IPNDOC

DISTRIBUTION

C.K. Bedient	ARH254
F.A. Bischke	ARH254
J.L. Carlson	ARH254
R.E. Dennis	ARH254
J.B. Farr	ARH254
G.J. Greve	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
M.D. Carter	ARH260
R.E. Erickson	ARH260
N.E. Fox	ARH260
D.J. Holm	ARH260
R.A. Mann	ARH260
G.S. Barrett	ARH263
A.J. Lawson	ARH263
A.C. Rupert	ARH280
H.A. Wohlwend	ARH280
R.D. Palm	ARH293
J. Sutherland	CANCDD
B.E. Southworth	MNA02B
J.H. Wick	MNA02B
P.Y. Liou	SVL145
G.D. McGhie	SVL128
J.D. Neuhaus	SVL128
S.L. Pittman	SVL162

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

J. L. Kappler	ARH254
R.H. Kingdon	ARH254
T.C. McGee	ARH254
R.M. Medin	ARH254
N.E. Meyer	ARH254
J.R. Ruble	ARH254
N.J. Lee	ARH260
D.J. Maguire	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.L. Harrell	SVL163
J.S. White	SVL102F
J.E. Jones	SVL143
J.A. Walters	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   00000
    C     M M  M M  E        M M  M M  0     0     C
    D     M  M  M  EEEEE    M  M  M  0     0     D
    C     M     M  E        M     M  0     0     C
          M     M  EEEEEEE  M     M   00000
```

DATE : MAY 21, 1982

TO : DISTRIBUTION LOCATION :

FROM : B. J. SWIERZBIN LOCATION : ARH260

SUBJECT : CYCLE 3 REVISION OF THE PROCEDURES NOTEBOOK

A Cycle 3 update of the Integration Procedures Notebook is now available. The Notebook has not changed a great deal since its extensive revision at Build Q, but there are some changes of interest in the utility procedures, described in Section 2.11, and the CYBIL build process, described in Appendix C. Keypoint information has been added to the document in Appendix D. A complete listing of this document can be obtained through the following command sequence:

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

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 the NOS Trap Handler 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.

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

Build Q reflects Phase 2 of the restructuring of the NOS/VE operating system into two distinct partitions - the system core and the job template. This restructuring is necessary in order to reach the R1 goal of deadstarting NOS/VE in 1MB of memory. In this system the system core will be able to deadstart and support tasking, without the job template. This was completed in Build P.

The system core contains monitor, the NOS Trap Handler, and all code that executes in ring 1, and consists of the following libraries:

XLMMTR - Monitor mode procedures.
XLS113 - Job mode procedures that have static data, write Mainframe Wired, Mainframe Fixed or Job Fixed, or make privileged monitor calls (these are most of the modules that used to reside in XLJ11F).
XLS133
XLS123 - A small subset of the procedures that used to reside in XLJ12F,
XLS13D XLJ13F, and XLJ1FF that are
XLS1DD needed by the system core.

In the restructured system, modules in the system core cannot XREF variables or procedures that are part of the job template, and indeed need not know nor depend upon any job template code. The job mode segments that come from the system core will be in the address space of every task (regardless of job type) and will have the same segment numbers. Code cannot be added to or deleted from the system core without a deadstart.

All the rest of the operating system code that does not reside in the system core is found in the job template. This code consists of Job Monitor and ring 3/run anywhere task services, contained in the following libraries: XLJ223, XLJ23D, and XLJ2DD (XLJ236 and XLJ266 are linked in, but are

empty in Build Q). In the restructured system, there may be multiple job templates depending upon the specific requirements of the various jobs in the system. These job templates will be staged across the memory link after the system core has been deadstarted.

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 MAPXX file, produced by the NVELINK procedure, for a list of the entry points available to a user task. A convenient listing of these entry points can be obtained by running the MAPXX linkmap file through the procedure NVEMAP and specifying the keyword "TWO". The gated entry points are flagged in the section entitled "PVA, NAME SORTS" at the end of the NVEMAP linkmap output. The NVEMAP keyword "GATED" will list only thoses entry points which are gated (see the documentation for the procedure NVEMAP in Section 2 of this document).

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 DUAL STATE MEMORY MAP

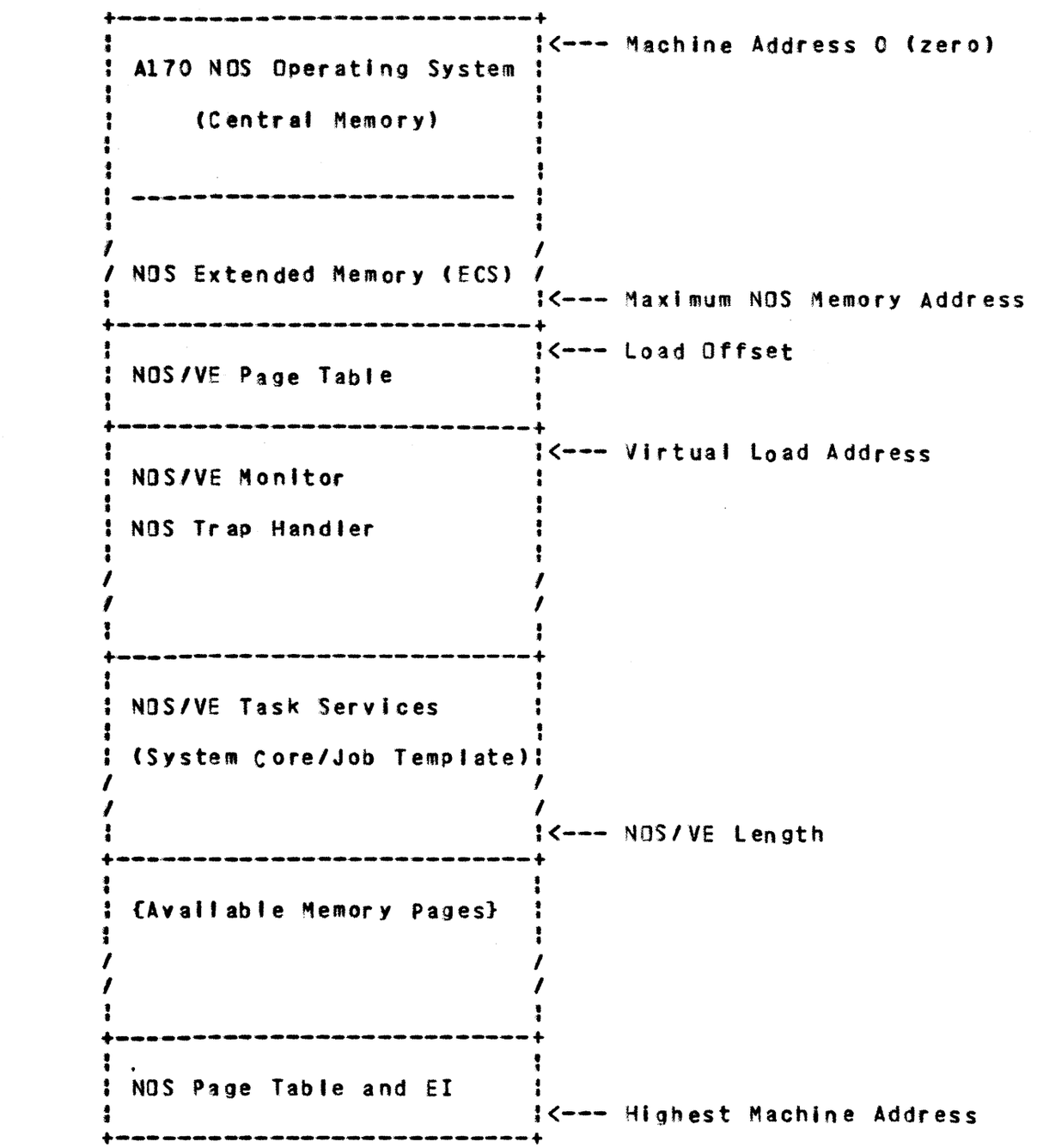
When dealing with a virtual memory system it is often necessary to understand the real memory aspects of the software which is present in the machine. The following map describes the real memory aspects of the software, and where it is mapped during the deadstart process. To make this map complete would 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 SYSXDIR file. This skeleton is dynamically edited when the NVELINK procedure is invoked to produce the <iw>XLDR file. The SYSXDIR 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 linkmap file. Specifically, the load addresses for Monitor, System Core, Job Template, etc. are contained in the Virtual Environment Generator output which immediately follows the LINKER output.

1.0 NOS/VE SYSTEM OVERVIEW
1.4 DUAL STATE MEMORY MAP



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, DEV1, or DEV2 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

2.0 OVERVIEW OF INTEGRATION PROCESS
2.4.1 INTRODUCTION

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

```
SES.LISTPROC B=SESPLIB 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 = 24378
NF = unlimited
MS = unlimited
DS = 4096
EC = 2008 (if simulator is to use LCM)
DB = unlimited (each library is built via batch job)
```

The current values may be obtained with the LIMITS control 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 CCL 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 CCL 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 CCL. 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 CCL 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 'CYBICMN'. Refer to the Simulated I/O ERS for documentation of these I/O interfaces.

The dual-state execution of NOS/VE, in conjunction with the NOS operating system, requires NOS system modifications and the presence of a set of NOS utilities and procedure files. The software which supports this dual-state environment from the 'NOS' side of the hardware is contained on a program library named 'VE170PL'. Included in this package of NOS/VE support programs is a software application called the 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 (SES53A6) and LOADER (SES53A5) 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 'LW=SIM' parameter to NVELINK should be used to produce the file named 'SIMXX'. 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 'LW=SYS' parameter must be used first,then 'LW=JOB'. The 'SYSXX,JOBXXYY' file is used by the NVESYS procudure to produce a deadstart tape image on disk named 'TPXXXK', which the dual-state deadstart procedure NVE will then find. Refer to Appendix D 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 (the NOS Trap Handler) is rebuilt using the NVEBILD procedure (deck named OSANTH).

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, if 'FULL' keyword is specified If 'FULL' is not specified, these procedures will create (or update existing) library file(s) in the current catalog, the resulting contents begin only the modules which were just compiled (added to any previously modifyed binaries). The names of the library file(s) will be the same as the destination integration library(s) for the modified modules to enable them to be merged with the integration binaries during the linking phase. This applies to selected module compilation only ('m' option). If the 'LISTING' parameter is specified a 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, will be created. If you specified 'listing = tape vsn' then the listing will be archived to the tape. THis listing can be 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. 'LJ23D' for the library 'XLJ23D'), 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>.

2.0 OVERVIEW OF INTEGRATION PROCESS
2.5 NVEBILD PROCEDURE DESCRIPTION

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

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>) ]
            [ listing= keyword or keyword=<tape vsn>]
            [ full= keyword ]
            [ ab = <alternate base> ]
            [ omit = (<module name>..<module name>) ]
            [ link = <parameter string for NVELINK> ]
            [ test = <parameter string for NVESIM> ]
            [ 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.

listing : For producing compilation listing as a permanent file or archive the listing to the tape. The default is not to save the listing.

full : To add or replace modified modules into object libraries. The default is only put modified module(s) into object library.

link : String passed as the NVELINK parameter list, to optionally invoke the linker after compiling. The default is to not link the system.

test : String passed as the NVESIM parameter list, to optionally invoke the simulator after linking. 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 ]
            [ listing = <tape vsn>]
```

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.

listing: To archive the listing to the tape. The default is not to archive the listing to the tape.

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> ]
            [listing = <tape vsn>]
```

[full = keyword]
[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.

listing : To archive the listing file to the tape. The default is no listing is archived.

full : To add or replace modified module(s) into object libraries.

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 'O' 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.

2.0 OVERVIEW OF INTEGRATION PROCESS
2.5.3 LISTNVE PROCEDURE DESCRIPTION

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 'AREA' parameter.

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

The format of the LISTNVE is as follows:

```
SES.LISTNVE        [ m = ( <module name>..<module name> ) ]
                   [ i = <file name> ]
                   [ vsn = <tape vsn> ]
                   [ o = <print file name> ]
                   [ area = <user name> ]
                   [ print ]
                   [ fiche = <tape vsn> ]
                   [ 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.

vsn : Tape vsn. For archiving the listing file to the tape. 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.

area : 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.

fiche : Write listing output to the tape with this VSN in a format suitable for microfiching. The default is to not microfiche the listing.

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 (supplied by the Development project) to produce various optional image files and link map files. 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.

