User Documentation

for the

Development Environment

of

NOS/VE

on

NOS/VE

Revision Date : 02/08/88

1.0 INTRODUCTION

1.0 INTRODUCTION

This document describes the use of the commands which maintain the environment established for design, development, evaluation and integration of the NOS/VE system, using NOS/VE as the basis for that environment.

Initially, of course, the user may need to convert NOS libraries, binaries, and files to 180 format. For a brief discussion of this, please see Appendix C. Migration will present different problems for different people. In an effort to make NOS/VE more usable, a utility has been provided as an "ear" for user complaints. Any bug (or feature) that a user encounters which is difficult to understand, makes life harder, or prevents one from accomplishing work should be "sent to integration".

The Integration Source Maintenance Commands enable access to the source code, object code, etc. in "logical" terms rather than in terms that require knowledge of the physical structure of the information. For example, if a developer wishes to compile a module at a certain build level of the system, he would need to specify the name assigned to that build level rather than the file(s) or catalog on which that level of the system reside. Creation, modification, compilation, formatting, and transmittal of decks are all handled by these procedures.

Some background is supplied first in the following sections, with command usage subsequent.

2.0 CATALOG STRUCTURE FOR SOURCE MAINTENANCE

2.0 CATALOG STRUCTURE FOR SOURCE MAINTENANCE

From the point of view of a developer of NOS/VE code the source exists in three levels.

- The main level is maintained by NOS/VE Integration and resides in the "development_base" (INTVE).
- In most cases a number of individual developers are working on the same feature, so there is a "feature catalog" shared by those developers which sits between their working catalog and the main integration source catalog.
- At the local level, each developer has their own "working catalog" which is a subset of the feature and/or system source.

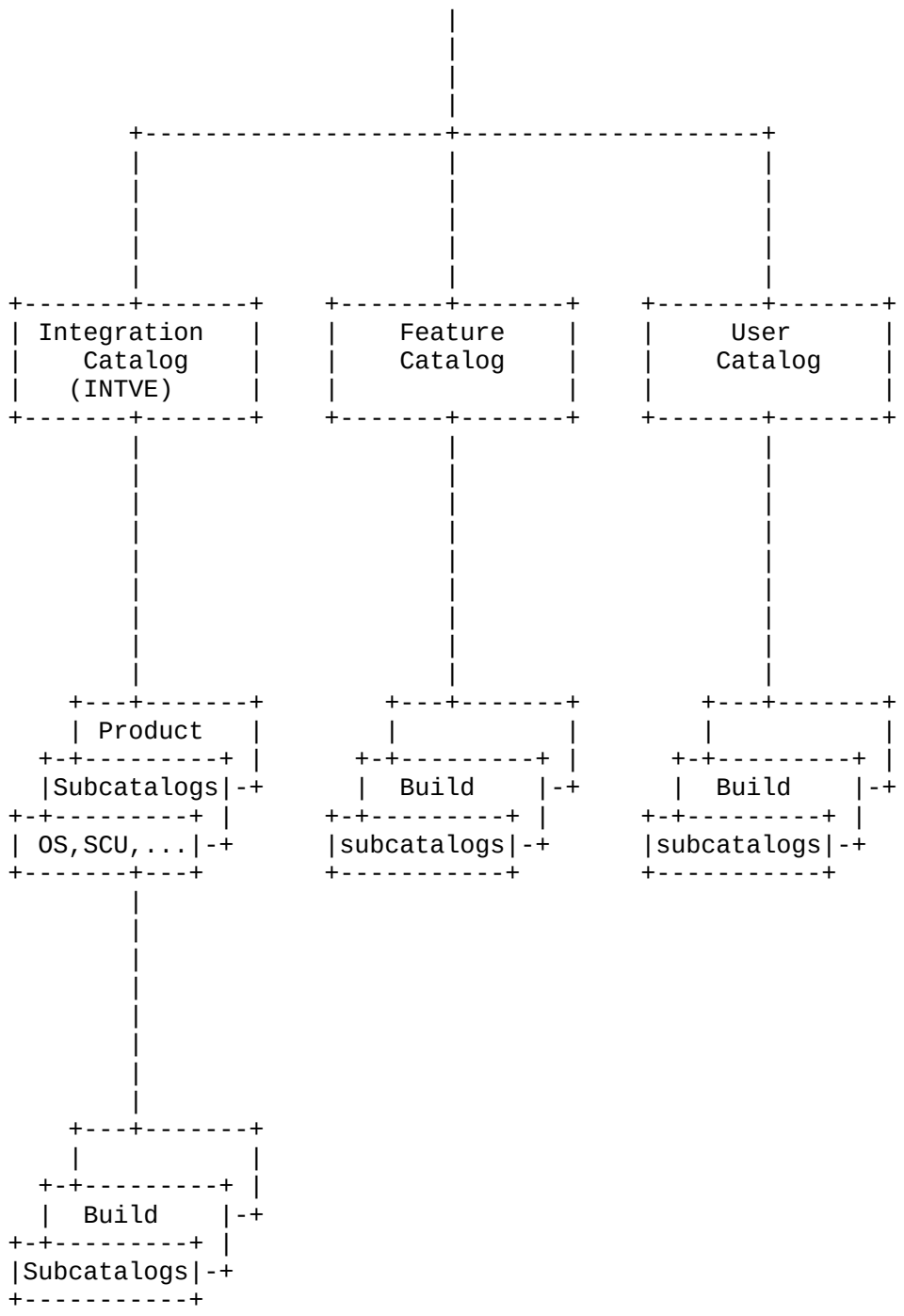
Thus each feature catalog contains a unique subset of the main data base and each working catalog contains a unique subset of the corresponding feature subset.

When a feature is being handled by a single developer, the "feature" level and "local" level may effectively be one and the same.

The information (source library, object libraries, etc.) in the feature and working subset catalogs will parallel the structure used in the main integration catalog; i.e. the same subcatalog hierarchy, file names, etc. will be used.

The relationship of these catalogs is illustrated below.

2.0 CATALOG STRUCTURE FOR SOURCE MAINTENANCE



2.0 CATALOG STRUCTURE FOR SOURCE MAINTENANCE
2.1 MAIN INTEGRATION CATALOG (INTVE)

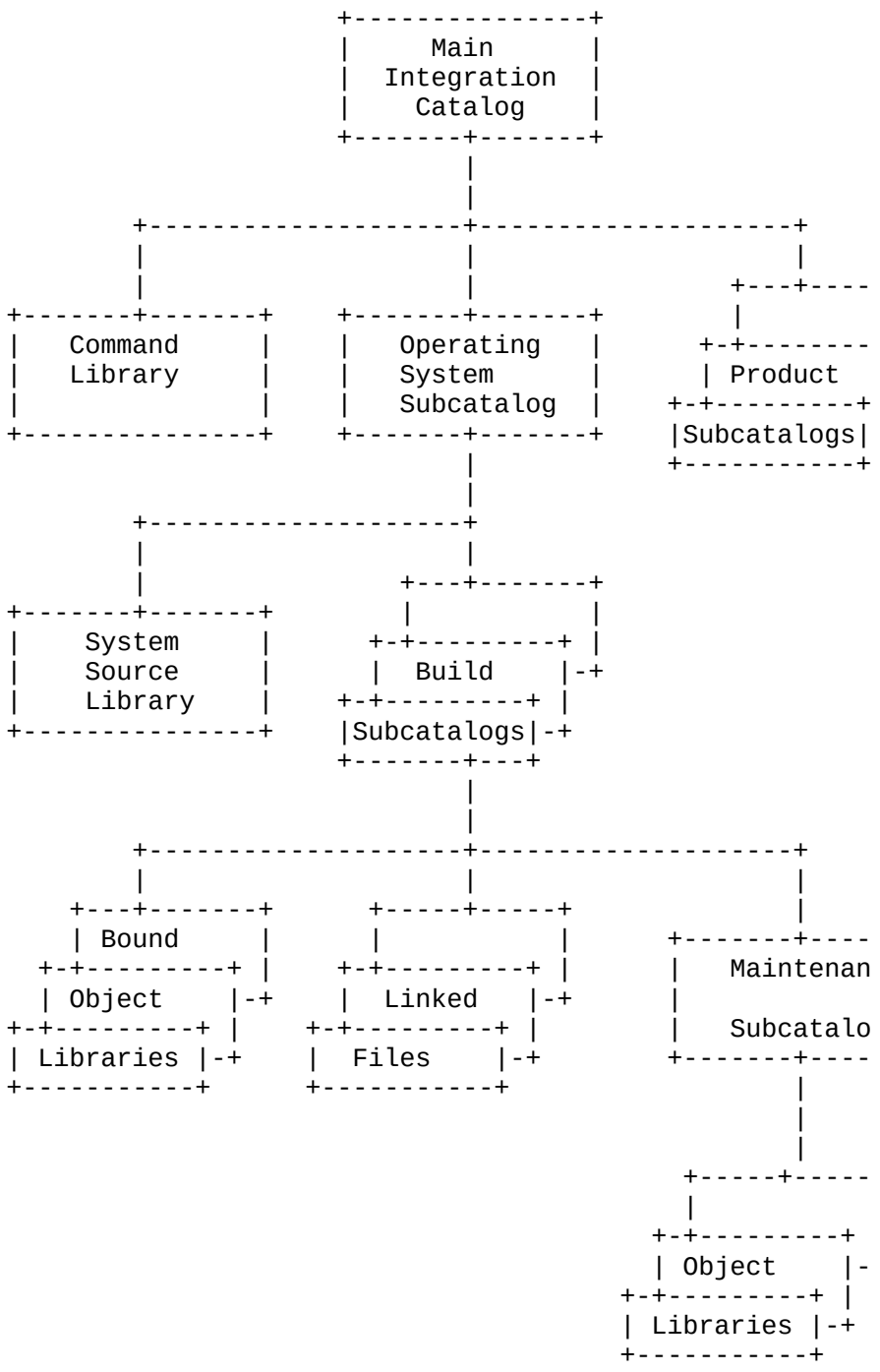
2.1 MAIN INTEGRATION CATALOG (INTVE)

This catalog holds the main level of the software development data base and all of the commands and tools needed to maintain and manipulate it. It contains the following :

- a library of commands used to access and manipulate the source (.INTVE.Command_Library)
- a subcatalog for the operating system (OS)
- a subcatalog for each product (SCU,ASSEMBLER,FTN,etc.) (Note that in this context the following are considered to be products :
  - "system tests"
  - test tools
  - system level documentation (e.g. AO/R, release bulletins, etc.)

The following diagram illustrates the overall structure of this catalog.

2.0 CATALOG STRUCTURE FOR SOURCE MAINTENANCE
2.1 MAIN INTEGRATION CATALOG (INTVE)



2.0 CATALOG STRUCTURE FOR SOURCE MAINTENANCE
2.1 MAIN INTEGRATION CATALOG (INTVE)

Remember that each product subcatalog contains a structure matching that of OS. OS is essentially one of several products. Each working catalog contains a structure matching that of the OS, also.

2.2 OPERATING SYSTEM SUBCATALOG (OS)

This subcatalog contains all the data necessary to build and test a NOS/VE system. In particular it contains the SCU source library for all versions of NOS/VE, and a subcatalog for each version (build level) of the system. These subcatalogs contain the object libraries, NOS libraries, and whatever else may be required for the corresponding level. See Appendix A on Representation of multiple build levels.

2.3 SYSTEM SOURCE LIBRARY

The SCU library for the operating system contains the following :

- the source code for the operating system and system supplied utilities for both virtual state and real state (OSS$SOURCE, OSS$REAL_STATE_SOURCE)
- the source code the system's external interfaces (OSS$PROGRAM_INTERFACE)
- the source code for NOS/VE feature tests
- miscellaneous "files" needed to build NOS/VE such as linker subcommands, selection criteria, etc.

The system source library will actually be kept in two files :

- SOURCE_LIBRARY contains the main library
- LATEST_CHANGES contains additions and changes to the main library that have not yet been merged into it. In other words it is the recipient of all code transmittals from development.

> Although there are physically two source libraries, they should be thought of as one "logical" library.

The reason for having two separate files is twofold. The primary reason is to remove the need for a developer to have to rewrite 100 plus Mbyte files (in the case of OS). The second reason ties into the algorithm used in parallel source library maintenance. See "User Documentation for NOS/VE Integration" for more details.

Integration will periodically merge the "latest changes" with the main library. This will happen approximately once per day. See "User Documentation for NOS/VE Integration" parallel source library maintenance chapter for more details.

2.3.1 GROUPS ON THE SOURCE LIBRARY

Critical to the structure of the Source Library, and to the efficiency of the source maintenance procedures, is the association of SCU "group names" with each deck on the library. These groups may be used to manipulate "blocks" or "groups" of decks, such as all decks in a job template or system core library, fairly easily. (See Appendix B for specific information on the naming conventions for these groups.)

Integration will set up the major groups on the Source Library (NOSVEPL et al) when the library is converted. After that, group attributes will generally be established at deck creation time.

2.3.2 USE OF "FEATURES" AND "MODIFICATIONS"

Every modification to the source generated by a developer will be associated with an SCU "feature". Features are used to identify what PSR, DAP, or increment the modifications associated with it are addressing. Modifications are used to identify code authored by an individual that is part of (or all of) a "feature". A modification may affect more than one deck.

Modification names will have the following form :

iii_nnnnn

where iii are the initials of the author of the modification and nnnnn is a one to five digit sequence number for modifications authored by iii. In the case where two developers have the same initials, the fourth character (normally an underscore) can be changed to resolve the conflict. The generation of these

modification names is handled and tracked automatically by the Integration procedures.

2.3.3 USE OF INTERLOCKS

SCU deck interlocks are used to avoid the problems caused by more than one person making a change to the same deck at the same time.

The commands described later use the following policies when moving information between the levels of the data base.

- Whenever a deck is extracted for the purpose of modifying it, the extraction is done using the SCU command EXTRACT_SOURCE_LIBRARY with interlocks enabled. This is true both when extracting from the main system library to a feature library, and from a feature library to a working library. This is accomplished via the Extract_Source procedure. When a deck is extracted from a library with interlocks it contains all modifications made to that deck up to that point.

- Whenever a modification is transmitted the affected decks must be recombined with the library from which they were extracted using the SCU command COMBINE_LIBRARY with interlock checking enabled. This is accomplished via the Transmit_To_Integration command.

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT

A library of commands (mostly SCL procedures) provides access to the source. These commands use standard facilities of NOS/VE and its product set. In addition tools provided by the Software Tools and Technology group will be utilized.

There is a command library for use by designers, developers, evaluators and integrators to access and maintain the source, and to access any "non-standard" tools.

These commands are maintained on the NOS/VE source library. Refer to Appendix G for copies of the actual procedure definitions.

To access these commands the user must add the command library to his/her command list. This is accomplished simply by entering :

```
Set_Command_List add=.INTVE.Command_Library
```

To facilitate this, the set_command_list could be added to the user's prolog. As a result, the commands appear to the user to be system commands.

Some of the commands described below will create a new cycle of a file. When this operation would take place on a temporary file that file will be overwritten (this is because NOS/VE does not support cycles for temporary files). When this operation takes place on a permanent file, cycle 1 of the file is first written (as a scratch file) and when this has successfully completed, the $NEXT cycle will be written and cycle 1 purged. In this way the integrity of the $HIGH cycle of the library should be maintained. To facilitate this, users are asked to keep cycle 1 of their Source_Library file empty. For such files in integration catalogs, the previously highest cycle will be deleted.

The deleting of old cycles of source libraries from feature and working catalogs will take place automatically via an archive during PF BACKUP. The expiration date of

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT

old cycles of files will be set by the procedures.

