DISTRIBUTION

C.K. Bedient	ARH254
F.A. Bischke	ARH254
R.E. Dennis	ARH254
J.B. Farr	ARH254
J.H. Koch	ARH254
L.E. Leskinen	ARH254
T.M. Miller	ARH254
J.A. Nauman	ARH254
M.J. Perreten	ARH254
R.A. Peterson	ARH254
J.F. Steiner	ARH254
R.J. Thielen	ARH254
S.C. Wood	ARH254
R.B. Beeson	ARH260
R.E. Erickson	ARH260
R.A. Mann	ARH260
R. Woodruff	ARH260
J.J. Krautbauer	ARH280
H.A. Wohlwend	ARH280
R.D. Palm	ARH293
J. Sutherland	CANCDD
J.H. Wick	MNA02B
C.A. Sheats	SVL114

Please help keep the above distribution list current. If your name should be removed from the list or another name added, contact Bonnie Swierzbin at ARH260 - extension 3460.

DISTRIBUTION

R.H. Kingdon	ARH254
T.C. McGee	ARH254
R.M. Medin	ARH254
J.R. Ruble	ARH254
D.J. Maguire	ARH260
N.E. Meyer	ARH260
S.W. Fewer	ARH263
C180 Central Dayfile	
E.B. Buckley	SVL173
R.S. Cummer	SVL163
R.K. Endo	SVL128
A.E. Hiebert	SVL143B
U.B. Lundh	SVL102
W.H. Moehrke	SVL163
J.S. White	SVL102M
D.H. Williams	SVL102
J.Y. Young	SVL173

Please help keep the above distribution list current. If your name should be removed from the list or another name added, contact Bonnie Swierzbin at ARH260 - extension 3460.

```
          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 : JULY 28, 1981 ;

TO : DISTRIBUTION LOCATION : ;

FROM : BJ SWIERZBIN LOCATION : ARH260 ;

SUBJECT : UPDATED PROCEDURES NOTEBOOK DOCUMENTATION

The Build N update of the Integration Procedures Notebook is now available. Change pages may be obtained through the following command sequence: ;

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

Some parts of the document are now out of date (see Document Deficiencies below) but have not yet been removed. However, a complete listing of the document can be obtained through the following command sequence: ;

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

Some highlights of this revision are as follows:

Catalog Management Policies: Section 2.3 has been updated to reflect how the Integration catalogs are managed for parallel builds. ;

New Procedures: Sections 2.4.5, 2.5.2, and all of Section 2.11 have been updated to describe the new procedures NVEBLD, NVEREP, GET, SAVE, and NVEMAP. ;

Updated Appendices: Appendix G now contains CYBIL installation information for version 81188. Appendix E contains the Build N Helpful Hints document. ;

Document Deficiencies: Appendices C, D and E are outdated and should not be used. Information formerly in Appendix H is now in Appendix E and Appendix H should no longer be referenced. This document will be completely revised for Build C. Your comments for that revision will be appreciated. ;

INSTRUCTIONS TO CONTRIBUTORS

* Subject and verb always has to agree.

* Being bad grammar, the writer will not use dangling participles.

* Parallel construction with coordinate conjunctions is not only an aid to clarity but also is the mark of a good writer.

* Do not use a foreign term when there is an adequate English quid pro quo.

* If you must use a foreign term, it is de rigor to use it correctly.

* It behooves the writer to avoid archaic expressions.

* Do not use hyperbole; not one writer in a million can use it effectively.

* Avoid cliches like the plague.

* Mixed metaphors are a pain in the neck and ought to be thrown out the window.

* In scholary writing, don't use contractions.

* A truly good writer is always especially careful to practically eliminate the too-frequent use of adverbs.

* Use a comma before nonrestrictive clauses which are a common source of difficulty.

* Placing a comma between subject and predicate, is not correct.

* Parenthetical words however should be enclosed in commas.

* Consult the dictionary frequently to avoid mispelling.

-- Reprinted from the Texas Newsletter for Medical Technology (Volume 5, Number 9, 1978)

1.0 NOS/VE SYSTEM OVERVIEW

1.0 NOS/VE SYSTEM OVERVIEW

1.1 INTRODUCTION

The basic components of NOS/VE include the following:

- A Hardware Initialization Verification Sequencer (HIVS) component
- Modifications to standard A170 NOS system components
- A170 NOS application programs (and procedure files) which execute in the A170 NOS (Real State) environment, and provide system to system communication facilities.
- The Virtual Environment code which is responsible for the execution of tasks in Native Mode in the Virtual State of the hardware.

The nomenclature used to describe NOS/VE components is rather confusing. Frequently, the Virtual system software is what is referred to as "NOS/VE". When A170 NOS and the supporting utilities are present, the term "Dual State" is used. To differentiate the two execution modes of the machine the terms "Native Mode", "Virtual Mode", and "Virtual State" are used to describe the execution of C180 instructions. The terms "Real State" and "NOS" are used to refer to the execution of C170 instructions.

The model which is often used to describe the execution of NOS/VE in Dual State mode is that of one machine front-ending another, and communication between the two machines occurring over a communications link. From the software's point of view, another perspective is used. To the Virtual State software, the NOS system is merely a job which happens to be executing in the Virtual State envelope created by EI. (The microcode translation of the C170 instruction set is "invisible" to both the NOS and NOS/VE software.) NOS's view of the Virtual State is merely that of a job which runs at a control point, and is communicated with through the K-display. The remainder of this section is meant to describe NOS/VE with regard to its components.

1.1.1 THE HIVS COMPONENT

Included in the diagnostic world, which establishes the initial Virtual Execution environment, is the microcode for the CPU as well as a Native Mode monitor-like program called the Error Interface (EI). The microcode is strictly supported by the diagnostic organizations, and neither the NOS nor the NOS/VE software will execute without microcode present in the CPU. The EI program is supported by Advanced Systems Development. There are currently two versions of EI, one which only supports A170 NOS and does tasks such as CMU instruction emulation, and one which also supports the switching between the NOS and NOS/VE CPU monitors. This latter version of EI requires a partner intermediary called "EIE" which is statically loaded with the Virtual State software, and is loaded during deadstart of NOS/VE. In order to assure that the right version of EI is present, the HIVS tape which is distributed by Advanced Systems Integration for use with NOS/VE is the correct version to install. The version of microcode present on this HIVS tape will hopefully be the right version, but we have not discovered the necessary utilities to replace versions of microcode on a HIVS tape from an installed configuration such as that in which we test systems.

1.1.2 A170 NOS MODIFICATIONS

The modifications required to support a Dual State execution environment are primarily assembled in the BLD170 procedure file. Few specifics will be given here other than to state that half a dozen Peripheral Processor routines are involved, as well as modification to NOS CPU Monitor. The key aspect to note about these components is that a special version of A170 NOS deadstart tape must be used. Again, this deadstart tape version is supplied by the Advanced Systems Integration project. There are additional procedure files which must also be present on this special deadstart tape, which are not documented here at this time.

1.1.3 A170 NOS APPLICATIONS

A portion of the software alluded to as A170 NOS modifications could be classified as A170 NOS Applications. The applications referred to, however, are not present on the deadstart tape, but exist as permanent files which are invoked or processed by procedure files present on the deadstart tape. In the strict sense of the word, those utilities which

are not execution order dependent or require system residence are placed on the deadstart tape. Utilities which must be run in a given sequence (and possibly as system origin) are governed by procedure files which are present on the deadstart tape.

1.1.4 THE VIRTUAL STATE COMPONENT

The Virtual State software consists of both a statically and dynamically linked component. The statically linked component is composed of Monitor and Task Services modules while most other tasks are dynamically linked. In order to statically link Monitor and Task Services, the SES utilities VELINK and VEGEN or their Virtual State equivalents must be used. In other systems this statically linked system component is commonly referred to as the "unconfigured deadstart tape" or "bootstrap system". Once the "bootstrap system" has been generated which has its own LINKER/LOADER equivalent, then it is possible to deadstart this bootstrap system and begin dynamic link/loads. One of the attributes of this statically linked component is that there is more than one partition associated with it. In order to keep these partitions separate during the CI build process, these partitions are placed on separate files. The content of these partitions is described in a subsequent section of this document.

The dynamically linked component of the Virtual State software consists of II format object text which is processed by the NOS/VE LOADER. In order to create II format object text, it is necessary to either use the utility "CITOII" to convert CI object text to II object text, or else use a II compiler or assembler. In order to make this CITOII utility available to each task created by the Virtual State software, the object text for this utility must be statically linked and loaded with Monitor and Task Services. Once this utility is made available to dynamically generated tasks, it is necessary to retrieve the other utilities which establish communications with their A170 NOS counterparts. This is accomplished by executing a post-deadstart SCL procedure file which initiates the Remote Host and Interactive Facilities from the library "OSLIB" (which in turn was created from the CI object text file "XLJOSL").

The components mentioned thus far have been the statically linked modules for Monitor, Task Services, the CITOII utility, and the dynamically linked modules from OSLIB. There are libraries other than OSLIB, some of which are necessary for the successful execution of user tasks. Such a library is SYSLIB which contains the Object Code Utility modules which

provide for the creation of II object text libraries. The SYSLIB library must be made part of a job's object library list if any II object library manipulations are to be made. From the compiler perspective, the run-time libraries CYBILIB, MATHLIB, FRTL, etc. must be created by a job which has SYSLIB as part of its object library list. The compiler generated object text for a compiler such as CYBIL names the appropriate run-time library in the object text records (eg. CYBILIB). Thus, a CYBIL program must access the appropriate run-time library (CYBILIB) and make this library part of the job's object library list. This explicit manipulation of a job's object library list will eventually be replaced by job prologues which are created during accounting and validation and/or user prologues which establish a job's execution environment.

1.2 VIRTUAL STATE PARTITIONING

The system can be thought of as being divided into three partitions which consist of six object text files. 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. The modules on file XLJBBB consist of user tasks which can be thought of as belonging to a third type of partition. This latter partition contains routines which run in job mode in the user ring (for the purpose of converting CI compiler generated output into II format after it has been transferred from the A170 NOS software to the NOS/VE execution environment). In the memory map, which is described in a subsequent section, the code for this user partition resides in the "User Task(s) Library" and is made a part of each task created for NOS/VE. The system restructure which occurred with Build M made this partition relatively small (since this partition must be loaded into memory at deadstart time). Although the build procedures allow for this partition to be replaced with one which contains additional user tasks, the preferred method of executing user tasks is to GET them from a NOS permanent file (after they have been generated by a CI compiler) to the NOS/VE execution environment and EXECUTE these tasks using the NOS/VE LOADER. If the execution environment is the simulator instead of the hardware, then new user tasks should be statically loaded in place of XLJBBB using the NVELINK procedure.

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. A variable or entry point may be gated in the source specification using CYBIL and CPU ASSEMBLER language constructs, or the object text may be modified by using the SES Object Code Utilities. Refer to the MAP<offset>K file, produced by the NVELINK procedure, for a list of the entry points available to a user task. This list of entry points preceeds the linkage of file XLJBBB 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.

Occasions arise in which procedures are of common utility to more than one partition, but should not be gated across partitions. In such instances, these procedures are placed upon a "run-time" library such as CYBILIB, and references to these procedures are satisfied at "LOAD" time from the appropriate library. "LOAD" time satisfying of externals can either be done statically or dynamically. A static load is accomplished through the SES Linker and Loader utilities. Dynamic loads occur through the use of the NOS/VE loader during the execution of a C180 job. Whenever possible, dynamic loading of routines is preferred (as in the case of a compiler satisfying externals from a run-time library) since this is the mechanism which customers of NOS/VE systems will be using.

1.3 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.

1.4 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.

1.5 DUAL STATE MEMORY MAP

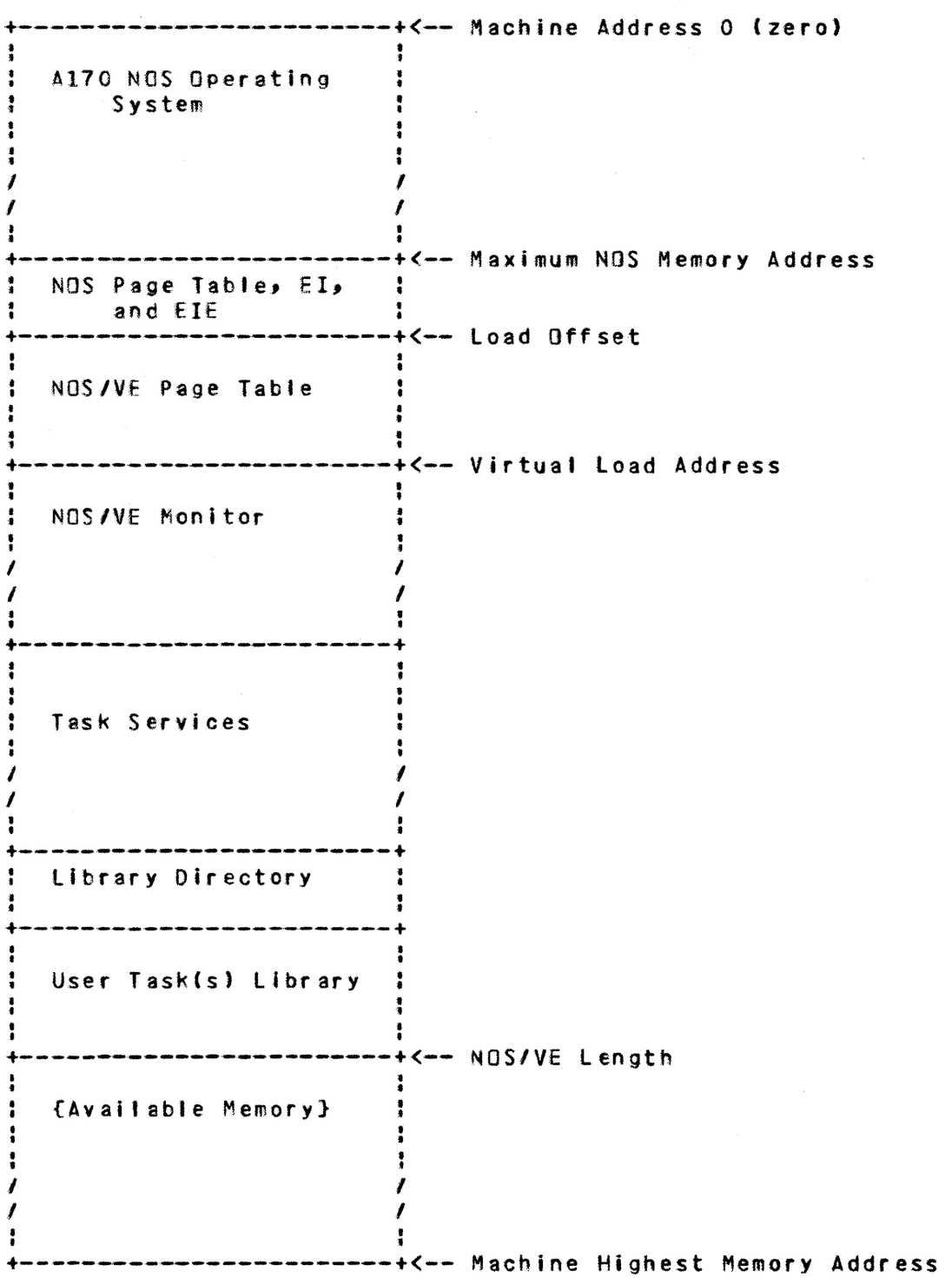
When dealing with a virtual memory system it is often necessary to understand the real memory aspects of the software which is present in the machine. The following map describes the real memory aspects of the software, and where it is mapped during the deadstart process. To make this map complete would require overlaying it with segment and page boundaries. Rather than attempt to produce this overlaying effect, suffice it to say that (by convention) the boundaries described in this map occur at even page boundaries. Whether or not the pages which constitute any given area in the map are paged is a function of the attributes of the segment to which the area belongs.

The real utility of this map is in showing the relationship of values which are supplied to the SES Virtual Environment Generator through the skeleton VELDCM file. This skeleton is dynamically edited when the NVELINK procedure is invoked to produce the LDR<offset>K file. The VELDCM variables are underlined in the relationships described after the map. By using the relationships given, it is possible to compute the

relative starting locations of different areas within a NOS/VE dump.

It should be noted that the relationships given here are expressed in decimal byte addresses, while the machine addresses are hexadecimal. To pursue hexadecimal addresses requires a copy of the MAP<offset>K file. Specifically, the load addresses for Monitor, Task Services, etc. are contained in the Virtual Environment Generator output which immediately follows the LINKER output.

1.0 NOS/VE SYSTEM OVERVIEW
1.5 DUAL STATE MEMORY MAP



1.0 NOS/VE SYSTEM OVERVIEW
1.5 DUAL STATE MEMORY MAP

Page Size = [1; 2; 4; 8; 16; 32; 64] * 1024
= PAGESIZEV

Page Table Size = [4; 8; 16; 32; 64; 128; 256; 512; 1024] * 1024

Page Table Length = MAXIMUM(<Page Size>,<Page Table Size>)
= PTLV

EI Length = 8 * 1024
= 8192
= EILENGTHV

Load Offset = [256; 128; 64; 0] * 1024 * 8
= [2048; 1024; 512; 0] * 1024
= LOADOFFSETV

Maximum NOS Memory Address = LOADOFFSETV - EILENGTH
= NOSLENGTHV

Virtual Load Address = LOADOFFSETV + PTLV
= LOADADRV

Page Table Address = LOADOFFSETV (By convention)
= PTAV

2.0 OVERVIEW OF INTEGRATION PROCESS

2.0 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.

2.1 RELATED DOCUMENTS

Document Title	Distributor
NOS/VE Procedures and Conventions	S.W. Fewer
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 CI 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 - ARH3883

2.2 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 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).

2.3 CATALOG MANAGEMENT POLICIES

The Integration project builds two systems in parallel and manages two catalogs for each system. The primary system is the system that is between the beginning of the build cycle and the feature code cutoff. The secondary system is between the feature code cutoff and the end of the build cycle. Primary system files begin in the INT1 catalog and move to the INT2 catalog after the system has passed Confidence Testing. After a system has reached its feature code cutoff, a stabilized feature build is moved to the DEV1 catalog, the build from INT2 is moved to DEV2, and this system is now considered the secondary system. Also at this time, a new primary system is started in the INT1 catalog by adding its planned feature code to the previous primary system. When a build has completed its integration cycle, the final build for that cycle is moved to the REL1 catalog. It is at this time that a build is considered a candidate for transmittal to other facilities for further development work, a final Build Content Report is distributed and whatever usage documentation that is available is distributed.

2.0 OVERVIEW OF INTEGRATION PROCESS
2.3 CATALOG MANAGEMENT POLICIES

The following diagram illustrates the function of each catalog:

	Primary System	Secondary System	Transmitted System
Working Catalog	INT1	DEV1	---
Latest Stable Build Catalog	INT2	DEV2	REL1

In general, procedures executing from a given catalog access only those files which have the same level of verification associated with them. The INT1 and INT2 catalogs will access the most recent compilers, SES tools, etc. while the DEV1 and DEV2 catalogs access a previous, more stable level of utilities.

The REL1 catalog represents the "frozen" catalog for which changes are no longer being accepted (typically a snapshot of the last build cycle). This is generally the system that is being run in SVL closed shop and is retained for duplicating problems found there. The REL1 catalog will change no more frequently than once for each build cycle. The INT2 and DEV2 catalogs contain the latest stable builds (i.e. the builds have passed Confidence Testing) for the primary and secondary systems, respectively. The INT1 and DEV1 catalogs, however, are "working catalogs" for the debug of new system fixes, new procedures, etc. The stability of these catalogs cannot be predicted.

2.4 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:

2.4.1 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. To

obtain a listing of the complete set of command language procedures provided in the Integration catalog, execute the following command language sequence:

```
SES.LISTMEM B=PROCLIB UN=<Integration_Catalog>
```

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 = 200B (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 statement. If they are not large enough, have the operations staff change them.

2.4.2 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.

Practically all of the procedures described in this document are written to execute in "BATCH" as well as local mode. In order to provide a consistent result when these

procedures are run, it is necessary to save many of the generated files as permanent files. While some purists see this as "catalog polution", we make all attempts to preserve only those files which are necessary for future reference or usage. Whenever possible local file names generated by the procedure are given unique names so as not to conflict with any user files. In numerous instances, convenient "reserved" file names are used to enhance the configurability of these procedures. For example, all files accessed by the procedures are searched for first as local files, then as permanent files in the catalog in which the procedure is executing, then (optionally) in the catalog specified by the 'AREA' parameter value, and finally in the catalog in which the procedure was found. Thus, if the name of a tool accessed by the procedures is known, several versions of this tool can be tried through iterative executions of the procedures without requiring procedure modification. To aid in the isolation of tools accessed by the procedures a "common deck" type structure is included in the procedure libraries which names many of the tools to be accessed by the procedures. These structures exist as records on the procedure library which are INCLUDEd into the relevant procedure file. Initially we have partitioned these tools into the records 'TOOLALL', 'TOOL170', and 'TOOL180'. Experience has shown this partitioning to be somewhat combersome for some of the procedures, and will probably be fine-tuned in subsequent revisions of the procedure library.

Some of the procedures contained on the procedure libraries change very frequently due to changes in system structure or for other reasons. It is inevitable that errors creep into the procedures at times. Often it is quicker to change the KCL generated as a result of invoking the SES procedure than to change the procedure library. The SES processor may be invoked via the SES,TEST.<Procedure_Name> mechanism and the resultant KCL is written to a file named SESTEST. This SESTEST file may then be edited, and the offending control statement corrected. A subsequent CALL,SESTEST statement may then be used to execute the corrected KCL. While we do not necessarily condone this approach to fixing procedure files we can hardly deny its existence. If, for example, it should prove necessary to provide our customers with installation procedures for software which we generate with SES procedures, it would be our intent to ship the SES generated KCL statements rather than the SES procedure and a copy of the SES processor.

2.4.3 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 'CY8ICMN'. 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 Remote Host Facility which supplies job-to-job communication between the Virtual State and NOS portions of the CY180 machine.

2.4.4 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 NVEBILD procedure. (Refer to the Source Maintenance Section of the SES User's Handbook if you have questions about source maintenance.)

2.4.5 COMPILE/ASSEMBLE FROM SOURCE

The efficiency of the Integration build procedures is a function of how much of the system is being built and how much information is supplied to the procedures when they are invoked. If the name of each module to be recompiled and its object file residency is known prior to invoking the procedures, then the most efficient method is to use the NVEBILD procedure and specify the lists of module names and library names via the 'M' and 'L' parameters. If only the module names are known, then the NVEBILD procedure with the 'M' parameter specified should be used (a search for the library names will be used). If only a modset file is available, the scope of changes is not readily apparent (i.e. several common decks are changed) or the number of modules to be recompiled is prohibitively large for manual specification to the NVEBILD procedure, then the NVEBLD procedure can to used to automatically generate the correct NVEBILD procedure calls (using a NOSVEPL cross reference). If it has been determined ahead of time that several modules on the same library are being changed, then it is more efficient to rebuild the entire library using the 'L' parameter of the NVEBILD procedure. If several libraries need to be rebuilt (as in a full system build), then the NVEBILF procedure should be used.

The general philosopy behind the NVEBILD 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 NVEBILD 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.

2.4.6 BEGIN THE LINKER-LOADER PHASE

The LINKER (SES.VELINK) and LOADER (SES.VEGEN) are packaged together in the Integration procedure NVELINK. 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 NVELINK options should be used to produce the file named 'LGBOK'. This file can be run on the simulator using the NVESIM procedure, or can be used to create a deadstart tape using the 'VSN' parameter of the NVESYS 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 NVESYS 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.

2.4.7 GENERATE THE DEADSTART FILE

In order to generate a deadstart tape for standalone NOS/VE, it is only necessary to run the NVESYS 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 NVEBILD procedure (deck named MLAEI).

2.5 NVEBILD PROCEDURE DESCRIPTION

The procedure NVEBILD is used to add or replace modules on a base object text file. NVEBILD 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 ERRORS (which has the same format as NOSLIST) in the current catalog. The direct access file ERRLIST will contain a one line error message which indicates the type of error detected for all errors diagnosed by the procedures. By listing the ERRLIST file from a terminal, a summary of the number and types of errors encountered can be determined. There are conditions such as unrecoverable disk errors which can cause erroneous messages to occur in this file. In such case it is necessary to examine the DAYFILE produced by the procedure to isolate the problem.

If a specified module is to be replaced (i.e. it is already part of the existing system), NVEBILD 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 NVEBILD will match everything up positionally. If these parameters are not specified, and NVEBILD 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 NVEBILD 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. NVEBILD 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>,<l2>)
   where
   - m = a 7-character left-justified module name
   - c = a 1-character compilation option ('0','1', or '3'; 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.
   - l2 = a 7-character left-justified secondary library destination 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>.

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

2.0 OVERVIEW OF INTEGRATION PROCESS
2.5 NVEBILD PROCEDURE DESCRIPTION

The format of the NVEBILD is as follows:

```
SES.NVEBILD [ m=(<module name>..<module name>) ]
            [ l=< library name > ]
            [ c=(<compilation option>..<compilation option>) ]
            [ area = < user name > ]
            [ xref=(< xref option>..<xref option>) ]
            [ 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(s) onto which a newly compiled module (or modules) is (are) to be added or replaced. If the 'M' parameter has not been specified, then the entire library is recompiled. To recompile several libraries at a time it is recommended that the NVEBILF procedure be used.

c : c = 0 to assemble a module
c = 1 to compile a CYBIL module (DEFAULT)
c = 3 to compile a CYBIL module using CYBICMN type declarations

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.

lo : List options to be in force during CYBIL compilations. May be any combination of A, C, F, O, I, W, R, or 0 (zero). The default is 0 (zero).

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 NVELINK. 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 NVESIM). 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 NVEBILD in BATCH mode. The default is to run it locally.

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

2.5.1 NVEBILF PROCEDURE DESCRIPTION

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

The format of the NVEBILF is as follows:

```
SES.NVEBILF [ 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 NVEBILF in BATCH mode. The default is to run it locally.

Note : To rebuild one library, the following are logically equivalent:

1) SES.NVEBILF l=< library name >
2) SES.NVEBILD l=< library name > batch

The expansion of either of the above procedures can be prohibitively long when being run from a terminal. The 'batch' keyword on the NVEBILD procedure is implemented for the express purpose of freeing up the terminal for other purposes (the procedure expansion is done within the BATCHed job).

2.5.2 NVEBLD PROCEDURE DESCRIPTION

The NVEBLD procedure generates and routes to the input queue a set of NVEBILD jobs which compile the modified or replaced decks and those decks which call modified or replaced common decks. First, NVEBLD finds the decks modified by creating a list of all the *DECK lines in the 'mf' file(s) and a list of all the decks in the 'dkf' file(s). It combines the two lists, sorting and deleting duplicate names. The combined list is checked against the current list of modules which make up NOS/VE, using the cross reference file from the 'xrf' parameter. Then the procedure again creates two lists: a list of modules to be compiled and a list of common decks to be checked. A subset of the cross reference is used to generate a list of all decks referencing the given common decks. The list of modules and the list of decks referencing the modified common decks are combined, sorted and duplicates are removed. This final list is then used to generate the NVEBILD jobs to compile the necessary modules.

The format of the NVEBLD is as follows:

```
SES.NVEBLD  [ mf = <file_name> ]
            [ dkf = <file_name> ]
            [ xrf = <file_name> ]
            [ nl = <library_name> ]
            [ area = <user_name> ]
            [ ab = <file_name> ]
            [ batch ! local ! defer ! batchn ]
```

mf : The file or list of files which contains the modsets or a list of *DECK deck_names. If the parameter is omitted the file MODFILE is used.

dkf : The file or list of files which contains the new or replacement decks in a group file format. If the parameter is omitted the file DECKFIL is used.

xrf : The NOS file name for the file containing the cross reference of NOSVEPL. If the parameter is

omitted the file XNVEPL is used.

nl : The list of library names to be omitted from the build. There is no default.

area : The search order to find any file. If the parameter is omitted the user names of the current user and the user name of the procedure are used for the default user names.

ab : The file name of the alternate base. The default is NEWDKPL.

batch : local : defer : batchn : The job run mode of the procedure. If none are defined LOCAL is used.

2.5.3 LISTNVE 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 <u>not</u> 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.

2.6 NVELINK PROCEDURE DESCRIPTION

NVELINK 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.

NVELINK will link either a stand-alone or a dual state system. The choice is made via the OFFSET parameter (see description below). NVELINK 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 NVELINK 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 NVELINK is as follows:

```
SES.NVELINK [ offset = < load offset > ]
            [ ps = < page size> ]
            [ ptl = < page table length > ]
            [ save ]
            [ quick ]
            [ add = < additional links file > ]
            [ dump ]
            [ area = < user name > ]
            [ print ]
            [ test = < test file 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

ps : Page size of the target NOS/VE system, expressed in multiples of 1024 bytes. Values may be 1, 2, 4, 8, 16, 32, or 64. Default is 8.

ptl : Page table length of target NOS/VE system, expressed in multiples of 1024 bytes. Values may be 4, 8, 16, 32, 64, 128, 256, 512, or 1024. Default is 16.

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 NVELINK Procedure". The default is not to do a quick link.

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.

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.

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 NVESIM). The default is to not run the simulator test.

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

2.6.1 LPF FILE DESCRIPTION

The LINK commands used in the NVELINK 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.

2.6.2 VELDCM / LDR FILE DESCRIPTION

The VELDCM file used by the procedure NVELINK 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 NVELINK procedure, depending upon the specification of the OFFSET, PS, and PTL parameters. 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 NVELINK has been executed, as the procedure NVESYS uses this file in building a deadstart tape.

2.7 ADDING USER TASKS TO NOS/VE

2.7.1 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.

2.7.2 QUICK LINK OPTION OF NVELINK PROCEDURE

To do a "quick link", specify the QUICK option on the call to NVELINK (see the section entitled "NVELINK Procedure Description). In this case NVELINK 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 NVELINK 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.

NVELINK, with the QUICK option specified, should not require any modification to execute in different user task configurations. The user can merely add the user object text file produced by an assembly or compilation to the XLJBBB file using the SES.GOF procedure. To link additional user object files into the system, 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 "NVELINK Procedure Description".

2.8 NOS/VE SIMULATION

2.8.1 RUNNING A SIMULATOR TEST (NVESIM PROCEDURE)

NVESIM 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.

NVESIM 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 NVESIM 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 NVESIM 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

NVESIM 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 NVEKEY before being written to the file TOUTPUT.

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

Additionally, if the 'NCPF' parameter is specified, NVESIM 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 NVESIM is as follows:

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

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

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

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

2.0 OVERVIEW OF INTEGRATION PROCESS
2.8.1 RUNNING A SIMULATOR TEST (NVESIM PROCEDURE)

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)".

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.

run : A count of the number of suimulated instructions to execute. The default is 800000 instructions (or the profile variable value for 'RUNCNT').

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 NVESIM output. The default is not to dump the checkpoint file.

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

2.8.2 NVEKEY PROCEDURE DESCRIPTION

NVEKEY 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 NVEKEY is as follows:

```
SES.NVEKEY  [ kpf = < keypoint file > ]
            [ format = < SIM ! HDW > ]
```

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

format : Specifies whether simulator or hardware format keypoints are being processed. Default is 'SIM'.