It is no longer necessary to keep complete object libraries in the current/area catalogs when modifications have only been made to a few modules. Instead, the changed modules should reside on (an) object file(s), with the same name(s) as their destination object library(s), in the current/area catalog(s). NVELINK then applies these modifications on top of the current <Integration> libraries, applying first the area catalog changes (if any) and then the current catalog changes (this function is performed by the SES GOF utility entirely on local files; at procedure end the libraries residing in the current/area catalogs remain exactly as they were before NVELINK was invoked). This made necessary an option to dynamically delete modules no longer needed on a library. This is done via the 'D' parameter, which specifies a file containing the name(s) of the module(s) to delete and the library(s) to delete it (them) from. This file must be in the format illustrated by the following example:

```
F=(XLS13D),MODULE=(PMM$PROGRAM_SERVICES,
JMM$PROGRAM_LEVEL_INTERFACES,PMM$SYSTEM_TIME_REQUESTS,
MLM$HANDLE_SIGNAL)
F=(XLS1DD),MODULE=(OSM$INTRINSICS)
F=(XLJ223),MODULE=(AVM$INITIALIZE,AVM$JOB_ACCOUNTING_KERNEL,
AVM$JOB_LIMITS_MANAGER,CLM$READ_INPUT_FILE)
etc...
```

Note: Each new entry (flagged by "F=") must begin in column one of a new line. Also,each library file name and list of modules must be enclosed in parantheses.

NVELINK will link any one of the following: the Production System Core and/or the Recovery System Core, or the Production Job Template and/or the Recovery Job Template. The choice is

made via the 'LW' and 'REC' parameters (see description below). The following table indicates what output files are saved in the current catalog after NVELINK has completed, and what their various names are depending upon the 'LW' and 'REC' options:

	LW=JOB	LW=SYS	LW=JOB REC	LW=SYS REC
checkpoint file	PJB<cid><jid>	PSY<cid>	RJB<cid><jid>	RSY<cid>
linkmap	PMP<cid><jid>	PMP<cid>	RMP<cid><jid>	RMP<cid>
outboard symbol tables	PJBST<cid>	PMTST<cid> PSYST<cid>	RJBST<cid>	RMTST<cid> RSYST<cid>
loader directives file	JOBXLDR	PSYXLDR	JOBXLDR	RSYXLDR
linker debug table	PJBXDBG	PMTXDBG PSYXDBG	RJBXDBG	RMTXDBG RSYXDBG

where <cid> = Value given the CID parameter, and
<jid> = Value given the JID parameter.

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 [ lw = < link option > ]
            [ rec ]
            [ cid = < core id > ]
            [ jid = < job id > ]
            [ ps = < page size> ]
            [ ptl = < page table length > ]
            [ d = < DEOM directives file name > ]
            [ add = < additional links file > ]
            [ uj = < list of object files > ]
```

```
[ nvesim = < parameter string for NVESIM > ]
[ nvesys = < parameter string for NVESYS > ]
[ dump/nodump ]
[ area = < user name > ]
[ debug ]
[ print = < link option > ]
[ batch ]
```

lw : The link option, used to determine which "pieces" or variations of the system are to be linked.
LW = JOB links only the system job template (DEFAULT).
LW = SYS links only the system core.

rec : Option to link both the Production and Recovery System Cores/Job Templates (depending on the 'LW' selection). The default is to link only the Production System Core/Job Template.

cid : Two character string which becomes the system core identifier and is appended to the names of the NVELINK output files to identify the system core version which was just linked or which is to be used in the current link of a job template. The default is 'XX'.

jid : Two character string which becomes the job template identifier and is appended to the names of the NVELINK output files to identify the job template version just linked. The default is 'YY'.

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 32.

d : The name of the file containing the information necessary to delete modules from libraries before linking. The format of this file is described above. The default is to not delete any object modules before linking.

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.

uj : (List of) object file(s) containing user programs which are to be linked into the system along with the library XLJBBB.

nvesim : String passed as the NVESIM parameter list, to optionally invoke the simulator after linking.

nvesys : String passed as the NVESYS parameter list, to optionally build a deadstart file or stand-alone deadstart tape after linking.

dump/nodump Option to print a memory dump of the system. The default is 'NODUMP'.

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.

debug : Option to save the linker output debug tables as permanent files in the current catalog. The names of these files, when saved, are given in the table above. The default is to not save these files.

print : The print option, used to determine which linkmaps to print. The default is not to print the linkmap. If only the print keyword is specified, the linkmap(s) matching the 'LW' option selected is printed.

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=<file_name> parameter on the LINK commands. For the monitor linkage this information is on LPF file MTRXLCB, system core/job template linkage information is on LPF files SYSXLCB and JOBXLCB, the XLJBBB (and other optional user object files) linkage information is on LPF file BBBXLCB, and EI linkage information is on EILCB.

2.6.2 SYSXDIR / LDR FILE DESCRIPTION

The SYSXDIR 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.

SYSXDIR is a "skeleton" file which is dynamically edited during the execution of the NVELINK procedure, depending upon the specification of the LW, PS, REC, and PTL parameters. The edited file is then put on an indirect access file, named according to the conventions outlined in the NVELINK Procedure Description section, 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 PARAMETER DESCRIPTOR TABLE AND MESSAGE TEMPLATE BUILDING

2.7.1 GENPDT AND BLDGPDT DESCRIPTIONS

An SCL Parameter Descriptor Table (PDT) is a sufficiently complicated "type" that its declaraton, in particular its initialization, in CYBIL is awkward. Therefore, a means of

easily generating a PDT has been devised.

GENPDT is an SES procedure which provides for the generation of a PDT from a specification that is virtually identical to an SCL proc declaration (see the NOS/VE Command Interface ERS, ARH3609).

The output of GENPDT is a file containing the CYBIL variable declarations for the PDT specified. As a general rule this file should be formatted using the CYBIL source code formatter.

The format of the GENPDT is as follows:

```
SES.GENPDT   [ i = <file name>]
             [ o = <file name>]
```

i : The name of one file containing one PDT declaration. Blank lines and continued lines are allowed. The default is INPUT.

o : The name of the output file. All lines from 'i' are echoed on this file in the form of "block" comments.

See the ERS for Parameter Descriptor Table Generator for the PDT declaration format and examples (GPDTERS/UN=SCL).

The SES procedure BLDGPDT builds the generate parameter descriptor table program that is used by GENPDT.

The format of the BLDGPDT is:

```
SES.BLDGPDT [ l = <file name> ]
            [ b = <file name> ]
```

l : The name of the file which will receive the listing. The default is LISTING.

b: The name of the file which will receive the binaries. The default is GPDTBIN.

2.7.2 GENMT, GNVEMT AND BLDGMT DESCRIPTIONS

GENMT is an SES procedure which takes as input 1 or more CYBIL common decks containing Exception Condition Code definitions with status severity and message template specified in an accompanying CYBIL comment and produces a

ready-to-compile CYBIL module which consists of CYBIL variables that represent the message templates. The common decks that are used as input should be stripped of the MODIFY headers and the line COMMON. The CYBIL module produced would next be compiled and included with other object files so the message formatter can locate the message template.

The format of the GENMT is:

```
SES.GENMT   [ i = <file name> ]
            [ id = <2-character identifier> ]
            [ o = <file name> ]
            [ e = <file name> ]
```

i ; f : The name of the file containing common decks to be used as input. The default is INPUT.

id : A two-character product identifier. The default is OS.

o : The name of the file to contain the CYBIL module which is output. The default is TEMPLAT.

e : The name of the file to be used for error listing output. The default is OUTPUT.

The SES procedure BLDGMT builds the generate message templates program, generates the message templates for the operating system, and produces the message template object module and puts it in object library XLJ2DD.

The format of the BLDGMT is as follows:

```
SES.BLDGMT   [ l = <file name> ]
             [ b = <file name> ]
```

l : The name of the file to receive the listing. The default is LISTING.

b : The name of the file to receive the binaries. The default is GMTBIN.

The SES procedure GNVEMT produces the message template object module for the operating system and puts it in the object libary XLJ2DD. (It is INCLUDEd by BLDGMT and it INCLUDEs GENMT.)

The format of the GNVEMT is as follows:

SES.GNVEMT [bl = <file name>]
[e = <file name>]

bl : The name of the file to receive the object module. The default is MTLGO.

e : The name of the output file. The default is OUTPUT.

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) IDAYF. 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 2 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 file (formed by adding the character '0' to the 'NCPF' file name - which must therefore be six or less characters long) is 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 ]
            [ area = < user name > ]
            [ 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 "SIMXX".

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 "500000(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.

area : The name of the catalog to search for the files needed to simulate the system should they not be found in the current catalog. The default is the <Integration> catalog.

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 > ]
            [ kd = < list of keypoint descriptor files > ]
            [ area = < user name > ]
```

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'.

kd : A file or list of files which define(s) the keypoint descriptions. The default is 'KEYDESC'.

area : The name of the catalog to search for the files needed to create the keypoint trace file should they not be found in the current catalog. The default is the <Integration> catalog.

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 > ]
            [ l = < output file > ]
            [ dump = STND ! ALL ]
            [ print ]
            [ area = < user name > ]
            [ batch ]
```

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

l : The file which is to receive the dump output. This file will be a local file after the procedure has finished execution. It is <u>not</u> 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.

area : The name of the catalog to search for the files

needed to make the DSDI dump should they not be found in the current catalog. The default is the <Integration> catalog.

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 image files created by the linking of the system. The 'REC' parameter allows the option of building either a Production System or a Recovery System deadstart file. If the parameter 'VSN' is specified, then the deadstart file will be written to tape; otherwise it is written to the file TPXXXK.

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) Console/Printer drivers (file XIDSP)
3) PP helper (file XIHLP)
4) PP Resident program (file XIRES)
5) NOS/VE disk drivers supporting multiple controllers (files XD4, XD5A, XD5B, XD5C, and XD5C2)
6) NOS/VE MCU Driver (file XMSPMCU)
7) System Monitor Unit (file XSMUPP)
8) Monitor Display Driver (file XMDD)
9) System Monitor Assistant (file XSMA)

A copy of the loader directives (file PSYXLDR) will be included on the NOS/VE deadstart file (a description of this file is included in a previous section). Also included on the deadstart file are the Production System Core Command File (DCFILE), the Recovery System Core Command File (RDCFILE) if 'REC' is specified, and the Configuration Prolog File (PROLOG).

If any of the above 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.)

The format of the NVESYS is as follows:

```
SES.NVESYS  [ vsn = < tape vsn > ]
            [ cpf = < Production System Core image file > ]
            [ jtf = < Production Job Template image file > ]
            [ rcpf = < Recovery System Core image file > ]
            [ rjtf = < Recovery Job Template image file > ]
            [ area = < user name > ]
            [ cmds = < tape generator commands file > ]
            [ pack ]
            [ nocti ]
            [ rec ]
            [ batch ]
```

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

cpf : The Production System Core image file. The default is PSYXX.

jtf : The Production Job Template image file. The default is PJBXXYY.

rcpf : The Recovery System Core image file. The default is RSYXX.

rjtf : The Recovery Job Template image file. The default is RJBXXYY.

area : The name of the catalog to search for the files needed to build the deadstart tape or file should they not be found in the current catalog. The default is the <Integration> catalog.

cmds : The name of the file containing directives for use by the deadstart tape generator. The default is to use procedure-defined directives.

pack : Option to pack the deadstart file for dual state. The default is to not pack the deadstart file.

nocti : Option to not put CTI on the deadstart file. The default is to write CTI to the beginning of the deadstart file.

rec : Option to build a Recovery System deadstart file. The default is to build a non-Recovery System 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. NOSVEPL comes from the <Integration> catalog.

The format of the CPP180 is as follows:

```
SES.CPP180  [ m = < module name > ]
            [ ab = < alternate base > ]
            [ area = < user name > ]
            [ listing = keyword or keyword =<tape vsn>]
            [ 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.

listing : To save the compilation listing as a permanent file or archive it to a tape. The default is no listing is saved.

area : The name of the catalog to search for the PL specified by the 'AB' parameter should it not be found in the current catalog. The default is the <Integration> catalog.

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 > ]
             [ listing = keyword or keyword=<tape vsn> ]
             [ 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.

listing : Option to save the compilation listing in a permanent file or to archive the listing to a tape. The default is no listing is created.