3.1 ENVIRONMENT COMMANDS

The commands described in this section are used to specify the "environment" in which work will be done.

Particular parts of the environment can be specified on most of the commands, but the commands are much more convenient to use if the environment parameters are established prior to work starting (via the Set_Working_Environment command).

Normally the SET_WORKING_ENVIRONMENT command is called from within the user's prolog thus establishing the environment at each login.

Other commands that care about an environment parameter will default that parameter to the value specified on the SET_WORKING_ENVIRONMENT command. If that command hasn't been called, the default used is that specified in the description of the corresponding parameter of the SET_WORKING_ENVIRONMENT command.

> All procedures (except those named) described in this document will work either within or outside of the working environment "utility". The exceptions to this are : EXTRACT_SOURCE, RE_EXTRACT_SOURCE, CHANGE_MODIFICATION_HEADER, CLEAR_INTERLOCK_REQUEST, TRANSMIT_TO_INTEGRATION, and TRANSIT_TO_FEATURE_CATALOG.

3.1.1 DISPLAY_WORKING_ENVIRONMENT (DISWE)

This command displays the values of the working environment parameters that have been established by Set_Working_Environment.

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT
3.1.1 DISPLAY_WORKING_ENVIRONMENT (DISWE)

```
display_working_environment
     display_options : list of name
     output : file
     status : status
```

display_option | display_options | do : specifies which working environment parameters are to be displayed. The legal values for this parameters are the names of the working environment parameters, and their abbreviations, as specified for the set_working_environment command, plus the keyword ALL meaning display all working environment parameters. Omission causes ALL to be used.

output | o : specifies the file to receive the display. Omission causes $OUTPUT to be used.

status : see NOS/VE error handling.

3.1.2 ENTER_WORKING_ENVIRONMENT (ENTWE)

This command should be used when a number of changes need to be made to the working source library. It calls SCU such that the base parameter is the source library in the working catalog and the result parameter is a new cycle of that source library.

This command provides a more efficient way of doing work than "popping in and out" of SCU to do editing, deck expansion, etc. For example, a user may enter the SCU utility, make some changes to a deck, compile it (without having to exit SCU and thus write a new cycle of the working library) , fix whatever problems may show up in the compile, compile again, etc. Only when finished with that particular work session does the user need to leave SCU.

When the working environment is entered, the highest cycle of source_library in the working environment is used as the base and cycle 1 of source library in the working environment is specified as the result file. Checkpoints of updates to the source library can be made via SCU's WRITE_LIBRARY subcommand with no parameters when in the working environment. This will overwrite cycle 1 of the

source library in the working environment working catalog. Cycle 1 of the source library is set to the highest cycle when exiting working environment via the EXIT_WORKING_ENVIRONMENT command.

The ENTER_WORKING_ENVIRONMENT command will support a prolog file called "WORKING_ENVIRONMENT_PROLOG" in the working catalog. This file will be invoked prior to a user's commands being accepted.

To inhibit writing the next cycle of source library so that no changes will be made, type in "END FALSE". Typing in "END" or "END TRUE" has the same effect as "END FALSE". In all of these cases, cycle 1 of source_library is purged so the changes are lost. The only way to save the changes is by typing "EXIWE" or "EXIT_WORKING_ENVIRONMENT".

Not all commands can be used from within the working environment. See section 3.1 for commands that work properly inside the working environment.

```
enter_working_environment
     author : string = $job(user)
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : file or key none = none
     feature_build_level : name = object
     working_catalog : file or key none = none
     working_build_level : name = object
     target_operating_system : key
     status : status
```

author | a : specifies the caller's name, to be recorded as the author of any decks or modifications created by this user.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

target_operating_system | tos : specifies which 170 system to generate a 180 system for. The only valid values are NOS and NOS/BE.

Omission causes NOS to be used.

status : see NOS/VE error handling.

3.1.3 EXIT_WORKING_ENVIRONMENT (EXIWE)

This command is used to exit from the working environment created by a preceding call to the ENTER_WORKING_ENVIRONMENT command.

If changes have been made since the working environment was "entered" the new cycle of the working source library is written. If no changes were made, the new cycle is discarded.

exit_working_environment
status : status

status : see NOS/VE error handling.

3.1.4 SET_WORKING_ENVIRONMENT (SETWE)

This command is used to create or change command language variables that specify the level of system or product to be used as a basis for work and the "feature" and "working" catalogs being used. The parameters specified in this command can also be specified on ALL of the other commands described in this document. The defaults for these parameters are the same in each command. However, once values for them have been established by the Set_Working_Environment command, they need not be specified again unless the user wishes to access a different catalog or build level.

Omission of a parameter will cause a default value to be picked if the corresponding variable is undefined, or if defined the corresponding value will be left unchanged.

All command language variables set by this command will have a scope of JOB and their names will begin with the characters WEV$. The variable names used are specified with the corresponding parameter.

This command is normally called from a user's prolog.

```
set_working_environment
     author : string = $job(user)
     development_base : file = .intve
     product_name : name = os
     build_level : name
     tools_library_build_level : name
     feature_catalog : file or key none = none
     feature_build_level : name = object
     working_catalog : file or key none = none
     working_build_level : name = object
     target_operating_system : key
     status : status
```

author | a : specifies the caller's name, to be recorded as the author of any decks or modifications created by this user.

The corresponding variable name is WEV$AUTHOR.

Omission causes the user name of the caller to be used.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the name of the product to be worked on. This is the name of the operating system or product subcatalog in the main integration catalog.

The corresponding variable name is WEV$PRODUCT_NAME.

The default is OS.

build_level | bl : specifies the name of the system or product build level to be used as the basis for work.

The corresponding variable name is WEV$BUILD_LEVEL.

The default is the latest product build level.

tools_library_build_level | tbl : specifies the build level of the tools command library to be added to the user's command list. If certain tools (compiler, linker, etc.) are unique for the build_level of a product, a tools command library will exist for the build_level of that product.

The corresponding variable name is WEV$TOOLS_LIBRARY_BUILD_LEVEL.

The default is the value of the BUILD_LEVEL parameter.

feature_catalog | fc : specifies the catalog in which the files and subcatalogs corresponding to a feature are located.

The corresponding variable name is WEV$FEATURE_CATALOG.

The default is to have no feature catalog (a value of 'NONE').

feature_build_level | fbl : specifies the name of the feature build level to be used. This is analogous to the system or product build level but designates the version of the feature to be used. The initial implementation of these commands will support only one "feature build level".

The corresponding variable name is WEV$FEATURE_BUILD_LEVEL.

The default is OBJECT.

working_catalog | wc : specifies the catalog in which the files and subcatalogs corresponding to a developers current work are located.

The corresponding variable name is WEV$WORKING_CATALOG.

The default is to have no working catalog (a value of 'NONE').

working_build_level | wbl : specifies the name of the working build level to be used. This is analogous to the feature build level but designates the "working" version to be used. The initial implementation of these commands will support only one "working build level".

The corresponding variable name is WEV$WORKING_BUILD_LEVEL.

The default is OBJECT.

target_operating_system | tos : specifies which 170 system to generate a 180 system for. The only valid values are NOS and NOS/BE.

Omission causes NOS to be used.

status : see NOS/VE error handling.

3.2 COMMANDS FOR CODE DEVELOPMENT AND INTEGRATION

The commands described in the following subsections are intended to be used by developers (and evaluators in their role as developers of tests) and integrators to access system oriented source. Parameters to these commands are the same as corresponding parameters in standard system utilities such as SCU and CYBIL. For example the Cybil list_options parameter can also be used in the call to the Compile_Source command.

3.2.1 ASSIGN_MODIFICATION(S) (ASSM)

This command can be used inside the Working_Environment.

This command creates one or more unique modifications in the working source library. Any time the user wishes to create or modify a deck, a modification name must be supplied to the appropriate command. This modification must have been created by Assign_Modification or must contain all information required by Assign_modification.

The names for the modifications are created from the "modification name seed" assigned to a user upon joining the project, and sequence number maintained in the user's prolog. This command will update the sequence number in the prolog, thus keeping track of modification names already used.

The user must initiate this tracking scheme by entering the following two create_variable commands in his/her prolog :

```
create_variable WEV$MODIFICATION_INDEX kind : integer ..
  scope= job value= 1 (or whatever number desired to start)

create_variable wev$modification_initials kind : string ..
  scope= job value= 'xyz_' (xyz is user's initials)
```

The feature name specified is added to WEF$FEATURE_LIST file in the user's working catalog. This file is a selection criteria file used by compile_source and get_source when expanding decks. There is no mechanism

for deleting features from WEF$FEATURE_LIST except by editing it.

The new version of the source library is written as the next cycle of the file.

```
assign_modification
     feature : name
     count : integer
     modification_description : string
     output : file
     author : string = $job(user)
     working_catalog : file or key none = none
     status : status
```

feature | f : specifies the feature with which the modifications are to be associated.

Required parameter.

count | c : specifies the number of modifications to be assigned.

Omission causes 1 to be used.

modification_description | md : specifies a brief description of the modifications.

Required parameter.

output | o : specifies the file to which are written the names of the modifications that have been created.

Omission causes $OUTPUT to be used.

author | a : specifies the creator of the contents of the modifications.

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

3.2.2 ASSIGN_AND_RETURN_MODIFICATION (ASSARM)

This procedure calls assign_modification and returns that modification in a stri to the modification parameter.

```
assign_and_return_modification
     feature, f : name = $required
     modification_description, ..
     md : string = $required
     output, o : file = $local.$output
     author, a : string = $job(user)
     working_catalog, wc : file = os
     modification, m : var of string = $optional
     status : var of status = $optional
```

feature | f : specifies the feature with which the modifications are to be associated.

Required parameter.

modification_description | md : specifies a brief description of the modifications.

Required parameter.

output | o : specifies the file to which are written the names of the modifications that have been created.

Omission causes $OUTPUT to be used.

author | a : specifies the creator of the contents of the modifications.

working_catalog | wc : specifies the working catalog to be used.

modification | m : This is the string which is returned with the new modification name.

status : See NOS/VE error handling.

3.2.3 BUILD_EXEC (BUIE)

The purpose of this procedure is to serve as a "watchdog" over the batch jobs produced by MAKE_BUILD_JOBS. This procedure will display each job's current status and allow the user to manipulate and resubmit the jobs.

```
build_exec
     products, p : list of name
     display_job_logs, djl : key all, errors, e, none
     edit_jobs, ej : key all, errors, e, none
     resubmit_jobs, rj : key all, errors, e, none
     veto, v : boolean
     output, o : file
     build_level, bl : name
     status : var of status
```

products | p : This a list of product names for which MAKBJ has been run. The procedure assumes that the catalog structure matches the MAKBJ default, with each product's information in $USER.<build level>.<product name>.BJC.

Default if omitted : the product setting from the working environment variable wev$product_name.

(Note : the keyword ALL will select all data on all products in the build job catalog.)

display_job_logs | djl : This parameter causes the log file, $USER.<build level>.<product name>.BJC.JOB_LOGS.<job> to be copied to <output>.

Default if omitted : none.

edit_jobs | ej : This parameter will allow the user to edit the job file, $USER.<build_level>.<product name>.BJC.<job>.

Default if omitted : none.

resumit_jobs | rj : This selects whether to resumit a job or not.

Default if omitted : no resubmit.

veto | v : This parameter causes a job-by-job interactive inquiry on each of the three operations selectable above.

Default if omitted : false.

output | This is the file that recieves all of the status information.

build_level | bl : specifies the system or product build level to be used.

status : see NOS/VE error handling.

3.2.4 CHANGE_MODIFICATION_HEADERS (CHAMH)

The purpose of this procedure is to change information in modification headers on the product source library of modifications at state 1. Feature, author and the modification_description are the only values that can be changed. The modification headers will not be updated immediately but at the next routine update of the product source library.

NOTE : This procedure can only be executed from the master machine.

```
change_modification_headers
     modification : list of name
     new_feature : name
     new_author : string = $job(user)
     new_description : string
     development_base : file = .intve
     product_name : name = os
     status : status
```

modification | m : specifies the modifications to be changed.

new_feature | nf : specifies the feature to be assigned to the specified modifications.

Omission results in no change.

new_author | na : specifies author to be assigned to the

specified modifications.

Omission results in no change.

new_description | nd : specifies modification_description to be assigned to the specified modifications.

Omission results in no change.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

status : see NOS/VE error handling.

3.2.5 CLEAR_INTERLOCK_REQUEST (CLEIR)

The purpose of this procedure is to clear the interlock on decks the caller has interlocked. If the interlocked deck is on latest_changes, it can be cleared immediately. If not, it is cleared at the next regular update of the product source library. If need arises to have someone else's interlock cleared, contact integration.

NOTE : If the deck does not exist on latest_changes, this request must be run on the master machine.

```
clear_interlock_request
     deck : list of name
     development_base : file = .intve
     product_name : name = os
     status : status
```

deck | d : specifies the deck whose interlock is to be cleared.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to use.

status : see NOS/VE error handling.

3.2.6 COMPILE_SOURCE (COMS)

This command can be used inside the Working_Environment.

This command compiles/assembles one or more decks.

The defaults have been chosen for convenience of developers who will typically be dealing with a small number of modules at a time.

Libraries are included in the list to be searched in the following order :

- Working library (if a working catalog is in effect)
- Feature library (if a feature catalog is in effect)
- Alternate bases (if alternate_bases are specified)
- Latest changes (if include_latest_changes is specified)
- OS or product source library
- Alternate product libraries (if alternate_products are specified)

At least one of the deck and selection_criteria parameters must be provided. If both are given the deck parameter is processed first. If either processor or object_library (or both) is not specified, defaults are taken from the first deck in the compile list.

Decks to be compiled are specified only by the DECK parameter and by the INCLUDE_DECK and INCLUDE_GROUP directives of the user selection criteria file (specified by the SELECTION_CRITERIA parameter). Modifications to be applied against the selected decks are selected by the BUILD_LEVEL and FEATURE parameters and by directives in the user selection criteria file. The INCLUDE_LATEST_CHANGES parameter indicates whether or not the LATEST_CHANGES file in the product catalog is to be included as an alternate base.

The selection of decks and modifications is passed on to the SCU EXPAND_DECK command (by this PROC) by parameter values on the command call and by the selection criteria file which this PROC always builds - whether or not the SELECTION_CRITERIA parameter is specified on the PROC call. As a result of the call to SCU by this PROC, the following operations are performed :

1. The various source libraries specified are merged. That is, certain directories such as the deck directories and the modification directories from the various libraries are merged. A deck which appears on more than one library is selected from the first library on which it is located. However, modification headers which may appear on more than one library are selected from the last library encountered.

2. All modifications resulting from the merge are flagged as "included"; all decks, "excluded".

3. All modifications in state 0 or 1 are excluded. This step is performed because there may be mods in these states which may be of no interest to the caller (they may actually be "bad"). This does mean that the current mods you are generating and testing will be excluded since they are in state 0, however, your mods will be added later either by the feature or selection criteria parameter or by your wef$feature_list file.

4. The directories are "rolled back" to the specified build level - that is, all new mods included in builds after the specified build are excluded. The directories should now be in the same configuration they were in when the build of the specified level was generated. This step is skipped if the value of the BUILD_LEVEL parameter is NONE.

5. Modifications associated with the features specified by the FEATURE parameter are now "included". If the FEATURE parameter is not specified, the features selected will be those in the WEF$FEATURE_LIST file of the working catalog. If this file does not exist or is empty and no value is specified for the FEATURE parameter, then no additional modifications will be included by this step. If the FEATURE parameter is given the keyword value of ALL, then

step 3 is essentially undone - that is, all modifications with state 0 or 1 are included. If the FEATURE parameter is given the keyword value of NONE, this step will result in no changes - that is, no modification introduced after the specified build level will be included.