2.8.3 DUMPING A SIMULATOR CHECKPOINT FILE (NVEDUMP PROCEDURE)

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

The format of the NVEDUMP is as follows:

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

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

i : 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 NVEDUMP in BATCH mode. The default is to run it locally.

2.9 BUILDING A DEADSTART FILE

2.9.1 INTRODUCTION

2.9.2 CREATING THE FILE (NVESYS PROCEDURE)

The SES procedure NVESYS 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.

NVESYS 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) - standalone only systems
4) PP helper (file XIHLP) - standalone only systems
5) PP Resident program (file XIRES) - standalone only systems
6) NOS/VE disk driver (file XDISK)
7) NOS/VE MCU Driver (file XMSPMCU) - standalone only systems
8) Dual State MCU Driver (file XMCUPP) Dual State only systems
9) 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 NVESYS is as follows:

```
SES.NVESYS  [ vsn = < tape vsn > ]
            [ offset = < load offset > ]
            [ cpf = < checkpoint file name > ]
```

[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 NVESYS in BATCH mode. The default is to run it locally.

2.9.3 COMPILING 180 PP CODE (CPP180 PROCEDURE)

CPP180 is an SES procedure file which compiles 180 PP code. The source for the PP code is retrieved from a source program library. 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 >]
[ab = < alternate base >]
[batch]

m : The module name of the PP program to be compiled.

ab : The alternate base searched by CPP180 to satisfy externals before searching NOSVEPL. The default is to search only NOSVEPL.

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

2.10 DUAL STATE PROCEDURES

2.10.1 BLDEI PROCEDURE DESCRIPTION

BLDEI is an SES procedure file which builds the absolute file for dual state EI. The AB parameter may be specified if a program library 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 'DSLIST' 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 > ]
            [ area = < user name > ]
            [ batch ]
```

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

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

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).

2.10.2 DSBILD PROCEDURE DESCRIPTION

DSBILD is an SES procedure file which builds the dual state binaries XDSTVE, XRUNVE, and XTRMVE. All assembly and CYBIL 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 appended to the compilation and assembly listings.

The format of the DSBILD is as follows:

```
SES.DSBILD  [ ab = < alternate base > ]
            [ batch ]
```

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

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

2.11 UTILITY PROCEDURES

2.11.1 NVEREP - REPORT SYSTEM CONTENT

NVEREP is a procedure which dynamically produces NOS/VE build content reports based upon build information contained in the Integration procedure library (PROCLIB), or that generated dynamically by partner procedures. The reports are sorted according to a user supplied primary sort key, and a procedure defined secondary key which is associated with the primary sort key. The amount of information contained on the Integration procedure library is limited by the SES Command Language processor to eighty characters, but the procedure is sufficiently generalized to work with expanded information produced by partner procedures. These partner procedures are not of a generalized nature so as to be documented at this time (primarily due to a series of deficiencies in the current CI tools and conventions).

The format of the NVEREP procedure is as follows:

```
SES.NVEREP  [ left = < primary key > ]
            [ right = < primary key> ]
```

```
                [ area = < alternate user name > ]                    ;
                [ f = < input source > ]                              ;
                [ o = < output destination > ]                        ;
                [ l = < library name > ]                              ;
                [ print ]                                             ;
                [ batch ]                                             ;

left :          Primary sort key for left side of two paged           ;
                report. Only the first two characters of this         ;
                parameter are significant. May be either              ;
                MOdule, MAdify, LIbrary, or LAnguage. (An             ;
                additional option, VErsion, is only available         ;
                when used in conjunction with partner                 ;
                procedures. Option BUild is under consideration       ;
                for future implementation.) The default for           ;
                this parameter is MOdule. No validity checking        ;
                is performed for either the 'left' or 'right'         ;
                parameter values, and an invalid specification        ;
                will result in a report which may differ from         ;
                that desired.                                         ;

right :         Primary sort key for right side of two paged          ;
                report. See parameter left for valid                  ;
                specifications. Default is MAdify.                    ;

area :          Alternate user name to search first for the           ;
                input file specified by the 'f' parameter.            ;
                Default value is 'null', and source for 'f' is        ;
                found in the user name where the procedure            ;
                resides (PRCUNAM).                                    ;

f :             Name of the file containing build content             ;
                information. Default is PROCLIB.                      ;

o :             Name of the file to receive the two paged             ;
                report. Default is VEREP. (The files LEFT and         ;
                RIGHT currently remain local after the procedure      ;
                has completed. These files contain the left and       ;
                right hand portions of the two paged report.)         ;

l :             Name(s) of NOS/VE library (or libraries) which        ;
                are to be included in the report. Default is to       ;
                report on all primary NOS/VE libraries.               ;

print :         Keyword to cause output file to be printed.           ;
                Default is to not print the report unless batch       ;
                execution has been selected.                          ;

batch :         Keyword to cause the batch execution of the           ;
```

procedure. Default is to run the procedure in 'LOCAL' mode.

2.11.2 PROCEDURE GET - GET A LOCAL FILE

The GET procedure makes some assumptions about files which prove convenient to the procedure writer. The first assumption is that any files named by the user represent files which are to be modified or interrogated without worry as to the access mode of the file (it may be either a DIRECT or an INDIRECT access permanent file). The second assumption is that the file named by the user may already exist as a local file, and if it does that the current local copy of that file is the correct version of that file. The procedure's function then is to leave currently existing local files alone (unless they are attached DIRECT files), and find any remaining files to add them to the list of currently active local files. The user may wish to specify a search order for the list of files to be added to the local file list, and may do so by providing a list of user names (in search order, left to right) via the 'UN' parameter. The user may also wish to accumulate a number of files upon one filename, and may do so by providing a filename for the 'MF' parameter.

The format of the GET procedure is as follows:

```
SES.GET     [ < lfn > ] ; [ < pfn > ]
            [ <un> ]
            [ < mf > ]
            [ msg ]
```

lfn : Local file name by which the file is to be known (may be a list of files). If no filename is specified for lfn, then the filename value for the 'PFN' parameter is used. One of either 'LFN' or 'PFN' must be specified. All parameters may be specified positionally, so the typical usage of this procedure would be:
SES.GET (MYFILE1,MYFILE2, ... ,etc.)

pfn : Permanent file name of the file to be made local. If no filename is specified for pfn, then the assumed value is that specified by the 'LFN' parameter. The list of pfn values is matched positionally with the 'LFN' specified values. This is illustrated as follows:
SES.GET (one,two) (stuff1,stuff2)

un : An optional list of user names which directs the procedure's search order for files which are currently not local. This is convenient if the user knows that the file exists in one of several catalogs, but isn't sure where to look. An example would be:
SES.GET ipndoc UN=(int1,int2,dev1,dev2,rel1)

mf : A filename upon which to stack the named files (one file per NOS record). For example:
SES.GET (ipndoc,bcr) MF=bildoc

msg : Keyword which causes the generation of messages to describe what the procedure is doing. If for example the location of a file is in several user catalogs, the msg keyword would issue messages indicating which catalogs (specified by the 'UN' parameter) had been searched for the file, and where it had finally been found.

2.11.3 PROCEDURE SAVE - MAKE A LOCAL FILE PERMANENT

Procedure SAVE may be used in conjunction with the GET procedure. The redeeming factor about the SAVE procedure is that the user need not be concerned about file size. The named local files are made permanent as INDIRECT access files, if possible, otherwise DIRECT access files. Files are SAVE'd as SEMI-PRIVATE, READ-ONLY files. This is intentionally done to preserve as much precious disk space as possible. (NOS allocates DIRECT access files much less frugally than INDIRECT access files.) Be forewarned that a slight penalty is imposed in access time for each sector of disk space saved in this manner, and that the actual sector savings is only visible from the operator's console (not via CATLIST). In the procedure writer's world of living, this procedure negates the worry of predicting file size prior to creating it.

The format of the SAVE procedure is as follows:

```
SES.SAVE    [ <lfn> ] : [ <pfn> ]
            [ <un> ]
            [ msg ]
            [ dir ]
```

lfn : Local filename to be made permanent. Defaults to 'PFN' value if not specified. One of 'LFN' or 'PFN' parameters must be specified. Parameters may be specified positionally, and

typical usage would be:
SES.SAVE (xlmmtr,xljlib, ... ,etc.)

pfn : Permanent filename for files to be saved. Defaults to 'LFN' if not specified. PFN values are matched positionally within the list to LFN values. This is illustrated as follows:
SES.SAVE (one,two) (stuff1,stuff2)

un : User Name in which to save the local file. This parameter is only valid for DIRECT access files for which write permission has been granted (in another user's catalog).

msg : Keyword which causes the generation of messages describing the names and destinations of files being saved.

dir : Keyword which directs the procecure to make all named files DIRECT access files, regardless of their size.

2.11.4 NVEMAP - REFORMAT NOS/VE LINKMAP

NVEMAP is a procedure to reduce the number of printed pages of a NOS/VE linkmap, while maintaining readability, and to provide summary reports of information contained wihtin the linkmap. Either all, or portions of the linkmap may be processed. The reformatted form of the linkmap is also suitable for microfiche, in the format defined for the NOS/VE operating system.

The format of the NVEMAP procedure is as follows:

```
SES.NVEMAP  [ i = < input file > ]
            [ o = < output file> ]
            [ area = < alternate user name > ]
            [ copy = < record count > ]
            [ skip = < record count > ]
            [ print ]
            [ gated ]
            [ fiche ]
            [ module ]
            [ save ]
            [ two ]
            [ batch ]
```

i :	Input file which contains generated output from the execution of the NOS/VE CI (or SES) Linker. Default is MAP256K for systems prior to the system restructure (Prior to Build 0), and MAPHXX for restructured systems.
o :	Name of the output file to receive the reformatted linkmap file. Default is to produce a local file of the same name as specified by the 'i' parameter.
area :	Alternate user name to search for the input file specified by the 'i' parameter. Default value is 'null'.
copy :	Count of the number of NOS records to process from the current file position of the input file (default position BOI). Each invocation of the linker produces a new record upon the output file. Thus, to process only the first portion of the linkmap (typically Monitor for the NOS/VE Operating System) 'COPY=1' would be specified. Default value for this parameter is to process the entire linkmap BOI to EOI.
skip :	Count of the number of NOS records to skip prior to processing. For the NOS/VE operating system 'SKIP=2' would suppress the Monitor and EI portions of a Dual State linkmap. 'SKIP=2 COPY=1' would process only the Task Services portion of a Dual State linkmap. Use of either the 'skip' or 'copy' parameters infers explicit knowledge of the content the linkmap. Due the the number of variations of linkmap which can be produced it would be impractical to generalize these parameters in a more logical manner.
print :	Keyword to cause output file to be printed. Default is to not print the reformatted linkmap. 'PRINT=TWOMAP' will print the contents of file named TWOMAP. Key 'PRINT' will print TWOMAP if key 'TWO' is also specified, otherwise the file specified by the 'o' parameter will be printed. 'PRINT=ALL' will print both TWOMAP and the file specified by the 'o' parameter.
gated :	Keyword which eliminates information for all entry points which do not have the GATED attribute. Conceivably, a combination such as

'SKIP=2 COPY=1 GATED' would produce information to a compiler project about which entry points within Task Services are GATED for their use. Default is to produce reports of all system entry points.

fiche : Directs the procedure to place the output of the procedure onto the file NOSLIST for subsequent microfiche processing. Default is to not add the linkmap to the NOSLIST file.

module : Keyword which causes the removal of all entry point information from the linkmap. This proves useful for auditing module attributes. The default is to retain all system entry point information.

save : Keyword which causes the output files to be retained on permanent files for subsequent inspection. This option is selected if 'batch' has been specified, otherwise it is left to the discretion of the user to dispose of the local copies of the output files.

two : Keyword which directs the procedure to twopage the linkmap onto the file TWOMAP. TWOMAP will always be generated, but will only contain the summary report information unless 'two' is specified. This twopage option is not the familiar SES TWOPAGE option, but rather a computed split and merge of the reformatted map.

batch : Keyword to cause the batch execution of the procedure. Default is to run the procedure in 'LOCAL' mode.

This procedure will always produce two output files. The primary output file is governed by the 'o' parameter. A secondary output 'TWOMAP' is always produced as well. The 'TWOMAP' file will only contain the summary reports and loadmap if parameter 'two' is not selected. The first summary report which is produced is a two paged report of PVAs found in the linkmap. The left hand portion of this report is a sort by PVA. The right hand portion of this report is a sort by module and/or entry point name. This report should answer the questions: 1. Given a PVA, in which module and/or procedure is that PVA contained?, 2. Given an entry point name, in which module is it defined and what is its PVA?, and

3. Given the name of a variable within a system defined table, what is its location within a dump?

The final report is a two part error summary for the linkmap. The first portion of this report identifies which pages of the linkmap contained one or more errors. The second portion of this report is a list of all of the errors found within the linkmap, in the sequence in which they appear in the map.

3.0 DUAL STATE INSTALLATION SEQUENCE

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

If HIVS and CTI 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 HIVS and CTI were used.

3.1 CLEAR_POINTERS_AND_INSTALL_CTI

To clear the pointers, deadstart from the Dual State NOS Deadstart tape which is NT, D=PE, F=I, LB=KU, and enter:

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

Deadstart again from the same deadstart tape as above and then enter:

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

3.2 INSTALL_HIVS

To install HIVS, deadstart from the HIVS tape which is NT, D=PE, F=SI, LB=KU, and then enter:

```
*U*--->*T* (Select TDX option)
```

3.0 DUAL STATE INSTALLATION SEQUENCE
3.2 INSTALL HIVS

```
'C' option (Build directory in CM and copy to disk)
Disk type=x
Disk Channel=xx
Disk Eq=xx
Disk Unit=xx
MT type=3
MT channel=xx
Eq=xx
Unit=xx
User type=02 (Shared O/S)
Install options=02 (Edited MSL)
Options=01 (Build without D/S sector since it was
                already created by CTI.)
```

When END appears, HIVS is installed.

3.3 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.) Hit carriage return. The system will display:

```
ENTER LOCATION
OF MSL/HIVS DEVICE
```

Enter the information for the same disk where HIVS was installed previously:

```
Channel=xx
Equipment=xx
Unit=xx
```

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 HIVS was installed previously.

3.4 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.

Deadstart from the disk upon which the NOS Dual State system was installed and LOADPF the files to the desired user number.

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

*RELOADCH=x. x should be an empty channel.
*DISKCH=y1,y2. NOS cannot use channel y1 or y2 with Dual State active.
*SYST,DISKUz=ON. z can only be 0 through 3.
(Unit z on channel y1 or y2 cannot appear in the NOS CMRDECK or EST.)

3.5 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. followed by K.*ENDRUN. 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 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:

```
LOGIN USER=DEV NAME=TESTBAM
LIU,USER=(DEV1,NVE),PA=DEV1X,A=NOTUSED,PR=NOTUSED
GET,SYSLIB,SYSLIB,,,NVE,B56
GET,CYBILIB,CYBIILB,,,NVE,B56
SET_OBJECT_LIST ADD=SYSLIB
EXECUTE,,,,,BAMTEST
TES1
TES2
TES3
TES4
TES5
TES6
TES7
TES8
TES9
```

4.0 NOS/VE HARDWARE REGRESSION TESTING
4.2.1 TESTBAM

```
TES10
TES11
TES13
TES14
TES15
TES16
TES20
BAMSTOP
GET,SCL180,SCL180,,,NVE,B60
SUBMIT,SCL180
JMEXIT
```

4.2.2 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:

```
LOGIN USER=DEV NAME=JOB
LIU,USER=(DEV1,NVE),PA=DEV1X,A=NOTUSED,PR=NOTUSED
GET,CITEXT180,CYBILGO,,,NVE,B60
EXECUTE,,'CITEXT180,IITEXT180',,,CITOII
EXECUTE,,,,,COL
ADD,F=IITEXT180
DIS,ON=ALL
GENERATE,LIBRARY=LIBRARY180
QUIT
REPLACE,LIBRARY180,CYBIILB,,,NVE,B56
GET,JOB2,JOB2,,,NVE,B60
SUBMIT,JOB2
JMEXIT
```

4.2.3 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_LIST command
- SET_PROGRAM_OPTIONS command
- GETPF B56
- GETPF B60
- CITOII conversion
- Load/Execute User Program + Library
- JMROUTE C180 print file
- JMEXIT

The command sequence follows:

```
LOGIN USER=DEV NAME=JOB2
LIU,USER=(DEV1,NVE),PA=DEV1X,A=NOTUSED,PR=NOTUSED
GET,NEWLIBRARY,CYBIILB,,,NVE,B56
SET_OBJECT_LIST,ADD=NEWLIBRARY
SET_PROGRAM_OPTIONS,MO=(BLOCK,ENTRY_POINT,XREF,SEGMENT),...     <*
TERMINATION_ERROR_LEVEL=FATAL
GET,XPETEST,XPETEST,,,NVE,B60
EXECUTE,,'XPETEST,LGO',,,CITOII
EXECUTE LGO
JMROUTE,NOTUSED,LOADMAP,PR
GET,CYBILIB,CYBIILB,,,NVE,B56
SET_OBJECT_LIST,DELETE=ALL
SET_PROGRAM_OPTIONS,TERMINATION_ERROR_LEVEL=ERROR
EXECUTE LGO
JMROUTE,NOTUSED,LOADMAP,PR
GET,TESTBAM,TESTBAM,,,NVE,B60
SUBMIT,TESTBAM
JMEXIT
```

4.3 S2_REGRESSION_TEST_SEQUENCE

1) Mount the disk labeled "DAHL-Large sector" on 844 Unit 0. Other disks will not work.

2) Deadstart A170 NOS:
   - Set the deadstart panel to disk deadstart from: CH=1

```
            UNIT=41
            WORD 13=0006
          - Push deadstart button.
          - Select *D* display.
          - Select *H* display.
          - Enter CM=10000
          - Hit carriage return.
          - Enter date/time.
```

3) If necessary, update the INT2 catalog and load the latest system files into the INT1 catalog:

```
          - Mount the INT1 and INT2 catalog DUMPPF tapes.
          - X.DIS.
            USER,INT1,INT1X.
            SES.UPCATS <INT1=tpu1> <INT2=tpu2> <SYSEDIT>
            Hit "." to go into AUTO mode.
            DROP.
```

The SES.UPCATS procedure works as follows:

a) Updates the INT2 catalog by retrieving the "fast files" (CMIMAGE, PPIMAGE, RGIMAGE) and CYBIILB from the INT1 catalog and by loading selected files from the DUMPPF tape mounted on the unit specified by the "INT2" parameter. This parameter must be specified for the INT2 catalog to be updated.

b) LOADPF's the latest system into the INT1 catalog from the DUMPPF tape mounted on the unit specified by the "INT1" parameter (defaults to "50").

c) SYSEDIT's the A170 Remote Host and Interactive binaries if the "SYSEDIT" keyword is specified.

4) Bring up dual state:

```
     X.UPMYVE(CAT=INT1)
     K,n.  (where "n" is the UPMYVE control point)
```

5) 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 0.
     K.DM P 100.
     ->  If system is hung at this point then paging is
     not working.
     K.SMCLOSE P.
```

6) Bring up A170 Remote Host and Interactive:

```
     TAFNVE.
```

7) Bring up C170 Remote Host:

9) Test input file route / job exit / output file route:

```
X.DIS.
USER,INT1,INT1X.
GET,TESTBAM,JOB1,JOB2.
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 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 all jobs have finished executing and their dayfiles have printed, then NOS/VE and Remote Host may be terminated by doing the following:

```
2.STOP.  (bring down TAFNVE)
K.*BYEVE.
```

When the K-display displays the message that NOS/VE has terminated, type:

```
K.*ENDRUN.
```

10) Bring down A170 NOS:

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

4.4 THE CONFIDENCE TESTS

The Confidence testbase contains a set of tests which are to be run after each build. These tests are created by the GENTEST procedure. This procedure MUST be rerun after any change to OSLPI or the CI CYBIL compiler. The GENTEST procedure is designed to be run on S/N 101, but can be run on the S2.

4.5 CONFIDENCE TEST SETUP INSTRUCTIONS

Do this setup procedure when the OSLPI, the CI CYBIL compiler, or the confidence testbase changes. The GENTEST procedure may be run on the NOS state of the CYBER 180 or on any NOS system with the lastest CI CYBIL compiler, OSLPI, and the confidence testbase PL(CONFPL).

4.5.1 FILES NEEDED TO RUN THE SETUP CONFIDENCE PROCEDURES

To run the confidence test setup procedures the following files are needed:

```
CYBILI/UN=INT1
OSLPI/UN=INT1
CONFPL/UN=INT1
CYBILC/UN=DEV1
```

4.5.2 RUN ON A NONE CYBER 180

1. Login to the same user as the test are to be run under on CYBER 180.
2. Enter SES,INT1.GENTEST (BA001..GENCVRT)
   (See section on GENTEST for more details.)
   The test will generate a set of Xfiles containing the binary file for each job, also the CYBER 180 job will be appended to the 180 load file(LD180). If it is A170 and 180 interacting test the jobs will be appended to A170 load file(LD170).
3. Dump the files to tape, use the SES.DUMPPF procedure.

You will then be ready to run the test on the CYBER 180 after loading the dump tape.

4.5.3 RUNNING ON A CYBER 180 WITH NOS/VE UP

After loging in to the user which the testbase is to be run under do the following:

```
SES,INT1.GENTEST (BA001..GENCVRT) CHAROFF
    (See section on GENTEST for more details.)
```

4.6 CONFIDENCE TEST OPERATOR INSTRUCTIONS

1. Deadstart the Dual State NOS system.
2. Load files into the INT1 catalog, if necessary:

```
vsn,<tape unit>,"vsnname".
x.dis.
user,int1,int1x.
     ENTER
request,dump,nt,d=pe,f=si,lb=kl,po=r,vsn="vsnnam".
loadpf.
drop.
     OR ENTER (can be done from a terminal also)
ses,int1.loadpf vsn="vsnnam"
```

3. Bring up NAM.
4. Login,int1,int1x.
5. Bring up VE and IRHF.
6. Run 180 tests(from terminal).

```
SES.LDTEST
      (See section on LDTEST for more details.)
```

7. To get 170 listings.

```
on32.
Form32,.
```

8. To get 180 listings.

```
on32.
Form32,tm.
```

4.7 CONFIDENCE TESTBASE PROCEDURES

4.7.1 GENTEST

This procedure will generate a set of A170 and/or C180 load files from a testbase program library (PL). The default PL is the confidence testbase on UN=INT1.

4.0 NOS/VE HARDWARE REGRESSION TESTING
4.7.1 GENTEST

The format for GENTEST is as following:

```
SES,INT1.GENTEST test = <test_name>
                [ base = <file_name> ]
                [ ld170 = <file_name> ]
                [ ld180 = <file_name> ]
                [ un = <user_name> ]
                [ lc = <leading_character> ]
                [ norun ]
                [ charoff ]
```

test ; deck ; d ; t ; m : The test which will be generated from the base PL. This parameter is required for running of this procedure.

base ; b ; pl ; b : The file name containing the confidence testbase. The default is the file name of CONFPL.

ld170 ; l7 : The file to get the 170 and 180 interacting tests. The default file name is LD170.

ld180 ; l8 : The file to get the 180 tests. The default file name is LD180.

un : The user name which contains the CONFPL file. The default is the owner of the procedure.

lc : The leading character of the confidence testbase PL. The default is a /.

norun ; nr : This keyword when used will create a load file on the LD170 file only. The default is not to use this parameter.

charoff ; choff : This keyword when used will run the tests. These keyword when used without the NORUN keyword should only be used when on a Dual State NOS system and NOS/VE is up. The default is not to use this parameter.

Example:

```
SES,INT1.GENTEST (BA001..GENCVRT)
```

(This will cause all the A170 jobs to be run and the C180 jobs to be saved on the LD180 file. Also the A170 and C180 interacting tests to be saved on the LD170 file.)

4.0 NOS/VE HARDWARE REGRESSION TESTING
4.7.1 GENTEST

```
SES,INT1.GENTEST (BA001..GENCVRT) CHAROFF
    (This will generate the complete testbase creating
    the A170 job which will route in a C180 job when the
    supporting binaries are created. This example will
    only work if running on a dual state NOS system and
    NOS/VE are up.)

SES,INT1.GENTEST (BA001..GENCVRT) CHAROFF NORUN
     (This will generate only the LD170 load file for the
     LDTEST procedure. The A170 part of the test will
     route the C180 job after the supporting binary files
     are created on the dual state NOS system.)
```

4.7.2 LDTEST

This procedure will load a set of tests from two load files (one file containing only C180 jobs and the other file containing jobs interacting tests between the A170 and the C180). The procedure has three speeds of loading (fast, normal, and low). It also has the option of reloading all the tests as many times as wanted.

Format of LDTEST is as follows:

```
SES,INT1.LDTEST [ ld180 = <file_name> ]
          [ ld170 = <file_name> ]
          [ speed = <high ! normal ! low> ]
          [ loop = <loop_count> ]
```

ld180 ! l8 : The file containing the CYBER 180 load file produced by the GENTEST procedure. Default is LD180 on the current user catalog.

ld170 ! l7 : The file containing the CYBER 170 load file produced by the GENTEST procedure. Default is LD170 on the current user catalog.

speed ! sp : The speed at which the tests will be loaded in to the system. The procedure will only test the first letter to get the value. The allowable values are F or H for high speed, M or N for normal speed, or any other letter for low speed. The default is normal speed.

loop ! lp : The number of times the complete testbase will be loaded into the system. The default is only one set of tests.

Example:

SES.LDTEST
(This will load one complete set of tests in to the system with a normal wait between loading each test job.)

SES.LDTEST SP=H LOOP=3
(This will load three complete copies of the confidence testbase with a one second rollout between loading jobs. This can be uses to load down the system.)

4.8 CONTENTS OF THE CONFIDENCE TESTBASE

BAM Tests
- BA001 Issue and verify amp$file and amp$get_file_attributes.
- BA004 Issue and verify amp$get_file_attributes and amp$fetch for default and modified attributes.
- BA023 Verify record movement, data integrity and file access changes.

Condition Handler Tests
- CH001 Test pmp$continue_to_cause from specific handler to a handler for all system conditions.
- CH006 Produce various error codes connected with condition handler procedure calls.
- CH008 Test that the appropriate error code is generated when pmp$continue_to_cause is called.

SCL Tests
- CL016 Test clp$declare_variable and clp$remove_variable with Test clp$remove_variable for an undeclared variable and for a task variable.
- CL017 Test string length boundaries in clp$declare_variable.
- CL018 Test clp$declare_variable dimension boundaries and dimension with all command languages.

Loader Tests
- LD001 Test static and global pointer initialization.
- LD002 Test XDCL procedure by using large array parameters.
- LD003 Test linking a user task to procedures in task services.
- LD004 Test linking a user task to procedures in task

services.

Remote Host Tests
- RH005 Test replace existing file with no conversion involved.
- RH006 Test replace existing file with conversion involved.
- RH009 Test replace non-existing file with no conversion involved.
- RH010 Test replace non-existing file with conversion involved.

GENCVRT Set up for remote host tests.

4.9 CONFIDENCE TESTBASE MAINTENANCE

When new tests are added or old tests are removed from the confidence the following maintenance is needed:

This document the new test description must be added or the old description must be removed.

The CONFPL the new deck must be added (see section on maintenance of the CONFPL) or the old deck removed.

The GENTEST procedure must have the new deck name added or the old deck name removed. See the section on maintenance of the GENTEST procedure.

4.9.1 CONFPL MAINTENANCE

When adding a new test to the confidence testbase the test should have the following call:

/CALL USER170 after the A170 job card

/CALL TO180 the last card in the A170 job

/CALL USER180 after the C180 LOGIN

The A170 job should NOT have any of the following commands:

SES.PRINT

ROUTE,file,DC=PR,FC=HR.

DEFINE, SAVE, or REPLACE

4.0 NOS/VE HARDWARE REGRESSION TESTING
4.9.1 CONFPL MAINTENANCE

The SAVE, REPLACE, or DEFINE should be replaced be a SES,INT1.SAVEPF call before the /CALL T0180.

4.9.2 GENTEST PROCEDURE MAINTENANCE

As new test are added to the testbase the GENTEST procedure is the only procedure which will needed any maintenance. The section of the GENTEST procedure which will change is the test names. The following is lines are those which will be changed:

```
='BA001RUN'
     .
     .
     .
='RH005SAVE'
     .
     .
     .
='GENCVRTRUN'
=n
```

The RUN parameter is for any job which has an A170 job to produce a file for the C180 test.

The SAVE parameter is for A170 and C180 interacting tests.

The n is the count of the last job which can be loaded.

(1) NOS/VE Transmittal Form

Originator_____ DATE ___/___/___ Target Build _____

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

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

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

(3) Code affects system user. yes _____ no _____

Usage Changes Description File: FN=_____ UN=_____

Modset Identifier(s) _____

Code Destination (if not NOSVEPL): PL= _____

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

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

PSR(s) Answered_____

COMMENTS TO INTEGRATION

(4) Installation procedure changes required? [____]

(4) Dependent upon other feature, fix, or tool? [____]

(4) Documentation changes required? [____]

(4) OSLPI or Internal Interface changes required? [____]

Module(s) to be recompiled _____

Has this code been tested? yes _____ (4) 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) Indicate whether this code implements changes that will affect a system user. Specify the location of the file containing descriptive text describing how the user is affected. Attach copy of this file to form (both 14 7/8 by 11).

(4) If any of the above are checked, then explain below.

Code Reviewer _____ Approval _____

07/29/81

Request Number _____

SOFTWARE CHANGE REQUEST

Requestor_____ DATE ___/___/___ PSR Number _____

Configuration _____ Target Entry Date ___/___/___

Software Needing Change

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

07/29/81

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

07/29/81

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 NVEBILD is used for EIE.
INT1 DEV1	NEWDKPL	Meaningless Madify program library which users may substitute for as an alternate base when using Integration compilation	Never, disappears when SCU conversion is complete.

		procedures.	
INT2/INT1 DEV1	MAP<offset>K	Contains link map of Dual State system created, where MAP0K 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.

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.

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

Files maintained by Integration

07/29/81

Object Text Files

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

BUILD N HELPFUL HINTS

This paper describes helpful hints on how to use build N of NOS/VE. It is intended to supplement, rather than to replace, the standard NOS/VE documentation. If you have any qustions or suggestions, please see Tom McGee or Geoff Barrett. Appendix A lists background documents and how to obtain them.

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

```
GET,NHINTS/UN=GSB
SES.FORMAT NHINTS TXTFORM
```

Update History

Date	Changes
12/22/80	Revisions for NOS/VE Phase C
2/12/81	Additional revisions for NOS/VE Phase C
6/9/81	Revisions for NOS/VE Build N
6/19/81	Additional revisions for NOS/VE Build N

E1.0 MAJOR CHARACTERISTICS OF THIS BUILD

E1.0 MAJOR CHARACTERISTICS OF THIS BUILD

- The CYBIL/CI compiler has been modified to support the SIS conformance procedure calling sequence. In conjunction with this change, the NOS/VE object text (V1.2) has been revised to support building PVAs with a negative byte offset. The result of this is that all NOS/VE object files and object libraries produced prior to build N are incompatible with the build N object code utilities, loader and compilers. Therefore, all users of CYBIL/CI and NOS/VE build N and build O must recompile their modules with the new compiler and reconvert the object files and rebuild their NOS/VE object libraries. All code generators should be upgraded to the new object text definitions.

  Any attempt to use the previous versions of the object text on build N will result in the appropriate processor issuing a diagnostic.

  NOTE: SES tools and the SCITOII and CITOII converters will continue to support the non CYBIL/CI V1.1 object text.

- DO NOT assign the loader map file to the terminal in interactive jobs (i.e., file 'OUTPUT' or '$OUTPUT'). This will cause a system crash. File names are assigned to the map file via SET_PROGRAM_OPTIONS, EXECUTE, and PMP$EXECUTE (program description).

- Permanent Files now work. NOS/VE permanent files are permanent only until a crash or deadstart.

  The files JEDN, SYSLIB and CYBILIB that everyone had to GET from the NOS should now be attached as NOS/VE permanent files as follows:

```
ATTACH .USER1.SYSLIB FAMILY=FAMILY1
ATTACH .USER1.CYBILIB FAMILY=FAMILY1
ATTACH .USER1.JED FAMILY=FAMILY1 " For editor "
```

  However users who use the SETUP, CITOII and COL commands (which we strongly recommend) should never have to attach these two files. If it is necessary to do so, be aware that the names

E1.0 MAJOR CHARACTERISTICS OF THIS BUILD

FAMILY1, USER1 will be changing to $SYSTEM, $SYSTEM during build 0.

- A temporary procedure is available that will perform several functions that will be handled by validation and permanent files in the R1 system. The procedure is named SETUP, and should be called first thing after gaining access to NOS/VE in interactive jobs and immediately after the LOGIN command in batch jobs.

- A parameter has been added to the AMP$OPEN request. See 'Program Interface Status - File Management' for further information.

- Issuing the LOGIN command will cause a permanent file catalog to be created for the current user/family. Do:

  ```
  LOGIN user=xxx
  ```

  Interactive jobs have family=NVE and batch jobs have family=REMOTE. An example:

  ```
  LOGIN user=ZZZ
  DEFINE .ZZZ.A
  ```

  For an interactive job creates:

  ```
  .ZZZ.A FAMILY=NVE
  ```

  For a batch job it creates:

  ```
  .ZZZ.A FAMILY=REMOTE
  ```

  Note that the current user name may be specified by the $USER construct -- as in:

  ```
  ATTACH $USER.A
  ```

  LOGIN should be the first command issued in a batch job; followed by SETUP. For an interactive job, the first NOS/VE command issued should be SETUP since interactive users login to the NOS NAM.

- Job and standard files, as documented in the 'Command Interface ERS, Rev.8', are available.

E1.0 MAJOR CHARACTERISTICS OF THIS BUILD

- For interactive jobs, files INPUT, OUTPUT and COMMAND are connected (assigned) to the terminal automatically for every task in a job. The RMP$REQUEST_TERMINAL request must be used to assign files other than COMMAND, INPUT, OUTPUT to a terminal device.

- You can cause SCL to output the commands being executed from a procedure or include file by doing:

  ```
  connect_file $ECHO $OUTPUT
  ```

- The NOS/VE editor is available on (B56) file JEDN/un=IFP. The editor can be accessed by the EDIT nonstandard command. If you want more information about the editor, contact Jack Bohnhoff.

- The REWIND and RETURN commands are available. These commands accept NAME rather than FILE REFERENCE parameters; this should only present a problem in an SCL procedure. All FMxxx commands (like FMRETURN) have been removed:

  ```
  RETURN A
  RETURN (A,B,C)
  ```

  Files cannot be returned if they are open.

- Files <u>will</u> be returned at job termination. Files will also be closed at task termination.

- The procedure CLP$GET_STND_INP has been deleted. Procedure CLP$PUT_STND_OUT will be deleted in Build O or P. Everyone should convert to use BAM.

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

- The file(s) specified by the FILE parameter on the EXECUTE command may not be object library files.

- The program option PRESET has no effect since the file system does not yet support presetting.

E1.0 MAJOR CHARACTERISTICS OF THIS BUILD
E1.1 NOS/VE USAGE EXAMPLES

E1.1 NOS/VE USAGE EXAMPLES

E1.1.1 EXECUTING PROGRAMS

LIMITATIONS

- The file(s) specified by the FILE parameter on the EXECUTE command may not be object library files.

- The program option PRESET has no effect since the file system does not yet support presetting.

- If you are an interactive user, DO NOT cause the map to be written to the file 'OUTPUT' or '$OUTPUT' (SET_PROGRAM_OPTIONS or EXECUTE) - doing so will crash the system.

PROCESS

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

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

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

    or

  - o Creating the library file via the object library generator

    or

  - o Staging the library file from NOS to NOS/VE using the GET_FILE command (with B56 conversion mode specified).

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

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

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

  - o The REPLACE_FILE command with A6 conversion mode specified

E1.0 MAJOR CHARACTERISTICS OF THIS BUILD
E1.1.1 EXECUTING PROGRAMS

if running on the simulator.

or

- The PRINT command if running on the hardware.

EXAMPLES

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

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

```
SETUP JLN JLNX
CITOII CITXTRS
EXECUTE CITXTRS PARAMETER='program parameters'
PRINT LOADMAP
```

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

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

```
SETUP JLN JLNX
GET_FILE MYLIB C=B56
SET_PROGRAM_OPTIONS MAP_OPTIONS=(B,E,X,S)
CITOII CITXTRS
EXECUTE CITXTRS 'program parameters' LIBRARY=MYLIB
PRINT LOADMAP
```

E1.1.2 CREATE OBJECT LIBRARY ON NOS/VE AND SAVE IT ON NOS

Notes:

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

E1.0 MAJOR CHARACTERISTICS OF THIS BUILD
E1.1.2 CREATE OBJECT LIBRARY ON NOS/VE AND SAVE IT ON NOS

- ILIB170 is NOS permanent file name for file containing the library.

NOS/VE Job Commands

```
SETUP JLN JLNX
CITOII CLGO170 IITEXT180
COL
ADD LIBRARY=IITEXT180
GENERATE LIBRARY=LIBRARY180
QUIT
REPLACE_FILE LIBRARY180 ILIB170 C=B56
```

E1.1.3 MODIFY A PREVIOUSLY SAVED OBJECT LIBRARY

Notes

- ILIB170 is NOS permanent file name for file containg the old library
- LIBRARY180 is NOS/VE local file name for file containing the old library
- CMOD170 is NOS permanent file name for file containing CI object text for the new module
- NEWIIMODULE is NOS/VE local file name for file containing II object text for the new module
- NEWLIBRARY is NOS/VE local file name for the library being created
- NLIB170 is NOS local file name for new library

NOS/VE Job Commands

```
SETUP JLN JLNX
GET_FILE LIBRARY180 ILIB170 C=B56
CITOII CMOD170 NEWIIMODULE
COL
ADD LIBRARY=LIBRARY180
REPLACE FILE=NEWIIMODULE
GENERATE LIBRARY=NEWLIBRARY
QUIT
REPLACE_FILE NEWLIBRARY NLIB170 C=B56
```

E1.0 MAJOR CHARACTERISTICS OF THIS BUILD
E1.1.4 ROUTE AN INPUT FILE FROM NOS TO NOS/VE

E1.1.4 ROUTE AN INPUT FILE FROM NOS TO NOS/VE

Running from an interactive terminal, enter:

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

The input file which is sent to NOS/VE must be in 6/12 ASCII (or display code subset). The job file must be a single partition NOS record containing NOS/VE commands. Multi partition input files are not supported by NOS/VE so NOS data files used by the program must be obtained through the GET_FILE command.

E1.1.5 PRINT A NOS/VE FILE

At NOS/VE job termination the job log will be automatically returned to NOS. The job log will be appended to the NOS/VE output file OUTPUT. NOS/VE print files must be written by BAM as 8/8 ASCII RT=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 NOS. Support for ASCII print files (8/12 ASCII) will be added at a later build. All NOS/VE output files will appear in the NOS output queue (NOS H,O display) with the name IRHFxxx as a banner. In order to route a NOS/VE print file to NOS, the following command must be contained in the NOS/VE 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 NOS/VE output queue.

filename - name of the local NOS/VE 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 NOS/VE family for the print file (must always be REMOTE).

Example of JMROUTE command:

```
JMROUTE,LISTING,LINKMAP,PR,REMOTE.
```

E1.0 MAJOR CHARACTERISTICS OF THIS BUILD
E1.1.5 PRINT A NOS/VE FILE

The SCL PRINT command can also be used to print files. See the command ERS for details.

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

```
ON32.
FORM32,TM.
```

E2.0 COMMAND INTERFACE STATUS

E2.0 COMMAND_INTERFACE_STATUS

E2.1 ACCESS_TO_NOS/VE_IN_DUAL_STATE

E2.1.1 LOGIN TO NOS/VE

To initially login to NOS/VE via TAF, you must cause the first login attempt to fail. This can be done by responding to the "FAMILY:" login 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 login 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:".

E2.1.2 TERMINAL USAGE

1) The question mark (?) is the 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.

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

3) Terminal breaks (control-T and control-P) now work. It is possible to terminate a task or to suspend a task and enter a new task to process SCL commands. When a break is entered, the options available to the terminal user are:

   R - resume from point of interruption

   T - terminate task; n.b. commands running in the job monitor

E2.0 COMMAND INTERFACE STATUS
E2.1.2 TERMINAL USAGE

task (e.g. COPY, EXECUTE etc.) cannot be terminated.

S - process SCL commands. To resume execution at the point of interruption use the HCS command TMEXIT (temporary).

E2.1.3 NOS/VE PROGRAM ACCESS TO THE TERMINAL

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

E2.2 COMMAND FUNCTIONS

Function	Status
$MOD	unchanged
$CHAR	unchanged
$CLOCK	unchanged
$DATE	unchanged
$FNAME	unchanged
$INTEGER	unchanged
$NAME	unchanged
$ORD	unchanged
$REAL	unchanged
$STRING	unchanged
$STRLEN	unchanged
$STRREP	unchanged
$SUBSTR	unchanged
$TIME	unchanged
$VAR	unchanged
$SPECIFIED	unchanged
$SET_COUNT	unchanged
$VALUE_COUNT	unchanged
$RANGE	unchanged
$DEBUG_MODE_ON	unchanged
$PARAMETER_LIST	unchanged
$PARAMETER	unchanged

E2.0 COMMAND INTERFACE STATUS
E2.2.0.0.0.1 System Access Commands

E2.2.0.0.0.1 System_Access_Commands

Commands_	Status
LINK_USER	unchanged
LOGIN	new - *1
LOGOUT	new

*1 A master catalog is created for the user who issues LOGIN command. No parameters are necessary.

E2.3 RESOURCE_MANAGEMENT

Command_	Status
REQUEST_TERMINAL	not available

E2.4 FILE_MANAGEMENT

Command_	Status
FILE	updated to Rev 8 ERS
COPY	unchanged
DUMP_FILE	unchanged
COMPARE	unchanged

E2.5 PERMANENT_FILE_MANAGEMENT

Command_	Status
HCS GET	unchanged
HCS REPLACE	unchanged
DEFINE	unchanged
ATTACH	unchanged
PURGE	unchanged
CHANGE	unchanged
PERMIT	unchanged
DELETE_PERMIT	unchanged
DEFINE_CATALOG	unchanged
PURGE_CATALOG	unchanged
DELETE_CATALOG_PERMIT	new
PERMIT_CATALOG	new

E2.0 COMMAND INTERFACE STATUS
E2.5 PERMANENT FILE MANAGEMENT

Build N Permanent File Program Interface Deficiencies

1. In order to access the NOS/VE permanent files of a user, a master catalog must exist for that user. At release one time, the validation facility will create a master catalog for a user when the user is created. At build N, however, the validation facility does not exist. To circumvent this problem, the LOGIN command has been modified to create a master catalog for the user being logged in. Thus, once a user has logged in it will be possible to perform NOS/VE permanent file operations for that user. Since a LOGIN command will eventually be required for all jobs, it should be no hardship to include it in jobs starting with build N.

2. Permanent files on NOS/VE are only permanent until a NOS/VE deadstart.

3. The WAIT parameter for ATTACH is always treated as if PFC$NO_WAIT is specified.

E2.6 SCL STATEMENTS AND PROCEDURES

Command	Status
PROC/PROCEND	unchanged
SET_COMMAND_LIST	unchanged
DISPLAY_COMMAND_LIST	unchanged
REPEAT/UNTIL	unchanged
WHILE/WHILEND	unchanged
DECLARE_VARIABLE	unchanged - *1
REMOVE_VARIABLE	unchanged
BLOCK/BLOCKEND	unchanged
LOOP/LOOPEND	unchanged
FOR/FOREND	unchanged
IF/ELSEIF/ELSE/IFEND	unchanged
CYCLE	unchanged
EXIT	unchanged
INCLUDE	unchanged
COLLECT_TEXT	unchanged
DISPLAY_VALUE	unchanged
EXIT_PROC	unchanged
ACCEPT	parameter changes
DO	added status parameter
CONNECT_FILE	new
DISCONNECT_FILE	new
DISPLAY_CONNECTION	new

E2.0 COMMAND INTERFACE STATUS
E2.6 SCL STATEMENTS AND PROCEDURES

change HCS variable	unchanged
display HCS variable	unchanged

*1 Variables can no longer be declared with the same names as the boolean constants.

E2.7 INTERACTIVE_COMMANDS

Command_	Status
RESUME	new
TERMINATE	new

E2.8 OBJECT_CODE_MAINTENANCE

Command_	Status
CREATE_OBJECT_LIBRARY	unchanged
DISPLAY_LIBRARY	unchanged
SELECT_DISPLAY_LEVEL	unchanged
ADD	unchanged
REPLACE	unchanged
COMBINE	unchanged
CREATE_MODULE	unchanged
BIND_MODULE	unchanged
DEFINE_PROGRAM	unchanged
DELETE	unchanged
CHANGE	unchanged
SATISFY	unchanged
REORDER	unchanged
GENERATE	unchanged
QUIT	unchanged
CI to II Conversion	unchanged

1) An II object file or library can be displayed using the nonstandard command OBJLIST.
Example:

OBJLIST II_OBJECT LISTING

Where II_OBJECT is an II object file or object library and LISTING is the file containing the formatted object text listing.

E2.0 COMMAND INTERFACE STATUS
E2.8 OBJECT CODE MAINTENANCE

2) Users of CREATE_OBJECT_LIBRARY and OBJLIST must now get the programs to NOS/VE via the following commands prior to their execution.