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 BLD170 PROCEDURE DESCRIPTION

BLD170 is an SES procedure file which builds the A170 dual state deadstart tape binaries containing the modifications to A170 NOS to support dual state. The assembled binaries are put on the direct access file SYSBINS. The COMPASS assembly listings may be saved either on disk (file DSLIST) or on tape, depending upon the specification of the 'LISTING' parameter.

The format of the BLD170 is as follows:

```
SES.BLD170  [ m = < list of module names> ]
            [ ab = < alternate base file > ]
            [ area = < user name > ]
            [ listing ! listing = < tape vsn > ]
            [ batch ]
```

m : The module name(s) of the COMPASS routines to be assembled. The default is to assemble all of the A170 NOS dual state support modifications.

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

area : Optional catalog specification to add to the search list for files needed by BLD170. The default is the current catalog.

listing : Specifying the keyword 'LISTING' saves the assembly listings on the direct access file DSLIST. Specifying 'LISTING=<tape vsn>' writes the assembly listings to the tape with the specified VSN in sorted order. The default is to not save any listings.

batch : Run BLD170 in BATCH mode. The default is to run it locally.

2.10.3 BLDICF7 PROCEDURE DESCRIPTION

BLDICF7 is an SES procedure file which builds the 170 library file LINKLIB. This library contains the binaries for the 170 side of the Interstate Communication Facility. The SYMPL/COMPASS listings may be saved either on disk (file DSLIST) or on tape, depending upon the specification of the 'LISTING' parameter.

2.0 OVERVIEW OF INTEGRATION PROCESS
2.10.3 BLDICF7 PROCEDURE DESCRIPTION

The format of the BLDICF7 is as follows:

```
SES.BLDICF7 [ m = < list of module names> ]
            [ ab = < alternate base file > ]
            [ area = < user name > ]
            [ listing ! listing = < tape vsn > ]
            [ batch ]
```

m : The module name(s) of the routines to be assembled/compiled. The default is to assemble all of the modules which make up the LINKLIB library.

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

area : Optional catalog specification to add to the search list for files needed by BLDICF7. The default is the current catalog.

listing : Specifying the keyword 'LISTING' saves the assembly listings on the direct access file DSLIST. Specifying 'LISTING=<tape vsn>' writes the assembly listings to the tape with the specified VSN in sorted order. The default is to not save any listings.

batch : Run BLDICF7 in BATCH mode. The default is to run it locally.

2.10.4 BLDIF7 PROCEDURE DESCRIPTION

BLDIF7 is an SES procedure file which builds the A170 deadstart tape binaries needed to support the NOS/VE Interactive and Operator Facilities. The linked binary absolutes are put on the direct access file SYSBINS. The compilation/assembly listings may be saved either on disk (file DSLIST) or on tape, depending upon the specification of the 'LISTING' parameter.

The format of the BLDIF7 is as follows:

```
SES.BLDIF7  [ ab = < alternate base file > ]
            [ area = < user name > ]
            [ debug ]
            [ listing ! listing = < tape vsn > ]
            [ batch ]
```

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

area : Optional catalog specification to add to the search list for files needed by BLDIF7. The default is the current catalog.

debug : Option to link Interactive with NETIOD. The default is to link with NETIO.

listing : Specifying the keyword 'LISTING' saves the assembly listings on the direct access file DSLIST. Specifying 'LISTING=<tape vsn>' writes the assembly listings to the tape with the specified VSN in sorted order. The default is to not save any listings.

batch : Run BLDIF7 in BATCH mode. The default is to run it locally.

2.10.5 BLDRH7 PROCEDURE DESCRIPTION

BLDRH7 is an SES procedure file which compiles/assembles the modules which make up the A170 side of the Remote Host Facility and produces the updated RHA170R library file containing the A170 relocatable Remote Host binaries. When RHA170R has been built, BLDRH7 will link to produce the five absolute files which are added to the A170 NOS system deadstart tape to support the Remote Host. This happens only if no MADIFY/compilation/assembly errors occurred during their respective phases. To be able to link successfully, the module must be recompiled/reassembled error-free. The linked absolutes are added to the direct access file SYSBINS in the current catalog. The compilation/assembly listings may be saved either on disk (file DSLIST) or on tape, depending upon the specification of the 'LISTING' parameter.

The format of the BLDRH7 is as follows:

```
SES.BLDRH7  [ m = < list of module names> ]
            [ c = < processor option > ]
            [ ab = < alternate base file > ]
            [ area = < user name > ]
            [ listing ; listing = < tape vsn > ]
            [ batch ]
```

m : The module name(s) of the Remote Host routines

to be compiled/assembled. The default is to compile/assemble all of the modules which make up A170 Remote Host.

c : For a new module, the processor option for compiling or assembling the module. Specify 'C=1' for CYBIL CC modules and 'C=0' for COMPASS modules. The default is 'C=1' for new modules, and internally defined defaults (stored with the module name within the BLDRH7 procedure itself) for existing modules.

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

area : Optional catalog specification to add to the search list for files needed by BLDRH7. The default is the current catalog.

listing : Specifying the keyword 'LISTING' saves the assembly listings on the direct access file DSLIST. Specifying 'LISTING=<tape vsn>' writes the assembly listings to the tape with the specified VSN in sorted order. The default is to not save any listings.

batch : Run BLDRH7 in BATCH mode. The default is to run it locally.

2.10.6 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 > ]
            [area = < user name > ]
            [listing = keyword or keyword=<tape vsn> ]
            [ batch ]
```

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

listing : Option to save a compilation listing as a permanent file or to archive the listing to a tape. The default is no listing will be created.

area : Option to obtain the PL specified by the 'AB' parameter from another user's catalog should it not be found in the current catalog.

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 provides a "working copy" of a file (ie. one which can be written to, or read from, without concern as to whether the file is accessed as a DIRECT or INDIRECT file). Several user catalogs may be searched for the file by specifying a list of values for the UN parameter. If a local

file already exists with the same name as that specified by the LFN or PFN parameters then the local file is either rewound or converted to a working file. One message per file is issued to indicate the action performed on the filenames specified on the procedure call. The MF parameter specifies a filename upon which the working files are to be appended.

The format of the GET procedure is as follows:

```
SES.GET     [ lfn = < local filename > ]
            [ pfn = < permanent filename > ]
            [ un = < user name > ]
            [ mf = < merge filename > ]
            [ a ! na ]
```

lfn : Local file name by which the file is known (may be a list of files). If no filename is specified for lfn, then the filename value for the PFN parameter is positionally used. A common usage of this procedure is as follows:
SES.GET (MYFILE1,MYFILE2, ... ,etc.)
In the above procedure call the permanent files MYFILE1, MYFILE2, etc., would be made local working files named MYFILE1, MYFILE2, etc.

pfn : Permanent file name to be made a local working file. If no filename is specified for pfn, then the value specified for the LFN parameteris used. The list of PFN values is matched positionally with the LFN specified values. This is illustrated as follows:
SES.GET (one,two) (stuff1,stuff2)
The above procedure call would create working files named ONE and TWO from the permanent files STUFF1 and STUFF2 respectively.

un : An optional list of user names to direct the 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. An example would be:
SES.GET IPNDOC UN=(int1,int2,dev1,dev2,rel1)
The above procedure call would search local files for a file named IPNDOC followed by the catalogs INT1, INT2, etc. until the file is found or the search is exhausted.

mf : A filename upon which to stack the resultant working files. For example:

SES.GET (IPNDOC,BCR) MF=BUILDOC
The above procedure call gets the files IPNDOC and BCR as working files, then appends them to the file BUILDOC. IPNDOC and BCR remain as working files along with BUILDOC.

a ; na : : Keyword which determines whether the procedure should abort in FILE NOT FOUND situations. Default is to REVERT,ABORT.

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. 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. Files are SAVE'd as SEMI-PRIVATE READ-ONLY files, unless changed via the CT and M parameter or profile variable values.

The format of the SAVE procedure is as follows:

```
SES.SAVE    [ lfn = < local filename > ]
            [ pfn = < permanent filename > ]
            [ ct = < catalog type > ]
            [ m = < access mode > ]
            [ dir ]
            [ a ; na ]
```

lfn : Local filename to be made permanent. Defaults to PFN value if not specified. Parameters may be specified positionally, and typical usage is as follows:
SES.SAVE (xlmmtr,xljlib, ... ,etc.)
In the above procedure call the files XLMMTR,XLJLIB, etc. are SAVEd as permanent files. A message is issued to indicate the type of file created (DIRECT or INDIRECT).

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)
The above procedure call saves files ONE and TWO as permanent files named STUFF1 and STUFF2.

ct : Catalog Type of the file when made permanent. Default is to make files semi-private (CT=S).

m : Mode of the file when accessed. Default is read access (M=R).

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

a ! na Keyword which determines whether the procedure should abort if FILE IS NOT A LOCAL FILE. Default is to REVERT,ABORT.

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 MAPXX for the system.

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. It is left to the discretion of the user to dispose of 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

2.0 OVERVIEW OF INTEGRATION PROCESS
2.11.4 NVEMAP - REFORMAT NOS/VE LINKMAP

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.

2.11.5 PROCEDURE FORMPROC - FORMAT PROCEDURE

This procedure reformats a single SES or CCL procedure which is present on a GROUP file. Nesting levels of the procedure are used to compute a floating margin to indent statements contained within IF, IFE, WHILE, ROUT, or SKIP blocks.

The first TOKEN of each line is processed as follows:

- IF a double quote (") then leave the line asis.
- IF a backward slant (\) then process the second TOKEN.
- IF a blank and DELETE boolean FALSE, then leave asis.
- IF AND or OR, then this line is continuation of SES IF or ORIF, adjust margins appropriately if INROUT boolean is FALSE.
- IF a CCL IFE, WHILE, SKIP, etc. and CCL boolean TRUE then adjust margins, and add blank lines as appropriate.
- IF none of the above, or a ROUT - ROUTEND block and DOROUT boolean is false, then leave line asis.

The second TOKEN, for lines having backward slant as their first TOKEN, is processed as follows:

- IF a valid SES reserved name, then uppercase the TOKEN and adjust margins as appropriate.
- IF not a valid SES reserved name, then GENLOWR the TOKEN and SUBSTR the value to a seven character value. (This is typically a statement assigning a value to a SES variable.)

Conventions for spacing and indentation were established through trial and error with several complex procedures. Most all of the values which govern these conventions have been externalized as parameters and PROFILE variables to allow tailoring to individual tastes. Blank lines are used to signify the start or end of a IF, ROUT, WHILE, or SKIP block or to highlight the presence of INCLUDE, CYCLE, ACCEPT, or EXIT statements.

Special limitations are imposed upon procedures formatted by this procedure. If the formatted length of a statement exceeds 79 characters (a SES restriction) then a terse diagnostic is issued and the line is left unformatted. Each diagnostic or message issued indicates the line number being processed as well as the line number being written. Thus, the growth or shrinkage of a procedure can be observed while formatting is taking place. Informative messages are issued to indicate when a new indentation "nest" level has been reached. These informative messages are intended to give the warm feeling that the procedure is doing something.

2.0 OVERVIEW OF INTEGRATION PROCESS
2.11.5 PROCEDURE FORMPROC - FORMAT PROCEDURE

The format of the FORMPROC procedure is as follows:

```
SES.FORMPROC [ i = < filename > ]
             [ o = < filename > ]
             [ control = < string > ]
             [ indent = < number > ]
             [ after = < number > ]
             [ blanks = < boolean > ]
             [ delete = < boolean > ]
             [ ccl = < boolean > ]
             [ dorout = < boolean > ]
```

i :	Input filename containing the procedure file source. Default name is GROUP.
o :	Output filename to which formatted procedure is to be written. Default is GROUP.
control :	A string which defines the SES directive character for the procedure to be formatted. Default is \ (backwards slash), "commercial at" character cannot be used.
indent :	Number of spaces to indent from left margin. Default value is two spaces (for unnested SES directives column 1 contains '\' and column 2 contains a blank character, for CCL IF, WHILE, or SKIP commands columns 1 and 2 contain blanks if the CCL boolean is TRUE).
after :	Number of spaces to indent lines occurring after IF, WHILE, ROUT or SKIP statements. Default value is two spaces.
blanks :	Boolean which if TRUE causes the insertion of blank lines into the formatted procedure (if needed). Default is TRUE.
delete :	Boolean which if TRUE causes the deletion of all unneccessary blank lines. Default is FALSE.
ccl :	Boolean which if TRUE causes CCL (including NOS Command Language Statements) to be indented along with other procedure statements. Default is TRUE.
dorout :	Boolean which if TRUE causes ROUT - ROUTEND statements other than SES directives to be formatted. Default is FALSE. WARNING!!! A TRUE value for this parameter can create havoc with HELP documentation.

Note: When formatting procedures which contain only CCL statements it is recommended that INDENT=0 and BLANKS=FALSE be specified.

2.11.6 PROCEDURE SIZES - REPORT MODULE SIZES

This procedure uses partner procedures NVEMAP and HEXTDEC to produce a "quick and dirty" report of module sizes in decimal and hexadecimal byte lengths from a VELINK link map. Very little sophistication has been worked into this procedure whose purpose is to provide data for the Maintenance Task Force in their analysis of Remote Maintenance techniques in a Binary Release environment.

The format of the SIZES procedure is as follows:

```
SES.SIZES     [ i = < filename > ]
              [ o = < filename > ]
              [ un = < user name > ]
```

i :	Input filename containing a VELINK link map. This parameter is required.
o :	Output filename to write report to. Default is same as input filename.
un :	User name in which input file is to be found. Default is current user's catalog.

2.12 PRE-INTEGRATION BUILDS

During the Build 0 timeframe, a small group of people consisting of 3 developers and 1 integrator formed the Pre-Integration Build Team. The purpose of this group is to ease and expedite the formal build process by turning around problems and generating fixes before a formal Integration build is performed. When a major feature, requiring a great deal of code and causing substantial impact upon the system, is ready for integration, the Pre-Integration Build Team goes to work on it first. The code is applied to the PL's in the Integration catalog (INT1), and the entire system is compiled quickly in the pre-integration build catalog. Modsets are generated by this group to correct any compilation errors, the system is linked and a deadstart tape is built for testing on the hardware. The regression tests are run on this system, and fixes are generated to solve problems found. When the Pre-Integration Build Team has fixed as many problems as it can or needs to, Integration makes its formal build in the Integration (INT1) catalog with the original code plus fixes.

In order to build the entire system in a timely manner, pre-integration build procedures have been developed and are outlined in the following sections.

2.12.1 GENDEK PROCEDURE DESCRIPTION

The GENDEK procedure takes the latest list of deck names of each of the OS libraries from the Integration PROCLIB (i.e. LMMTR, LS113, LJ23D, etc.), and creates two files of MADIFY directives - one file (the last 4 characters of the library file appended to the string 'ADK') contains a *EDIT directive for each assembler deck, and the other file (the last 4 characters of the library file appended to the string 'CDK') contains a *EDIT directive for each CYBIL deck. If a particular library does not contain any assembler decks (or any CYBIL decks), GENDEK will issue an informative message stating as much, and no ADK____ (CDK____) file will be created. The ADK____/CDK____ files will be saved in the current catalog for a subsequent BILDLIB run.

The format of the GENDEK is as follows:

```
SES.GENDEK
```

2.12.2 BILDLIB PROCEDURE DESCRIPTION

The BILDLIB procedure takes the ADK____ and CDK____ files (if they exist) for a particular library (created by GENDEK), and assembles/compiles all the modules for that library. For each assembler deck on the ADK____ file a separate call to the C180 assembler is performed, and the packed assembled binaries are copied to the library file. The CDK____ file is used to make one call to MADIFY to write <u>all</u> CYBIL modules to one compile file, which in turn is fed to CYBIL. The CYBIL binaries are copied to the library file following the assembler binaries, and the library file is saved in the current catalog. <u>No</u> compilation listings are produced. The dayfile for the job is saved in the current catalog (the last 4 characters of the library name appended to the string 'DAY') for input to the CHKLIB procedure. The GENDEK procedure <u>must</u> be run prior to a BILDLIB run.

The format of the BILDLIB is as follows:

```
SES.BILDLIB [ lib = < library indicator > ]
            [ plu = < user name > ]
            [ b = < PL name > ]
            [ ab = < alternate base > ]
            [ abu = < alternate base user name > ]
            [ local ]

lib :       The last 4 characters of the name of the library
```

to be built. This parameter is required.

plu : The name of the catalog to search for the PL's and the compiler. The default is the <Integration> catalog.

b : The name of a PL in the current catalog to include in the MADIFY PL search list. This parameter is optional.

ab : The name of a PL in another user's catalog to include in the MADIFY PL search list. This parameter is optional.

abu : The name of the catalog to search for the PL specified by the 'AB' parameter. This parameter is required if the 'AB' parameter is specified and ignored if the 'AB' parameter is not specified.

local : Run BILDLIB locally. The default is to run it in BATCH mode.

NOTE: The modules RHM$INTERIM_SIMULATED_IO (deck RHMSIO in library XLJ23D) and OCM$OMC_SIMULATED_IO_ROUTINES (deck OCMCIO in library XLJBBB) will always have compilation errors. Both modules require common decks from CYBICMN, but including CYBICMN in the MADIFY PL search list for these two libraries causes many more compilation errors in other modules. Currently, these two modules must be rebuilt, after running BILDLIB for their respective libraries, using the procedure NVEBILD (see the documentation for this procedure in a previous section).

2.12.3 BILDALL PROCEDURE DESCRIPTION

The BILDALL procedure simply submits a batch BILDLIB run for each of the OS libraries.

The format of the BILDALL is as follows:

```
SES.BILDALL [ plu = < user name > ]
            [ b = < PL name > ]
            [ ab = < alternate base > ]
            [ abu = < alternate base user name > ]
            [ batch ]
```

plu : The name of the catalog to search for the PL's

and the compiler. The default is the <Integration> catalog.

b : The name of a PL in the current catalog to include in the MADIFY PL search list. This parameter is optional.

ab : The name of a PL in another user's catalog to include in the MADIFY PL search list. This parameter is optional.

abu : The name of the catalog to search for the PL specified by the 'AB' parameter. This parameter is required if the 'AB' parameter is specified and ignored if the 'AB' parameter is not specified.

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

2.12.4 CHKLIB PROCEDURE DESCRIPTION

The CHKLIB procedure uses the dayfile saved from a BILDLIB run ('DAY____' file, see BILDLIB description) to report any MADIFY/assembler/CYBIL errors. It simply uses XEDIT to extract the necessary information, which is displayed on the file 'OUTPUT'.

The format of the CHKLIB is as follows:

```
SES.CHKLIB  [ lib = < library indicator > ]
            [ batch ]
```

lib : The last 4 characters of the name of the library which was built. This parameter is required.

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

2.12.5 PURDEK PROCEDURE DESCRIPTION

The PURDEK procedure simply purges all the ADK____, CDK____, and DAY____ files from the current catalog at the end of a pre-integration when they are no longer needed.

The format of the PURDEK is as follows:

2.0 OVERVIEW OF INTEGRATION PROCESS
2.12.5 PURDEK PROCEDURE DESCRIPTION

SES.PURDEK

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

If MSL 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.

3.1 RELEASE RESERVED SPACE AND INSTALL CTI

WARNING--This process should only be done by the site analyst.

To release reserved disk space, deadstart from the Dual State NOS Deadstart tape which is NT, D=PE, F=I, LB=KU, and enter:

U for the Utilities display
I for Install CTI to disk

-This display will appear:

ENTER ONE OF THE FOLLOWING
(CR) - INSTALL DEADSTART MODULE ON DISK
R - RELEASE CTI-MSL/HIVS RESERVED DISK SPACE

Enter R and this display will appear:

RELEASE CTI-MSL/HIVS
RESERVED DISK SPACE

Enter the disk parameters as they are requested.

CH 00
EQ 00
UN 00

3.0 DUAL STATE INSTALLATION SEQUENCE
3.1 RELEASE RESERVED SPACE AND INSTALL CTI

-This display will appear:

ENTRY OF CR WILL CAUSE RELEASE OF
CTI-MSL/HIVS RESERVED DISK SPACE

Enter a carriage return and RELEASE COMPLETE, (CR) TO PROCESS DIFFERENT DEVICE will appear. Enter a carriage return.

-The ENTER ONE OF THE FOLLOWING display will appear again; this time enter a carriage return. A WARNING message will appear; enter another carriage return.

This display will appear:

INSTALL DISK DEADSTART MODULE

Enter the disk parameters as above. The following messages will appear:

INSTALLING CTI TO DISK

INSTALL COMPLETE.

3.2 INSIALL_MSL

-Deadstart from the MSL tape (CTI is on the tape also.) From the *A* display, type in U for Utilities. From the *U* display, type in T for INSTALL MSL/HIVS TO RMS.

-This display will appear:

TDX
DISK AND TAPE TRANSFER UTILITY
CR TO CONTINUE

Enter a carriage return and enter disk and tape parameters as they are requested. These parameters are for the disk to which MSL is to be loaded and the tape from which it is to be loaded.

DISK CH 01
DISK UN 00

TAPE TYPE 03
0=60X,1=65X,2=66X,3=67X

TAPE CH 13

3.0 DUAL STATE INSTALLATION SEQUENCE
3.2 INSTALL MSL

TAPE EQ 00

TAPE UN 00

-This display will appear:

```
A-BUILD MSL FROM TAPE
B-BUILD CB LIBRARY ON MSL
C-ADD PROGS.  TO MSL
D-ADD CB-S TO MSL
E-COPY PROGS.  TO TAPE
F-COPY CB-S TO TAPE
G-DIS SYS TBLS
```

Enter A and the following display will appear:

```
MSL INSTALLATION OPTION

A=HIVS
B=MSL/OS SHARED
C=MSL/MAINTENANCE ONLY
```

Enter B.

-This display will appear:

```
HAS A COMMAND BUFFER LIBRARY BEEN BUILT AT
STARTING CYL 1420 THAT YOU WANT TO SAVE
Y=YES,N=NO
```

Enter N, and the screen will show CHECKING STARTING CYLINDER, BUILDING SRT, and BUILDING PNT.

The next displays and corresponding entries are:

```
COPY FROM
-CR- = FIRST NAME
```

Enter a carriage return.