6. Decks specified by the DECK parameter are now included. Note that these are the only decks which have been selected for compilation.

7. The SCU selection criteria file specified by the SELECTION_CRITERIA parameter is now processed. The directives in this file may - intentially or inadvertantly - undo some of what has been done before.

8. Unless both the PROCESSOR and OBJECT_LIBRARY parameters are specified on the call to COMPILE_SOURCE, the PROC will include some directives here to determine those values for the deck(s) under consideration.

Specifying both the PROCESSOR and OBJECT_LIBRARY parameters causes all decks to be expanded and then compiled together. This is quicker than having the procedure determine processor and object library for each deck if there are a lot of them.

When the procedure is run a sub_catalog called compilation_catalog is created in the working_catalog. As the modules are processed the object_text (lgo file) is written to a file in this catalog. The file name is the same as the destination object_library which will be written to the maintenance sub_catalog. Upon successful generation of the object_library, the object_text file is deleted. If there are no other files in the compilation_catalog sub_catalog, it is deleted also.

When the procedure is run to compile something on the 170 side, any existing working catalog copies of 170 libraries are replaced to the 170 (if CLEANUP_170= BOTH or START) along with the text to compile. A partner job is executed on the 170 to compile the text. The NOS partner job generates the following : a compilation listing file, a status file and a dayfile of the 170 job. When compilation completes, the object text generated is replaced onto the specified object library. The three

special files mentioned above are then retrieved to the 180 and massaged as follows :

1. The 170 job dayfile is placed on $LOCAL.DAYFILE_170,
2. If SAVE_LISTINGS<>NONE the listing file has its page headers altered to match the NOS/VE SIS.
3. The status file is interpreted to report the result of the compilation.

When all compilations have completed, the 170 object libraries are retrieved (if CLEANUP_170= BOTH or END) back to your working catalog.

```
compile_source
     deck : list of name
     selection_criteria : file
     feature : list of name
     compile : file
     processor : string
     object_library : list of name
     list : file
     list_options : list of name
     listing_library : name
     save_listings : list of save_options
     alternate_product : list of list of name
     alternate_base : list of file
     target_operating_system : key
     cleanup_170 : key START END BOTH NONE = BOTH
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : file or key none = none
     feature_build_level : name = object
     working_catalog : file or key none = none
     working_build_level : name = object
     status : status
```

deck | decks | d : specifies the decks to be compiled.

selection_criteria | sc : specifies the file containing SCU selection criteria commands that alone or in conjunction with the decks parameter select the decks to be compiled.

Omission causes $NULL to be used.

NOTE : The selection criteria file cannot be an interactive file because there is no way to end it and allow SCU to continue processing selection criteria file directives which the procedure has to append. The selection criteria file should not contain an "END" command.

feature | f : specifies what features are to be included when compiling specified decks. The keywords ALL or NONE are also allowable values.

Omission causes file WEF$FEATURE_LIST in the user's working catalog to be used if it exists. This file is maintained by the ASSIGN_MODIFICATION procedure.

compile | c : specifies the name of the compile file to be generated. It is not disturbed when procedure completes.

Omission causes a unique name to be used and the file is detached when procedure completes.

processor | p : specifies the compiler or assembler to be used to process each deck. Processor options may be selected by giving this parameter as a string (e.g. 'CYBIL DA=NONE OPT=HIGH RC=NONE' - which is the value used for closed shop and release system compiles).

NOTE : If specifying the processor field and the processor is CYBIL, it is absolutely necessary to also specify DA=NONE (no debug tables). Modules with debug tables do not work when linked into the system.

Omission causes each deck to be processed according to its "processor" attribute. Obviously the procedure will be more efficient if a processor is specified, as it will not have to process each deck individually.

object_library | object_libraries | ol : specifies the object libraries in the working catalog into which the compiled/assembled modules will be placed. If an object library already exists the newly compiled modules are combined with it to form a new version of the library which effectively overwrites the

original. If NONE is specified, no object_library or object_file will be written.

Omission causes the object libraries to be selected according to each deck's "destination group" attributes.

list | l : specifies the file to which the compilation/assembly listings are to be written.

Omission causes $LIST to be used. (Warning - in a batch job $LIST is connected to OUTPUT. If a listing is not desired in a batch job, $NULL should be specified for list.)

list_option | list_options | lo : specifies the listing options to be used on each call to a compiler/assembler. See the documentation for each "processor" for valid options.

Omission causes (S,R) to be used, i.e. a listing of the source and a cross reference (excluding unreferenced identifiers). This default is not currently valid for the 180 Assembler. If the user wants the processor default to take effect, DEFAULT should be specified in the call to the procedure. If the processor is CP or PP COMPASS, no list options are specified on processor call if the defaults (s,r) are specified.

listing_library | ll : specifies the name of the listing_library.

save_listings | sl : specifies whether listings generated by compiler are to be saved on an SCU library in the working environment. Options are ALL, GOOD, BAD and NONE. GOOD and BAD refer to the absence or presence of compilation errors.

Omission causes NONE to be used.

alternate_product | alternate_products | ap : specifies other product catalogs in which to find source_libraries to be used as alternate_bases in the expansion of the code. This parameter accepts either single names or pairs or names as values. If an entry is a single value, it specifies an alternate

product name. If a pair of values is specified, the second element specifies the build level for the alternate product. If the second value is NONE or only a single value specified, a build level of NONE, which is all active lines at state 2 or higher is assumed.

Omission causes no alternate_products to be accessed.

alternate_base | alternate_bases | ab : specifies other files that may be required to properly expand the desired decks.

Omission causes NONE to be used.

cleanup_170 | c7 : This parameter is only used when compiling 170 language decks (COMPASS, SYMPL, CYBIL CC, CCL PROCS, etc.). It is used to control the replacement and retrieval of the 170 object libraries from the 180 to allow performance improvements. It is not recommended that this parameter be used unless the user is familiar with the implications it has for the procedures LINK_170 and GENDF as well as COMPILE_SOURCE. The valid values are :

START : replace files, do NOT retrieve or purge them.
END : do NOT replace files, retrieve and purge when done.
BOTH : replace files, retrieve and purge when done.
NONE : do NOT replace, retrieve, or purge files.

Omission causes BOTH to be used.

target_operating_system | tos : specifies which 170 system to generate a 180 system for. The only valid values are NOS and NOS/BE.

Omission causes NOS to be used.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be

used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.2.7 CREATE_SOURCE (CRES)

This command creates the new decks in the working source library. Once this is accomplished the text of the deck may be added via EDIT_LIBRARY or EDIT_SOURCE, or (indirectly) by any other editor.

The new version of an affected source library is written as the next cycle of the corresponding file.

```
create_source
     deck : list of name
     modification : name
     source_group : name
     destination_group : list of name
     group : list of name
     processor : string
     author : string = $job(user)
     deck_description : string
     expand : boolean
     same_as : name
     development_base : file = .intve
     product_name : name = os
     feature_catalog : file or key none = none
     working_catalog : file or key none = none
```

status : status

deck | decks | d : specifies the decks to be created.

modification | m : specifies the modification under which the decks are to be introduced. This modification must exist in the working library at state 0.

source_group | sg : specifies the group of decks to which the new decks are to be assigned, such as OSS$SOURCE. (See the section on "use of groups" above.)
Required parameter.

destination_group | destination_groups | dg : specifies the file(s) in which the "processed" (compiled, assembled, etc.) form of the modules is to be placed. (See Appendix B) If this is not applicable (in the case of a common deck) NONE should be specified.
Required parameter.

group | groups | g : specifies the "arbitrary" groups to which the decks are to be assigned. (See Appendix B)

Omission causes no "arbitrary" groups to be assigned.

processor | p : specifies the processor for the decks. The processor is also added as a group unless it is the keyword NONE.

Omission causes CYBIL to be used.

author | a : specifies the creator of the contents of the decks.

deck_description | dd : specifies a brief description of the decks. If no description is desired, specify the string NONE.
Required parameter.

expand | e : specifies whether the decks are to be "directly expandable" i.e. written directly to a compile file (expand= true) or expandable only by being the object of a SCU *copy or *copyc text embedded directive (expand= false). (i.e. a common

deck)

Omission causes a default to be chosen according to the syntax of the deck name (each deck will have its own default). If the fourth character of the deck name is "$" and the third character is not "M", the default is FALSE; otherwise the default is TRUE.

same_as | sa : specifies that SCU deck attributes not specified with parameters above are to be taken from the named deck.

Omission causes the normal SCU defaults to be used.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

3.2.8 CREATE_TESTNAMES_LISTS (CREATE_TESTNAMES_LIST

CREATE_TESTNAME_LISTS CREATE_TESTNAME_LIST CRETL)
This procedure creates the data files needed to drive the automated testing utility. These files contain the names of the tests to be executed; broken down into various subsets and categories. New testname lists must be generated whenever tests are added or removed from the system. Canned directives exist on a deck called TDI$TESTNAMES_product_name or can be supplied by the TESTNAMES_DATA_FILE.

create_testnames_lists
testname_data_file, tdf : file or key none

```
        alternate_bases, alternate_base, ab : list of file or
        key none
        development_base, db : file
        product_name, pn : name
        feature_catalog, fc : file or key none
        working_catalog, wc : file or key none
        build_level : name
        working_build_level, wbl : name
        mystery_maintainer_of_future, mysmf : boolean
        help, h : file
        status : var of status

    testnames_data_file | tdf : This is a  file  of  alternate
        directives for determining test subsets.

    alternate_bases  | ab : This is a list of source libraries
        to  be  searched  in   addition   to   the   standard
        (integration, f_c, w_c) set

    development_base | db : specifies the catalog in which all
        products and build levels reside.
        Default if omitted : .INTVE

    product_name  | pn : specifies the system or product to be
        used.

    build_level | bl : specifies the system or  product  build
        level to be used.

    feature_catalog | fc : specifies the feature catalog to be
        used (if any).

    working_catalog | wc : specifies the working catalog to be
        used.

    working_build_level  |  wbl  : specifies the working build
        level to be used.

    status : see NOS/VE error handling.
```

3.2.9 DISPLAY_BUILD_INFORMATION (DISBI)

This procedure displays the build information of specified build or cycle level. The information displayed is about various components of a NOS/VE build, such as NOS

deadstart tape, product tape, test tape and the build levels of various products. The information concerning how many products are displayed controlled by the information_level parameter. The default is the products that are built in NOS/VE development.

Note : The product level represents all of the Sunnyvale products.

```
display_build_information
     build_level : list of name
     cycle_level : list, range of integer
     information_level : key
     list : file
     development_base : file = .intve
     status : status
```

build_level : bl : specifies the build level for which build information is displayed.

Omission causes the build level to be used from the OS source library.

cycle_level : cl : specifies the cycle level for which build information displayed. Cycle level is the XX part of build number build_1XX07 for instance. All build levels for that cycle are displayed.

Omission causes no cycle level to be selected.

information_level : il : specifies amount of information display for each build level, valid entries are full and brief. Brief, will display the level of nos deadstart tape, product tape, test tape, ocu, cybil and scu while full will display all of the brief information plus the level of many more products.

Omission causes BRIEF to be used.

list | l : specifies the file to receive the information displayed.

Default if omitted : $OUTPUT

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

status : see NOS/VE error handling.

3.2.10 DISPLAY_FEATURE_BUILD_LEVEL (DISFBL)

The purpose of this procedure is to show a user the build level at which a feature was built into the system. This is accomplished by expanding the build decks and searching for the feature in question.

```
display_feature_build_level
     feature, f : name
     minimum_build_level, mbl : name
     output, o : file = $reponse
     product_name, pn : name
     status : var of status
```

feature | f : This is the name of the feature to search for.

minimum_build_level | mbl : This is the oldest level of the system that the feature COULD HAVE gone into.

Default if omitted : BUILD_00000.

output | o : This is the file to write the results onto.

product_name | pn : specifies the system or product to be used.

status : see NOS/VE error handling.

3.2.11 DISPLAY_SYMBOL_TABLE (DISST)

This command invokes a utility that displays CYBIL symbol tables. This utility requires an object library or object file as input. The object file must be compiled with debug symbol tables. This is the default on the CYBIL command. To use compile_source, specify P= 'CYBIL' OL : file_name.

During program execution, a local file name, PD, is

generated. This file name contains the directives to generate a graphic display of the types.

To generate graphical display :

1) REPF PD
2) LOGOUT
On 170 side :
3) SES.FORMAT I= PD LOCAL
4) SES.PRINT LISTING

```
display_symbol_table
     input : file
     output : file
     object_library : file
     module : name
     status : status
```

input | i : specifies the file where the utility reads its input from.

Omission causes $input to be used.

output | o : specifies the file where the utility writes its output.

Omission causes $output to be used.

object_library | ol : specifies the object library or file where the symbol tables are that are to be displayed. If it is an object library and no module name is specified, it defaults to the first module on the object library.

Required parameter.

module | m : selects what module on object library that the symbols will be displayed for.

Omission causes the first module on an object library to be used.

status : see NOS/VE error handling.

3.2.12 EDIT_SOURCE (EDIS)

This command can be used inside the Working_Environment.

This command is used to modify a deck that resides on the working source library.

The result of the editing will appear on a new cycle of the working source library.

```
edit_source modification : name
     deck : name
     output : file
     working_catalog : file or key none = none
     status : status
```

modification | m : specifies the modification to be used for editing the decks. The modification must exist in state 0 on the working library.

deck | d : specifies the deck to be edited.
Omission will necessitate selection of the deck (deck) to be edited from within the editor via the Select_Deck command.

output | o : specifies the file to receive displays from the editor.
Omission causes $OUTPUT to be used.

working_catlaog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

3.2.13 EXTRACT_SOURCE (EXTS)

This command can NOT be used inside the Working_Environment.

This command is used to extract decks from the feature or system/product level to the working level in

preparation for making changes. In addition to the decks themselves, the relevant "parts" of all associated modifications, features and groups are extracted (i.e. SCU's EXTRACT_SOURCE_LIBRARY facility is used).

If a deck does not exist at the feature level it is first extracted from the system/product level to the feature level.

This extraction is performed using deck interlocks.

The new version of an affected source library is written as the next cycle of the corresponding file.

At least one of the deck and selection_criteria parameters must be provided. If both are given the deck parameter is processed first. Currently a deck name MUST be specified.

When extracting a deck without interlock, the deck is extracted at specified build level. If build level is specified as none, the deck is extracted with all active lines, as if it was extracted with interlocks. Decks extracted without interlock can not be transmitted back to integration or a feature catalog.

```
extract_source deck : list of name
     interlock : boolean
     selection_criteria : file
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : file or key none = none
     working_catalog : file or key none = none
     status : status
```

deck | decks | d : specifies the decks to be extracted.

interlock | i : specifies whether decks are to be extracted with or without interlocks.
Omission causes TRUE to be used.

selection_criteria | sc : specifies the file containing SCU selection criteria commands that alone or in conjunction with the decks parameter select the decks to be extracted.

Omission causes $NULL to be used.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

3.2.14 FORMAT_SOURCE (FORS)

This command can be used inside the Working_Environment.

This command formats a CYBIL or SCL_PROCEDURE source deck.

If an error is detected, the replacement is suppressed.

```
format_cybil_source deck : name
     modification : name
     working_catalog : file or key none = none
     status : status
```

deck | d : specifies the deck to be formatted.

modification | m : specifies the modification under which the update is to be made. The modification must exist in state 0 on the working source library.