```
ATTACH .USER1.SYSLIB FAMILY=FAMILY1
ATTACH .USER1.CYBILIB FAMILY=FAMILY1
SETOL ADD=SYSLIB
```

The default system prolog used by the SETUP command eliminates the necessity of issuing these commands. Using SETUP is the recommended approach.

3) CREATE_OBJECT_LIBRARY expects V1.2 of the object text and V0.2 object libraries. This is incompatible with previous releases. All NOS/VE object files and object libraries must be regenerated.

E2.9 USER_SERVICES

Command_	Status
DISPLAY_LOG	UPON changed to OUTPUT - *1

*1 type = <system ! sys ! job> instead of type = <system ! account ! engineering ! statistics ! job ! job_statistic>

Warning - If you print display log output on a line printer, you must use the PRINT <file name> SHIFT=YES command or you will get one line per page.

E2.10 FILE_ROUTING

Command_	Status
HCS JMROUTE	unchanged

E2.11 PROGRAM_EXECUTION

Command_	Status
SET_OBJECT_LIST	unchanged
SET_PROGRAM_OPTIONS	unchanged

E2.0 COMMAND INTERFACE STATUS
E2.11 PROGRAM EXECUTION

DISPLAY_PROGRAM	unchanged
EXECUTE	unchanged - *2
"name call"	unchanged - *1
PUSH_DEBUG_MODE	unchanged
POP_DEBUG_MODE	unchanged
SET_DEBUG_MODE	unchanged
SET_DEBUG_RING	unchanged
TERMINATE_TASK	new

*1 Warning - "name call" works only for SCL procedures unless a FILE command has been issued to specify that the file is an object file and the structure is data. The FILE command must be reissued every time the file is attached or brought over from NOS.

*2 If you are an interactive user, do not specify the values 'OUTPUT' or '$OUTPUT' for the parameter MAP; doing so will crash the system.

E2.12 JOB_MANAGEMENT

Command_	Status
SUBMIT	unchanged
DISPLAY_JOB_STATUS	unchanged
DROP_JOB	unchanged
PRINT	unchanged
DROP_FILE	unchanged
DISPLAY_PRINT_STATUS	unchanged

E2.13 NOS/VE_COMMANDS_IMPLEMENTED_AS_PROCS

In this build, several NOS/VE commands have been implemented as SCL procedures in order to make the system look more like the final version. Users are urged to use these procedures rather than their interim counterparts since the interim commands will ultimately be withdrawn.

In order to have these procedures available in your job, you should use the SETUP command using the default system prolog. The procedures will be local files in your job, so name conflicts should be avoided.

E2.0 COMMAND INTERFACE STATUS
E2.13.1 GET_FILE

E2.13.1 GET_FILE

This procedure is identical to the version documented in the NOS/VE ERS except that the default conversion is A6. We intend to change the ERS to use this default as well.

E2.13.2 REPLACE_FILE

This procedure is identical to the version documented in the NOS/VE ERS except that the default conversion is A6. We intend to change the ERS to use this default as well.

E2.13.3 PRINT

This procedure is similar to the version documented in the NOS/VE ERS except that it returns the local file after routing it to NOS. This is a deficiency of the JMROUTE command as well.

In addition, the PRINT command does not add carriage control to the file to be printed, if necessary. This can be done by using the SHIFT parameter of the command.

The following is a description of the build N PRINT command:

```
print file=list of <file>
      [shift=<boolean>]
      [status=<status variable>]

      file: This parameter specifies a list of files to be printed.
           These files will be returned after the files are routed
           to the printer.

      shift : s: This parameter specifies whether or not carriage
           control characters are to be added to the file.

           Omission will cause NO to be assumed.

      status: See ERROR HANDLING.
```

E2.0 COMMAND INTERFACE STATUS
E2.13.4 CREATE_OBJECT_LIBRARY ; COL

E2.13.4 CREATE_OBJECT_LIBRARY ; COL

This procedure is identical to the version documented in the NOS/VE ERS except that only the command alias COL is available.

E2.14 NON_STANDARD_COMMANDS

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

E2.14.1 SETUP

The purpose of this command is to provide a "prolog" capability for a user. This is an interim command and will be withdrawn when the full validation and permanent file capabilities are available.

Only a single SETUP command should be issued in a batch job or terminal session. Any SETUP commands issued after the first will fail due to the presence of the LINK_USER command. The SETUP command should be the first NOS/VE command issued in an interactive job and should be issued immediately following the LOGIN command in a batch job.

```
setup user=<name>
      password=<name>
      [prolog=<name>]
      [system_prolog=<name>]
      [family=<name>]
      [status=<status variable>]
```

user ; u: This parameter specifies the name of the user responsible for the processing to be accomplished.

password ; pw: This parameter specifies the user password. The password together with the user and family name will be used to construct a link_user command. This link_user command will be processed as a part of the setup command.

prolog ; p: This parameter specifies the name of a NOS file which will be copied to the current session environment (get operation). This file will then be interpreted

E2.0 COMMAND INTERFACE STATUS
E2.14.1 SETUP

using the include capability. Omission will cause a file name of PROLOG to be used.

system_prolog ; sp: This parameter specifies the name of a NOS/VE file, within the USER1 catalog, which will be attached to the current session environment (ATTACH operation). This file will then be interpreted using the include capability. Omission will cause a file named SYSPROF to be used. The functions performed by SYSPROF can be determined by either doing a COPY SYSPROF or COPY NOTE command to see the contents at a terminal.

family ; f: This parameter specifies the name of the family to be accessed. Omission will cause a family name of NVE to be used.

status: See ERROR HANDLING.

E2.14.2 CITOII

The purpose of this command is to get a CI object file or object library from NOS and to convert it to an II object file suitable for processing by the NOS/VE loader and/or object library generator.

```
citoii ci=<NOS file name>
       [ii=<NOS/VE lfn>]
       [user=<NOS user name>]
       [status=<status variable>]
```

ci: This parameter specifies the NOS permanent file name of the CI object file or object library to be converted.

ii: This parameter specifies the NOS/VE file name on which the converted (i.e. II) object file is to be written.

Omission will cause the CI file name to be used.

user: This parameter specifies the NOS user name in whose catalog the CI object file is located.

Omission will cause the user name of the user issuing the command to be used.

status: See ERROR HANDLING.

E2.0 COMMAND INTERFACE STATUS
E2.14.3 OBJLIST

E2.14.3 OBJLIST

The purpose of this command is to produce a formatted listing of a NOS/VE object module produced on NOS/VE (i.e. II object text).

```
objlist object=<lfn>
        [list=<lfn>]
        [status=<status variable>]
```

object : o: This parameter specifies the object file or object library to be listed.

list : l: This parameter specifies the file on which the formatted listing is to be written.

Omission will cause the listing to be printed on the local printer.

status: See ERROR HANDLING.

E2.14.4 LINK_USER : LIU

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

E2.14.5 GET

This command is an interim implementation of the final GET_FILE command that is documented in the NOS/VE command interface. All users should use the GET_FILE procedure now available as the GET command will be withdrawn in subsequent builds.

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. 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:

E2.0 COMMAND INTERFACE STATUS
E2.14.5 GET

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 Binary

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 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

E2.0 COMMAND INTERFACE STATUS
E2.14.5 GET

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) 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 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 and Record Type = Variable (RT=W). The 170 logical record structure EORs are dropped.

Example of GET command:

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

E2.14.6 REPLACE

This command is an interim implementation of the final REPLACE_FILE command that is documented in the NOS/VE command interface. All users should use the REPLACE_FILE procedure now available as the REPLACE command will be withdrawn in subsequent builds.

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. REPLACE command. All parameters 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 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 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 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 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:

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

E2.0 COMMAND INTERFACE STATUS
E2.14.6 REPLACE

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. ;

E3.0 PROGRAM INTERFACE STATUS

E3.0 PROGRAM_INTERFACE_STATUS

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

E3.1 _COMMAND_PROCESSING

Procedure_	Status
CLP$SCAN_PARAM_LIST	unchanged
CLP$TEST_PARAMETER	unchanged
CLP$GET_KEYWORD	unchanged
CLP$GET_SET_COUNT	unchanged
CLP$GET_VALUE_COUNT	unchanged
CLP$TEST_RANGE	unchanged
CLP$GET_VALUE	unchanged
CLP$DECLARE_VARIABLE	unchanged
CLP$REMOVE_VARIABLE	unchanged
CLP$READ_VARIABLE	unchanged
CLP$WRITE_VARIABLE	unchanged
CLP$SCAN_COMMAND_FILE	unchanged
CLP$END_SCAN_COMMAND_FILE	unchanged
CLP$SCAN_COMMAND_LINE	unchanged
CLP$PUSH/POP COMMAND LIST	unchanged
CLP$CREATE_FILE_CONNECTION	new
CLP$DELETE_FILE_CONNECTION	new

E3.2 _MESSAGE_GENERATOR

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

E3.0 PROGRAM INTERFACE STATUS
E3.3 RESOURCE MANAGEMENT

E3.3 RESOURCE MANAGEMENT

Procedure	Status
RMP$REQUEST_MASS_STORAGE	unchanged
RMP$REQUEST_TERMINAL	unchanged

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

- auto_input
- transparent_mode
- prompt_file
- prompt_string

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

- AMP$OPEN
- AMP$GET_NEXT
- AMP$GET_DIRECT
- AMP$GET_PARTIAL
- AMP$PUT_NEXT
- AMP$PUT_DIRECT
- AMP$PUT_PARTIAL
- AMP$CLOSE

E3.4 PROGRAM EXECUTION

Procedure	Status
PMP$EXIT	unchanged
PMP$EXECUTE	unchanged - *1
PMP$TERMINATE	unchanged
PMP$AWAIT_TASK_TERMINATION	unchanged
PMP$MODULE_TABLE_ADDRESS	unchanged
PMP$ENTRY_POINT_TABLE_ADDRESS	unchanged
PMP$PUSH_JOB_DEBUG_MODE	unchanged
PMP$POP_JOB_DEBUG_MODE	unchanged
PMP$SET_JOB_DEBUG_MODE	unchanged
PMP$JOB_DEBUG_MODE_ON	unchanged
PMP$PUSH_TASK_DEBUG_MODE	unchanged
PMP$SET_TASK_DEBUG_MODE	unchanged
PMP$TASK_DEBUG_MODE_ON	unchanged

E3.0 PROGRAM INTERFACE STATUS
E3.4 PROGRAM EXECUTION

PMP$SET_DEBUG_RING	unchanged	;
PMP$DEBUG_RING	unchanged	;
PMP$CHANGE_DEBUG_LIBRARY_LIST	unchanged	;
PMP$POP_TASK_DEBUG_MODE	unchanged	;

*1 DO NOT specify the values '$OUTPUT' or 'OUTPUT' for the field LOAD_MAP_FILE of the program description - doing so will crash the system. ;

E3.5 PROGRAM COMMUNICATION ;

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

E3.6 CONDITION PROCESSING ;

Procedure	Status	
PMP$ESTABLISH_CONDITION_HANDLER	unchanged	
PMP$DISESTABLISH_COND_HANDLER	unchanged	
PMP$CAUSE_CONDITION	unchanged	
PMP$CONTINUE_TO_CAUSE	unchanged	
PMP$TEST_CONDITION_HANDLER	unchanged	
PMP$VALIDATE_PREVIOUS_SAVE_AREA	unchanged	;
PMP$ESTABLISH_DEBUG_CFF	unchanged	;
OSP$SET_STATUS_FROM_CONDITION	new	;

E3.7 PROGRAM SERVICES ;

Procedure	Status
PMP$GET_TIME	unchanged

E3.0 PROGRAM INTERFACE STATUS
E3.7 PROGRAM SERVICES

PMP$GET_MICROSECOND_CLOCK	unchanged
PMP$GET_TASK_CP_TIME	unchanged
PMP$GET_DATE	unchanged
PMP$GET_USER_IDENTIFICATON	unchanged
PMP$GET_ACCOUNT_PROJECT	unchanged
PMP$GET_JOB_NAMES	unchanged
PMP$GET_JOB_ID	unchanged
PMP$GET_JOB_MODE	unchanged
PMP$GET_PROGRAM	unchanged
PMP$GET_TASK_ID	unchanged
PMP$MANAGE_SENSE_SWITCHES	unchanged
PMP$GET_OS_VERSION	unchanged
PMP$GET_PROCESSOR_ATTRIBUTES	unchanged
PMP$DEFINE_DEBUG_ENTRY	unchanged
PMP$GET_DEBUG_ENTRY	unchanged
PMP$MODIFY_DEBUG_ENTRY	unchanged
PMP$REMOVE_DEBUG_ENTRY	unchanged

E3.8 LOGGING

Procedure	Status
PMP$LOG	unchanged
PMP$LOG_ASCII	unchanged

E3.9 FILE MANAGEMENT

Procedure	Status
Sequential Access	*1
Byte_Addressable Access	unchanged
Record Access	*1
Segment Access	unchanged
W_System Specified	unchanged
W_User Specified	new
U_System Specified	unchanged
U_User Specified	new
F_System Specified	unchanged
F_User Specified	new
AMP$DESCRIBE_NEW_FILE	unchanged
AMP$FILE	unchanged
AMP$GET_FILE_ATTRIBUTES	unchanged
AMP$FETCH	unchanged

E3.0 PROGRAM INTERFACE STATUS
E3.9 FILE MANAGEMENT

AMP$STORE	unchanged
AMP$COPY_FILE	new
AMP$RENAME	new
AMP$RETURN_LOCAL_FILE	new
AMP$OPEN	unchanged added parameter ;
AMP$CLOSE	unchanged
AMP$FETCH_ACCESS_INFORMATION	unchanged
AMP$SKIP	*4
AMP$REWIND	*2
AMP$WRITE_END_PARTITION	new
AMP$GET_NEXT	*1, *5
AMP$GET_DIRECT	unchanged ;
AMP$GET_PARTIAL	*1, *6
AMP$GET_PARTIAL_DIRECT	unchanged ;
AMP$PUT_NEXT	*1 ;
AMP$PUT_DIRECT	unchanged ;
AMP$PUT_PARTIAL	*1, *3
AMP$PUT_PARTIAL_DIRECT	unchanged ;
AMP$SEEK_DIRECT	unchanged ;
AMP$GET_SEGMENT_POINTER	unchanged ;
AMP$SET_SEGMENT_EOI	unchanged ;
AMP$SET_SEGMENT_POSITION	unchanged ;
AMP$SET_LOCAL_NAME_ABNORMAL	new
AMP$SET_FILE_INSTANCE_ABNORMAL	new
AMP$ACCESS_METHOD	new
AMP$FETCH_FAP_POINTER	new
AMP$STORE_FAP_POINTER	unchanged ;

*1 user specified blocking is now supported

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

*3 AMP$PUT_PARTIAL
PUT_PARTIAL with the TERM_OPTION = AMC$START does <u>not</u> start a ;
new record.

*4 AMP$SKIP
If the number of units to skip = 0 and the file is positioned mid-unit the file will remain positioned mid-unit.

*5 AMP$GET_NEXT
A GET_NEXT with a working_storage length = 0 will return an undefined file position. A request of zero length is invalid and will result in abnormal termination in later builds.

*6 AMP$GET_PARTIAL
A GET_PARTIAL of record type = undefined never returns file ;

E3.0 PROGRAM INTERFACE STATUS
E3.9 FILE MANAGEMENT

position of end-of-record.

E3.10 PERMANENT FILE MANAGEMENT

Procedure	Status
PFP$DEFINE	parameter changes
PFP$ATTACH	parameter changes
PFP$PURGE	parameter changes
PFP$CHANGE	parameter changes
PFP$PERMIT	parameter changes
PFP$DELETE_PERMIT	parameter changes
PFP$DEFINE_CATALOG	parameter changes
PFP$PURGE_CATALOG	parameter changes
PFP$PERMIT_CATALOG	parameter changes
PFP$DELETE_CATALOG_PERMIT	parameter changes

Build N Permanent File Program Interface Deficiencies

1. In order to access the NOS/VE permanent files of a user, a master catalog must exist for that user. At release one time, the validation facility will create a master catalog for a user when the user is created. At build N, however, the validation facility does not exist. To circumvent this problem, the LOGIN command has been modified to create a master catalog for the user being logged in. Thus, once a user has logged in it will be possible to perform NOS/VE permanent file operations for that user. Since a LOGIN command will eventually be required for all jobs, it should be no hardship to include it in jobs starting with build N.

2. Permanent files on the NOS/VE are only permanent until a NOS/VE deadstart.

3. The WAIT parameter for ATTACH is always treated as if PFC$NO_WAIT is specified.

E3.11 MEMORY MANAGEMENT

MMP$ADVISE_IN	unchanged
MMP$ADVISE_OUT	unchanged
MMP$ADVISE_OUT_IN	unchanged
MMP$WRITE_MODIFIED_PAGES	unchanged
MMP$CREATE_SEGMENT	unchanged

E3.0 PROGRAM INTERFACE STATUS
E3.11 MEMORY MANAGEMENT

MMP$DELETE_SEGMENT	unchanged
MMP$STORE_SEGMENT_ATTRIBUTES	unchanged
MMP$FETCH_SEGMENT_ATTRIBUTES	unchanged
MMP$VERIFY_ACCESS	unchanged
MMP$FREE	unchanged
MMP$LOCK_PAGES	unchanged
MMP$UNLOCK_PAGES	unchanged
MMP$FETCH_PVA_UNWRITTEN_PAGES	unchanged

E3.12 STATISTICS FACILITY

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

E3.13 NOS/VE EXCEPTIONS

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

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

Exception Code	Product Identifier	Product Name
1 - 158,999	Reserved	
159,000 - 159,999	SY	System Core
160,000 - 169,999	AM	Basic Access Methods
160,000 - 163,999	BA	Basic Access
164,000 - 164,999	LN	Local Name Mgr
165,000 - 165,999	JF	Job File Mgr
166,000 - 166,999	SR	Conversion Services
167,000 - 167999	CM	Configuration Mgmt
170,000 - 179,999	CL	Command Language
180,000 - 189,999	JM	Job Management
190,000 - 199,999	LL	Loader

E3.0 PROGRAM INTERFACE STATUS
E3.13 NOS/VE EXCEPTIONS

200,000 - 209,999	MM	Memory Management
200,000 - 204,999	MM	Monitor Level
205,000 - 205,999	MM	Task Level
210,000 - 219,999	OS	Operating System
210,000 - 210,999	OS	OS
211,000 - 211,999	MT	EXEC
212,000 - 212,999	IO	MS I/O
213,000 - 213,999	IO	Tape I/O
214,000 - 214,999	DM	Device Management
215,000 - 215,999	ML	Memory Link
216,000 - 216,999	IF	Interactive
217,000 - 217,999	TM	TM Monitor
218,000 - 218,999	TM	TM Task
219,000 - 219,999	JS	Job Swappers
220,000 - 229,999	PF	Permanent File Management
221,000 - 221,999	ST	Set Management
222,000 - 222,999	PU	Permanent File Utilities
230,000 - 239,999	PM	Program Management
240,000 - 249,999	RM	Resource Management
250,000 - 259,999	OF	Operator Facility
260,000 - 269,999	AV	User Administrator
270,000 - 279,999	IC	Interstate Communication
280,000 - 289,999	RH	Remote Host Facility
290,000 - 299,999	OC	Object Code Utilities
300,000 - 309,999	DB	Deadstart/Recovery
310,000 - 319,999	MS	Maintenance Services
320,000 - 329,999	Reserved	
340,000 - 349,999	SF	Statistics Fac.
330,000 - 339,999	US	User Errors
500,000 - 509,999	AA	Advanced Access Method
510,000 - 519,999	AG	ALGOL
520,000 - 529,999	AL	Assembly Language
530,000 - 539,999	AP	APL
540,000 - 549,999	BA	BASIC
550,000 - 559,999	CA	Conversion Aids System
560,000 - 569,999	CB	COBOL
570,000 - 579,999	CY	CYBIL
580,000 - 589,999	FT	FORTRAN
590,000 - 599,999	PA	PASCAL (Wirth)
600,000 - 609,999	P1	PL/1
610,000 - 619,999	SM	Sort Merge
620,000 - 629,999	SC	Source Code Utility
640,000 - 649,999	DB	Debug

E4.0 DUAL STATE DEADSTART AND OPERATION

E4.0 DUAL STATE DEADSTART AND OPERATION

E4.1 A170 NOS DEADSTART

- The system is configured to run with three FMD units (41, 42 and 43).
- Set the D/S panel to deadstart from the primary system disk. This is Unit 43 for all Build N systems.
- Push D/S button
- Select "O" display
- Select "H" display
- Enter CM=10000
- Enter (CR)
- Enter date/time

Wait for deadstart to complete.
Note: The deadstart tape DUAL6N (which is currently installed on unit 43) is found in the area in the northeast corner of the room where the tape cabinet is found.

E4.2 CURRENT DUAL STATE CONFIGURATION

- FMD Unit 43

  This unit contains the following:

  - A170 NOS (Build 6 level), CTI, CMSE, EI binaries, NOS deadstart files
  - Files associated with user number LIBRARY
  - Files associated with user number SES
  - Files associated with DEV1, DEV2, REL1, INT1.

- FMD Unit 41

  This is a scratch unit

- FMD Unit 42

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.2 CURRENT DUAL STATE CONFIGURATION

This unit contains the following:

- NOS/VE Development Area PL's and Member PL's
- NOS/VE Deadstart Files to be tested (saved in individual user's catalogs)
- Files associated with user number INT2

E4.3 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 COMSOURCE (R.K. Cooper - x3092) to be assigned a user index

2) PF dumping and loading

   You may use "SES.DUMPPF" on SN/101 to dump your permanent fies 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".

E4.4 NOS/VE DEADSTART

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

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

  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.

- o For Build N: Mount the disk labeled "DAHL-Large sector" on 844 unit 0 (other disks will not work). The HCS disk driver (for

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.4 NOS/VE DEADSTART

small sector 844's) will not work in Build N.

- For Builds 0 greater than 05: Mount one of the disks labeled "S2 S/N 104 system scratch" on 844 unit 1. This cannot be a large sector disk! For Builds n > 23 and 0 > 5 the remote host/interactive 170 binaries must be sysedited into the running system until the deadstart tape has been updated:

  ```
  X.DIS.
  USER,SCAT,SCATX.
  where "scat" = DEV1 for Build N
  "scat" = INT1 for build 0
  CALL,SYEDNVE.
  DROP.
  ```

- Bring up dual state:

  ```
  X.UPMYVE (CAT=mycat, DEV1=scat)
  where mycat = user catalog (as before)
        scat = system ctalog - INT2, INT1 or DEV2
  ```

- 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.

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

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

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.4 NOS/VE DEADSTART

installation deadstart and a deadstart.

E4.4.1 DS

The purpose of this command is to perform an installation deadstart or a deadstart of NOS/VE.

```
ds [install=<boolean>]
   [load_source_libraries=<boolean>]
   [load_products=<boolean>]
   [status=<status variable>]
```

install : i: This parameter specifies whether an installation deadstart is to be performed. Valid specifications are:
TRUE - installation deadstart to be performed
FALSE - deadstart to be performed

Omission will cause a deadstart to be performed.

load_source_libraries : lsl: This parameter specifies whether SCU libraries are to be loaded. Valid specifications are:
TRUE - libraries are loaded
FALSE - libraries are not loaded

On the Arden Hills closed shop S2 system, the SCU libraries to be loaded are:
OSLPI - operating system program interface
Subset of NOSVEPL - operating system source library

Omission will cause SCU libraries to be loaded.

load_products : lp: This parameter specifies whether the object libraries defining the current product set are to be loaded. Valid specifications are:
TRUE - the products are loaded
FALSE - the products are not loaded

On the Arden Hills closed shop S2 system, the product set to be loaded consists of:
CYBIL
SCU

Omission will cause the product set to be loaded.

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

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.4.1 DS

If you change any of the following decks you MUST use the installation deadstart from your own catalog (with files CYBILGO, XLJOSL, and XLJLIB):

AMMTSA	BAMDVR	BAMPC4	BAMPC2	BAMPC1	BAMPC3	IIMRSE	IIMRLE	IIMRUM
IFMEXEC	IIMA72H	IIMTDEL	IIMRUSM	IIMDC2S	OCMREO	OCMMUR	OCMBIM	
OCMSDL	OCMEND	OCMLP	OCMCPY	OCMCRM	OCMGEN	OCMMOMS	OCMDEF	OCMREP
OCMOLG	OCMCOL	OCMOBJ	OCMCHA	OCHOFH	OCMADD	OCMNP	OCMDEL	OCMDLB
OCMCOM	OCMSAT	RHMQAF		RHMQIP	RHMSFM	RHMLOF	RHMMLI	RHMQOP
RHMQTE	RHMLCF		RHMLOG		RHMQRE	RHMA12	RHM12A	RHMROU
RHMWLF	RHMRTN		RHMGDM	USORT	UUSER1	UUTL		

E4.4.2 EXAMPLE OF NOS/VE INSTALLATION DEADSTART

Type
K,n. where n is the UPMYVE control point number.
K.LIU (your un,NVE) your_password.
K.GET,DS,DS,,SCAT,NVE,A6.
K.DS TRUE FALSE FALSE.

The system is up when the following message comes up:

SYSTEM IS NOW ALL UP AND RUNNING
WAITING FOR MORE INPUT

E4.4.3 EXAMPLE OF NOS/VE DEADSTART

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

Type (where DEV1 is the same as the CAT=value in the UPMYVE call):
K,n. where n is the UPMYVE control point number.
K.LIU (DEV1,NVE) DEV1X.
K.GET,DS,DS,,,NVE,A6.
K.DS.

The system is up when the following message comes up:
SYSTEM IS NOW ALL UP AND RUNNING
WAITING FOR MORE INPUT

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.5 NOS/VE TERMINATION

E4.5 NOS/VE TERMINATION

o Bringing down dual state:

```
K.*BYEVE.
```

o If not a normal termination

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

E4.6 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=13).

6) Set ECUU (S2=01uu)

   E = equipment

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

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

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

8) Answer "dump number" with a CR.

9) Answer "dump controlware" with a CR.

   * - Warning if this step is omitted, DSDI canot process the dump tape.

To create a listing of the EDD tape:

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.6 DSDI INFORMATION

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

2) GET,DSDI/UN=DEV1. (On S/N 101.)

   or

   GET,DSDI/UN=DEV1. (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.

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.7 A170 NOS SHUTDOWN

E4.7 A17C_NOS_SHUIDOWN

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

4) Push deadstart button.

E4.8 INIERIM_MEMORY_LINK_SIORAGE_MOVE_CONSIDERAIIONS

The following precautions must be taken when running dual state with the Interim Memory Link.

1) Drop IRHF170, PASSON and all permanent file partner jobs before doing *BYEVE (via 2.STOP.)

2) Do not rollout IRHF17C, PASSON or permanent file partner jobs at any time.

3) Before doing a CHECKPOINT SYSTEM, drop IRHF170, PASSON and all permanent file partner jobs (via 2.STOP.)

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.

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.8 INTERIM MEMORY LINK STORAGE MOVE CONSIDERATIONS

- 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.

E4.9 NOS/VE INTERACTIVE FACILITY OPERATION

E4.9.1 OPERATOR INITIATION

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

1) Bring up NOS/VE.

2) Bring up NAM

At the system console enter:
3.NAM.

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

4) Bring up A170 part of interactive:

- TAFNVE.

Control point two must be free or rolloutable (i.e. NAM should not be there). This also brings up PASSON and the MLI subsystem control points.

E4.9.2 OPERATOR TERMINATION

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

- 3.CFO.DI,AP=TAF. (3 is the NAM control point number)

  This is the preferred method. To bring NOS/VE interactive back up, you must first do a 3.CFO.EN,AP=TAF.

- 3.CFO.DI,NE. (3 is the NAM control point number)

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.9.2 OPERATOR TERMINATION

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

E4.9.3 OTHER OPERATOR CAPABILITIES

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

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

- To send a message to all terminals do:

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

- 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

E4.10 IO_RELOAD_CONTROLWARE_FOR_THE_DAHL-LARGE_SECTOR_844

The following commands need to be entered from deadstart

- o Push D/S button
- o Select "U" display
- o Select "S" display
- o Select TYPE=3
- o Select CM=1
- o Select CH=1

E4.0 DUAL STATE DEADSTART AND OPERATION
E4.10 TO RELOAD CONTROLWARE FOR THE DAHL-LARGE SECTOR 844

- Select EQ=0
- Select UN=the unit number of the system pack (43)
- Select "M" display
- Carriage return
- Type in "CW MA8 22" carriage return wait until LOADED
- Then redeadstart using section 1

E4.10.1 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.
```

E4.11 K_DISPLAY_ASCII

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

INPUT	TRANSLATED_TO	INPUT	TRANSLATED_TO
/1	^	/(	[
/2	"	/)	]
/3	#	/+	>
/4	$	/-	<
/5	(reversed /)	/=	.
/6	;	/*	' (single quote)
/7	?	//	/
/8	{	/,	:
/9	}	/A to /Z	a - z (lower case)
/0	_ (underscore)		

(The major incompatibilities with earlier systems are for characters for ; and '. To get a semicolon, type /6, to get a ' (single quote), type /*

E5.0 ARDEN HILLS DEVELOPMENT LAB SUPPORT BY INTEGRATION

E5.0 ARDEN HILLS 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. If you project would like to reserve a section of this tape rack, contact Tim McGibbon or Bonnie Swierzbin.

- 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 in the southeast corner 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 and listing of the NOS/VE PPU routines, system link map. This rack is along the west wall 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 West 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 Bonnie Swierzbin of any problems or deficiencies of these materials. Leave a note if we are not available.

E5.0 ARDEN HILLS 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 - ARH3610 - obtained from Karen Rubey (482-3966) or via SES.TOOLDOC
- 3.2 Command Interface ERS - ARH3609 - obtained from Karen Rubey (482-3966) or via SES.TOOLDOC
- 3.3 NOS/VE Procedures and Conventions - SESD010 - obtained by SES.TOOLDOC
- 3.4 Listing of all NOS/VE Modules - obtained by SES,DEV1.LISTNVE. See Integration Procedures Notebook

E5.0 ARDEN HILLS DEVELOPMENT LAB SUPPORT BY INTEGRATION

for details.

3.5 NOS/VE Internal Interface Maintenance Procedures

Memo available from S.C. Wood.

3.6 Integration Procedures Notebook

Obtained by:

Acquire,IPNDOC/UN=DEV1. SES.PRINT,IPNDOC.

4.0 Tools Reference Documents

4.1 CYBIL Interactive Debugger - ARH3142

4.2 SES User's Guide - ARH1833

4.3 CYBIL Specification - ARH2298

4.4 C180 Assembler ERS - ARH1693

4.5 Simulator ERS - ARH1729

4.6 VEGEN ERS - ARH2591

4.7 VELINK ERS - ARH2816

4.8 Simulated I/O ERS - ARH3125

4.9 Object Code Utilities ERS - ARH2922

4.10 CYBIL Implementation Dependent Handbook - ARH3078

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:

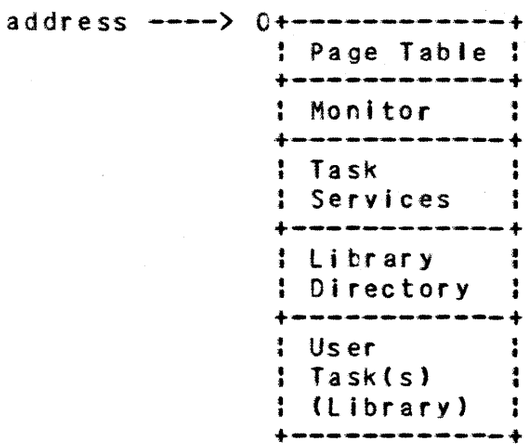
```
LOADLIB NS PNAME
```

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 NVELINK, 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

F3.0 EXECUTION
F3.1 INTRODUCTION

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

F3.0 EXECUTION
F3.2.30 SSET

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

F3.0 EXECUTION
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:

F3.0 EXECUTION
F3.4.2.2.5 TRACE BACK

- 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:

<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.

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

F3.0 EXECUTION
F3.5.1.5 File Manager

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_INPUT

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_OUTPUT

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:

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

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 RNVEKEY 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.

F5.0 KEYPOINTS
F5.3.1 KEYPOINT REFORMATTING UTILITY

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 RNVEKEY 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:

F5.0 KEYPOINTS
F5.3.3 REFORMATTED FILE DESCRIPTION

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 exec:

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).

o 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.

o 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.

o Select the U (utilities) option. The CTI *U* display should appear.

o 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 IESI 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 EXISIING IESI 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'

F7.1.3.5 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 IESIMEM

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 IIMEQUI

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 017B).

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).