```
CTI ON TAPE (Y/N)
```

Enter Y

```
COPY THRU
-CR- = LAST NAME
```

Enter a carriage return.

3.0 DUAL STATE INSTALLATION SEQUENCE
3.2 INSTALL MSL

DATA VERIFY (Y/N)

Enter N

-The MSL Tape will load at this point and this display will appear:

LAST USED
CYL 14nn TRK 0000 SEC 0000m

3.3 CMRDECK CHANGES AND CMDS1 FILE

3.3.1 NOS CMRDECK AND LIBDECK CHANGES

a. When deadstarting NOS to run a dual state environment, it is required that one or more of the following commands be processed by the NOS CMRDECK command processor. The command(s) which must be specified by the operator will be dictated by the type of NOS environment which is to be supported during dual state operation.

MINCM=xxxxx.	The parameter 'xxxxx' defines the minumum amount of central memory words which NOS will use for the following operating system execution. The parameter is an octal value expressed in multiples of 100. If this command is not specified the system assumes a default of 300,000(8) or 98K(10) central memory words.
VE.	This command establishes a Dual State Communication Block (DSCB) in NOS Central Memory Resident (CMR). The DSCB is required to enable the operator to deadstart NOS/VE.
VE=xxxxx.	This command establishes a DSCB in NOS CMR and also reserves, for use by NOS/VE, the number of

	central memory words expressed by the parameter 'xxxxx'. The parameter must be an even number and is an octal value expressed in multiples of 1000.
EQxx=DE,ST,OP,SZ,FD.	This command is required if UEM (soft ECS) is to be used by NOS. The SZ parameter is an octal value (minimum of 10) expressed in multiples of 1000. The remaining parameters are defined in the NOS Installation Handbook. NOTE: If this command is specified in the NOS CMRDECK, it is mandatory that the VE=xxxxx. command be specified if NOS/VE will be deadstarted.

The algorithm used by NOS to determine if the memory size parameters specified by these commands are 'legal' is:

(Machine Field Length) >= (MINCM + UEM + NOS/VE)

Some examples of how to use these commands when deadstarting NOS for execution in a dual state environment are as follows, assuming a 16MB mainframe:

VE.	No UEM and the default MINCM will be used. NOS will allow NOS/VE to use all but 98K or 300,000(8) words of the machine field length.
VE. MINCM=10000.	No UEM and NOS will allow NOS/VE to use all but 262K or 1,000,000(8) words of the machine field length.
VE=5000.	No UEM and NOS will allow NOS/VE to use a maximum of 10MB or 5,000,000(8) words of the machine field length. NOS will use 6MB.
VE=5000. EQ5=DE,ST,OP,2000,FD.	NOS will allow NOS/VE to use a maximum of 10MB or

3.0 DUAL STATE INSTALLATION SEQUENCE
3.3.1 NOS CMRDECK AND LIBDECK CHANGES

5,000,000(8) words of the machine field length. NOS will allocate 4MB or 2,000,000(8) words for UEM and will use 2MB or 1,000,000(8) words for CM.

b. All CM= lines should be deleted as all memory allocation is dynamic.

c. All NOS/VE mass storage devices must be specified in the NOS CMRDECK as DOWN devices, e.g. EQnn=DQ-1,DOWN,0,40,2.

d. The disk controlware that NOS loads to FM type controllers at deadstart is the correct controlware for the NOS/VE 7155-1x disk controllers. Add the following line to the CMRDECK to cause controlware to be loaded to both the NOS and NOS/VE disk controllers: LBC,FM,ch1,ch2,ch3. (The chi are disk channels as appropriate.)

e. The NOS LIBDECK must be modified to include the new procedures necessary for deadstart. *PROC SETVE and *PROC OPERFAC must be added and *PROC UPMYVE should be deleted.

3.3.2 CMDS1 FILE

a. A feature has been added to Cycle 3 to improve the transportability of the DSTVE directive file CMDS1. If *MEMORY=0. is entered in CMDS1, DSTVE will always request all available memory from A170 NOS. It is essential with this feature to enter in the NOS CMRDECK the entry MINCM=10000. to prevent NOS from giving NOS/VE so much memory that A170 NOS will not run well.

b. The default value for the NOS/VE system device disk channel is specified on the following command in the CMDS1 file:

*SYST,DSPANEL,OFFF00(16)=c.

This means that it is no longer necessary to rebuild IOPQUER to connect the NOS/VE disk to the proper channel.

c. The default value for the NOS/VE deadstart command file number (DCF) is specified on the following command in the CMDS1 file:

*SYST,DSPANEL,OFFFFF(16)=n.

d. The channel number x in the following command should be an empty channel: *RELOADCH=x.

e. Debug flag number 2 should be TRUE to enable recovery to work automatically, e.g. *SYST,DEBUG2=TRUE.

3.4 INSIALL_SYSIEM

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. 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 MSL 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.

3.0 DUAL STATE INSTALLATION SEQUENCE
3.4 INSTALL SYSTEM

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

3.5 LOADPE_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.

3.6 BRING_UP_DUAL_STATE

Refer to the current Helpful Hints document, Section 4, for information regarding the operation and execution of NOS/VE.

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 running the S2 Regression Test Sequence, as outlined in the following sections, on the hardware.

4.2 S2 REGRESSION TESTS

4.2.1 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
- SUBMIT of batch job
- JMEXIT

The command sequence follows:

```
LOGIN USER=DEV1 NAME=JOB2
LIU,USER=(DEV1,NVE),PA=DEV1X,A=NOTUSED,PR=NOTUSED
GET,NEWLIBRARY,CYBIILB,,DEV1,NVE,B56
SET_OBJECT_LIST,ADD=NEWLIBRARY
SET_PROGRAM_OPTIONS,LOADMAP,...
MO=(BLOCK,ENTRY_POINT,XREF,SEGMENT),...
TERMINATION_ERROR_LEVEL=FATAL
GET,XPETEST,XPETEST,,DEV1,NVE,B60
```

4.0 NOS/VE HARDWARE REGRESSION TESTING

4.2.1 JOB2

```
EXECUTE,,'XPETEST,LGO',,,CITOII
EXECUTE LGO
JMROUTE,NOTUSED,LOADMAP,PR
SET_OBJECT_LIST,DELETE=ALL
GET,CYBILIB,CYBIILB,,DEV1,NVE,B56
SET_PROGRAM_OPTIONS LIST1 TERMINATION_ERROR_LEVEL=ERROR
EXECUTE LGO
JMROUTE,NOTUSED,LIST1,PR
GET,JOB3,JOB3,,,NVE,A6
SUBMIT,JOB3
GET,JOB4,JOB4,,,NVE,A6
SUBMIT,JOB4
JMEXIT
```

4.2.2 JOB3

JOB3 is the same NOS/VE procedure file as JOB2 and tests the same NOS/VE features with one addition: instead of doing a GET of the II version of CYBILIB from the 170, JOB3 does a NOS/VE permanent file ATTACH of CYBILIB from the $SYSTEM catalog.

The command sequence follows:

```
LOGIN USER=DEV1 NAME=JOB3
LIU,USER=(DEV1,NVE),PA=DEV1X,A=NOTUSED,PR=NOTUSED
ATTACH $SYSTEM.CYBILIB NEWLIBRARY,...
ACCESS_MODE=(READ,EXECUTE)
SET_OBJECT_LIST,ADD=NEWLIBRARY
SET_PROGRAM_OPTIONS,LOADMAP,...
MO=(BLOCK,ENTRY_POINT,XREF,SEGMENT),...
TERMINATION_ERROR_LEVEL=FATAL
GET,XPETEST,XPETEST,,DEV1,NVE,B60
EXECUTE,,'XPETEST,LGO',,,CITOII
EXECUTE LGO
JMROUTE,NOTUSED,LOADMAP,PR
SET_OBJECT_LIST,DELETE=ALL
RETURN NEWLIBRARY
ATTACH $SYSTEM.CYBILIB ACCESS_MODE=(READ,EXECUTE)
SET_PROGRAM_OPTIONS LIST1 TERMINATION_ERROR_LEVEL=ERROR
EXECUTE LGO
JMROUTE,NOTUSED,LIST1,PR
GET,TESTBAM,TESTBAM,,,NVE,A6
SUBMIT,TESTBAM
JMEXIT
```

4.2.3 JOB4

JOB4 is a file containing the NOS/VE commands which execute the NOS/VE SETUP command to set up the 180 job environment (see Appendix D for the description of the SETUP command), followed by a CITOII conversion of a 170 object file user program and an EXECUTE of this program. The loadmap is staged back to the 170 side to be printed. JOB4 tests the following NOS/VE features:

- CONNECT_FILE command
- SETUP command
- CITOII conversion
- SET_PROGRAM_OPTIONS command
- Load/Execute User Program + Library
- JMROUTE C180 print file
- JMEXIT

The command sequence follows:

```
LOGIN USER=DEV1 NAME=JOB4
CONNECT_FILE $ECHO OUTPUT
SETUP DEV1 DEV1X
CITOII II=LGO CI=XPETEST USER=DEV1
SET_PROGRAM_OPTIONS LIST1 TERMINATION_ERROR_LEVEL=ERROR
EXECUTE LGO
JMROUTE,NOTUSED,LIST1,PR
JMEXIT
```

4.2.4 TESTBAM

TESTBAM is a file containing the statements necessary to execute all of the BAM test cases supplied by the BAM project. 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=DEV1 NAME=TESTBAM
LIU,USER=(DEV1,NVE),PA=DEV1X,A=NOTUSED,PR=NOTUSED
COLLECT_TEXT BAMINP
TES1
TES2
TES3
TES4
TES5
```

4.0 NOS/VE HARDWARE REGRESSION TESTING
4.2.4 TESTBAM

```
TES6
TES7
TES8
TES9
TES10
TES11
TES12
TES13
TES14
TES15
TES16
TES20
TES21
TES22
TES23
TES24
TES25
TES26
TES27
TES28
TES29
TES31
TES32
TES33
TES34
TES35
TES36
TES37
TES38
BAMSTOP
**
ATTACH $SYSTEM.SYSLIB SYSLIB ACCESS_MODE=(READ,EXECUTE)
ATTACH $SYSTEM.CYBILIB CYBILIB ACCESS_MODE=(READ,EXECUTE)
SET_OBJECT_LIST ADD=(SYSLIB,CYBILIB)
EXECUTE,,'BAMINP',,,BAMTEST
GET,SCL180,SCL180,,,NVE,A6
SUBMIT,SCL180
JMEXIT
```

4.3 S2_REGRESSION_TEST_SEQUENCE

1) Deadstart A170 NOS:
   - Set the deadstart panel to disk deadstart from:
     CH=1
     UNIT=41
     WORD 13=0006
   - Push deadstart button.
   - Select *0* display.

- Hit carriage return.
- Enter date/time.

2) If necessary, update the INT2(DEV2) catalog and load the latest system files into the INT1(DEV1) catalog:
   - Mount the INT1(DEV1) and INT2(DEV2) catalog DUMPPF tapes.
   - X.DIS.
     USER,INT1(DEV1),INT1(DEV1)X.
     SES.UPCATS <WC=tpu1> <SC=tpu2> <SYSEDIT>
     Hit "." to go into AUTO mode.
     DROP.

   The SES.UPCATS procedure works as follows:

   a. Updates the INT2(DEV2) catalog by retrieving OSLIB, SYSLIB, COLLIB, and CYBIILB from the INT1(DEV1) catalog and by loading selected files from the DUMPPF tape mounted on the unit specified by the "SC" parameter. This parameter must be specified for the INT2(DEV2) catalog to be updated.

   b. LOADPF's the latest system into the INT1(DEV1) catalog from the DUMPPF tape mounted on the unit specified by the "WC" parameter (defaults to "50").

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

   d. LOADPF's the Confidence Test binaries from the file CONFTST, and then purges CONFTST from the catalog.

3) Bring up A170 Remote Host and Interactive:
   TAFNVE.

4) Bring up dual state:
   Make sure the NVE Subsystem Environment is set up:
   X.SETVE(DEV1,UN=DEV1,C=6) Then bring up the NVE Subsystem: n.NVEDEV1. (where n is any free NOS control point)

5) When the NVE control point requests the Operator

Facility control point, do:
K,n. (where n is the OPFAC control point)

6) Bring up NAM:
FNC5,7700.
N.NAMS2. (where N /= 2 is NAM control point)

7) Build and save the 180 object files needed to complete deadstart of NOS/VE (and needed to perform subsequent deadstarts of this system), and begin execution of the 180 system tasks to support Interactive, Remote Host, and the dayfile displays:
K.LIU (DEV1,NVE) DEV1X.
K.GETF DS
K.DS TRUE FALSE FALSE TRUE SIF=TRUE.

8) When the system displays the message "READY FOR COMMANDS", start loading the Confidence Test Base into the system (see Section 4.6 below). Make sure that the ASCII printer is ready, i.e. that the START light is lit and that a "FORM33,TM." has been entered. To monitor the 180 jobs as they enter the system,do K.VED CP to bring up the NOS/VE control point display.

9) The system is now ready to be taken down:
N.CFO.DI,NE. (N is the NAM control point; this disables NAM)
2.STOP. (Bring down 170 Interactive, Remote Host, and Operator Facility)
K.*BYEVE. (Bring down NOS/VE)
IMPORTANT: This last step must be performed at the NVE Subsystem K display, NOT the Operator Facility K display.

When all control points are "quiet", proceed to the next step.

10) Bring down A170 NOS:
AB.
CHECKPOINT SYSTEM.
E,M. (make sure that all checkpoints complete)
STEP.

4.4 INTRODUCTION TO CONFIDENCE TESTS

The confidence test base is a set of tests,used to determine if a build is ready, for installation in an closed shop environment. The test base consists of a set of tests in each area of the operating system, presently under analysis by

the evaluation unit. The tests are described in section 4.7. Any questions or problems should be addressed to R. E. Jarosz, x6834.

4.5 INSTALLATION

The following procedure should be followed when installing this test base.

1. REQUEST,TAPE,VSN=CNFTAP,NT,D=PE,LB=KL,F=I,PO=R.
2. LOADPF,B=TAPE,TY=ALL
   This will install two files.
   1. The test source PL (CONFPL)
   2. The SES procedure library (CNFPROC)
3 Load any files from a tape created with the CLEANUP procedure.
4. Two variables must then be added or changed in the user PROFILE.
   They are as follows.
   \ USEBIN = 'OLD'
   \ RUNLVL = 'DEV1'

For more information on these profile variables refer to sections 4.8.2 and 4.8.4.

At this point the confidence test base is installed and ready to be run.

4.6 EXECUTION

Before the tests are executed it is necessary to establish the ACR routines which collect the test results. Because of the changes, that are on going in ACR, the routines will remain under the control of the Evaluation Unit. To use ACR, do the following before executing any tests.