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

3.2.15 GENERATE_PRODUCT_SET_TAPE (GENPST)

This procedure is used to generate product set tapes for dedicated machine time. The following is performed to create the tape :

1. Make a copy of the installation table from the working, feature, os_build_level or installation_catalog.

2. IF product = 'ALL' THEN
   Change the product name to 'GENPST' for all products defined by the variable raf$nos_ve_products.
   IFEND

3. Change all the required products product name to 'GENPST'.

4. Change the path descriptor for all products specified by the product_name_and_path parameter. Also change the product name to 'GENPST'.

5. Delete all entries in the installation table that do not have have the product name 'GENPST'. And combine all product with an object_library format from the build_level, feature, and working catalogs, with the exception of bound_products.

6. Assemble release materials for the product 'GENPST'.

7. Generate the product tape.

8. Verify the tape with the restore_permanent_files command. Dependent upon the value of the verify_tape parameter.

9. Delete the installation catalog dependent upon the value of the save_installation_table parameter.

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT
3.2.15 GENERATE_PRODUCT_SET_TAPE (GENPST)

```
generate_product_set_tape
     os_build_level, obl : name
     product_name_and_path, pnap : list 1..$max_value_sets
     1..2 of any
     product_tape_type, ptt : key required, all = all
     verify_tape, vt : boolean = true
     installation_catalog,      ic      :      file      =
     $user.installation_catalog
     save_installation_catalog, sic : boolean = false
     external_vsn, evsn : list of string 1..6 = $required
     type, t : key mt9$800 mt9$1600 mt9$6250 = mt9$1600
     delete_modules, dm : file = $local.$null
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog, fc : file or key none = none
     feature_build_level, fbl : name = object
     working_catalog, wc : file or key none = none
     working_build_level, wbl : name = object
     status
```

os_build_level | obl : specifies the os build level.

product_name_and_path | pnap : specifies additional products or changed path names for included products. Both the product and path_name must be specified.

product_tape_type | ptt : specifies the type of product set tape to be generated (required or all). Required is a subset of all.

verify_tape | vt : specifies rather to verify the tape or not.

installation_catalog | ic : specifies the location of the installation
catalog that is being built.

save_installation_catalog | sic : specifies rather to save the installation table or not.

external_vsn | evsn : specifies the external vsn on the product tape.

type | t : specifies the product tape's density and format.

delete_modules | dm : specifies file containing one or more lines each of which is a module deletion directive. See delete_modules parameter in Link_Operating_System command for directive format.

Omission causes no modules to be deleted.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the product catalog to be used. See working environment conventions for details.

Default if omitted : w-e defaults.

build_level | bl : specifies the version of the product to be used.

Default if omitted : w-e defaults.

development_base | db : specifies the catalog in which all products and build levels reside.

feature_catalog | fc : specifies the feature catalog to be used.

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.2.16 GET_SOURCE (GETS)

This command can be used inside the Working_Environment.

This command writes one or more decks to a legible file. The decks may be written in expanded (EXPAND_DECK) or unexpanded (EXTRACT_DECK) form.

The decks are taken from the Source Library without interlocks, thus allowing the user to include (or exclude) any modifications or features desired. This command allows the user to look at source that may already be interlocked by someone else, for informational purposes only, or for editing by a non-SCU editor.

At least one of the deck and selection_criteria parameters must be provided. If both are given the deck parameter is processed first.

NOTE : See the compile source write up for how decks are selected and how the selection criteria files are processed.

CAUTION : Because get_source can be used to get a source module from a library without all active lines, caution must be exercised when using get_source and replace_source to get a module from a library, modify it and replace it back on the library. The replace_source procedure uses the generate_scu_edit_commands command to produce the changes necessary to the module on the source library to make it look like the modified version. If the module modified was not pulled off with all active lines, this would have the effect of deleting all changes made after the build level used and all code not currently in a build level that is not part of features or modifications specifically included (code at state 0 or 1). When doing something like this, BL= NONE and FEATURE= ALL must be specified on the get_source command to retrieve the module from the library with all active lines.

```
get_source
     deck : list of name
     source : file
     selection_criteria : file
     feature : list of name
     expand : boolean or key forced_expand, fe = false
     expansion_depth : integer
     alternate_product : list of list of name
     alternate_base : list of file
     development_base : file = .intve
```

```
target_operating_system : key
line_identifier,  li:  key  right, r, left, l, none =
none
development_base, db: file = .intve
product_name : name = os
build_level : name
feature_catalog : file or key none = none
feature_build_level : name = object
working_catalog : file or key none = none
working_build_level : name = object
status : status
```

deck | decks | d : specifies the decks to be written.

source | s : specifies the file to receive the decks.

Omission causes source to be used.

selection_criteria | sc : specifies the file containing SCU selection criteria commands that alone or in conjunction with the decks parameter select the decks to be written.IST

Omission causes $NULL to be used.

See the compile source write up for a more detailed explanation of this parameter.

feature | f : specifies what features are to be included when compiling specified decks. The keywords ALL or NONE are also allowable values.
Omission causes file WEF$FEATURE_LIST in user's working catalog to be used if it exists. This file is maintained by the ASSIGN_MODIFICATION procedure.

See the compile source write up for a more detailed explanation of this parameter.

expand | e : specifies whether the decks are to be expanded. The FORCED_EXPAND attribute will cause normally non-expandable decks to be expanded. The FORCED_EXPAND selection may not be used while within the working environment (ENTWE).
Omission causes FALSE to be used.

expansion_depth | ed : specifies the number of levels of

COPY and COPYC directives to process.
Omission causes $max_integer to be used.

alternate_product | alternate_products | ap : specifies other product catalogs in which to find source_libraries to be used as alternate_bases in the expansion of the code. This parameter accepts either single names or pairs or names as values. If an entry is a single value, it specifies an alternate product name. If a pair of values is specified, the second element specifies the build level for the alternate product. If the second value is NONE or only a single value specified, a build level of NONE, which is all active lines at state 2 or higher is assumed.

Omission causes no alternate_products to be accessed.

alternate_base | alternate_bases | ab : specifies other files that may be required to properly expand the desired decks.

Omission causes NONE to be used.

target_operating_system | tos : specifies which 170 system to generate a 180 system for. The only valid values are NOS and NOS/BE.
Omission causes NOS to be used.

line_identifier | li : This is the scu line_identifier option. It can cause modset/line-number information to be produced for each line in a deck and on either the left or right side of a page.

Omission causes no identifiers to be externalized.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.2.17 MAKE_BUILD_JOBS (MAKBJ)

This procedure generates a set of batch jobs to recompile a selected set of decks. The procedure will sort the selected list of decks by destination object library and by processor type and generate one batch job for each destination library. Within the batch job, there will be one section for each processor type. Decks can be selected by any valid scu selection criteria file directives. These directives are input on the selection_criteria parameter. It is recommended that any common decks have a 'include_copying_decks' entry so that the full scope of effect is realized. Likewise, it is recommended that the 'include_modified_decks' entry have the include_copying_decks parameter set to true. All jobs will be created in a catalog called the build_job_catalog. This catalog will also contain the job_logs after the jobs finish executing. This catalog will also contain a special file called the status_file (after release 121). This file contains a set of entries which will list a job's current status. This file can be read by the build_exec procedure or manually. The batch jobs will be created to run in the current usernumber. They will have all of the working_environment's environment variables set to the values supplied to make_build_jobs itself.

make_build_jobs
selection_criteria | sc : file = $required
list | l : file = $null
cybil_list_options | clo : list of key a, f, o, r,

ra, s, x, none = none
save_listings, save_listing | sl : key all, none, good, fatal, warning = none
build_job_catalog | bjc : file = $optional
submit_jobs | submit_job | sj : key immediate, i, delay, d, no_submit, ns = no_submit
micro_fiche | mf: boolean = false
job_class | jc: any of key none, keyend, name, anyend = none
compiler_options | co: list of list 1..2 of string = $optional
os_build_level | obl : name = $optional
development_base | db : file = .intve
product_name | pn : name = os
build_level | bl : name = $optional
feature_catalog | fc : file or key none = none
feature_build_level | fbl : name = object
working_catalog | wc : file or key none = none
working_build_level | wbl : name = object
password | pw : name = $required
status : var of status = $optional

selection_criteria | sc : This parameter is used to select which decks to compile.

list | l : This parameter is passed to compile_source to use as it's list parameter.

cybil_list_options | clo : This parameter specifies the values to use as list options when compiling cybil decks.

save_listings | save_listing | sl : This value is passed to the compile_source parameter.

build_job_catalog | bjc : This parameter specifies a scratch catalog that the procedure is to use to sort all of the decks selected. WARNING - IT IS CRITICAL THAT THIS PARAMETER BE A SCRATCH CATALOG. THE CATALOG SPECIFIED ON THIS PARAMETER MAY NOT HAVE ANY ITEMS WHICH THE USER WISHES TO KEEP AS THE PROCEDURE WILL DELETE THE ENTIRE CONTENTS OF THIS CATALOG.

submit_jobs | submit_job | sj : This parameter controls automatic job submission. jobs can either be submitted automatically or manually. The value of 'delay' causes all jobs to be created with the job_class of maintenance. This allows these jobs to

be submitted but not initiated until that job_class is opened for use by the operators (usually at night when the jobs won't compete with interactive users for machine resources).

micro_fiche | mf : This selects the special operations required to create jobs that will produce microfiche-able listings.

job_class | jc : This allows the user to select any job class desired. The default is the users default job class.

compiler_options | co : This allows a user to select the options (such as 'DA=ALL', 'OPT=LOW', etc.) for a compiler. This parameter has two string components. The first is the compiler - such as 'CYBIL', 'FORTRAN'; etc. The second is the parameter options to be passed to COMPILE_SOURCE when this compiler is to be used. The default if omitted is integration's standard parameters.

os_build_level | obl : This specifies the level to use when the product name is set to something other than os. (Non-os products automatically have an alternate_base of os.)

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted: .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build

level to be used.

password | pw : The password for the current user number. (All jobs are created to run under the current user number.)

3.2.18 MAKE_BUILD_REQUEST (MAKBR)

This command is used to request that a feature(s) be considered for a subsequent build. This command generates a build request form to be signed by your manager, after it is signed it should be put in the DCT input basket.

When specifying more than one feature per build request each feature is processed individually, creating a separate build request form for each. Also note that with multiple feature lists, parameters : modifications, dependencies and associated_srb_feature cannot be used. If these parameters are needed list features individually, making several requests.

When making a build request, all code that the request affects must have already been transmitted to integration. After that has been done the MAKE_BUILD_REQUEST procedure is run. It will produce a form detailing all elements which have been selected, print one copy and retain a copy in the user's catalog in the subcatalog BUILD_REQUEST_CATALOG under the feature name as a file. These files are saved in case the build request form is lost, delete them after you get the form. The procedure will also put information into the provisional build content files in the product sub-catalog of INTVE. The DCT determines what build the code will go into.

When a transmittal requires a change to the external system specifications, you will have to follow the following steps :

1. Create a deck of text which contains the SRB item. This deck is created just like any new code deck.
2. Transmit the SRB change to product_name = SYSTEM_DOCUMENTATION.
3. Transmit the code to the appropriate product catalog.

4. Run MAKE_BUILD_REQUEST for your SRB transmittal.
5. Run MAKE_BUILD_REQUEST for your code transmittal and specify the feature name used for the SRB transmittal as the ASSOCIATED_SRB_FEATURE. The SRB transmittal should use the same feature name.

This general sequence should be followed whenever a code transmittal requires a documentation change, as well.

```
make_build_request
     features : list of name
     modifications : list of name
     rebuild_message_templates : boolean
     rebuild_keypoint_descriptors : boolean
     dependencies : list of name
     associated_srb_feature : name
     comments_file : file
     development_base : file = .intve
     product_name : name = os
     author : string = $job(user)
     status : status
```

features | feature | f : This parameter specifies the feature(s) which are ready to put in a build. All modifications that are members of the selected feature(s) are selected.

Required parameter.

modifications | modification | m : This parameter is used to specify a selected subset of the modifications from a feature which are ready to put in a build. This parameter would be used when doing a subsequent request for the same feature name. The build requests for a feature are cumulative.

Omission causes all modifications to be selected.

NOTE : When specifying more than one feature, this parameter cannot be used.

rebuild_message_templates | rmt : This parameter sends a note to integration stating that message templates must be rebuilt when this code is integrated.
Omission causes FALSE to be used.

rebuild_keypoint_descriptors | rkd : This parameter sends

a note to integration stating that the keypoint descriptor file must be rebuilt when this code is integrated.

Omission causes FALSE to be used.

dependencies | dependency | d : This parameter sends a note to integration to make sure that the specified build requests have already been put into this or a previous build.

Omission indicates no dependencies.

NOTE : This parameter can not be specified when more than one feature is specified.

associated_srb_feature | asf : This parameter specifies the feature name of an associated SRB item.

Omission causes no SRB feature name to be listed.

NOTE : When specifying more than one feature, this parameter cannot be used.

comments_file | cf : This parameter specifies a file containing additional comments to be sent to integration.

Note : You should not specify an interactive file since you cannot supply it with an EOF when you are done.

Omission is no comments.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : This parameter specifies the product name for which the build request is made.

author | a : This parameter specifies the originator of the request.

status : see NOS/VE error handling.

3.2.19 QUERY_DEBUG_TABLE (QUEDT)

This command allows the user to interactively interrogate the debug file produced by link_operating_system. The user types in a PVA, procedure or variable name. The procedure responds with the procedure name and offset or the PVA respectively.

```
query_debug_table
     debug_table, dt   :   file   or   key   system,   s,
     running_system, rs, boot, b = system
     input : file
     output : file
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : file or key none = none
     feature_build_level : name = object
     working_catalog : file or key none = none
     working_build_level : name = object
     status : status
```

debug_table | dt : specifies the debug table (output by link_operating_system) to be used for date. The keywords SYSTEM, S, RUNNING_SYSTEM, RS, BOOT and B, are also allowable values. If SYSTEM or S is specified, the procedure finds the proper debug table in the working catalog, feature catalog or build catalog. If RUNNING_SYSTEM or RS is specified, the debug table of the running system is used. If the BOOT or B options are specified, the boot_debug_table, from the appropriate place, will be used.

Omission causes SYSTEM to be used.

input | i : specifies the file from which input commands are read, i.e., a PVA or procedure or variable name.

Omission causes standard input to be used.

output | o : specifies the file to which output is written.

Omission causes output to go to standard output.

development_base | db : specifies the catalog in which all

products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the product catalog to be used. See working environment conventions for details.

Default if omitted : w-e defaults.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.2.20 RE_EXTRACT_SOURCE (REES)

This command cannot be used inside the working environment.

This command extracts the selected modifications and decks from the caller's working source library, extracts the decks with or without interlock from the product source library and reapplies the selected modifications on the caller's working source library. If there are any problems reapplying the modifications, that modification has to be reapplied manually. Modifications being reapplied are saved in the re_extract subcatalog in the working environment. When satisfied that the modifications are reapplied properly, the re_extract subcatalog should be deleted.

The decks selected for reextraction can be specified

directly with the deck parameter or implicitly with the feature and modification parameters. If the deck parameter is not specified, the decks reextracted are the ones affected by the selected modifications. Specifying decks on the deck parameter selects only those decks.

The modifications selected for reextracting are explicitly selected by the feature and modification parameters and implicitly selected by the deck parameter. If neither feature nor modification parameter is specified, all modifications at state zero in specified decks are reextracted. Specifying only the feature parameter selects only modifications that belong to the specified feature. Specifying only the modification parameter selects only those modifications. If both the feature and modification parameter are specified, only the specified modifications that belong to the specified features are selected. In all cases, only modifications at state zero are selected.

