowed before monitor goes into wait loop.
- DBRING – Lowest ring that can be executed while in debug mode.
- PQTHRESH* – Number of pages kept in the page queues.
- KM – Keypoint mask used for every task created after execution of this command.
- MM – Monitor mask used for every task created after execution of this command.
- UM – User mask used for every task created after execution of this command.

PITVAL* - The value that the PIT is reset to after every PIT interrupt.

DISDELAY - How often (milliseconds) the system status display (3 lines on the console) is updated.

NV - The new value for the specified control parameter. If NV is omitted, the current value will be displayed.

It is important to note that these commands are used primarily for hardware and monitor debugging and may change or disappear at any time.

F3.2.31 FMCREATE

This command makes a file known to the system.

syntax: FMCREATE FILENAME

FILENAME - The name of the file being created (1 to 8 characters).

F3.2.32 FMDELETE

This command deletes a file previously made known to the system with the FMCREATE command.

syntax: FMDELETE FILENAME

FILENAME - Name of the file being deleted.

F3.2.33 FMDOWNAU

This command identifies bad areas on disk and keeps them from being allocated.

syntax: FMDOWNAU UNIT CYLINDER TRACK SWLBUG SECTOR

UNIT - Unit number of the disk device.

CYLINDER - Cylinder number.

TRACK - Track number.

SWLBUG - This parameter is present because of a compiler bug.

SECTOR - Sector to be marked as bad within the specified unit/cylinder/track.

F3.3 CONSOLE COMMANDS

Commands to the system are entered via the console keyboard. With the exception of messages to the operating system, all commands entered must include a two-character command identifier or a two-character operating system display identifier. Some commands require parameters, others do not. All command input lines are restricted to 60 characters or less; all are terminated by depressing the carriage return key.

F3.3.1 DISPLAY CENTRAL MEMORY

F3.3.1.1 Display - Partial Mode

The following commands provide display of only the right-most 60 bits of central memory words (they use the 60 bit PPU cm read/write instructions).

F3.3.1.1.1 DP,<ADDRS>

Displays an installation-specified number of central memory words; two words are displayed per display line along with the byte address of the left-most word of the line.

<addrs>: A 1-8 digit hexadecimal real memory byte address which defines the first word to be displayed. The specified address is forced to zero module eight if it is not so specified by the command.

F3.3.1.1.2 DP,+

Increments the most recently specified memory address and displays a set of memory words which are contiguous with those most recently displayed. This command is used to "roll" forward through memory.

F3.3.1.1.3 DP,-

Decrements the most recently specified memory address and displays a set of memory words which are contiguous with those most recently displayed. This command is used to "roll" backward through memory.

F3.3.1.1.4 DP

This command may be used to reinstate the most recent central memory display after the screen has been used for other purposes.

F3.3.1.2 Display = Full Mode

The following commands provide display of all 64 bits of central memory words. There are a number of characteristics of these commands of which the user should be aware:

- These commands use the 64-bit central memory access mechanism.

F3.3.1.2.1 DF,<ADDRS>

Displays an installation-specified number of central memory words; two words are displayed per display line along with the byte address of the left-most word of the line.

<addrs>: A 1-8 digit hexadecimal real memory byte address which defines the first word to be displayed. The specified address is forced to zero module eight if it is not so specified by the command.

F3.3.1.2.2 DF,+

Increments the most recently specified memory address and displays a set of memory words which are contiguous with those most recently displayed. This command is used to "roll" forward through memory.

F3.3.1.2.3 DF,-

Decrements the most recently specified memory address and displays a set of memory words which are contiguous with those most recently displayed. This command is used to "roll" backward through memory.

F3.3.1.2.4 DF

This command may be used to reinstate the most recent central memory display after the screen has been used for other purposes.

F3.3.2 CHANGE CENTRAL MEMORY

F3.3.2.1 Change-Partial Mode

F3.3.2.1.1 CP,<ADDRS>=<VALUE>

This command inserts a specified value into the right-most 60 bits of a 64-bit central memory word; the left-most 4 bits of the central memory word are unconditionally set to zero.

<addrs>: A 1-8 digit hexadecimal real memory byte address which defines the central memory word which is to be modified. The specified address is forced to zero module eight if it is not so specified by the command.

<value>: A 16 digit hexadecimal value which is to be inserted into the central memory word; all 16 digits must be specified. Blank characters may separate hex digits if desired to simplify value specification; for example, the two value specifications shown below yield the same result:

```
value1 0123456789ABCDEF
value2 0123 4567 89AB CDEF
```

F3.3.2.2 Change-Full Mode

F3.3.2.2.1 CF,<ADDRS>=<VALUE>

This command inserts a specified value into the full 64 bits of a 64-bit central memory word.

<addrs>: A 1-8 digit hexadecimal real memory byte address which defines the central memory word which is to be modified. The specified address is forced to zero module eight if it is not so specified by the command.

<value>: A 16 digit hexadecimal value which is to be inserted

into the central memory word; all 16 digits must be specified. Blank characters may separate hex digits if desired to simplify value specification; for example, the two value specifications shown below yield the same result:

```
value1 0123456789ABCDEF
value2 0123 4567 89AB CDEF
```

F3.3.3 PRINT CENTRAL MEMORY

F3.3.3.1 PM,<addrs>,<words>

This command provides a listing of central memory to a line printer. Four words are listed per line along with the byte address of the left-most word of the line.