1. GET,ACRJOB/UN=EVAL.
2. ROUTE,ACRJOB,DC=IN.

You are now ready to execute the tests in the confidence test base. To begin do the following:

```
SES,LPFN=CNFPROC.LDTB D=(BA001..RH021) B=CONFPL OLD,..
..? DELAY=20
```

At this point, you can run test IF094 by following the

4.0 NOS/VE HARDWARE REGRESSION TESTING
4.6 EXECUTION

instructions in section 4.6.1 of this document.

After all the tests have completed, do the following replacement of ACR results files to the 170:

```
GET,JOB1/UN=DEV1
ROUTE,JOB1,DC=IN
```

At this point, run the RESLTS procedure to recieve the ACR results listing.

This job will also produce a 180 listing, of the system statistics, for comparasion purposes.

```
SES,UN=EVAL.RESLTS I=RESULTS O=listing_file
```

4.6.1 EXECUTION OF IF094

This test is somewhat different in that it requires running by hand under NOS/VE IAF. To execute this test do the following.

```
On NOS
  SES,LPFN=CNFPROC.GENBIN D=IF094,B=XIF094,..
  PL=CONFPL,CI,S=180,OLD
This will create the object code part of the test.
ON NOS/VE  do the following:
  ATTACH_FILE .EVAL.EVAL_SETUP
  INCLUDE_FILE EVAL_SETUP
  ACR TEST_NAME=IF094 PRODUCT_IDENTIFIER=IF
  GET_FILE TO=XIF094 DATA_CONVERSION=B60
  EXECUTE_TASK PARAMETERS='XIF094,LGO',SP=CITOII,LIBRARY=SYSL
  LGO
At this point do what the program tells you to.
If you feel it performed as expected do the following:
  ACR STATE=PC REPORT=TRUE ACTION=FULL
Otherwise
  ACR STATE=FAIL REPORT=TRUE ACTION=FULL
```

4.7 IESIS

TEST	TEST FUNCTIONS
******	**********
BA001	AMP$FILE

	AMP$GET_FILE_ATTRIBUTES
BA024	AMP$GET_DIRECT
	AMP$PUT_DIRECT
BA027	AMP$GET_NEXT
	AMP$PUT_NEXT
	AMP$GET_PARTIAL
	AMP$PUT_PARTIAL
BA048	AMP$GET_SEGMENT_POINTER
	AMP$GET_SEGMENT_POSITION
	AMP$GET_SEGMENT_EOI
BA055	AMP$GET_FILE_ATTRIBUTES
BA062	COPY
CH001	PMP$ESTABLISH_CONDITION_HANDLER
	PMP$DISESTABLISH_COND_HANDLER
	PMP$CONTINUE_TO_CAUSE
CL029	DISPLAY_VALUE
CL039	DECLARE_VARIABLE
CL058	EXIT, CYCLE
CL061	IF CLAUSE
CL103	FILE CONNECTIONS
LD034	BASIC LOADER OPERATIONS
LD037	PMP$EXECUTE
OC044	RETAIN IN CREATE_OBJECT_LIBRARY
PF101	VERIFY FILE CYCLES (DEFINE)
PF125	VERIFY LFN,PFN RELATIONSHIP
PF140	VERIFY FILE CYCLES (PURGE)
PF145	CHANGE
PF156	DEFINE_CATALOG
PF157	PURGE_CATALOG
PM016	OSP$AWAIT_ACTIVITY_COMPLETION
PM050	JOB LOCAL QUEUES
QF100	SUBMIT
RH001	GET B60
RH011	REPLACE A6
RH014	REPLACE A8
RH021	GET B56
IF094	VERIFY CONDITIONAL_BREAKS
	INTERACTIVE INPUT/OUTPUTS
	TERMINAL ATTRIBUTES

NOTE: IF094 MUST BE EXECUTED BY HAND. SEE Section 4.6.1.

4.8 TOOLS

This section describes the ACR and SES procedures, used by the tests, in the confidence test base.

4.8.1 AUTOMATIC CHECKING ROUTINES - ACR

The 180 common ACR is described in the Interim Test Tools ERS. (DCS number ARH4207)

4.8.2 LDTB

This SES procedure is used to load a set of tests into the 170 input queue, from a Madify source program library. The parameters are shown below.

DECK ! D: This required parameters is a list of deck names that are to be expanded and loaded. This parameter may be a list, a range or both. Because these decks are expanded by MADIFY they can not be common decks.

PL ! B: This required parameter is the library where the decks listed on the preceding parameter reside.

DELAY: This parameter specifies the number of seconds between submission of jobs. If it is ommitted, 60 is assumed.

UN: This parameter is the user number where the program library specified on the PL parameter resides. If it is ommitted, the user number of the executing job is utilized.

OLD ! NEW: This keyword is used to replace the profile variable USEBIN. This profile variable is used by the GENBIN procedure (see Section 4.8.4). This parameter allows for a decision, to be made at execution time, on whether or not to create any binary files needed with this test. The default is NEW.

PRINT ! NOPRINT: This keyword determines if the 170 output file is printed or not. The default is NOPRINT.

4.8.3 CLEANUP

When tests use the GENBIN procedure, to create CI object code, two files are created (DUMPDIR and PURGEF). The CLEANUP procedure uses these files to archieve the object code files and purge them. The parameters for CLEANUP are shown below.

VSN: This required parameter is the VSN of the tape, where the DUMPPF is done.

KL ! KU: This optional keyword describes the label characteristics of the tape specified above. The default is KL.

NOTE: The following table shows the other characteristics of the tape specified on the VSN parameter.

TRACK	NT	(nine track)
DENSITY	PE	(1600 bpi)
FORMAT	I	(internal)

The file written on this tape can be reloaded using the LOADPF UTILITY.

4.8.4 GENBIN

This procedure optionally creates a CC or CI object file from an expanded cybil source file or MADIFY deck. The parameters are shown below.

D ! J ! ALL: This is the list of MADIFY decck names to be expanded and compiled.

PL: This parameter is the library where the decks specified on the previous parameter reside.

AB ! APL: This optional parameter describes the user numbered file names, to be considered as alternate base for expansion of decks or source files.

UN: This optional parameter is the user number of the file specified on the PL parameter.

SF: This parameter is an optional file name, telling the procedure to work from a source file instead of a MADIFY deck.

CF: This is the file name where the expanded deck will reside, and it is the input file to CYBIL.
The default is COMPILE.

L : This is the file where the CYBIL compilation listing is placed. The default is LISTING.

SYS ; S: This parameter causes a specfic file to be used as a alternate base as shown by the following table.

SYS	FILE
101	CYBICMN/UN=SES
170	CYBCCMN/UN=SES
180	OSLPI/UN=lvl

LVL: This optional parameter is the location of OSLPI that will be used as an alternate base. The default is either the profile variable RUNLVL or DEV1.

BIN ; B: This required parameter is the name of the file checked for when the keyword OLD is specified. It is also the name of the file where the object code will reside if NEW is specified.

CC ; CI: This parameter is unique in that the value is used along with the keyword. If just the keyword is coded the proper compiler is recieved from a default user number. If the value is coded the proper compiler is received from the used specified. The default user number is either the value of the RUNLVL profile variable or DEV1. The default value for the keyword is CC.

NEW ; OLD: This keyword determines if the object code file coded for the B parameter will be produced or not. If NEW is coded the file coded on the B parameter will be created. If OLD is coded and the file exists then that file will be used and no compilation takes place. If the file does not exist then the file will be created.
The default value used is taken from the USEBIN profile variable.

4.8.5 PRTLIST

This procedure is totally internal to the tests and the discussion is left to the help file.

05/22/82

Originator_____ DATE ___/___/___ Transmittal No. _____

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

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

Code Description File: FN=_____ UN=_____ *

IF module has a call bracket of D OR
code affects system user in some other way THEN

Usage Changes Desc. File: FN=_____ UN=_____ **

Code Destination (if not NOSVEPL): PL= _____

B: Modset Identifier(s) _____

New Feature [____] OR Corrective Code [____]

Module(s) to be recompiled _____

How has this code been tested? (Use right margin.)

COMMENTS TO INTEGRATION
NOTE: If any of these are checked, then explain in right margin.

Installation procedure changes required? [____]
Dependent upon other feature, fix, or tool? [____] (List below)
OSLPI or Internal Interface changes required? [____]
Should GNVEMT (Generate NOS/VE Message Templates) be run? [____]

Notes regarding code submittal:
More forms are on FN = XMIT10 UN = DEV1.
* Attach copy of description file to form (both 14 7/8 by 11).
Format is: #MODSET_IDENTIFIER (or NEW_DECK_NAME) (upper case)
&PSR Number (Omit if feature code.)
Descriptive text which describes code content
**DECK_MODIFIED (or NEW_DECK_NAME) (upper case)
** Attach copy of Usage Changes Description File to form
(both 14 7/8 by 11).

ATTACHMENT CHECKLIST
Description File [____]
Proof of Compilation [____]
Proof of Execution [____]
(Usage Changes Desc. [____]
(PSR) [____]

DEPENDENCY LIST
-
-
-
-
(Continue in right margin)

Target Build _____
Should this code be added to the successor build cycle? [____]

Files maintained by Integration

05/22/82

Source Files

USER NUMBERS	FILENAME(S)	FUNCTION	VERSION/FREQUENCY OF UPDATE
INT1/INT2 DEV1/DEV2	NOSVEPL	MADIFY program library of Virtual State code	Matches the level of system binaries contained in same catalog. Updated on periodic scheduled basis.
INT1/INT2 DEV1/DEV2	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/INT2 DEV1/DEV2	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.
LIBRARY	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).
INT1/INT2 DEV1/DEV2	SESPLIB	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.
magnetic tape	listing files	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/INT2 DEV1/DEV2	MTRXLCB, EILCB,SYSXLCB JOBXLCB,BBBXLCB	Linker directives files for monitor, error interface, system core, job template, and user modules respectively.	Matches the level of system binaries contained in the same catalog. EI is built using the BLDEI procedure.
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.

Source Files

		procedures.	
INT1/INT2 DEV1/DEV2	PMPXX PMPXXYY RMPXX	Contains link map of particular system created, where RMPXX RMPXX are the Production/ Recovery System Core Maps and PMPXXYY/RMPXXYY are the Production/Recovery Job Template maps.	Each Velink of a Production/Recovery System Core/Job Template.
INT1/INT2 DEV1/DEV2	PSYXLDR RSYXLDR	Contains VE generator directives for Dual State offset loads.	As required by system content or structure changes.
INT1/INT2 DEV1/DEV2	KEYDESC	Contains Keypoint descriptions for the Keypoint report program XXM7KEY.	Non-standard, updated upon development's request.
INT1/INT2 DEV1/DEV2	PMTXDBG,RMTXDBG PSYXDBC,RSYXDBG PJBXDBG,RJBXDBG	Contains debug tables produced by the linking of the system.	Each Velink of a Production/ Recovery System Core/Job Template.

Files maintained by Integration

05/22/82

Object Text Files

INT1/INT2 DEV1/DEV2	XLMMTR	Object text file of modules which execute in monitor mode.	Each recompilation of a monitor mode module.
INT1/INT2 DEV1/DEV2	XLS113, XLS133,XLS13D, XLS1DD	Object text files of system core modules which run in job mode and execute in ring 1.	Each recompilation of a system core module within these libraries.
INT1/INT2 DEV1/DEV2	XLJ223,XLJ236, XLJ266,XLJ23D, XLJ2DD	Object text files of job template modules.	Each recompilation of a job template module within these libraries.
INT1/INT2 DEV1/DEV2	XLJOSL,XLJLIB, XLJBBB,XLJOCM XLJSG	Object text files of Remote Host, Interactive, CITOII, the Object Library Generator, and various user utility programs.	Each recompilation of a module within these libraries.
INT1/INT2 DEV1/DEV2	PMTSTXX, PSYSTXX, RMTSTXX RSYSTXX	Outboard symbol table files for Production/Recovery system Monitor, and Production/ Recovery System Core produced by VELINK.	Each Velink of the system.
INT1/INT2 DEV1/DEV2	PSYXX,PJBXXYY RSYXX,RJBXXYY	The Virtual Environment files produced by VEGEN.	Each VELINK/VEGEN of the system.
INT1 DEV1	NOSTEXT,PSSTEXT SSYTEXT	A170 NOS system texts for current NOS version.	Each A170 NOS update.
INT1/INT2 DEV1/DEV2	XXM7KEY	Program to report NOS/VE Keypoints encountered during a simulation run.	Non-standard ISWL utility.
INT1/INT2	XXM7DSI	NOS/VE deadstart file generator.	Non-standard, supplied by the Deadstart project.
INT1/INT2 DEV1/DEV2	XIDST,XMOD, XIDSP,XIHLP, XIRES,XSMA,	CYBER 180 PPU programs.	Upon demand.