If reextracting without interlocks, the build level parameter affects what source lines are extracted in the same way as in extract_source without interlocks.

NOTE : At least one of the feature, deck, or modification parameters must be specified.

```
re_extract_source
     features : list of name
     decks : list of name
     modifications : list of name
     interlock : boolean
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : name
     working_catalog : name
     status : status
```

features | feature | f : specifies what features are to be reextracted.

Omission causes the selected decks and modifications to be determined by the deck and/or modification parameters.

decks | deck : specifies what decks are to be reextracted.

Omission causes the selected decks to be determined by the feature and/or modification parameters.

modifications | modification | m : specifies what modifications are to be reextrcted.

Omission causes the selected modifications to be determined by the feature and/or deck parameters.

interlock | i : specifies whether the deck is to be interlocked when reextracting.

Omission causes the decks to be reextracted with interlocks.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

3.2.21 REPLACE_SOURCE (REPS)

This command can be used inside the Working_Environment.

This command is used to update a deck on the working source library. The new version of a deck on a file is compared with the old version on the working library using SCU's generate_scu_edit_commands facility. The SCU editor is then invoked to apply the resulting edit commands.

This command, in conjunction with the get_source command, provides the mechanism for making changes to a deck that are difficult to make by direct use of the SCU editor. The best example of this is automated formatting of source code.

The new version of the working source library is written as the next cycle of the file.

CAUTION : Because get_source can be used to get a source module from a library without all active lines, caution must be exercised when using get_source and replace_source to get a module from a library, modify it and replace it back on the library. The replace_source procedure uses the generate_scu_edit_commands command to produce the changes necessary to the module on the source library to make it look like the modified version. If the module modified was not pulled off with all active lines, this would have the effect of deleting all changes made after the build level used and all code not currently in a build level that is not part of features or modifications specifically included (code at state 0 or 1). When doing something like this, BL= NONE and FEATURE= ALL must be specified on the get_source command to retrieve the module from the library with all active lines.

```
replace_source
     source : file
     deck : name
     modification : name
     working_catalog : file or key none = none
     status : status
```

source | s : specifies the file which contains the new version of the deck.

deck | d : specifies the deck to be updated.

modification | m : specifies the modification under which the update is to be made. The modification must exist in state 0 on the working source library.

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

3.2.22 TRANSMIT_TO_FEATURE_CATALOG (TRATFC)

This command can NOT be used inside the Working_Environment.

This command transmits code (decks, modifications, features) from a working catalog to a feature catalog.

Once the transmitted code has been written to its destination, it is removed from its source. The new version of an affected source library is written as the next cycle of the corresponding file.

At least one of the deck, modification, or feature parameters must be provided. If more than one of them is given they are processed in the order just listed.

Unlike when directly using SCU, e.g. in a selection criteria file, selecting a modification or feature via this commands parameters will result in selection of all decks affected by that modification or feature.

```
transmit_to_feature_catalog
     decks : list of name
     modification : list of name
     feature : list of name
     feature_catalog : file or key none = none
     working_catalog : file or key none = none
     status : status
```

deck | decks | d : specifies the decks to be transmitted.

modification | modifications | m : specifies the modifications to be transmitted.

feature | features | f : specifies the features to be transmitted.

feature_catalog | fc : specifies the feature catalog to be used.

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

3.2.23 TRANSMIT_TO_INTEGRATION (TRATI)

This command can NOT be used inside the Working_Environment.

This command transmits code (decks, modifications, features) from a working catalog to a feature catalog (if specified) and then integration. (If no feature catalog is used, it will transmit directly to integration.)

Once the transmitted code has been written to its destination, it is removed from its source. The new version of an affected source library is written as the next cycle of the corresponding file.

The deck, modification or feature parameter must be provided. Unlike when directly using SCU, e.g., in a selection criteria file, selecting a modification or feature via this command's parameters will result in selection of all decks affected by that modification or feature. When the appropriate decks are subsequently deleted from the working source library ALL mods contained in those decks are also deleted. Thus, if there are modifications you are still working on, the modification should be extracted and saved and deleted from the source library before transmitting the deck.

```
transmit_to_integration
     decks : list of name
     modification : list of name
     feature : list of name
     development_base : file = .intve
     product_name : name = os
     feature_catalog : file or key none = none
     working_catalog : file or key none = none
     status : status
```

deck | decks | d : specifies the decks to be transmitted.

modification | modifications | m : specifies the modifications to be transmitted.

feature | features | f : specifies the features to be transmitted.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

working_catalog | wc : specifies the working catalog to be used. This parameter is ignored if a feature catalog is used.

status : see NOS/VE error handling.

3.3 PRODUCT SPECIFIC COMMANDS

All of the procedures in this section are specialized. They only work on one particular product. Most of these are linking/binding procedures. Also included procedures used to generate one flavor of deadstart tape (GENDF and BUICT). Each procedure in this set expects that when called; the user's product name is set to the product for which the procedure is intended and that build level values reflect those for the appropriate product.

3.3.1 BIND_AV (BINA)

The purpose of this procedure is to bind the AV product.

bind_av
os_build_level, obl : name

cybil_level, cl : name
map, m : file
delete_modules, dm : file
build_level, bl : name
feature_catalog, fc : file or key none
feature_build_level, fbl : name
working_catalog, wc : file or key none
working_build_level, wbl : name
status : var of status

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.
Default if omitted : selected from the tying file of the selected os level.

map | m : This is the location to place the linkmap.
Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.2 BIND_DE (BIND)

This procedure will bind the Desktop using only information obtained from the ty The object libraries are merged using Merol so developers can use this to genera

```
bind_de
     os_build_level, obl : name
     cybil_level, cl : name = $optional
     development_base, db : file = .intve
     delete_modules, dm : file
     build_level, bl : name = $optional
     feature_catalog, fc : file or key none = none
     feature_build_level, fbl : name = object
     working_catalog, wc : file or key none = none
     working_build_level, wbl : name = object
     status : var of status = $optional
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.
Default if omitted : selected from the tying file of the selected os level.

development_base | db : specifies the catalog in which all products and build levels reside.
Default if omitted: .INTVE

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

3.3.3 BIND_HPA (BINH)

The purpose of this procedure is to bind the HPA product.

```
bind_hpa
     os_build_level, obl : name
     cybil_level, cl : name
     common_level, cml : name
     bound_product, bp : file
     map, m : file
     delete_modules, dm : file
     build_level, bl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     status : var of status
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.

Default if omitted : selected from the tying file of the selected os level.

common_level | cml : This is the level of the sunnyvale product COMMON's MLF$LIBRARY to be used.

Default if omitted : selected from the os build

level's tying file entry.

bound_product | bp : This is the location to place the link results.
Default if omitted : <wc>.BOUND_PRODUCT

map | m : This is the location to place the linkmap.

Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.4 BIND_KERMIT (BINK)

This procedure will bind the kermit product and use the tying file to determine where to obtain the required files.

bind_kermit
os_build_level, obl : name

```
     cybil_level, cl : name = $optional
     bound_kermit, bk : file = $optional
     map, m : file = $optional
     delete_modules, dm : file
     build_level, bl : name = $optional
     feature_catalog, fc : file or key none = none
     feature_build_level, fbl : name = object
     working_catalog, wc : file or key none = none
     working_build_level, wbl : name = object
     status : var of status = $optional
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.
Default if omitted : selected from the tying file of the selected os level.

bound_kermit | bk : The file to put the newly created kermit_bound_product onto.

map | m : This is the location to place the linkmap.
Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

3.3.5 BIND_MAILVE

The purpose of this procedure is to bind the MAILVE product.

```
bind_mailve
     os_build_level, obl : name
     tdu_level, tl : name
     cybil_level, cl : name
     bound_product, bp : file
     map, m : file
     delete_modules, dm : file
     build_level, bl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     status : var of status
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

tdu_level | tl : This is the version of the screen manager code to use.

Default if omitted : selected from the tying file of the selected os level.

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.

Default if omitted : selected from the tying file of the selected os level.

bound_product | bp : This is the location to place the

link results.

Default if omitted : <wc>.BOUND_PRODUCT

map | m : This is the location to place the linkmap.

Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.6 BIND_MALET (BINM)

The purpose of this procedure is to bind the MALET product.

bind_malet
os_build_level, obl : name
cybil_level, cl : name
bound_product, bp : file
map, m : file

delete_modules, dm : file
build_level, bl : name
feature_catalog, fc : file or key none
feature_build_level, fbl : name
working_catalog, wc : file or key none
working_build_level, wbl : name
status : var of status

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.

Default if omitted : selected from the tying file of the selected os level.

bound_product | bp : This is the location to place the link results.

Default if omitted : <wc>.BOUND_PRODUCT

map | m : This is the location to place the linkmap.

Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.7 BIND_MENU_VE (BINMV)

The purpose of this procedure is to bind the MENU_VE product.

```
bind_menu_ve
     os_build_level, obl : name
     cobol_level, cl : name
     bound_library, bdl : file
     map, m : file
     delete_modules, dm : file
     build_level, bl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     status : var of status
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cobol_level | cl : This allows the user to select the cobol level to match with the new product.
Default if omitted : cobol level from the tying file.

bound_library | bdl : This is the location to place the link results.

Default if omitted : <wc>.BOUND_PRODUCT

map | m : This is the location to place the linkmap.

Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.8 BIND_NAMVE (BINN)

The purpose of this procedure is to bind the NAMVE product. The NAM/VE product consists of the object library osf$network_management in an OS build catalog. The source is maintained on the OS source library.

```
bind_namve
     cybil_level, cl : name
     map, m : file
     delete_modules, dm : file
     build_level, bl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
```

status : var of status

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.

Default if omitted : selected from the tying file of the selected build level.

map | m : This is the catalog to place the linkmaps.

Default if omitted : <wc>.MAINTENANCE.NETWORK_MAINTENANCE

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.9 BIND_OCU (BINO)

The purpose of this procedure is to bind the OCU product. The OCU product consists of the file OCF$OBJECT_CODE_UTILITY in an OS build level catalog. The

source is maintained on the OS source library

```
bind_ocu
     cybil_level, cl : name
     use_system_ocu, uso : boolean
     bound_product, bp : file
     map, m : file
     delete_modules, dm : file
     build_level, bl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     status : var of status
```

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.

Default if omitted : selected from the tying file of the selected build level.

use_system_ocu | uso : This parameter specifies whether the link will be done using the standard ocu from the system or the the unbound ocu that is being linked.

Default of omitted : the unbound ocu being linked.

bound_product | bp : This is the location to place the link results.

Default if omitted : <wc>.OCU_BOUND_PRODUCT

map | m : This is the location to place the linkmap.

Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.10 BIND_PERFORMANCE_TOOLS (BINPT)

The purpose of this procedure is to bind the PERFORMANCE_TOOLS product. The PERFORMANCE_TOOLS product contains benchmark tests; including IBL, CBL and SBL.

```
bind_performance_tools
     os_build_level, obl : name
     cybil_level, cl : name
     bound_product_ring_3, bpr3file
     bound_product_ring_d, bprdfile
     map, m : file
     delete_modules, dm : file
     build_level, bl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     status : var of status
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.

Default if omitted : selected from the tying file of the selected os level.

bound_product_ring_3 | bpr3 : This is the location to place part of the link results.

bound_product_ring_d | bprd : This is the location to place part of the link results.

map | m : This is the location to place the linkmap.

Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.11 BIND_PFTF (BINP)

This is the front end for the pftf binding procedure supplied by the pftf group. This procedure will merge object_libraries found in the working environment with the libraries in the pftf build catalogs and create the bound_products with all the changes found.

```
bind_pftf
     os_level, ol : name = $optional
     os_build_level, obl : name
```

```
cybil_level, cl : name = $optional
common_level, cml : name = $optional
bound_pftf, bp : file = $optional
bound_pftf_trace, bpt : file = $optional
map, m : file = $optional
delete_modules, dm : file
build_level, bl : name = $optional
feature_catalog, fc : file or key none = none
feature_build_level, fbl : name = object
working_catalog, wc : file or key none = none
working_build_level, wbl : name = object
status : var of status = $optional
```

os_level | ol : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.
Default if omitted : selected from the tying file of the selected os level.

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be

used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

3.3.12 BIND_SCU (BINS)

This command is used to merge and bind the pieces of the SOURCE_CODE_UTILITY. This command can be used inside of the working environment.

```
bind_scu
     os_level : name
     cybil_level : name
     common_level : name
     workbench_level : name
     bound_product : file
     bound_editor : file
     map : file
     editor_map : file
     delete_modules, dm : file
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : file or key none = none
     feature_build_level : name = object
     working_catalog : file or key none = none
     working_build_level : name = object
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : specifies the level of cybil to be used to satisfy external references.

Default if omitted : derived from the TYING file contained in the os build level selected.

common_level | cml : specifies the level of common to be

used to satisfy external references.

Default if omitted : derived from the TYING file contained in the os build level selected.

workbench_level | wl : specifies the level of workbench to be used to satisfy external references.

Default if omitted : derived from the TYING file contained in the os build level selected.

bound_product | bp : specifies the location to place the results of the bind.

Default if omitted : <working_catalog>.<working_build_level>. BOUND_PRODUCT

bound_editor | be : specifies the location to place the results of the bind of the standalone editor.

Default if omitted : <working_catalog>.<working_build_level>. MAINTENANCE.OSF$COMMAND_LIBRARY

map | m : specifies the location to place the bind map.

Default if omitted : <working_catalog>.<working_build_level>. MAINTENANCE.MAP

editor_map | em : specifies the location to place the editor's bind map.

Default if omitted : <working_catalog>.<working_build_level>. MAINTENANCE.EDITOR_MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be

used.

Default if omitted : SCU

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

3.3.13 BIND_SVS (BINS)

The purpose of this procedure is to bind the SVS product.

```
bind_svs
     os_build_level, obl : name
     bound_cptime, bc : file
     bound_kernel_checker, bkc : file
     map, m : file
     delete_modules, dm : file
     build_level, bl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     status : var of status
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

bound_cptime | bc : This is the location to place part of the link results.

bound_kernel_checker | bkc : This is the location to place part of the link results.

map | m : This is the location to place the linkmap.

Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.14 BIND_TDU (BINT)

The purpose of this procedure is to bind the TDU product.

```
bind_tdu
     os_build_level, obl : name
     cybil_level, cl : name
     bound_product, bp : file
     map, m : file
     delete_modules, dm : file
     build_level, bl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
```

working_build_level, wbl : name
status : var of status

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.

Default if omitted : selected from the tying file of the selected os level.

bound_product | bp : This is the location to place the link results.

Default if omitted : <wc>.BOUND_PRODUCT

map | m : This is the location to place the linkmap.

Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.15 BIND_TDU_POST_PROCESSOR (BINTPP)

This procedure will bind the tdu post_processor and use the tying file to determine where to obtain the required files.

```
bind_tdu_post_processor
     os_build_level, obl : name
     cybil_level, cl : name = $optional
     bound_product, bp : file = $optional
     map, m : file = $optional
     delete_modules, dm : file
     build_level, bl : name = $optional
     feature_catalog, fc : file or key none = none
     feature_build_level, fbl : name = object
     working_catalog, wc : file or key none = none
     working_build_level, wbl : name = object
     status : var of status = $optional
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level. (which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.
Default if omitted : selected from the tying file of the selected os level.

map | m : This is the location to place the linkmap.
Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

3.3.16 BIND_TEST_TOOLS (BINTT)

The purpose of this procedure is to bind the TEST_TOOLS product.