<addrs>: A 1-8 digit hexadecimal real memory byte address which defines the first central memory word to be listed. The specified address is forced to zero module eight if it is not so specified by the command.

<words>: A 1-5 digit decimal value which specifies the number of central memory words to be listed.

A listing operation may be terminated prior to its normal completion by depressing the carriage return at the keyboard.

F3.3.4 DISPLAY/CHANGE SYSTEM ELEMENT REGISTERS

F3.3.4.1 Display Element Registers

F3.3.4.1.1 DR,<ELID>

This command causes display of an installation defined set of registers of a system element.

<elid>: A two-character system element identifier which specifies the element of which registers are to be displayed. Valid system element identifiers are listed under the section entitled "System Element Identifiers". The registers displayed for each system

element are listed under the section entitled "System Element Registers".

F3.3.4.2 Change Element Registers

F3.3.4.2.1 CR,<ELID>,<REGID>=<VALUE>

This command permits modification of system element registers for which the maintenance channel has write access.

<elid>: A 2 character system element identifier which specifies the element of which a register is to be modified. Refer to the section entitled "System Element Identifiers" for a list of valid identifiers.

<regid>: A 1-4 character register identifier which specifies the register which is to be modified. Refer to the section entitled "System Element Registers" for a list of valid register identifiers for each system element.

<value>: A 16 digit hexadecimal value which is to be inserted into the register; all 16 digits must be specified. Blank characters may separate hex digits if desired.

F3.3.4.3 System Element Identifiers

Following is a list of valid system element identifiers.

- M2 Identifies the central memory element
- P2 Identifies the central processor unit

F3.3.4.4 System Element Registers

The following subsections list, according to system element, those registers which may be displayed and which may be modified (assuming that the maintenance channel has write access to the specific register).

F3.3.4.4.1 CENTRAL MEMORY REGISTERS

F3.3.4.4.1 CENTRAL MEMORY REGISTERS

REGISTER MNEMONIC	REGISTER NAME	ACCESS ATTRIBUTES
SS	Status Summary	R
EC	Environment Control	R/W
BR	Bounds Register	R/W
CEL0	Corrected Error Log, Distributor 0	R/W
UC10	Uncorrected Error Log 1, Distributor 0	R/W
UC20	Uncorrected Error Log 2, Distributor 0	R/W

F3.3.4.4.2 CENTRAL PROCESSOR REGISTERS

REGISTER MNEMONIC	REGISTER NAME	ACCESS ATTRIBUTES
SS	Status Summary	R
EC	Environment Control	R/W
P	Program Address	R/W
MCR	Monitor Condition Register	R/W
UCR	User Condition Register	R/W
UP	Untranslatable Pointer	R/W
JPS	Job Process State	R/W
PFS	Processor Fault Status	R/W
CEL1	Retry Corrected Error Log	R/W
CEL2	Control Memory Corrected Error Log	R/W
CEL3	Cache Corrected Error Log	R/W
CEL4	Map Corrected Error Log	R/W
KC	Keypoint Code	R/W
KCN	Keypoint Class Number	R/W
TE	Trap Enables	R/W
SIT	System Interval Timer	R/W
CMA	Control Memory Address	R/W
CMB	Control Memory Breakpoint	R/W
PTM	Processor Test Mode	R/W
MDW	Model Dependent Word	R/W
DEC	Dependent Environment Control	R/W
MSL	Maintenance Scan Limit	R/W

F3.3.5 DISPLAY PPS PROGRAM ADDRESS REGISTERS

F3.3.5.1 PP

This command causes display of the program address register of each PPU in the PPS.

F3.3.6 CLEAR DISPLAY

F3.3.6.1 CD,<screen>

This command is used to deactivate a currently active display.

<screen>: Is L, R, or B to specify left screen, right screen, or both screens, respectively.

F3.3.7 START SYSTEM

F3.3.7.1 SS

This command is used during system deadstart after the message "PROCEED" is displayed at the console; this command causes final initialization to occur and the CPU to be started. The command is valid at no other time.

F3.3.8 HALT CENTRAL PROCESSOR

F3.3.8.1 HI

This command is used to halt the central processor.

F3.3.9 START CENTRAL PROCESSOR

F3.3.9.1 GO

This command is used to start the central processor after it has been halted with the HT command.

F3.3.10 OPERATING SYSTEM DISPLAYS

Various operating system displays are available to the console; a specific display may be called by typing its unique 2-character identifier and a carriage return.

F3.3.10.1 Display Identifiers and Descriptions

DD - Dayfile of the system job.

F3.3.11 CONSOLE MESSAGES TO THE OPERATING SYSTEM

Single line messages of 60 characters or less may be sent to the operating system from the console keyboard. Any line of input from the keyboard is sent to the operating system if both of the following conditions are met:

1. The first two characters of the line do not match a console command or an operating system display identifier.
2. The number of characters in the input line equals or exceeds 1 character.

F3.4 DEBUG FACILITY

The debug facility of NOS/VE provides a set of capabilities intended to assist in testing of programs which execute under control of NOS/VE. Services provided by the facility are task oriented: selection of the debug facility is at the option of the user at the time of task invocation. NOS/VE uses the CYBER 180 debug hardware to provide these capabilities.