Object Text Files

	XD5B,XD5C,XD5C2, XMSPMCU		
INT1/INT2 DEV1/DEV2	TPXXXK	Dual State deadstart file created by the NVESYS procedure.	Each time a new deadstart file is generated (upon demand).

C1.0 BUILD CATALOG SETUP

C1.0 BUILD_CATALOG_SETUP

Before beginning the CYBIL compiler builds, perform the following setup process in the build catalog:

1) GET,PROCFIL/UN=LP3.
   SAVE,PROCFIL/CT=S,M=R.

2) DEFINE,CYBPLIB/CT=S,M=R.
   ATTACH,SESPLIB/UN=LP3.
   COPY,SESPLIB,CYBPLIB,V.
   RETURN,SESPLIB,CYBPLIB.

3) Check that the PROFILE variable "PASSWOR" is defined, and set to the password of the build catalog.

4) Check that the Deferred Batch job limit is set to UNLIMITED in the build catalog.

All CYBIL build procedures are on the INT1 SESPLIB. They will generally be run, however, in a catalog other than an official Integration catalog. Therefore, all procedures outlined below should be called via "SES,INT1.<Procedure Name>", or else add INT1 to the PROFILE variable SEARCH list in the build catalog.

The general CYBIL build process is documented in the next section. The individual procedures are documented in subsequent sections.

C2.0 CYBIL BUILD PROCESS

C2.0 CYBIL_BUILD_PROCESS

The various CYBIL compilers are built and tested in the following order, via the procedures indicated:

BLDCC	1) Build the CC compiler front end and code generator; link them together to form the CYBILC compiler; build CYBCLIB.
TESTCC	2) Test the execution of the CC compiler and CYBCLIB.
CNVRGCC	3) Test the CC compiler for convergence (i.e. can it compile itself and produce identical binaries); rerun tests from 2) above.
BLDCI	4) Build the CI code generator with the CYBILC compiler built in 1)-3) above; link it with the common front end built in 1) above to form the CYBILI compiler.
TESTCI	5) Test the execution of the CYBILI compiler.
BLDILIB	6) Build CYBILIB with the CYBILI compiler built in 4) above; test the new CYBILIB with CYBILI.
RUNREG	7) Set up the environment for running the CI compiler regression tests and submit them.
BLDIIS	8) Build and test the II compiler for the simulator.
BLDII2S	9) Build and test the II OPT=2 compiler for the simulator.
BLDIIH	10) Build the II compiler for the hardware.
RUNREG	11) Set up the environment for running the CI compiler regression tests with the "OPT=2" option specified on the compiler call, and run them.

1)-6) must be run serially; 7)-10) may be run concurrently. 11) cannot be run until 7) has completed.

C3.0 CYBIL BUILD PROCEDURES

C3.0 CYBIL_BUILD_PROCEDURES

Each CYBIL build procedure that resides on the Integration SESPLIB is outlined below. Note that most procedures have a "CHAIN" parameter. If "CHAIN" is keyed on any one of the procedures, all subsequent procedures will be automatically submitted at the appropriate times in the order specified above. This allows you to submit all build jobs by typing in only one procedure call ("SES.BLDCC CHAIN"); it also allows you to restart the build process at any point should errors be discovered.

Note also that the CC and CI procedures write status messages to the indirect access file CYBSTS in the build catalog. This file can be interrogated on-line or printed to obtain the results of the building and testing stages of these compilers. The II compiler builds produce listings which must be checked to verify that the compilers were indeed correctly built (see procedure descriptions below).

C3.1 CC_COMPILER_BUILD_AND_TEST_PROCEDURES

C3.1.1 BLDCC PROCEDURE DESCRIPTION

The SES procedure BLDCC builds the CYBIL CC front end and code generator binary libraries (PFELIB and PCGLIB7, respectively) and then links them together to generate the CYBILC compiler. At the same time, it recompiles the changed CYBCLIB modules, saves them on an object file (CYBCOBJ), and REPULIB's them onto the existing SES version of CYBCLIB to generate the updated CYBCLIB. Status messages are written to the file CYBSTS in the current catalog (BLDCC is the only procedure which purges this file if it exists; all other procedure append information to the end of the file).

BLDCC creates the following permanent files in the build catalog: PFELIB, PCGLIB7, CYBCOBJ, CYBCLIB, CYBILC, CMAP, CYBSTS. The format of the BLDCC is as follows:

```
SES.BLDCC    [ m = < (list of) module name(s) > ]
             [ fe ! cc ]
```

C3.0 CYBIL BUILD PROCEDURES
C3.1.1 BLDCC PROCEDURE DESCRIPTION

[chain]
[local]

m : The name(s) of the CC compiler modules to compile. The default is to compile all CC compiler modules.

fe ! cc : Keyword indicating whether the modules specified by "M" are front end (fe) or code generator (cc) modules. This keyword is required if "M" is specified.

chain : Option to submit subsequent CYBIL build jobs after BLDCC completes. The default is to not submit these jobs.

local : Run BLDCC in LOCAL mode. The default is to run it in BATCH.

C3.1.2 CNVRGCC PROCEDURE DESCRIPTION

The SES procedure CNVRGCC tests the CC compiler built by the BLDCC procedure for convergence. This means that the compiler must be able to compile itself, producing binaries identical to those which make it up. First it saves all the files created by BLDCC by copying them to files named by changing the first character of the file name to "A" (e.g. PFELIB -> AFELIB, CYBCLIB -> AYBCLIB, etc.). It then rebuilds the front end, code generator, and CYBCLIB binaries with the CYBILC compiler built by the BLDCC procedure, generates and tests a new compiler, and compares the new binaries to the previously built ("A"-prefixed) ones. If the binaries are not identical, CNVRGCC makes a second attempt at convergence (this time saving the binaries on "B"-prefixed files) and again compares the binary libraries. If the binaries verify, the job to build the CI compiler (BLDCI) is submitted; if they do not verify after 2 attempts at convergence, the procedure ends and the CYBIL project must be notified. Status messages are written to the indirect access file CYBSTS in the current catalog.

CNVRGCC creates the following permanent files in the build catalog: AFELIB, ACGLIB7, AYBCOBJ, ACYBILC, AYBCLIB, ACMAP (also BFELIB, BCGLIB7, BYBCOBJ, BCYBILC, BYBCLIB, BCMAP if convergence does not occur on the first iteration). The format of the CNVRGCC is as follows:

C3.0 CYBIL BUILD PROCEDURES
C3.1.2 CNVRGCC PROCEDURE DESCRIPTION

```
SES.CNVRGCC [ chain ]
            [ local ]
```

chain : Option to submit subsequent CYBIL build jobs after CNVRGCC completes. The default is to not submit these jobs.

local : Run CNVRGCC in LOCAL mode. The default is to run it in BATCH.

C3.1.3 TESTCC PROCEDURE DESCRIPTION

The SES procedure TESTCC runs the CYBIL compiler tests SEQUEN, EXITLP, and PPROC2 against the CYBILC compilers built by the BLDCC and CNVRGCC procedures. These tests reside on the test base TESTPL in the HAW catalog. Status messages are written to the indirect access file CYBSTS in the current catalog.

TESTCC creates no new permanent files. The format of the TESTCC is as follows:

```
SES.TESTCC  [ cnvg ]
            [ chain ]
            [ local ]
```

cnvg : Keyword indicating that the compiler being tested was built using the CNVRGCC procedure. This parameter is needed to make the status messages written to CYBSTS more meaningful. The default assumes that the CYBILC being tested was built by BLDCC - i.e. it has not gone through convergence yet.

chain : Option to submit subsequent CYBIL build jobs after TESTCC completes. The default is to not submit these jobs.

local : Run TESTCC in LOCAL mode. The default is to run it in BATCH.

C3.2 CI_COMPILER_BUILD_AND_TEST_PROCEDURES

C3.2.1 BLDCI PROCEDURE DESCRIPTION

The SES procedure BLDCI builds the code generator binary library (PCGLIB8) for the CYBIL CI compiler. It then links this file with the common front end (PFELIB) produced by the BLDCC procedure to generate the CYBILI compiler. Status messages are written to the indirect access file CYBSTS in the current catalog.

BLDCI creates the following permanent files in the build catalog: PCGLIB8, CYBILI, CMAP8. The format of the BLDCI is as follows:

```
SES.BLDCI   [ m = < (list of) module name(s) > ]
            [ chain ]
            [ local ]
```

m : The name(s) of the CI code generator module(s) to be compiled. The default is to compile all CI code generator modules.

chain : Option to submit subsequent CYBIL build jobs after BLDCI completes. The default is to not submit these jobs.

local : Run BLDCI in LOCAL mode. The default is to run it in BATCH.

C3.2.2 TESTCI PROCEDURE DESCRIPTION

The SES procedure TESTCI runs the CYBIL compiler tests SEQUEN, EXITLP, and PPROC2 against the CYBILI compiler and CYBILIB (built by BLDCI and BLDILIB, respectively). These tests reside on the test base TESTPL in the HAW catalog. Status messages are written to the indirect access file CYBSTS in the current catalog.

TESTCI creates no new permanent files. The format of the TESTCI is as follows:

```
SES.TESTCI  [ lib ]
            [ chain ]
            [ local ]
```

lib : Keyword indicating that this run of TESTCI tests the CYBILIB built by BLDILIB. This keyword is needed to make the status messages written to

C3.0 CYBIL BUILD PROCEDURES
C3.2.2 TESTCI PROCEDURE DESCRIPTION

CYBSTS more meaningful. The default assumes that this run of TESTCI tests the CYBILI compiler built by BLDCI.

chain : Option to submit subsequent CYBIL build jobs after TESTCI completes. The default is to not submit these jobs.

local : Run TESTCI in LOCAL mode. The default is to run it in BATCH.

C3.2.3 BLDILIB PROCEDURE DESCRIPTION

The SES procedure BLDILIB compiles the CYBILIB modules which have changed and saves them on the object file CYBIOBJ. It then GOL*s them on the current SES version of CYBILIB to generate the updated CYBILIB. BLDILIB also creates the CYBILGO file which is used in creating the version of CYBILIB which is used on the hardware (done in DS). Finally, it resubmits the CI compiler tests (TESTCI with "LIB" option specified) which now use the new CYBILIB as a means of testing CYBILIB. Status messages are written to the indirect access file CYBSTS in the current catalog.

BLDILIB creates the following permanent files in the build catalog: CYBIOBJ, CYBILIB, CYBILGO, LBSRC80. The format of the BLDILIB is as follows:

```
SES.BLDILIB [ chain ]
            [ local ]
```

chain : Option to submit subsequent CYBIL build jobs after BLDILIB completes. The default is to not submit more jobs.

local : Run BLDILIB in LOCAL mode. The default is to run it in BATCH.

C3.2.4 RUNREG PROCEDURE DESCRIPTION

The SES procedure RUNREG sets up the build catalog environment to be able to run the CI compiler regression tests. The tests reside on an SCU test base PL in the HAW catalog. RUNREG first creates the alternate SCU base "MYBASE" in the build catalog which allows for the regression tests to be run from that catalog. It then submits the deferred batch

C3.0 CYBIL BUILD PROCEDURES
C3.2.4 RUNREG PROCEDURE DESCRIPTION

job which runs the tests (CY001..CY999). To obtain the test results, run

```
SES,LPFN=CYBPLIB.RUNANL
```

the morning after the regression tests are run. In order to run this procedure, the deferred batch limit in the build catalog must be set to UNLIMITED, and the user must have the PROFILE variable "PASSWOR" defined.

RUNREG creates the following permanent files in the build catalog: MYBASE, CIDAY. The format of the RUNREG is as follows:

```
SES.RUNREG  [ opt2 ]
            [ local ]
```

opt2 : Option to test the compiler with the OPT=2 option. The default is to run the tests without the OPT=2 option.

local : Run RUNREG in LOCAL mode. The default is to run it in BATCH.