```
bind_test_tools
     os_build_level, obl : name
     cybil_level, cyl : name 1..11
     common_level, cl : name 1..11
     bound_product, bp : file
     map, m : file
     delete_modules, dm : file
     build_level, bl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     status : var of status
```

os_build_level | obl : This allows the user to select the os system level to match with the new product.

Default if omitted : os level is equal to the build level.

(which is the first os level that the product is introduced at).

cybil_level | cl : This is the version of the CYF$RUN_TIME_LIBRARY to use.

Default if omitted : the value specified in the tying file of the selected os level.

common_level | cl : This is the level of the sunnyvale product COMMON's MLF$LIBRARY to be used.

Default if omitted : selected from the os build level's tying file entry.

bound_product | bp : This is the location to place the link results.

Default if omitted : <wc>.BOUND_PRODUCT

map | m : This is the location to place the linkmap.

Default if omitted : <wc>.MAINTENANCE.MAP

delete_modules | dm : This is a list of object modules to delete from the INTEGRATION level of the unbound libraries. It does NOT delete modules from feature or working catalog level libraries.
Default if omitted : nothing is deleted.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.17 BUILD_CIP_TAPE (BUICT)

The purpose of this procedure is to build a CYBER INTIAILIZATION PACKAGE (CIP) tape with the records produced by NOS/VE replaced with the versions from the specified build level.

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT
3.3.17 BUILD_CIP_TAPE (BUICT)

```
build_cip_tape
     volume_serial_number, vsn : name 1..6
     machine_class, mc : key of s0_51, s0_52, s1, s2,  s3,
     s4, s5, none = s1
     output, o : file
     cip_file, cf : file
     build_level, bl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     status : var of status

volume_serial_number | vsn | This is the tape to write the
     CIP onto.  Required parameter.

machine_class | mc : This is version of CIP to produce.

     Default if omitted : S1.

output | o  :  This  is  the  file  to  write  informative
     messages to.

     Default if omitted : $RESPONSE.

cip_file | cf : This is the file from which the non-NOS/VE
     portion of the cip information is taken.

     Default  if  omitted  :  From  the  appropriate  cip
     catalog, with the build_level selected from the TYING
     file.

build_level | bl : specifies the system or  product  build
     level to be used.

feature_catalog | fc : specifies the feature catalog to be
     used (if any).

feature_build_level | fbl : specifies  the  feature  build
     level to be used.

working_catalog | wc : specifies the working catalog to be
     used.

working_build_level | wbl : specifies  the  working  build
     level to be used.

status : see NOS/VE error handling.
```

3.3.18 CREATE_TEST_TAPE (CRETT)

This procedure creates the deadstart tape which contains the test base.

```
create_test_tape
     rvsn : string = $required
     evsn : string = $required
     status : var of status = $optional
```

rvsn : The internal vsn of the test tape.

evsn : The external vsn of the test tape.

status : See NOS/VE error handling.

3.3.19 GENERATE_CIP_COMPONENTS (GENCC)

The purpose of this procedure is to generate all of the CIP (Cyber Initialization Package) "pp" components needed for the CIP tape. The pp's are converted using BUILD_CIP_PP and the membory_image components are converted using PACKAGE_EI.

```
generate_cip_components
     version, v : integer 0..99999
     full_build, fb : boolean
     build_level, bl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     status : var of status
```

version | v : This is the CIP version being built.

full_build |fb : This controls whether pp's are to be selected from the master catalog (TRUE) as well as the feature and working catalogs (FALSE).

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.20 GENERATE_DEADSTART_FILE (GENDF)

This command can NOT be used inside the Working_Environment.

This command puts together a file suitable for deadstarting NOS/VE using the output from calls to the LINK_OPERATING_SYSTEM command.

The files needed for this operation are obtained by searching the build subcatalogs within the working catalog and feature catalog, and the operating system build subcatalog. If present, the files at the working and feature levels are subsets of the corresponding files at the integration level.

```
generate_deadstart_file
     volume_serial_number : string
     delete_modules : file
     delete_linked_files : boolean
     cleanup_170 : key START END BOTH NONE
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : file or key none = none
     feature_build_level : name = object
     working_catalog : file or key none = none
     working_build_level : name = object
     status : status
```

volume_serial_number | vsn : specifies the tape to which the deadstart file is to be written.

delete_modules | dm : specifies file containing one or more lines each of which is a module deletion directive. See delete_modules parameter in Link_Operating_System command for directive format.

Omission causes no modules to be deleted.

delete_linked_files | dlf : specifies whether the image file and the symbol table files created by the link_operating_system procedure are to be deleted after successful creation of a deadstart file.

Omission causes TRUE to be selected.

NOTE : If both the system core and job template are linked and this option set to true, next time this system is linked both system core and job templates have to be relinked even if only the job template portion is modified because the system core images and symbol tables have been deleted. If the system core files were saved in a feature catalog, or this option set to false, this would not be required if feature catalog specified on link.

cleanup_170 | c7 : determines how 170 side files are transferred by the procedure. Allowable values are START, END, BOTH or NONE. Specifying START will move all files to the 170 side. Specifying END will purge all 170 side files when procedure completed, no files are moved to 170 side. BOTH moves the files to the 170 side and purges them when the procedure completes. Specifying NONE will neither move files to the 170 side or purge them when the procedure completes. The parameter is of no use in normal development operations but is used by integration.

Omission causes BOTH to be used.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the product catalog to be

used. See working environment conventions for details.

Default if omitted : w-e defaults

build_level | bl : specifies the system build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.21 GENERATE_NOS_DEADSTART_TAPE (GENNDT)

The purpose of this procedure is to produce a new NOS deadstart tape version from a previous version; adding files from the working environment.

```
generate_nos_deadstart_tape
     old_nos_deadstart_tape, ondt : name 1..6
     new_nos_deadstart_tape, nndt : name 1..6
     nos_level, nl : string 4
     nos_ve_level, nvl : string 5
     print_catalog, pc : boolean
     product_name, pn : name
     build_level, bl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     status : var of status
```

old_nos_deadstart_tape | ondt : This is the VSN of the previous version of the NOS deadstart tape.

new_nos_deadstart_tape | nndt : This is the VSN of the new

version of the NOS deadstart tape.

nos_level | nl : This is the string containing the NOS level. All CMR decks are changed to display this at login.

nos_ve_level | nvl : This is the string containing the NOS/VE level. All CMR decks are changed to display this at login.

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.22 LINK_170

This procedure links 170 relocatable binaries which reside on the 170 libraries NOSBINS and NVELIB. The binaries used in linking are gotten from the working, feature, and product build levels of NOSBINS, NVELIB and NVEBINS. The linked object libraries and link maps are retrieved back to the 180 along with the dayfile of the 170 partner job when linking completed.

```
link_170
     module : key
     listing : file
     target_operating_system : key
     cleanup_170 : key
     full_build : boolean
     build_iiapas_with_debug : boolean
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : file or key none = none
```

working_catalog : file or key none = none
status : status variable

module | m : specifies the 170 module to be linked. Valid entries are DSMDSTG, DSMDST, DSMRUN, DSMTRM, IIAPAS, RHMPFP, RHAQAC, FMSLAVE, ALL (all 8), NONE.

Required parameter.

The keyword NONE merges the libraries OSF$NOSBINS, OSF$NVEBINS and OSF$NVELIB with the feature and build catalog version and saves the result libraries in the working catalog. The result library names are NOSBINS, NVEBINS and NVELIB.

listing | l : specifies the 180 file to retrieve the 170 link maps to.

Omission causes the maps to be written to 'MAPS_170' in the 'maintenance' subcatalog of the working environment

target_operating_system | tos : specifies the type of 170 operating system the binaries are being linked for. Valid entries are NOS and NOSBE.

Omission causes NOS to be used.

cleanup_170 : this parameter allows a job to override the retrieval and replacing of the working catalog files for better performance when compiling and linking several 170 modules. It is not recommended that this parameter be used unless its use is fully understood with respect to COMPILE_SOURCE and GENDF. Valid entries are :

START : replace files, do not retrieve results.
END : do not replace files, DO not retrieve results.
BOTH : replace files, retrieve results.
NONE : do not replace files, do not retrieve results.

Omission causes BOTH to be used.

full_build | fb : this parameter specifies whether complete libraries should be generated consisted of

the INTVE, feature and working_catalog copies merged together. This parameter is intended for integration use. Most other users should use the default value. The merged copies are moved into the working_catalog.working_build_level subcatalog.

Omission causes FALSE to be used.

build_iiapas_with_debug | biwd : causes a debug version of IIAPAS (using NETIOD in place of NETIO) to be generated.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.23 LINK_BOOT (LINB)

The purpose of this procedure is to link the object libraries OSF$BOOT_JOB and OSF$BOOT_MONITOR. The results of this link is the BOOT_MEMORY_IMAGE file.

link_boot

```
display_options, do : key errors, e, full, f
delete_modules, dm : file
build_level, bl : name
product_name, pn : name
feature_catalog, fc : file or key none
feature_build_level, fbl : name
working_catalog, wc : file or key none
working_build_level, wbl : name
status : var of status
```

display_options | do : This controls whether the link maps are retained always (FULL) or only if errors occur (ERRORS).

delete_modules | dm : This parameter allows the user to delete existing modules that may have been added from the integration copy of the files.

build_level | bl : specifies the system or product build level to be used.

product_name | pn : specifies the system or product to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.24 LINK_ENVIRONMENT_INTERFACE (LINEI)

The purpose of this procedure is to link the library OSF$C170_EI. This procedure then produces the environment_interface that is placed on the CIP tape after being converted by BUILD_CIP_PP.

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT
3.3.24 LINK_ENVIRONMENT_INTERFACE (LINEI)

```
link_environment_interface
     display_options, do : key errors, e, full, f = errors
     delete_modules, dm : file
     build_level, bl : name
     product_name, pn : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     status : var of status
```

display_options | do : This controls whether the linkmaps are retained always (FULL) or only if errors occur (ERRORS).

delete_modules | dm : This parameter allows the user to delete existing modules that may have been added from the integration copy of the files.

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.3.25 LINK_OPERATING_SYSTEM (LINOS)

This command links the various components of the operating system. The components can be linked independently using output from previous calls to this command, or the entire operating system can be linked with

one call.

The files needed for linking are obtained by merging files from build_subcatalogs within the working catalog and feature catalog with the corresponding files in an operating system build subcatalog. If present, the files at the working and feature levels are subsets of the corresponding files at the integration level.

NOTE : If both link_system_core and link_job_template are specified as false, osf$tasks and osf$builtin_library are linked if it is necessary.

```
link_operating_system
     link_job_template : boolean
     link_system_core : boolean
     delete_modules : file
     operating_system_identifier : string 1..5
     product_set_identifier : string 1..3
     debug : boolean
     display_options : list of name
     page_size : integer
     page_table_length : integer
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : file or key none = none
     feature_build_level : name = object
     working_catalog : file or key none = none
     working_build_level : name = object
     status : status
```

link_job_template | ljt : specifies whether a job template is to be linked.

Omission causes TRUE to be used.

link_system_core | lsc : specifies whether a system core is to be linked.

Omission causes FALSE to be used.

delete_module | delete_modules | dm : specifies a file containing one or more lines each of which is a module deletion directive. The format of these directives should be :

```
delete_module module status= ignore_status
```

The status must be specified as shown. The directives are executed for each library in job_template and/or system_core and the module won't exist in all libraries. The Linker will issue a warning each time the module is "not found". If not specified in this manner the procedure will abort.

Omission causes no modules to be deleted.

operating_system_identifier | osi : specifies the operating system part of the system header.

Omission causes the build level to be used.

product_set_identifier | psi : specifies the product set part of the system header.

Omission causes SVL to be used.

debug | d : specifies whether to include debug information (for use with dump analysis tools).

Omission causes TRUE (include the debug information) to be used.

display_options | do : specifies whether the link map is saved or not. The options are errors, e, full or f. If errors is specified, the link map is saved fi there are linker errors. If no errors, the link maps are deleted. Specifying full saves link maps regardless of errors.

Omission will cause the ALL option to be used.

NOTE : When any option other than FULL is selected, the full maps produced by the linker are deleted. Also, the binary maps are created when any option other than FULL is selected.

page_size | ns : specifies the virtual memory page size to be used.

Omission will cause the "standard" page size to be used.

page_table_length | ptl : specifies the virtual memory page table length to be used.

Omission will cause the "standard" page table length to be used.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the product catalog to be used. See working environment for details.

Default if omitted : w-e defaults

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

3.4 EDITOR ORIENTED COMMANDS

3.4.1 COPY_BOX (COPB CB)

The procedure is to be used in conjunction with the screen editor. After a user has marked a box of text; he can call this procedure to copy it to another portion of the screen. The cursor position is expected to be the upper, left hand, corner of the boxed text's new location.

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT

3.4.1 COPY_BOX (COPB CB)

```
copy_box
     status : var of status

status : see NOS/VE error handling.
```

3.4.2 DELETE_BOX (DELB DB)

This is a procedure for use with the screen editor. It will delete a block of text marked with the box_mark key.

```
delete_box
     status : var of status

status : see NOS/VE error handling.
```

3.4.3 MOVE_BOX (MOVB MB)

This procedure is used in conjunction with the screen editor. It will move a marked block of text.

```
move_box
     status : var of status

status : see NOS/VE error handling.
```

3.5 MISCELLANIOUS USEFUL PROCEDURES

3.5.1 BACKUP_USER_CATALOG (BACUC)

The purpose of this procedure is to perform a BACKUP_PERMANENT_FILES on a user's entire catalog and replace the resultant backup file to a 170 file called BACK180.

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT

3.5.1 BACKUP_USER_CATALOG (BACUC)

```
backup_user_catalog
     status : var of status

status : see NOS/VE error handling.
```

3.5.2 CATALOG_MULTI_RECORD_TAPE (CATMRT)

This procedure will list the contents of a NOS/VE or CIP deadstart tape.

```
catalog_multi_record_tape
     external_vsn, evsn : name = $required
     kind, k : key of, s0_cip, cip, nos_ve = $required
     type, t : key of, mt9$800, mt9$1600, mt9$6250 =
     mt9$6250
     output, o : file = $response
     status : var of status = $optional

external_vsn | ev : The vsn of the tape to catalog.

kind | k : The expected data on the tape.

type | t : The tape density.

output | o : The file to write the listing onto.

status : See NOS/VE error handling.
```

3.5.3 COMPARE_CATALOG_CONTENTS (COMCC)

This is a utility procedure used to compare two NOS/VE catalogs. It will do a depth-first comparison and list any differences.

```
compare_catalog_contents
0catalog_1, c1 : file = $required
     catalog_2, c2 : file = $required
     result, r : file = $response
     status : var of status
```

catalog_1 | c1 : The first catalog to compare.

catalog_2 | c2 : The first catalog to compare.

result | r : The file that recieves the difference listings.
Note: the output is written to the END of this file.

status : see NOS/VE error handling.

3.5.4 COMPILE_DECKS_IN_ISOLATION (COMDII COMPILE_DECK_IN_ISOLATION)

```
*****************************
DESCRIPTION NEEDED <<<<-------
*****************************
compile_decks_in_isolation
     decks, deck, d : list of name or key all = $optional
     include_decks_file, ..
     include_deck_file, idf : file = $optional
     base, b : file = source_library
     alternate_base, ab : file or key none =
     .intve.os.source_library
     width, w : integer 10..110 = 110
     list, l : file = $list
     status : var of status = $optional

-->> PARAMETER DESCRIPTIONS NEEDED <<--
```

status : See NOS/VE error handling.

3.5.5 COPY_NOS_TAPE (COPNT)

This is a utility procedure used to copy NOS deadstart tapes from the VE side. It will only work in dual-state mode.