F3.4.1 SUMMARY OF DEBUG FACILITY SERVICES

Set Breakpoint:	Selects a program interrupt which is to occur upon occurrence of a specified condition within a specified virtual address range.
Remove Breakpoint:	Deselects a previously selected program interrupt.
Change Breakpoint:	Changes the virtual address range of a previously specified breakpoint.
List Breakpoint:	Provides a list of currently selected breakpoints and associated conditions.
Trace Back:	Provides information relevant to stack frames associated with an interrupted procedure and its predecessor procedures.
Display Stack Frame:	Display selected information from a specified stack frame.
Display Register:	Display the contents of a specified register of an interrupted procedure.
Change Register:	Sets a specified value into a specified register of an interrupted procedure.
Display Memory:	Displays the contents of a specified area of virtual memory.
Change Memory:	Sets a specified value into a specified location of virtual memory.
Run:	Invokes program execution after a selected program interrupt has occurred.

F3.4.2 DEBUG FACILITY COMMANDS

F3.4.2.1 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..100
<regid> ::= X¦A¦P
<regno> ::= 0..15¦0..0F(16)
<hex_vstring> ::= 'hex string'
<time> ::= 1..(2**31)-1
<vstring> ::= "charstring"
<datatype> ::= HEX¦ASCII¦ASC¦DECIMAL¦DEC
<selector> ::= FULL¦AUTO¦SAVE
```

F3.4.2.2 Command Descriptions

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

F3.4.2.2.1 SET BREAKPOINT

Selects a program interrupt which is to occur upon occurrence of a specified condition within a specified virtual address range.

syntax: BP <name> <condition> [<base>] [<offset>] [<length>]

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

F3.4.2.2.2 REMOVE BREAKPOINT

Deselects a previously selected program interrupt.

syntax: RB <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.

F3.4.2.2.3 LIST BREAKPOINT

Provides a list of currently selected breakpoints and associated conditions.

syntax: LB [<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.

F3.4.2.2.4 CHANGE BREAKPOINT

Changes the virtual address range of a previously specified breakpoint.

syntax: CB <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

F3.4.2.2.5 TRACE BACK

Provides information relevant to stack frames associated with an interrupted procedure and its predecessor procedures.

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.

syntax: TB [<frame>] [<count>]

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

Default parameter values:

<frame>: 1
<count>: 1

F3.4.2.2.6 DISPLAY STACK FRAME

Display selected information from a specified stack frame.

syntax: DS [<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:

16
8

580
256 + 5
1280
128
1408

<frame>: 1
<selector>: FULL

F3.4.2.2.7 DISPLAY REGISTER

Display the contents of a specified register of an interrupted procedure.

syntax: DR <regid> [<regno>] [<datatype>]

Default parameter values:

<regno>: 0
<datatype>: HEX

F3.4.2.2.8 CHANGE REGISTER

Sets a specified value into a specified register of an interrupted procedure.

syntax: CR <regid> <regno> [<datatype>] <vstring>

Default parameter values:

<datatype>: HEX

F3.4.2.2.9 DISPLAY MEMORY

Displays the contents of a specified area of virtual memory.

syntax: DM <base> [<length>]

Default parameter values:

<length>: 8

F3.4.2.2.10 CHANGE MEMORY

Sets a specified value into a specified location of virtual memory.

syntax: CM <base> <hex_vstring>

F3.4.2.2.11 RUN

Invokes program execution after a selected program interrupt has occurred.

F3.0 EXECUTION
F3.4.2.2.11 RUN

syntax: RUN <time>

The time parameter specifies the maximum number of microseconds the program is to execute; a program interrupt will occur upon attaining this execution limit.

Default parameter values:

<time>: infinite

F3.5 ERROR CODES

Error codes are displayed as 6 hex digits in the following format:

AANNNN

The AA field designates the functional area that issued the error. The possible values are:

01 - Signal Handler
02 - Circular Buffer Handler
03 - Heap Manager
04 - Misc Task Services
05 - N/A
06 - File Manager
07 - N/A
08 - Task Manager
09 - Memory Manager
0A - Job Manager
0C - Loader
0D - Central I/O
0E - Dispatcher
0F - Command Language Processor
10 - Logical Name Manager
11 - Debug Processor
12 - Configuration Manager

The NNNN field is a detailed error number within the specified functional area.

F3.5.1 DETAILED ERROR CODES

All numeric values are given in hex.

F3.5.1.1 Signal_Handler

1 = timeout
2 = signal buffer full
3 = task swapped
4 = invalid signal id
5 = incompatible signal type
6 = message buffer too small
7 = empty wait list

F3.5.1.2 Circular_Buffer_Handler

1 = buffer not initialized
2 = buffer full
3 = message too long

F3.5.1.3 Heap_Manager

None

F3.5.1.4 Misc_Task_Services

None

F3.5.1.5 File_Manager

1 = KFD not available
2 = File active
3 = File not active
4 = PFD not available
5 = File exists
6 = File not created
7 = File open
8 = Multiple usage
9 = File not temporary
0A = File not permanent