C3.3 II_COMPILERS_BUILD_AND_TEST_PROCEDURES

C3.3.1 BLDIIS PROCEDURE DESCRIPTION

The SES procedure BLDIIS builds the front end and code generator binary files (CYBIIFE and CYBIICG, respectively) for the II compiler for the simulator, and then links these two files together to produce the compiler checkpoint file (IICPF) used as input to the simulator. It then submits a simulator test run of this compiler. The output of this test run consists of a compilation listing (which should contain no compilation errors), the simulator SESLOG, and a dayfile (which should also show no compilation errors).

BLDIIS creates the following permanent files in the build catalog: CYBIIFE, CYBIICG, IICPF, IIMAP, SESLOG, LISTX. The format of the BLDIIS is as follows:

```
SES.BLDIIS  [ local ]
```

local : Run BLDIIS in LOCAL mode. The default is to run

C3.0 CYBIL BUILD PROCEDURES
C3.3.1 BLDIIS PROCEDURE DESCRIPTION

it in BATCH.

C3.3.2 BLDII2S PROCEDURE DESCRIPTION

The SES procedure BLDII2S builds the front end and code generator binary files for the II OPT=2 compiler for the simulator (CYBI2FE and CYBI2CG, respectively), and then links these two files together to produce the compiler checkpoint file (I2CPF) used as input to the simulator. It then submits a simulator test run of this compiler. The output of this test run consists of a compilation listing (which should contain no compilation errors), the simulator SESLOG, and a dayfile (which should also show no compilation errors).

BLDII2S creates the following permanent files in the build catalog: CYBI2FE, CYBI2CG, I2CPF, I2MAP, SESLOG, LISTX. The format of the BLDII2S is as follows:

```
SES.BLDII2S [ local ]

local :     Run BLDII2S in LOCAL mode. The default is to
            run it in BATCH.
```

C3.3.3 BLDIIH PROCEDURE DESCRIPTION

The SES procedure BLDIIH builds the front end and code generator binary files (CYBHIFE and CYBHICG, respectively) for the II compiler binary that is used on the hardware. At the same time, it generates the II hardware version of CYBILIB (CYBHOBJ). It then takes these three files and some other miscellaneous routines and GOF's them together to create the UNCONVERTED and UNBOUND version of CYBIL II (CYBHBIN).

BLDIIH creates the following permanent files in the build catalog: CYBHIFE, CYBHICG, CYBHOBJ, CYBHBIN. The format of the BLDIIH is as follows:

```
SES.BLDIIH  [ local ]

local :     Run BLDIIH in LOCAL mode. The default is to run
            it in BATCH.
```

D1.0 KEYPOINTS

D1.0 KEYPOINTS

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

D1.1 ISSUING_KEYPOINTS_FROM_CYBIL_CODE

The general form of the keypoint instruction is:

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

D1.1.1 KEYPOINT CLASSES

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

```
OSDKEYS
COMMON

  CONST

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

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

D1.0 KEYPOINTS
D1.1.1 KEYPOINT CLASSES

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

  CONST
    osk$m = 4096;
```

D1.1.2 NOS/VE KEYPOINT CLASSES

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

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

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

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

conjunction with a keypoint of class unusual.

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

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

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

```
OSDKEYC
COMMON
{Define KEYPOINT CLASS Codes.

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

{*callc,osdkeys
```

D1.1.3 KEYPOINT DATA AND IDENTIFICATION

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

D1.1.4 EXAMPLE ISSUING KEYPOINTS

ENTRY keypoint with data:

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

UNUSUAL keypoint with no data:

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

ENTRY keypoint with extra data:

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

D1.1.5 KEYPOINT IDENTIFIERS

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

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

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

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

D1.0 KEYPOINTS
D1.1.5 KEYPOINT IDENTIFIERS

```
        fmk$base = 550, {550 - 599}
        ick$base = 600, {600 - 649}
        ifk$base = 650, {650 - 699}
        iik$base = 700, {700 - 749}
        ink$base = 750, {750 - 799}
        jmk$base = 800, {800 - 849}
        lgk$base = 850, {850 - 899}
        llk$base = 900, {900 - 949}
        lok$base = 950, {950 - 999}
        luk$base = 1000, {1000 - 1049}
        mlk$base = 1050, {1050 - 1099}
        mmk$monitor_base = 1100, {1100 - 1149}
        mmk$job_base = 1150, {1150 - 1199}
        msk$base = 1200, {1200 - 1249}
        mtk$base = 1250, {1250 - 1299}
        ock$base = 1300, {1300 - 1349}
        ofk$base = 1350, {1350 - 1399}
        osk$base = 1400, {1400 - 1449}
        pfk$base = 1500, {1500 - 1549}
        pmk$base = 1600, {1600 - 1699}
        rhk$base = 1750, {1750 - 1799}
        srk$base = 1800, {1800 - 1819}
        stk$base = 1850, {1850 - 1899}
        tmk$monitor_base = 1900, {1900 - 1949}
        tmk$job_base = 1950, {1950 - 1999}
        jsk$monitor_base = 2000, {2000 - 2049}
        jsk$job_base = 2050, {2050 - 2099}
        avk$base = 2100, {2100 - 2149}
        sfk$base = 2150, {2150 - 2199}
        lok$base = 2200, {2200 - 2249}
        rmk$base = 2250, {2250 - 2300}
        mtk$assembly_language_base = 4000; {4000 - 4095}
    {   OS assembly language      4000 - 4095}
    {*callc,osdkeyc
```

D1.2 COLLECTING KEYPOINTS

D1.2.1 ON THE SIMULATOR

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

D1.2.2 ON THE HARDWARE

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

Three commands are supplied to utilize the keypoint feature: KEYON, KEYOFF, and KEYPOINT.

D1.2.2.1 KEYON command

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

```
KEYON,recording_mode, environment, kmm, kmj,
   keypoint_class_start, keypoint_class_stop,
   keypoint_file_name, keypoint_buffer_size,
   collector_timeout_period

recording_mode = 'software' or 'hardware', default is software

environment = 'job' or 'system'          , default is job

keypoint_mask_monitor = 0 .. 0ffff(16)   , default is 0ffff(16)

keypoint_mask_job = 0 .. 0ffff(16)       , default is 0ffff(16)

keypoint_class_start = 0 .. 15
        This specifies that keypoint collection should not
        start until a keypoint of this class in encountered,
```

default is to begin collecting immediately.

keypoint_class_stop = 0 .. 15
This specifies that keypoint collection should stop when a keypoint of this class is encountered.

keypoint_file_name = file name on which keypoints are saved, used with software keypoints only, default is KEYFILE.

keypoint_buffer_size = 0 .. halfword , default is 2000
For software keypoints only.

collector_timeout_period = 0 .. halfword , default is 50 milliseconds.
For use with software keypoints only.

D1.2.2.2 KEYOFF command

The KEYOFF command terminates keypoint collection.

KEYOFF= environment

D1.2.2.3 KEYPOINT command

This command is used to issue keypoints.

KEYPOINT = keypoint_class_number, keypoint_code

keypoint_class_number = 0 .. 15 , default is 15

keypoint_code = 0 .. halfword , default is 0

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

REPLACE_FILE,keyfile,keyfile,b56

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

D1.0 KEYPOINTS
D1.2.2.3 KEYPOINT command

D1.3 KEYPOINT ANALYZER UTILITY

D1.3.1 NVEKEY

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

```
SES.NVEKEY [KPF= ] [FORMAT= ] [KD= ] [AREA= ]
NVEKEY creates a simulator generated keypoint trace file.
The "kpf" parameter is the keypoint file used as input.
The "kd" parameter is a file or list of files which define(s)
the keypoint descriptions.
```

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

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

D1.3.2 KEYPOINT DESCRIPTION FILE

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

D1.3.2.1 keypoint_decks

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

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

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

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

SUB_ID_FIELD - optional - (described later)

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

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

DATA_FIELD_DESCRIPTOR - optional - This consists of data format and length.
data_format
- H Hex
- I Integer (decimal)
- A ASCII

Concatenated to this is the length of the data portion of the keypoint, in decimal bits.
For example: I20

D1.3.2.1.1 EXAMPLE KEYPOINT DECK

```
STDKEY
COMMON

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

  CONST
```

D1.0 KEYPOINTS
D1.3.2.1.1 EXAMPLE KEYPOINT DECK

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

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

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

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

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

D1.3.2.1.2 SUB_ID_FIELD

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

SUB_ID_LENGTH.SUB_ID_MATCH

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

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

Example:

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

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

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

D1.0 KEYPOINTS
D1.3.2.1.2 SUB_ID_FIELD

```
        mmk$page_fault);
```

D1.3.2.2 generating_the_descriptor_file

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

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

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

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

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

D1.3.2.3 osmkeys_format

This section will only be useful to those desiring to add additional keypoint classes, keypoint class base constants, or new keypoint description decks.

The classes, identifiers, and descriptions are each buffered by a comment. For example, to add another keypoint class:

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

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

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

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

D1.0 KEYPOINTS
D1.3.2.3 osmkeys format

```
OSMKEYS
?? LEFT := 1, RIGHT := 110 ??
MODULE keypoint_description_file;
{*calic,osdkeys
{$$$ START KEYPOINT CLASSES $$$}
{*calic,osdkeyc
{$$$ END KEYPOINT CLASSES $$$}
{$$$ START KEYPOINT IDENTIFIER BASES $$$}
{*calic,osdkeyd
{$$$ END KEYPOINT IDENTIFIER BASES $$$}
{$$$ START KEYPOINT DESCRIPTIONS $$$}
{*calic,amdkey
{*calic,badkey
{*calic,cidkey
{*calic,cmdkey
{*calic,dbdkey
{*calic,dmdkey
{*calic,fmdkey
{*calic,icdkey
{*calic,ifdkey
{*calic,iidkey
{*calic,indkey
{*calic,jmdkey
{*calic,igdkey
{*calic,lidkey
{*calic,iodkey
{*calic,iudkey
{*calic,midkey
{*calic,mmdmkey
{*calic,mmdjkey
{*calic,msdkey
{*calic,mtdkey
{*calic,ocdkey
{*calic,ofdkey
{*calic,osdkey
{*calic,pfdkey
{*calic,pmdkey
{*calic,rhdkey
{*calic,srdkey
{*calic,stdkey
{*calic,tmdmkey
{*calic,tmdjkey
{*calic,jsdmkey
{*calic,jsdjkey
{*calic,avdkey
{*calic,sfdkey
{*calic,iodkey
```

D1.0 KEYPOINTS
D1.3.2.3 osmkeys format

```
{*calic,rmdkey
{$$$ END KEYPOINT DESCRIPTIONS $$$}
MODEND keypoint_description_file;
```

D1.4 REFORMATTED_FILE_DESCRIPTION

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

Each line in the first section has the following format;

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

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

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

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

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

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

M - Monitor mode
J - Job mode

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

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

D1.0 KEYPOINTS
D1.4 REFORMATTED FILE DESCRIPTION

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

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

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

D1.5 BNF_KEYPOINT_DESCRIPTION

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

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

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

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

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

<base> ::= <integer>

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

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

<class_definitions> ::= <keypoint_class> <spc> = <spc>
```

D1.0 KEYPOINTS
D1.5 BNF KEYPOINT DESCRIPTION

```
                          <class_base_id>  <offset>
                          { <keypoint_class_id> <cybil comment>

   <keypoint_class> ::= <identifier>

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

   <delimiter> ::= , | ;

   <keypoint_class_id> ::= <character>

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

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

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

   <base_id> ::= <identifier>

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

   <low_base> ::= <integer>

   <high_base> ::= <integer>

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

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

   <keypoint_info> ::= <keypoint_constant_line> <delimiter> <eol>
                              [ <keypoint_descriptor> ... ]
                              [ <blank lines> ]

   <keypoint_constant_line>  ::= <keypoint_constant> <spc> = <spc>
                              <keypoint_base> <spc> [ <offset> ] <spc>
```

D1.0 KEYPOINTS
D1.5 BNF KEYPOINT DESCRIPTION

```
<keypoint_constant> ::= <identifier>

<keypoint_base> ::= <identifier>

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

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

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

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

<sub_id_length> ::= <field_length>

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

<sub_id_match> ::= <small_integer>

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

<keypoint_label> ::= <label>

<label> ::= ' <character_string> '

<character_string> ::= any visible characters except '

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

<data_label> ::= <label>

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

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

<data_field_length> ::= <field_length>

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

D1.0 KEYPOINTS
D1.5 BNF KEYPOINT DESCRIPTION

superceeded by a deck which redefines the
area to which it pertains)

Table of Contents