```
copy_nos_tape
     source_vsn, svsn : name 1..6
     target_vsn, tvsn : list of name 1..6
```

```
          source_density, sd : key of, pe, ge = pe
          target_density, td : key of, pe, ge
          format, f : key of, i, si = i
          status : var of status
```

source_vsn | svsn : The label on the tape to copy.

target_vsn | tvsn : The label on the tape to copy to.

source_density | sd : The bit density of the source tape.

target_density | td : The bit density of the target tape.

Default if omitted : source_density.

format | f : The NOS standard tape formats.

status : see NOS/VE error handling.

3.5.6 COPY_NOS_VE_TAPE (COPNVT)

This is a utility procedure used to copy NOS/VE deadstart and product tapes (ON NOS - this procedure only works on dual state machines - not standalone).

```
copy_nos_ve_tape
     source_vsn, svsn : name 1..6
     target_vsn, tvsn : list of name 1..6
     source_density, sd : key of, pe, ge = pe
     target_density, td : key of, pe, ge
     status : var of status
```

source_vsn | svsn : The label on the tape to copy.

target_vsn | tvsn : The label on the tape to copy to.

source_density | sd : The bit density of the source tape.

target_density | td : The bit density of the target tape.

Default if omitted : source_density.

status : see NOS/VE error handling.

3.5.7 COPY_UNLABELLED_TAPES (COPUT COPY_UNLABELLED_TAPE)

This procedure copies unlabelled tapes on the 180 side by first requesting both the source tape and the target tape, then doing a COPY_FILE on them. It has to copy the tapes file by file and it needs to write a tape mark on the target tape after every file.

If a list is specified for the TARGET_VSN, the procedure will continue copying the remaining tapes if one of the tapes does not copy correctly. It will specify the external vsn of the bad tape and it will also return bad status.

```
copy_unlabelled_tapes
     source_vsn, svsn : name 1..6 = $required
     target_vsn, tvsn : list of name 1..6 = $required
     type, t : key mt9$800, mt9$1600, mt9$6250 = mt9$6250
     status : var of status = $optional
```

source_vsn | sv : The external name of the original tape.

target_vsn | tv : The external name of the new copy.

type | t : The tape density.

status : See NOS/VE error handling.

3.5.8 DELETE_JOB (DELJ)

The purpose of this procedure is to totally remove an entry for a job from a build_job catalog. The following files are affected :
1) The build job is deleted.
2) The build job's job log is deleted.
3) All entries for the job are removed from the status file.

```
delete_job
     build_job_catalog, bjc : file
     job_name, jn : name
     status : var of status
```

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT
3.5.8 DELETE_JOB (DELJ)

build_job_catalog | bjc : This is the catalog created by MAKE_BUILD_JOBS and which contains the status retrieval file that is to have it's entry removed.

job_name | jn : This is the entry to remove

status : see NOS/VE error handling.

3.5.9 DISPLAY_CATALOG_CONTENTS (DISCC)

This is an alternative to the system's DISPLAY_CATALOG command. This procedure will display subcatalogs in a breadth-first manner as opposed to the depth-first mode used by DISPLAY_CATALOG.

```
display_catalog_contents
     catalog, c : file = $USER
     output, o : file = $OUTPUT
     status : var of status
```

catalog | c : This is the catalog to display.

output | o : This is the file to receive the display.

status : see NOS/VE error handling.

3.5.10 GET_MULTI_RECORD_FILE (GETMRF)

The purpose of this procedure is to retrieve NOS ULIB format files.

```
get_multi_record_file
     nos_ve_file, nvf : file
     nos_file, nf : name 1..7
     development_base : file = .intve
     status : var of status
```

nos_ve_file | nvf : The 180 file name to retrieve the data to.

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT
3.5.10 GET_MULTI_RECORD_FILE (GETMRF)

nos_file | nf : The 170 file to retrieve.

development_base | db : specifies the catalog in which all products and build levels reside.
Default if omitted : .INTVE

status : see NOS/VE error handling.

3.5.11 GENERATE_SCL_LISTINGS (GENSL GENERATE_SCL_LISTING)

The purpose of this procedure is provide headers for scl procedures which look like those from a compiler.

```
generate_scl_listings
     scl_command_library, scl : file = $required
     listing_file, lf : file = $required
     status : var of status = $optional
```

scl_command_library | scl : This is the object library that contains the scl procs which are to have headers put on their listings.

listing_file | lf : This is the file to receive the listings (with headers).

status : See NOS/VE error handling.

3.5.12 REPLACE_MULTI_RECORD_FILE (REPMRF)

The purpose of this procedure is to send NOS ULIB format files to the 170 side of the system.

```
get_multi_record_file
     nos_ve_file, nvf : file
     nos_file, nf : name 1..7
     development_base : file = .intve
     status : var of status
```

nos_ve_file | nvf : The 180 file name to retrieve the data to.

nos_file | nf : The 170 file to retrieve.

development_base | db : specifies the catalog in which all products and build levels reside.
Default if omitted : .INTVE

status : see NOS/VE error handling.

3.5.13 RESTORE_USER_CATALOG (RESUC)

The purpose of this proc is to restore a users catalog from the backup file BACK180 saved on the 170-side. BACK180 is produced by BACKUP_USER_CATALOG (BACUC).

```
restore_user_catalog
     status : var of status
```

status : see NOS/VE error handling.

3.5.14 SEQUENCE_LIBRARY (SEQL)

This is a utility procedure which is used during library cleanup. It is used by the update jobs to change modifications to state 4 and then resequence them. It should be used by development to perform the same task. This procedure expects to be executed from within the working environment (inside ENTWE). It will reference a file called SEQUENCE_LIST which exists on the INTVE product catalog which the user is using. This file will have the list of modifications to change to state 4.

```
sequence_library
     status : var of status
```

status : see NOS/VE error handling.

4.0 SUBROUTINES

4.0 SUBROUTINES

The procedures in the following section are not intended for direct use. This section of documentation is intended for procedure writers who may need to make use of one of these components.

4.1 LANGUAGE_PROCESSOR_SUBROUTINES

The purpose of the language processor subroutines are to provide compile_source with a uniform interface to a set of "unusual" (mostly NOS-resident) languages which do not follow the NOS/VE System Interface Standards. Each of these procedures appears to compile_source as a compiler for the language in question. Each procedure is responsible for massaging the input source text, moving files to the 170, initiating a 170 compile job and retrieving the compile status, "object" file/library and listings. The listings may be massaged to convert the header to 180 standards if they are destined to be placed on a listing library. Each 170 processor also communicates with compile_source via some xdcl/xref'ed variables that control whather compile source should retrieve the result files.

A second set of processor procedures exist for 180 processes for which there is no compiler or for which some special manipulation must take place prior to compilation. Examples of the former are scl_procedures and program_descriptions. An example of the latter is generate_command_table_module.

4.1.1 CCL_PROCEDURES

This procedure puts a NOS/170 command langauge procedure onto a ULIB format library.

ccl_procedures

4.0 SUBROUTINES
4.1.1 CCL_PROCEDURES

```
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

4.1.2 CONVERT_TEXT_TO_OBJECT

This is a processor that will convert an object library, that has been encoded into a text deck as hex bytes, back into the object library again.

```
convert_text_to_object
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status : var of status = $optional
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This is a dummy parameter that is used to provide compliance with COMPILE_SOURCE. It is ignored.

status : See NOS/VE error handling standards.

4.1.3 CP_COMPASS

This is the standard NOS (or NOS/BE depending on the value of the wev$target_operating_system variable) compiler. Note : this procedure will satisfy external references from the following 'text' files which exist in the 170 catalog pointed to by the tying file variable wev$nos_base (or the NOS/BE psudo text files pointed to by wev$nosbe_base).

```
cp_compass
     input : file
     list : file
     binary : file
     list_options : list of name
     get_system_text : list of name
     status : status : var of status = $optional
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

get_system_text | gst | g : Additional 'text' files from wev$nos_base (or wev$nosbe_base) to satisfy external references from.

list_options | lo : This parameter allows the user to select the type of listing he prefers. Only standard compass options are used - anything else is ignored.

status : See NOS/VE error handling standards.

4.1.4 CREATE_FASLAVE (CREFS)

The purpose of this procedure is to produce the deck components for FASLAVE. This procedure works in conjunction with a dummy deck that exists on os source_library. This dummy deck has all of the attributes

needed to invoke this procedure when an attempt is made to compile it. This procedure will then compile the decks off of the FMA product and put them into NVERELS as required.

```
create_faslave
     input, i : file = $local.$null
     binary, b : file = $local.$null
     list, l : file = $local.$null
     list_options, lo : list of name = $optional
     status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling.

4.1.5 CREATE_MANUAL (CREM)

The purpose of this procedure is to overlay the system standard CREATE_MANUAL procedure with an interface which is compatable with COMPILE_SOURCE.

```
create_manual
     input, i : file = $local.$null
     binary, b, output, o : file = $local.$null
     error, e : file = $local.$errors
     list, l : file = $local.$null
     list_options, lo : list of name = $optional
     status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If

compile_source's save_listing parameter is not none, the listing is massaged prior to output.

error | e : This is the file where all detected errors are listed.

binary | b | output | o : This is the file where the result object library is to end up. The 'binary' parameter is supplied for compile_source, the 'output' parameter is supplied for compatibility with the standard create_manual utility.

list_options | lo : This parameter selects whither a cross-reference is generated. The only valid values are 'R' and 'NONE'. Anything else is ignored by this procedure.

status : See NOS/VE error handling.

4.1.6 CREATE_INSTALLATION_TABLE (CREIT)

This is a processor for COMPILE_SOURCE. It is used to produce an installation table from a deck on the OS source_library.

```
create_installation_table
     input, i : file = compile
     binary, b : file = lgo
     list, l : file = $list
     list_options, lo : list of name
     status : var of status
```

input | i : The file containing the installation table source deck.

binary | b : The destination file name for the installation table.

list | l : This is the file which receives a display of the resultant installation table.

list_options | lo : This parameter is included for compatibility with other processors. It is ignored.

status : see NOS/VE error handling.

4.1.7 CYBIL_CC

This procedure processes cybil_cc procedures. Note : this procedure will use version of the cybil compiler from the wev$cybil_cc_base catalog. If target_operating_system is equal to NOS/BE, it will use the version from the wev$nosbe_cybil_cc_base catalog. Also note that all cybil_cc interface decks (LGxxxxx, BIxxxxx, DIxxxxx, etc.) need to be satisfied at expansion time, not at compilation time. It is recommended that users of this language specify ap=COMMON_170 on the compile_source call.

```
cybil_cc
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter allows the user to select the type of listing he prefers. Only standard cybil options are used - anything else is ignored.

status : See NOS/VE error handling standards.

4.1.8 CYBIL_OBJECT_CHECKING

This proccessor front ends the cybil processor then sets weak parameter checking on the modules generated. The cybil_parameters parameter is a string that will contain

any parameters of cybil that the user wants to feed into cybil. When called from compile_ source, these parameters must be specified within double_quotes, ie. coms p='cybil_object_checking cp=''opt=high rc=none da=all'''.

```
cybil_object_checking
     input : file
     binary : file
     list : file
     list_options : list of name
     cybil_parameters : string
     status : status
```

input | i : This file contains the source text.

binary | b : This is the file where the result object library is to end up.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

list_options | lo : This parameter allows the user to select the type of listing he prefers. Only standard cybil options are used - anything else is ignored.

cybil_parameters | cp : This parameter allows the user to use any of the parameters of cybil that are not listed above.

status : See NOS/VE error handling standards.

4.1.9 DEFINE_MULTI_TERMINAL (DEFMT)

The purpose of this procedure is to trick compile source into compiling terminal definition decks. This procedure will handle multiple definition decks on one input file.

```
define_multi_terminal
     input, i : file = compile
     binary, b : file = lgo
     list, l : file = $list
     list_option, lo : list of name
     status : var of status
```

4.0 SUBROUTINES
4.1.9 DEFINE_MULTI_TERMINAL (DEFMT)

input | i : This is the file which contains the source data.

binary | b : This is the location to place the results.

list | l : This is the file to receive the compilation listings.

list_option | lo : This parameter controls the items which appear on the listings.

status : see NOS/VE error handling.

4.1.10 DESKTOP_CONFIGURATION (DESCF)

The purpose of this procedure is to trick COMPILE_SOURCE into creating desktop environment toolboxes from toolbox text

```
desktop_configuration
     input, i : file = $required
     binary, b : file = $required
     list, l : file = $NULL
     list_options, lo : list of name = $optional
     status : var of status = $optional
```

input | i : This file contains the source text.

binary | b : This is the file where the result object library is to end up.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling.

4.1.11 DESKTOP_CONTEXT (DESC)

The purpose of this procedure is to trick COMPILE_SOURCE into compiling desktop environment context files.

```
desktop_context
     input, i : file = $required
     binary, b : file = $required
     list, l : file = $NULL
     list_options, lo : list of name = $optional
     status : var of status = $optional
```

input | i : This file contains the source text.

binary | b : This is the file where the result object library is to end up.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling.

4.1.12 DESKTOP_TOOLBOX (DEST)

The purpose of this procedure is to trick COMPILE_SOURCE into creating desktop environment toolboxes from toolbox text

```
desktop_toolbox
     input, i : file = $required
     binary, b : file = $required
     list, l : file = $NULL
     list_options, lo : list of name = $optional
     status : var of status = $optional
```

input | i : This file contains the source text.

4.0 SUBROUTINES
4.1.12 DESKTOP_TOOLBOX (DEST)

binary | b : This is the file where the result object library is to end up.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling.

4.1.13 FORTRAN_CC

This procedure processes FTN5 procedures. Note : this procedure will use the version of the fortran compiler from the system running on the 170.

```
fortran_cc
     input : file
     list : file
     binary : file
     list_options : list of name
     optimization : integer 0..3
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter allows the user to select the type of listing he prefers. Only standard fortran options are used - anything else is ignored.

optimization | opt | o : This parameter allows the user to specify the fortran optimization level to use. Default is set to two because opt level three is considered "potentially dangerous" in the FTN5 manual.

status : See NOS/VE error handling standards.

4.1.14 GENERATE_COMMAND_TABLE_MODULE (GENCTM)

This procedure causes an expanded deck with this processor to be put through a 'precompiler' before being sent through the standard cybil compiler. This precompiler changes the command table deck into a real cybil module.

```
generate_command_table_module
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | list_option | lo : This parameter allows the user to select the type of listing he prefers. Only standard cybil options are used - anything else is ignored.

status : See NOS/VE error handling standards.

4.1.15 GENERATE_MSG_TEMPLATE_MODULE (GENMTM)

This command is used to create an object file from message template definitions.

This command utilizes the GENERATE_MESSAGE_TEMPLATE command to generate commands for the CREATE_MESSAGE_MODULE sub_utility of CREATE_OBJECT_LIBRARY.

The message template source must be expanded and

enclosed by MODULE/MODEND statements before being used as input to GENERATE_MSG_TEMPLATE_MODULE. GENERATE_MESSAGE_TEMPLATE will then generate CREATE_MESSAGE_MODULE, CREATE_STATUS_MESSAGE and END_MESSAGE_MODULE commands. CREATE_OBJECT_LIBRARY is invoked, the subcommands are processed and a library is generated.