0B = File not attached
0C = File attached

0D = File not open
0E = Invalid KFDL index
0F = KFD not active
10 = Invalid PFDL index
11 = PFD not active
12 = Active I/O
13 = No units configured
14 = No active units

F3.5.1.6 Task Manager

1 = Task not found

F3.5.1.7 Memory Manager

1 = Page already in page table
2 = Page table full
3 = No free pages
4 = Locked page free request
5 = Page not in page table
6 = Invalid PVA
7 = Page frame not locked
8 = Page frame not assigned
9 - 1E = N/A
1F = Invalid ring number
20 = Segment table is full
21 = Segment number is in use
22 = Segment number not in use
23 = Nil segment pointer invalid
24 = Segment number too big
25 = Cannot change segment number
26 = Unsupported keyword

F3.5.1.8 Job Manager

None

F3.5.1.9 Loader

1 = program not found

F3.5.1.10 Central_I/O

1 = IORP not available
2 = End of file
3 = Invalid byte count
4 = Invalid buffer length
5 = Invalid buffer address
6 = Invalid MS function code
7 = Invalid CIO hardware type
8 = Feature not supported
9-40 = N/A
41 = End of device
42 = End of allocation
43 = File not allocated
44 = File already allocated
45 = Invalid cylinder
46 = Invalid track
47 = Invalid sector
48 = AUD not defined
49 = AUD allocated
4A - 80 = N/A
81 = CBD not available

F3.5.1.11 Dispatcher

1 = Invalid task id
2 = PTL full
3 = Invalid running job ordinal

F3.5.1.12 Command_Language_Processor

1 = missing parameter
2 = invalid character
3 = undefined parameter type
4 = integer out of range
5 = unknown command
6 = unknown syntax
7 = not supported
8 = bad parameter type
9 = token too long
0A = bad combination of parameters
0B = bad value for pointer
0C = unknown parameter name
0D = valid password required
0E = SCB not event

F3.5.1.13 Logical Name Manager

1 = Entry not found
2 = Type mismatch on put
3 = Entry already exists
4 = Illegal put request
5 = Buffer wrong size
6 = Buffer too small

F3.5.1.14 Debug Processor

1 = Debug not initialized
2 = Breakpoint name exists
3 = Base parameter not specified
4 = Invalid breakpoint condition
5 = Condition already selected
6 = Maximum number of breakpoints already set
7 = Invalid address range
8 = Invalid breakpoint name
9 = Condition not selected
0A = No trap has occurred
0B = Invalid stack frame specified
0C = Register not in stack frame
0D = Invalid data type definition
0E = Invalid register type
0F = Invalid P-reg value
10 = Invalid A-reg value
11 = Invalid X-reg value
12 = Invalid selector
3-0FE = N/A
0FF = Unused error code

F3.5.1.15 Configuration Manager

1 = Invalid function code
2 = Not mass storage
3 = Unit not active

F3.5.1.16 CPU MONITOR

These values are stored in register XE just before NOS/VE halts the processor:

1 = Hardware failure

2 = Task aborted in ring 1 or 2
3 = Task aborted while abort in progress
4 = Task executing had negative PIT
301 = Hardware failure

F4.0 CODING CONVENTIONS AND SOURCE USAGE

F4.0 CODING_CONVENTIONS_AND_SOURCE_USAGE

The conventions presented in this section reflect the current state of the NOS/VE system, formerly known as the Hardware Checkout System (HCS). The final NOS/VE system will differ from the current system in many ways, such as different functional areas, new functional areas, different naming conventions, etc. The conventions to be used in the final NOS/VE system are documented in the NOS/VE Project - Procedure and Conventions document. The information presented in this section is intended to assist users while HCS conventions are still in use.

F4.1 NAMES

All global NOS/VE names have the following format:

AAT$NNN

The AA field designates the functional area to which the name applies, and can be one of the following:

SH - Signal Handler
HP - Heap Manager
FM - File Manager
PM - Task Manager
MM - Memory Manager
JM - Job Manager
LL - Loader
CI - Central I/O
DS - Dispatcher
CL - Command Language Processor
LN - Logical Name Space Manager
DB - Debug Processor
CM - Configuration Manager
OS - General NOS/VE
MH - Machine Code Breakout
DM - Data Management
MT - Monitor Interrupt Processor

The T field represents the type of the name, and can be one of the following values:

F4.0 CODING CONVENTIONS AND SOURCE USAGE
F4.1 NAMES

P - Procedure
V - Variable
E - Error Constant
K - Keypoint Constant

The NNN field is a descriptive string describing the object.

F4.2 IEXI_INPUI

NOS/VE contains a routine that allows text input to be performed easily, and without regard to whether execution is on the hardware or the simulator. When running on the simulator the system executes the IO machine instruction (opcode FF) which reads from the file specified on the I parameter when the simulator was called. When running on the hardware, text is input from the console. If more than one task attempts to read input at the same time when running on the simulator, the 'first' task will get the data. If this happens on the hardware, any one of the tasks may get the data.

The name of the routine is CLP$GET_STND_INP and has the following declaration:

```
*call RCLGETS
```

The variable cstring contains the input text and is defined as follows:

```
RECORD
LHI,RHI : 0..255,
S : STRING (255),
RECEND;
```

The LHI field points to the leftmost character of the string. Any number of blanks may precede the first character of the string. A semicolon will be added as the last non blank character of the string and the RHI field will point to the semicolon.

F4.3 IEXI_OUIPUI

NOS/VE contains a routine that allows text output to be performed easily, and without regard to whether execution is on

the hardware or the simulator. When running on the simulator, the system executes the IO machine instruction (opcode FF) which writes to the file specified on the O parameter when the simulator was called. When running on the hardware, text is output to the dayfile, which is scrolled on the console screen. If more than one task outputs to the console, the outputs will be intermixed. NOS/VE does not identify the output as to which task issued it.

The name of the routine is CLP$PUT_STND_OUT and has the following declaration:

```
*call RCLPUTS
```

Note that the string S is a VAR parameter, and as such, a literal string cannot be passed.

F4.4 COMMAND UTILITIES

NOS/VE contains various routines that will aid in the process of command cracking. The user could read a line of text input via CLP$GET_STND_INP and then use these utilities to crack the command line. The following capabilities are available:

CLP$CRACK_COMMAND -

This routine uses a parameter descriptor table to crack the syntax of a command. A parameter value table is built specifying the actual values from the command.

CLP$GET_TOKEN -

This routine returns the next token from a command string.

PMP$ASCII_TO_BINARY -

This routine converts an ASCII string to a binary value.

PMP$BINARY_TO_ASCII -

This routine converts a binary number to its ASCII representation.

For more information on these routines, see a current source listing of them.

There are two ways to use these routines. One way would be to write a user task program which called on them directly. The second way is to modify the NOS/VE command language interface (routine CLP$JOB_COMMAND_PROCESSOR) to process the desired commands. This second method might remove the need for a user task program to aid in program checkout.

NOS/VE provides a set of routines which externalize certain hardware instructions to a CYBIL program. These routines are described fully in a DAP written by Jack Steiner.

F4.5 PROGRAM_HEADER_DESCRIPTION

When a user task is executed via the EXEC command, the text string portion of the command is made available to the program. Also, the program can set the status return variable and have it displayed by the system at task termination.

This communication is performed via the parameters of the PROGRAM statement, which has the following format:

```
PROGRAM NAME (P : ^STRING (255);
              SL : 0..4096;
              VAR STATUS : OST$STATUS);
```

The P parameter points to the string specified on the EXEC command, and SL is the length of the string. For internal program calls, P can be declared as a ^CELL and then any structure can be passed.

F4.6 COMMON_DECK_NAMING_CONVENTIONS

The general format of a NOS/VE common deck name is:

XAANNNN

The X field denotes the type of deck and is one of the following:

T - TYPE/CONST deck
R - Procedure XREF deck

The AA field denotes the functional area the deck deals with. The field may take on the same values as the AA field described

in section 5.1.

Then NNNN field contains the first four characters of the name of the item the deck represents. For XREF's it is the first four characters from the descriptive part of the name, ignoring the characters P$, # and _. For a TYPE/CONST deck the value will be TYPE. For a TYPE/CONST deck defining tables the value will be TBLS.

Some examples would be-

Deck name rmcompa from m#compare_swap.

Deck name rciputs from name cip$put_stnd_out.

F4.7 IMPORTANT COMMON DECKS

The following NOS/VE common decks are worthy of note:

TOSTYPE - General OS definitions.
THDWTYP - Hardware structure definitions.
TMMTYPE - Memory management definitions.
TSMTYPE - Segment management definitions.
TCLTYPE - Command language definitions.

F4.8 DECK USAGE

The following rules apply when using NOS/VE common decks:

1) TYPE/CONST decks include all necessary keywords and end with a semicolon.

2) Procedure XREF decks end with a semicolon. Other common decks may be required to define the types for the parameters specified.

3) Variable XREF decks do not contain the VAR keyword and end with a comma rather than a semicolon. Both the VAR keyword and ending semicolon must be supplied in the surrounding text. Other common decks may be required to define the type symbol.

F5.0 KEYPOINTS

F5.0 KEYPOINTS

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

F5.1 SOURCE CODE CONVENTIONS

The general format of the source statement used to generate a keypoint from a CYBIL program is:

```
#INLINE('KEYPOINT',SECTION,DATA*256,ID);
```

The SECTION parameter identifies the functional area that is issuing the keypoint. It must be in the range 1 to 15. The following values are currently defined:

- 0 = Denotes a continuation of data from the previous keypoint (the occurance of a trap may not allow this feature to work correctly).
- 1 = System information (ID numbers 1-63 are reserved for use by assembler code routines) (OSK$)
- 2 = Memory Manager (MMK$)
- 3 = Command Language (CLK$)
- 4 = Debug (DBK$)
- 5 = CIO (CIK$)
- 6 = File Manager (FMK$)
- 7 = Task Services (TSK$)
- 8 = Dispatcher (DSK$)
- 9 = Unused
- 10= Job Manager (JMK$)
- 11= Signal Handler (SHK$)
- 12= Loader (LLK$)
- 13-15 = Unused

The DATA parameter can be any 24 bit or less integer value, and is normally used to display data that relates to a particular keypoint. The value must be shifted left 8 bits (multiplied by 256) so that it will not overlap with the ID value.

F5.0 KEYPOINTS
F5.1 SOURCE CODE CONVENTIONS

The ID parameter is used to identify the keypoint within a section. The value must be in the range 0 to 255. Values less than 31 are considered critical, unexpected, unusual, etc.

A user may add his own keypoints by using section 15 with appropriate DATA and ID values.

F5.2 KEYPOINT DATA USING THE HARDWARE

A circular buffer of the last 200 keypoints is maintained in memory. These may be displayed on the console using the following command:

DM <real memory address of the buffer>

The buffer is defined by the symbol OSV$KEYPOINT_BUFFER, which will appear in the LINKMAP where the monitor tables are defined. The real memory address is computed by adding the address of OSV$KEYPOINT_BUFFER to the length of the page table (monitor tables follow the page table).

Keypoints produced on the hardware are not nearly as useful as simulator keypoints because only the last 200 are available and they are displayed in an unedited format (i.e., a hex memory dump). However, they can be useful in showing the sequence of events leading up to a system crash.

The format of the keypoint buffer is an array of words with each word having the following format:

Left 28 bits - Value of free running microsecond clock when the keypoint occured.
Next 4 bits - Keypoint class
Next 24 bits - Data value.
Next 8 bits - Keypoint id value

F5.3 KEYPOINT DATA USING THE SIMULATOR

When executing on the simulator, all keypoint instructions cause an entry to be added to the local file SESSMKF. When execution is complete, this file may be processed by a utility program to produce a listing of the keypoint information in a readable format.

F5.3.1 KEYPOINT REFORMATTING UTILITY

The SESSMKF file produced by the simulator can be reformatted into a readable listing by executing the following procedure:

```
XEQ,RNOSKEY[,B](PR,[FN])
```

The B parameter, if present, causes the procedure to be run as a batch job.

The FN parameter specifies the name of the file containing keypoint data. The default is SESSMKF. If this file is not a local file, or the procedure is running in batch mode, the file will be obtained from the current catalog.

The PR parameter, if present, causes the reformatted listing to be sent to the printer.

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

The RNOSKEY procedure requires two additional files as input. The first file defines how the keypoint information is to be reformatted. The name of this file is KEYDESC and it is obtained from the current catalog or, if not present there, from the NOS/VE catalog. The format of this file is described in section 6.3.2.

The second input file provides directives to the utility program which direct its execution. The following directives are supported:

```
CV MAXPROCID N
```

This directive causes all keypoints with id values greater than N to be ignored.

```
CV UNDEFINED
```

This command causes both defined and undefined keypoints to be printed. An undefined keypoint is one that does not have a definition in the KEYDESC file.

```
CV DEFINED
```

This command causes only defined keypoints to be printed.

CV IDENT

This command causes the keypoint processor to indent the output produced based on the NS field of the KEYDESC file.

RUN

This command causes the processor to make one pass over the keypoint file. It is used after the CV commands have specified how to process the keypoint information.

END

This command terminates the keypoint processor. It must be the last command.

These directives are read from file RNOSKEZ. This file is obtained from the current catalog or, if not found there, from the NOS/VE catalog.

F5.3.2 KEYPOINT DESCRIPTION FILE

The keypoint description file is used by the keypoint reformatting utility to direct the reformatting of the keypoint information. Each line in the file describes one keypoint, and has the following format:

SID SCN PID LN F LEN FMT CSTR NS DT

The SID field represents the section ID and is 2 characters long. An example would be MM for memory manager.

The SCN field is the section class number, which equals the SECTION value from the keypoint instruction.

The PID field is the procedure ID, which equals the ID value from the keypoint instruction. The SCN and PID values uniquely define a keypoint - all of the other information is used for reformatting.

The LN field is used to cause a line on the keypoint listing to be preceeded by a * if the LN value is zero. This feature is used to mark a given keypoint as special.

The F field specifies that this is a special keypoint (i.e., has special meaning to the program that formats the keypoint

file). The values must be one of the following:

0 - Not special
1 - Task switch
2 - Begin trap
3 - End trap
4 - Begin monitor
5 - End monitor

Any new keypoint descriptions should specify this field as zero unless the utility program is modified to handle the new value(s).

The LEN field specifies the length of the data portion of the keypoint in bytes.

The FMT field specifies in what format the data portion of the keypoint should be displayed, and can be one of the following:

H - Hex
I - Integer
A - ASCII

The CSTR field is a 1 to 8 character string describing the data portion of the keypoint.

The NS field specifies the number of spaces to indent the DT string on the reformatted file. This feature can be used to show procedure nesting via keypoints.

The DT field is a text string that describes the purpose of the keypoint. It may fill the rest of the current line.

The user may add his own keypoint descriptions to this file and save it in his catalog. When RNOSKEY is run, the existing NOS/VE keypoints and user defined keypoints will be listed together. If the NOS/VE keypoints are not wanted, their descriptions may be deleted from the file.

F5.3.3 REFORMATTED FILE DESCRIPTION

The reformatted listing file contains two sections. The first is a summary of the number of times each keypoint occurred. The second section is a listing of all the keypoints in the order they were issued. Each line of the second section has the following format:

```
RT TSL DATA CSTR S TN SID DT
```

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 CSTR field is the CSTR field from the keypoint description file for this keypoint.

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, i.e., 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 SID and DT fields are just the SID and DT fields from the keypoint description file for this keypoint, with the DT field indented as specified in the keypoint description.

F6.0 DEADSTART PROCEDURES

F6.0 DEADSTART PROCEDURES

F6.1 STAND ALONE DEADSTART (WITHOUT NOS/170)

o Set the PPS deadstart panel as follows:
(Note that this is not the setting for the IOU)

Location	Setting	Instruction
1	7513	DCN
2	2001	LDC
3	0000	
4	7713	FNC
5	0060	(Warmstart read: 556 bpi, unit 0)
6	7413	ACN
7	7113	IAM
10	6200	

o Set SWEEP/LOAD/DUMP switch on the PPS panel to LOAD position.

o Mount deadstart tape on unit 0.

o Verify that the 512 printer is READY.

o Depress DEADSTART button at keyboard/display console (the message "PROCEED" should appear on the left screen).

o Type SS (followed by a carriage return) to complete system initialization.

F6.2 DEADSTART WITH NOS/170

This deadstart procedure uses the Common Test and Initialization (CTI) facility of NOS/170 to deadstart NOS/VE. It requires two tape units, one for the NOS/170 deadstart tape and one for the NOS/VE deadstart tape.

Detailed information on the use of CTI is located in chapter 2

F6.0 DEADSTART PROCEDURES
F6.2 DEADSTART WITH NOS/170

of the NOS Version 1 Operators Guide (60435600 J).

- Set the deadstart panel as follows:

Location	Setting
1	0000
2	0000
3	0000
4	75TT
5	77TT
6	EDDD
7	74TT
10	71TT
11	7301
12	RPXX
13	RPXX
14	0000

  TT is the channel number of the tape unit containing the NOS/170 deadstart tape.

  E is the equipment number of the tape controller.

  DDD is broken down as follows:

      FFU

  FF specifies the type of unit being deadstarted from and is 12 for 67X tapes, 26 for 66X tapes and 3U for 844/885 disks.

  U (or UU) is the unit number of the device.

  Words 12 and 13 (RPXX) can be ignored as they are only used during NOS/170 deadstart.

- Mount the NOS/170 deadstart tape on the unit described by the deadstart panel settings. Mount the NOS/VE deadstart tape on another unit. Press the deadstart button. The CTI *A* display should appear.

- Select the U (utilities) option. The CTI *U* display should appear.

- Select the A (alternate deadstart) option. CTI will ask for the device type, channel, equipment and unit of the device to be deadstarted from. The values supplied by the user should

F6.0 DEADSTART PROCEDURES
F6.2 DEADSTART WITH NOS/170

be the ones for the NOS/VE deadstart tape unit.

- The message "PROCEED" should appear. Type "SS" to start NOS/VE.

F7.0 NOS/VE TEST PROGRAMS

F7.0 NOS/VE TEST PROGRAMS

NOS/VE has a set of programs which run as user tasks. These programs provide the following features:

- checking specific hardware features.
- checking specific NOS/VE software features.
- creating a heavy and/or uniform load on the system.

F7.1 EXISTING TEST CASES

F7.1.1 SORT

This program creates an array of records with random keys, then sorts this array and checks the results.

The program allows a controlled amount of load to be applied to the system paging mechanism.

syntax: EX SORT 'NR,RS'

NR - The number of records to be created and sorted.

RS - A factor affecting the size of each record. By varying record size, the user can change the ratio of CP time to IO time for a test. The approximate record size is given by the formula:

32+8*RS

F7.1.2 USER1

This program will display the line:

'user task executing'

and then terminate. It is used to show that the system can at least execute a very small test case.

syntax: EX AAAA 'STRING'

STRING - If this parameter is present, it will be output after the 'user task executing' message.

F7.1.3 UUTL

This program provides a variety of different test cases, and also allows them to be run in a repetitive manner.

syntax: EX UUTL 'STR'

STR - A string describing which program to run and how to run it. The various programs are described below.

F7.1.3.1 ENVSPEC

This program generates an environment specification error.

syntax: EX UUTL 'ENVSPEC'

F7.1.3.2 AROVFL

This program generates an arithmetic overflow error.

syntax: EX UUTL 'AROVFL'

F7.1.3.3 INSSPEC

This program generates an instruction specification error.

syntax: EX UUTL 'INSSPEC'

F7.1.3.4 DIVFLT

This program generates a divide fault error.

syntax: EX UUTL 'DIVFLT'

F7.0 NOS/VE TEST PROGRAMS
F7.1.3.5 LA

F7.1.3.5 LA

This program performs an LA (load address) instruction on a specified PVA. It can be used to test various forms of memory protection.

syntax: EX UUTL 'LA,PVA'

PVA - The virtual address to be loaded from.

F7.1.3.6 SA

This program performs an SA (store address) instruction on a specified PVA. It can be used to test various forms of memory protection.

syntax: EX UUTL 'SA,PVA'

PVA - The virtual address to be stored into.

F7.1.3.7 RETURN

This program modifies the previous stack frame area and then returns to see what effect the modifiction will have.

syntax: EX UUTL 'RETURN,N'

N - Modification option. Must be one of the following:

1 - Set value of A2 so it is not 0 mod 8.
2 - Set A2 bit 32 to 1.
3 - Set A2 segment number invalid.
4 - Set A3 segment number to segment without read access.
5 - Set P register segment number invalid.
6 - Set P register so it is not 0 mod 2.
7 - Set P register bit 32 to 1.
8 - Set P register segment number to non executable segment.
9 - Set final A0 < > A2.
10 - Cause VMID error.
11 - Cause inward return.
12 - Cause return to C170 mode.

F7.0 NOS/VE TEST PROGRAMS
F7.1.3.8 TESTMEM

F7.1.3.8 IESTMEM

This program creates, writes and verifies an array of records. The record size is such that some records will be located on a word boundary, some on all seven (other) byte boundaries, some will cross pages, etc.

syntax: EX UUTL 'TESTMEM,BC'

BC - Number of bytes to be allocated to the records. The number of records created is: BC DIV 17+1.

F7.1.3.9 IESIMOVE

This program is similar to TESTMEM, except that BDP instructions are used to compare and move records.

syntax: EX UUTL 'TESTMOVE,BC'

BC - Number of bytes to be allocated to the records. The number of records created is: BC DIV 255*2 + 1.

F7.1.3.10 RECURSE

This program calls a procedure recursively.

syntax: EX UUTL 'RECURSE,N'

N - The number of times to recursively call the procedure.

F7.1.3.11 CYCLE

This program cycles for a specified number of milliseconds. It can be used to load the system with 'idle tasks.

syntax: EX UUTL 'CYCLE,MS'

MS - The number of milliseconds to cycle for.

F7.1.3.12 TIMEDUI

This program delays for a specified amount of time in increments.

F7.0 NOS/VE TEST PROGRAMS
F7.1.3.12 TIMEOUT

syntax: EX UUTL 'TIMEOUT,T1,T2'

T1 - The number of milliseconds in each pmp$delay request.

T2 - The total number of milliseconds delayed.

F7.1.3.13 LOOP

This program loops (executes) for a specified amount of time. It can be used to load the system with active tasks.

syntax: EX UUTL 'LOOP,MS'

MS - The number of milliseconds to execute for.

F7.1.3.14 A170

This program creates a segment with read, write, execute and binding attributes, places C170 code into it and executes that code. The code executed counts down an X register and when it gets to zero, executes an illegal C170 instruction (op code 0178).

syntax: EX UUTL 'A170,N'

N - This value times 1000000 is placed in the X register being decremented to zero.

F7.1.3.15 REPEAT

This program synchronously executes a program a given number of times.

syntax: EX UUTL 'REPEAT,PN,N,'STR''

PN - Name of the program to execute. It can be any program present in the NOS/VE library.

N - The number of times to execute program PN.

STR - The parameter string passed to the program when it begins execution. Note the use of double quote marks within a quoted string.

F7.0 NOS/VE TEST PROGRAMS
F7.1.3.16 CALLER

F7.1.3.16 CALLER

This program is similar to REPEAT, but the tasks are run asynchronously.

syntax: EX UUTL 'CALLER,PN,N,'STR''

The parameter definitions are the same as REPEAT.

F7.1.3.17 BULK

This program runs a number of different programs a specified number of times. This test is used primarily to load the system with a random mix of programs. Some programs will terminate abnormally.

syntax: EX UUTL 'BULK,N,X'

N - The number of times to execute the entire list. The following programs (with parameters) are executed:

```
LOOP,5
TIMEOUT,5
CYCLE,5
INSSPEC
ADRSPEC
ENVSPEC
PRIVINS
LA,257800000000(16)
SA,100200000000(16)
AROVFL
```

X - If this parameter is specified (any value except 1) then the test will be run asynchronously - that is, each test in the list will be started and after they are all running, the BULK test will wait for them all to complete. This procedure is repeated N times.

If this parameter is not specified, or is specified as 1, each test will be allowed to complete before the next test in the list is started.

F7.0 NOS/VE TEST PROGRAMS
F7.1.3.18 BULKNTC

F7.1.3.18 BULKNTC

This program is similar to BULK, except that all of the programs run are expected to complete normally.

syntax: EX UUTL 'BULKNTC,N,X'

N - Same as for BULK, except the program list is the following:

```
LOOP,500
LOOP,5
TIMEOUT,5,500
CYCLE,500
RECURSE,5000
LOOP,10
TESTMEM,100000
TESTMOVE,100000
LOOP,1000
```

X - Same as for BULK.

F7.1.3.19 ADRSPEC

This program causes an address specification error.

syntax: EX UUTL 'ADRSPEC'

F7.1.3.20 PRIVINS

This program causes a privileged instruction error.

syntax: EX UUTL 'PRIVINS'

F7.2 EXAMPLES

To run the sort program on 1000 records of size 200 do:

```
EX SORT '1000,200'
```

To run 53 asynchronous copies of same sort test do:

F7.0 NOS/VE TEST PROGRAMS
F7.2 EXAMPLES

```
EX UUTL 'CALLER,SORT,53,'1000,200''
```

To run the same sort test 45 times in succession, do

```
EX UUTL 'REPEAT,SORT,45,'1000,200''
```

HINT: When running programs which may run for a long period of time, use the 'A' option on the EX command. This runs the programs asychronously with the command processor. This allows you to status the test to see what is happening (TSTATUS, PFSTATS) or to terminate the test if it runs too long (TMTERM).

Table of Contents