G1.0 CYBIL VERSION 1.0 LEVEL 81188

G1.0 CYBIL_VERSION_1.0_LEVEL_81188

Tool ID CY

Tool Files

Source PL's

PFESRC - Compiler Front End
PCGSRC7 - CC code generator
PCGSRC8 - CI code generator
PAXCMN - Global common decks
PLBSRC7 - CC run time library
LBSRC80 - CI run time library

Binary Files

PFELIB - Compiler Front End
PCGLIB7 - CC code generator
PCGLIB8 - CI code generator
CYBCOBJ - CC run time binaries
CYBIOBJ - CI run time binaries
CYBCLIB - CC run time library
CYBILIB - CI run time library
CYBHBIN - II front end, code generator and run time binaries

Absolute Files

CYBILC - CYBIL-CC compiler
CYBILI - CYBIL-CI compiler

G2.0 BUILD INTERDEPENDENCIES

G2.0 BUILD_INTERDEPENDENCIES

The following tools and files must be available for the build process to execute.

MADIFY - ASCII version of Modify
SESULIB - I/O and miscellaneous routines supplied by the SES project. Found on SES numbered file SESxxxx.
PROCLIB - File containing SES procedure COMPII for building the II front end and code generator. Found in LP3.
SIMCOBJ - Simulated NOS/VE interfaces supplied by the SES project. Found on SES numbered file SESxxxx.
SIMIOBJ - Simulated NOS/VE interfaces supplied by the SES project. Found on SES numbered file SESxxxx.
C180 ASSEMBLER - C180 Assembler and SYSTEXT supporting the new object text
C180 OCU - Generate Object Library command
CYBIL-CC - An existing CYBIL-CC compiler
CYBIL-CI - An existing CYBIL-CI compiler
CYBCCMN - SES common decks for CYBIL-IO and SES utilities. Found on SES numbered file SESxxxx.
CYBCIMN - SES common decks for Simulated-IO and SES utilities. Found on SES numbered file SESxxxx.
CCCOMPL - Interfaces to the Common Compiler Modules.
CCMLIB - Binaries of the Common Compiler Modules.

G3.0 CYBIL COMPILER BUILDS

G3.0 CYBIL_COMPILER_BUILDS

This paper describes the procedures required to build the CYBIL CC, CI and II compilers. All of the builds are adaptable to run in any user catalog. The builds for the CC and CI compilers and all of the libraries are written predominantly in CCL with associated usage of SES procedures. The build for the II compiler is an SES procedure.

The general approach to building the compilers is first to generate an object library for each logical entity of the compilers and then for the CC and CI compilers to combine these objects into an absolutized compiler. For the II compiler, the object libraries are combined with the run time library binaries. When the compilers are built for a non-CYBIL project catalog, the run-time checking and debug code should be turned off.

G3.1 PROCFIL

Get file PROCFIL from LP3 and save in the catalog where the builds will be processed.

G3.2 BUILD_PROCEDURES

To compile the source PLs, the following command is used:

```
SES.DO,('CCL_begin_command'),defer.
```

The format of the CCL_begin_command is:

```
BEGIN,proc_name,PROCFIL,src,srcun,lib,chk,debug.

or BEGIN,proc_name,PROCFIL,src,srcun,obj,chk,debug.
```

For building the CI/II library the CCL begin_command is:

```
BEGIN,BCYBIOB,PROCFIL,src,srcun,obj,chk,debug,SIM/HDWR.
```

G3.0 CYBIL COMPILER BUILDS
G3.2 BUILD PROCEDURES

proc_name - Name of the build procedure, default is appropriate file name as listed in section 1.0.

src - Name of madify PL containing the source, default is appropriate file name listed above.

srcun - User name for src, default is LP3.

lib/obj - Name of file to receive new object library or object file, default as for src. If the file already exists it will be overwritten, otherwise it will be DEFINEd. For BCYBIOB with the HDWR option the file name must be specified if default was used with the SIM option.

chk - Run-time checking mode for compiler being built, default is no checking (CHK=0), CHK key alone implies all checking options (CHK=NRS), otherwise specify desired CHK parameters.

debug - Debugging option for compiler being built, default is no debug, keyword alone implies compile internal compiler debug code.

SIM/HDWR - Parameter for BCYBIOB, SIM specifies that library will be used with the CI compiler, HDWR specifies that library will be used with the II compiler. ;

Allowable CCL_build_commands are:

```
BEGIN,BPFELIB.  {Compile front end}
BEGIN,BCYBIOB.  {Compile ci library}
BEGIN,BCYBCOB.  {Compile cc library}
BEGIN,BPCGLB7.  {Compile cc code generator}
BEGIN,BPCGLB8.  {Compile ci code generator}
```

To build a compiler with checking mode turned on add the CHK parameter. To enable the compiler debug code add the DEBUG parameter.

G3.3 PROCLIB

The SES Proclib must now be moved from the LP3 catalog to access the procedures that complete the compiler builds.

G3.0 CYBIL COMPILER BUILDS
G3.4 BUILD PROCEDURE FOR II COMPILER

G3.4 BUILD PROCEDURE FOR II COMPILER

To compile the source PL's, the following command is used

```
SES.COMPII (mods),obj,type=typun,H,DEFER.
```

mods - module names to be compiled, for the front end, mods would be ABSNEG..WRITEPR,BIGQIO..PXQOPEN,CYBILII. For the code generator mods would be IBABSR1..UNSATRF

obj - Name of file to receive new object file. There is no default. If the file already exists it will be overwritten, otherwise it will be DEFINEd.

type - specify FE for front end or CG for code generator

typun - user name for compiler to be used. Default is LP3.

G3.5 BUILD COMPLEIION

G3.5.1 TO BUILD CYBIL/CC

```
SES.GENCIL CC.
```

G3.5.2 TO BUILD CYBIL/CI

```
SES.GENCIL CI.
```

G3.5.3 TO BUILD THE II COMPILER

```
SES,SSS.GOF BASE = (FEobj, CGobj, LIBobj) UPON = CYBHBIN
```

feobj - name of the file containing the II compiler front end binaries
cgobj - name of the file containing the II compiler code generator binaries
libobj - name of the file containing the II run time binaries

G3.0 CYBIL COMPILER BUILDS
G3.5.4 TO BUILD CC RUN TIME LIBRARY

G3.5.4 TO BUILD CC RUN TIME LIBRARY

```
ACQUIRE(CYBCOBJ)

ACQUIRE(SESULIB=SESxxxx/un=SES

ACQUIRE(SIMCOBJ=SESnnnn/un=SES

SES.REPULIB(CYBCOBJ,SESULIB,SIMCOBJ) NL=CYBCLIB NX=1
```

Note: See SES release manager for actual permanent file names SESxxxx and SESnnnn shown above.

G3.5.5 TO BUILD THE CI LIBRARY

```
ACQUIRE(CYBIOBJ)

ACQUIRE(SIMIOBJ=SESnnnn/un=SES)

RETURN(libtemp)

SES.GOL f=(CYBIOJB,SIMIOBJ) UP=libtemp

SES.REWRITE libtemp CYBILIB
```

Note: See SES releae manager for actual permanent file names SESnnnn shown above.

This library now includes the simulated NOS/VE interfaces.

G3.6 IESI_COMPILERS

G3.6.1 CC REGRESSION TESTING

The basic testing for the CC compiler is the convergence of the compiler.

G3.0 CYBIL COMPILER BUILDS
G3.6.2 CI REGRESSION TESTING

G3.6.2 CI REGRESSION TESTING

The test cases exist on file CIBASE/UN=HAW. Common deck PROLOG contains the attach for the compiler to be tested. Common deck COMPILE contains the compiler call statement and associated options. These common decks should be checked to ensure they are setup for the intended run.

To submit all the tests do:

```
SES.RUNCI,m=(CY001..CY999),CLEAR                    ;
```

After completion of the execution of the test cases do:

```
SES.RUNANL                                          ;
```

to analyze the results of the regression run.

G3.6.3 II TESTING

The current tests to be run are Test1, Test2, Test3, Test4, TestL from Testpl/un=TJR, TestM and TestN from testpl/un=ROD. To get the tests from testpl, use the SES procedure gencomp.

G3.0 CYBIL COMPILER BUILDS
H1.1 A170 NOS DEADSTART

H1.1 A170 NOS DEADSTART

Build N15
Date: 4/30/81

- The system is configured to run with three FMD units (41, 42 and 43).
- Mount the pack labeled "Advanced Systems Integration EI Pack" on any 844 disk drive EXCEPT UNIT 0.
- Set the D/S panel to deadstart from the primary system disk. This is Unit 43 for all Build N systems.
- Push D/S button
- Do an alternate deadstart to tape DUAL6N:
  - Select "U" display
  - Select "S" display
  - Deadstart Device Type - 2 (CR)
  - Channel - 13 (CR)
  - Equipment - 0 (CR)
  - Unit - xx (CR) (where "xx" is the tape unit number)
- Select "D" display
- Select "H" display
- Enter CM=10000
- Enter CACHE0-3=OFF
- Enter (CR)
- System will display the message "ENTER LOCATION OF MSL/HIVS DEVICE". Enter:
  - Channel - 03 (CR)
  - Equipment - 0 (CR)
  - Unit - xx (CR) (where "xx" is the unit number of the 844 drive where the EI pack is mounted)
- Enter date/time

Wait for deadstart to complete.
Note: The deadstart tape DUAL6N and the EI pack are both found in the area in the northeast corner of the room where the tape cabinet and bookcase of disk packs are found.

H1.2 CURRENT DUAL STATE CONFIGURATION

- FMD Unit 43
  This unit contains the following:
  - A170 NOS (Build 6 level), CTI, MSL, EI binaries, NOS deadstart files
  - Files associated with user number LIBRARY
  - Files associated with user number SES
  - Files associated with DEV1, REL1, INT1.
- FMD Unit 41

G3.0 CYBIL COMPILER BUILDS
H1.2 CURRENT DUAL STATE CONFIGURATION

This is a scratch unit

- o FMD Unit 42
  This unit contains the following:
  - NOS/VE Development Ara PL's and Member PL's
  - NOS/VE Deadstart Files to be tested (saved in individual user's catalogs)
  - Files associated with user number INT2

H1.3 NOS/VE DEADSTART

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

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

- o Mount the disk labeled "DAHL-Large sector" on 844 unit 0 (other disks will not work). The default disk driver for build M is the NOS/VE driver.
- o Bring up dual state: X.UPMYVE (CAT=mycat, DEV1=scat) where mycat = user catalog (as before)
  scat = system catalog - INT2 or INT1 (defaults to DEV1)
- o 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.

H1.4 BRING UP C170 REMOTE HOST

Type
TAFNVE.
Control point two must be free or rollroutable (i.e., NAM should not be there). This also brings up PASSON and the MLI subsystem control points.

G3.0 CYBIL COMPILER BUILDS
H1.5 BRING UP C180

H1.5 BRING UP C180

The following sections use the files of L0 or L1 to enter the needed commands to bring up all the needed 180 tasks to run NOS/VE at this level.

If you change any of the following decks you MUST use the L0 deadstart from your own catalog (with files CYBILOG, XLJOSL, and XLJLIB):

AMMTSA BAMDVR BAMPC4 BAMPC2 BAMPC1 BAMPC3 IIMRSE IIMRLE IIMRUM IFMEXEC IIMA72H IIMTDEL IIMRUSM IIMDC2S OCMREQ OCMMUR OCMBIM OCMSDL OCMEND OCMLP OCMCPY OCMCRM OCMGEN OCMOMS OCMDEF OCMREP OCMOLG OCMCOL OCMOBJ OCMCHA OCMOFH OCMADD OCMNP OCMDEL OCMDLB OCMCOM OCMSAT RHMQAF RHLFN RHMQIP RHSEND RHMLOF RHMMLI RHMQOP RHMQTE RHMLCF RHTIME RHLOG RHMGRJ RHMQRE RH88812 RH81288 RHROUTF RHWRTLF RHRTRNF RHFETCH RHGETDM USORT UUSER1 UUTL

H1.5.1 BRING THE SYSTEM UP FOR THE FIRST TIME FROM YOUR CATALOG

Type
K,n. where n is the UPMYVE control point number.
K.LIU (yourun,NVE) your_password.
K.GET,L0,L0,,DEV1,NVE,A6.
K.INCLUDE L0.
The system is up when the following message comes up:
SYSTEM IS NOW ALL UP AND RUNNING
WAITING FOR MORE INPUT

H1.5.2 BRING UP THE INTEGRATION SYSTEM (OR YOUR'S AFTER 1ST TIME)

The Integration system has had the L0 file ran on it. Also the file produced by the L0 deadstart have beed made semi-public and at found on the catalog used in the UPMYVE call.

Type (where DEV1 is the same as the CAT=value in the UPMYVE call):
K,n. where n is the UPMYVE control point number.
K.LIU (DEV1,NVE) DEV1X.
K.GET,L1,L1,,,NVE,A6.
K.INCLUDE L1.
The system is up when the following message comes up:
SYSTEM IS NOW ALL UP AND RUNNING
WAITING FOR MORE INPUT

G3.0 CYBIL COMPILER BUILDS
H1.6 NOS/VE TERMINATION

H1.6 NOS/VE TERMINATION

Bringing down dual state:

```
K.*BYEVE.
K.*ENDRUN
```

H1.7 DSDI INFORMATION

H1.7.1 TO CREATE AN EXPRESS DEADSTART DUMP (EDD) TAPE:

- Mount scratch tape (ring in) on a 9-track drive.
- Push D/S button.
- Select U (utilities) display.
- Select E (EDD) display.
- Set channel (S2=13).
- 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.
- Answer "non zero inhibits rewind" with a CR.
- Answer "dump number" with a CR.
- Answer "dump controlware" with a CR.

H1.7.2 TO CREATE A LISTING OF THE EDD TAPE:

- REQUEST,DUMP,NT,D=PE,F=S,LB=KU,PO=R,VSN=your choice.
  GET,DSDI/UN=DEV1. (On S/N 101.)
  or
  GET,DSDI/UN=DEV1. (On S2.)
- Create DSDI directives file:
  A DSDI directive file should include the following:
  IOUMR.
  PROMR.
  MEMMR.
  PRORF.
  W,first_byte_address,last_byte_address,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)
  or
  W,first_byte_address_last_byte_address,segment_

G3.0 CYBIL COMPILER BUILDS
H1.7.2 TO CREATE A LISTING OF THE EDD TAPE:

number,exchange_package. Exchange_package is the RMA of an exchange package or the keywords MPS or JPS.

- Execute DSDI:
  RFL,60000.
  DSDI,M,D,I="input directives file".
- To run (after the first time):
  DSDI,I=n.
  (Does not read tape again.)
- To run interactively:
  Same as above, except to do W command must first do:
  OUTPUT,LISTFIL.
- C170 DSDI information can be found in Chapter 10 of the NOS SYSTEM MANTENANCE Manual. A170 DSDI info can be found in document ARH3060 - GID for A170 NOS/S2.

H1.8 TO RELOAD CONTROLWARE FOR THE DAHL-LANGE SECTOR 844

The following commands need to be entered from deadstart

- Push D/S button
- Select "U" display
- Select "S" display
- Select TYPE=3
- Select CM=1
- Select CH=1
- Select EQ=0
- Select UN=43 !
- Select "M" display
- Carriage return
- Type in "CW MA8 22" carriage return wait until LOADED !
- Then redeadstart using section 1

H1.9 ADDITION NOTES ON BUILD

- For the use of COL (CREATE_OBJECT_LIBRARY) the following 180 commands are needed:

```
GET,SYSLIB,SYSLIB,,DEV1,NVE,B56
GET,CYBILIB,CYBIILB,,DEV1,NVE,B56
SETOL ADD=SYSLIB
```

- The above commands will be needed for all utilities other then CITOII.

Table of Contents

Table of Contents ATION RE)