```
generate_msg_template_module
     input : file
     binary : file
     list : file
     errors : file
     product_identifier : name
     list_options : list of name
     status : status
```

input | i : specifies the file which contains the expanded message template.

binary | b : specifies the file into which the compiled message template module will be placed.

Omission causes $LOCAL.LGO to be used.

list | l : specifies the file to which the compilation listings are to be written.

Omission causes $LIST to be used. (Warning - in a batch job, $LIST is connected to OUTPUT. If a listing is not desired in a batch job, $NULL should be specified for list.)

errors | e : specifies the file which contains errors encountered by GENERATE_MESSAGE_TEMPLATE. If this file is connected to $ERRORS, then this file will also contain errors encountered by CREATE_MESSAGE_MODULE subcommands (i.e., CREATE_STATUS_MESSAGE).

Omission causes $ERRORS to be used.

product_identifier | pi : specifies the two character product name concatenated onto the module name generated by GENERATE_MESSAGE_TEMPLATE.

Omission causes OS to be used.

list_options | list_option | lo : specifies the list options to be used on the CYBIL call. (Warning - this parameter is ignored. Thus, the default options (S,R) are used.)

status : see NOS/VE error handling.

4.1.16 META

This is the NOS META assembler.

```
meta
     input : file
     list : file
     binary : file
     list_options : list of name
     get_system_text : list of name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling standards.

4.1.17 PP_COMPASS

This is the processor procedure for NOS/VE pp drivers. Prior to 13500, this was a seperate, modified, compass assembler. At NOS/VE 13500 (NOS level 51F3), this capability was added to the standard nos compass assembler.

4.0 SUBROUTINES
4.1.17 PP_COMPASS

```
pp_compass
     input : file
     list : file
     binary : file
     list_options : list of name
     get_system_text : list of name
     name_170 : name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter allows the user to select the type of listing he prefers. Only standard compass options are used - anything else is ignored.

get_system_text | gst | g : Additional 'text' files from wev$nos_base (or wev$nosbe_base) to satisfy external references from.

name_170 | n : This parameter was used to allow the ident card value to be overridden. It is now obsolete, but is retained for backward compatability.

status : See NOS/VE error handling standards.

4.1.18 PROGRAM_DESCRIPTIONS (PROGRAM_DESCRIPTION,

PROGRAM_DESCRIPTORS, PROGRAM_DESCRIPTOR, PROD)
This procedure exists to put program_descriptions, specified as text, onto object libraries.

```
program_descriptions
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status
```

4.0 SUBROUTINES
4.1.18 PROGRAM_DESCRIPTIONS (PROGRAM_DESCRIPTION,

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling standards.

4.1.19 SCL_PROCEDURES (SCL_PROCEDURE, SCLP, SCL)

This procedure puts VE command language procedures onto an object library.

```
scl_procedures
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling standards.

4.1.20 SYMPL

This procedure will compile decks written in 170 SYMPL and put the results onto a ULIB object library.

```
sympl
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter allows the user to select the type of listing he prefers. Only standard sympl options are used - anything else is ignored.

status : See NOS/VE error handling standards.

4.1.21 TEXTCODE_TEXTFORM

This procedure will "compile" decks written with 170 textcode/textform embedded formatting directives.

```
textcode_textform
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result text file is to end up.

list_options | lo : This parameter is ignored. It only exists for compatibility with other processor types.

status : See NOS/VE error handling standards.

4.1.22 TEXT_FILES

The purpose of this procedure is to trick COMPILE_SOURCE into generating text files instead of object libraries. Note: list_options are ignored. The only purpose of this parameter is to provide easier interfacing with compile_source.

```
text_files
     input, i : file = compile
     binary, b : file = lgo
     list, l : file = $list
     list_options, lo : list of name = $optional
     status : var of status = $optional
```

input | i : This file contains the source text.

binary | b : This is the file where the result object library is to end up.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling.

4.1.23 TEXT_170

This procedure puts decks onto a 170 ULIB library as NOS "TEXT" type records.

```
text_170
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling standards.

4.1.24 Z80_COMPASS

This procedure uses a modified compass assembler to produce Z80 object code. It is used to produce the console drivers which use the CDC721 terminal as a console. It is also used by RAA's key utility.

```
z80_compass
     input : file
     list : file
     binary : file
     list_options : list of name
     status : status
```

input | i : This file contains the source text.

list | l : This is the file to put the listing onto. If

compile_source's save_listing parameter is not none, the listing is massaged prior to output.

binary | b : This is the file where the result object library is to end up.

list_options | lo : This parameter allows the user to select the type of listing he prefers. Only standard compass options are used - anything else is ignored.

status : See NOS/VE error handling standards.

4.2 MISCELLANIOUS SUBROUTINES

The following is a list of subroutines for the main working environment procedures. This list is provided as a reference for procedure writers.

4.2.1 ACQUIRE_FILE (ACQF)

The purpose of this procedure is to search a list of catalogs for a specified file name and attach the copy from the first catalog it is found in.

```
acquire_file
     file_name, fn : name
     local_file_name, lfn : name
     search_catalogs, sc : list of file
     status : var of status
```

file_name | fn : This is the file to search for.

local_file_name | lfn : This is the name to use when attaching the file.

search_catalog | sc : These are the catalogs to look in. The catalogs are searched in the order specified.

status : see NOS/VE error handling.

4.2.2 APPEND_TO_SPECIAL_REQUESTS (APPTSR)

The purpose of this procedure is to provide a mechanism for user and integration procedures to add information to a product's SPECIAL_REQUESTS. This is a subroutine of MAKE_SPECIAL_REQUESTS, CHANGE_MODIFICATION_HEADER, CLEAR_INTERLOCK_REQUESTS and others.

```
append_to_special_requests
     append_file, af : file
     requestor, r : string
     output, o : file
     development_base, db : file
     product_name, pn : name
     status : var of status
```

append_file | af : This is the file of information to be added to the SPECIAL_REQUESTS file.

requestor | r : This is the person or procedure appending the information. It is used for audit-trail purposes.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

status : see NOS/VE error handling.

4.2.3 BACKUP_USER_CATALOG_TO_TAPE (BACUCT)

The purpose of this proc is to backup the current $user catalog to a tape. The catalog can then be restored using RESTORE_USER_CATALOG_FROM_TAPE.

NOTE: when doing backups of the user catalog, any file attached in write mode is not backed up.

4.0 SUBROUTINES
4.2.3 BACKUP_USER_CATALOG_TO_TAPE (BACUCT)

```
backup_user_catalog_to_tape
     vsn, v : string = $job(user)
     status : var of status = $optional
```

vsn : The external and internal vsn of the tape to use for backup.

status : See NOS/VE error handling.

4.2.4 BIND_OS_LIBRARY (BINOL)

This procedure is a subroutine for the procedure LINK_OPERATING_SYSTEM. The purpose of this procedure is to bind the specified operating system library. NOTE : If the same file is specified for the unbound library and bound library, the bound library is written over the unbound library.

This procedure is driven by a set of decks which exist on the OS source library. There is one deck for each object library to be bound. These decks contain the actual binding directives for the library in question. The format of these decks is RAF$BD_<library name> where <library name> is one of the component libraries of the OS - minus the "OSF$" prefix (eg. JOB_TEMPLATE_2DD, SYSTEM_CORE_113, etc.).

```
bind_os_library
     binding_directives, bd : file
     unbound_library, ul : file
     bound_library, bl : file
     status : var of status
```

binding_directives | bd : This file contains the library specific directives for the os library. Required parameter.

unbound_library | This is the file that contains the data to be linked. Required parameter.

bound_library | This is the file to recieve the results of the link. Required parameter.

status : see NOS/VE error handling.

4.2.5 BUILD_ANAD_COMMAND_LIBRARY (BUIACL)

The purpose of this procedure is to create a command library to aid in examining the dumps associated with a particular build level. This procedure will create a library of scl procedures to process most of the type definitions in the system.

```
build_anad_command_library
     anad_command_library, acl : file = acl
     working_catalog, wc : file = $optional
     working_build_level, wbl : name = $optional
     status : var of status = $optional
```

anad_command_library | acl : This is the file to put the dump analysis library on.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

4.2.6 BUILD_BUILTIN_LIBRARY (BUIBL)

The purpose of this procedure is to build osf$builtin_library. NOTE : This procedure expects to have access to externally defined working_environment variables.

```
build_builtin_library
     preserved_file_path, pfp : file
     cybil_run_time_library, crtl : file
     delete_modules, dm : file
     status : var of status
```

preserved_file_path | pfp : This is the destination of the library
after it is built. Required parameter.

cybil_run_time_library | crtl : This is the cybil CYF$RUN_TIME_LIBRARY to use to satisfy external references. Required parameter.

delete_module | delete_modules | dm : specifies a file containing one or more lines each of which is a module deletion directive. The format of these directives should be :

delete_module module status= ignore_status

The status must be specified as shown. If not specified in this manner the procedure will abort.

Omission causes no modules to be deleted.

status : see NOS/VE error handling.

4.2.7 BUILD_SYSTEM_IDENTIFIERS

The purpose of this request is to initialize the variables BUILD_ID and VERSION_ID. These variables are used by the linking process to initialize the system level identifiers.

The IDENITIFIER_FILE is needed for future linking done at the site and contains code to create and initialize the BUILD_ID and VERSION_ID at that time.

The formats for the BUILD_ID and VERSION_ID are as follows:

```
 BUILD_ID = 'NOS/VE '//operating_system_level//' '// ..
product_set_level//' '
 VERSION_ID = 'NOS/VE '//release_level//' '// ..
psr_summary_level//' '
```

Where the OPERATING_SYSTEM_LEVEL is a 5 character field containing alphanumeric. The PRODUCT_SET_LEVEL is a 6 character field containing alphanumeric. Both of these

are left justified blank filled. The RELEASE_LEVEL is a 5 character field containing a numeric followed by a '.' then a numeric followed by another '.' and finally another numeric. The PSR_SUMMARY_LEVEL is a 6 character field containing an 'L' followed by a 3 character numeric and finally an optional 2 character alpha (bcu level). This is left justified blank filled as well. An example of each is given below :

```
12345678901234567890
    BUILD_ID = NOS/VE 12615 4RD
  VERSION_ID = NOS/VE 1.1.3 L644AA
```

```
build_system_identifiers
     build_id, bi : var of string
     version_id, vi : var of string
     identifier_file, if : file
     operating_system_level, osl : string 1..5
     product_set_level, psl : string 1..6
     release_level, rl : string 1..5
     psr_summary_level, pl : string 1..6
     status : var of status
```

build_id | bi : The NOS/VE internal level id codes.

version_id | vi : The NOS/VE external level id codes.

identifier_file | if : A file released as part of the system code, which identifies the system.

operating_system_level | osl : The numeric part of an OS build_level.

product_set_level | psl : The numeric part of a sunnyvale build_level.

release_level | rl : The release management id level.

psr_summary_level | pl : The customer services BCU/PSR id level.

status : see NOS/VE error handling.

4.2.8 BUILD_TASKS_LIBRARY (BUITL)

The purpose of this procedure is to build osf$tasks. NOTE: This procedure expects to have access to externally defined working_environment variables.

```
build_tasks_library
     preserved_file_path, pfp : file
     cybil_run_time_library, crtl : file
     delete_modules, dm : file
     status : var of status
```

preserved_file_path | pfp : This is the destination of the library after it is built. Required parameter.

cybil_run_time_library | crtl : This is the cybil CYF$RUN_TIME_LIBRARY to use to satisfy external references. Required parameter.

delete_module | delete_modules | dm : specifies a file containing one or more lines each of which is a module deletion directive. The format of these directives should be :

```
delete_module module status= ignore_status
```

The status must be specified as shown. If not specified in this manner the procedure will abort.

Omission causes no modules to be deleted.

status : see NOS/VE error handling.

4.2.9 CHANGE_OBJECT_LIBRARY (CHAOL)

This procedure will combine an object text file with an existing object library or will create a new object library from the object text. If an old object library is specified but no value is given for a new object library, the old object library will be overwritten.

4.0 SUBROUTINES
4.2.9 CHANGE_OBJECT_LIBRARY (CHAOL)

```
change_object_library
     new_object_text, not : file
     old_object_library, ool : file
     new_object_library, nol : file
     status : var of status

new_object_text | not : This is the data to be added to to
     the library.  Required parameter.

old_object_library  |  ool  :  This is the library to have
     data added to it.

     Default if omitted : no old library.

new_object_library | nol : This is the file to  write  the
     combined library to.

status : see NOS/VE error handling.
```

4.2.10 CHANGE_TEST_GROUPS

```
This  proc  changes  the  groupname for all decks with the
specified groupname

change_test_groups
     old_group, og : name = $optional
     new_group, ng : name = $optional
     delete_group, dg : list of name = $optional
     status : var of status = $optional

old_group | og : This is the group whose attributes are to
     be changed.

new_group  |  ng : This is new group attribute to be added
     to all decks.

delete_group | dg : This is an existing group attribute to
     be removed from all decks.

status : See NOS/VE error handling.
```

4.2.11 CHECK_PARAMS_FOR_TRAT

This procedure checks that all decks, modifications, features, etc. exist and creates a selection criteria file that includes all of them and a verify_states file which includes a unique list of feature names that is passed back to TRANSMIT_SOURCE (the main subroutine of TRATI and TRATFC) for further checking.

```
check_params_for_trat
     base : file
     decks : list of name or key all
     modifications : list of name
     features : list of name
     selection_criteria : file
     verify_states : file
     status : var of status
```

base : This is the scu source_library upon which all decks modifications and features, to be checked; reside.

decks : This is a list of decks whose associated modifications and features are to be verified. All features associated with state 0 modifications associated with these decks are put onto the verify_states file (if it is specified).

modifications : This is a list of modifications whose assocated features are to be verified. All features assocated with modifications on this list are added to the verify_states file (if it is specified).

features : This is a list of features to be verified. All entries in this list are added to the verify_states file (if it is specified).

selection_criteria : This is a scu selection criteria file. It will have include_deck (or modification, or feature) entries added for each of the items listed on the parameters above.

verify_states : This is a file of unique feature names which is passed back to transmit_source for further checking.

status : see NOS/VE error handling.

4.2.12 CHECK_SPECIFIED_DEPENDENCIES (CHESD)

The purpose of this procedure is to check the specified dependencies. First the feature list is searched for each dependencies. If some aren't found in the feature_list, then the state of the code is checked. State 1 code is assumed missing. State 3 is assumed built if the feature cannot be found the expanded build decks for the base_build_level.

```
check_specified_dependencies
     feature_list, fl : file = $required
     base_build_level, bbl : name = $required
     output, o : file = $response
     product_name, pn : name = os
     status : var of status = $optional
```

feature_list : fl : A file of features (WEF$FEATURE_LIST format) which are assumed to be dependencies of a specified feature.

base_build_level | bbl : This is the standard level against which any state three dependencies will be compared.

output | o : This is the file to which any problems are written.

product_name | pn : specifies the system or product to be used.

status : See NOS/VE error handling.

4.2.13 CLEANUP_FILE_CYCLES (CLEANUP_FILE_CYCLE CLEFC)

The purpose of this procedure is to provide a consistant mechanism for procedures to use for cleaning up whenever they write cycles of files.

```
cleanup_file_cycles
     file, f : file
     retention_lenth, rl, r : integer
```

```
     retention_count, rc : integer
     set_expiration_date, sed : boolean
     status : var of status
```

file | f : This is the file to be cleaned up. It should be the n-1st cycle of the file if at all possible. If it is not possible to point to the n-1st cycle; then set_expiration_date should be set false to prevent accidental loss of all cycles of the file.

retention_length | rl : This is the length of time to set the expiration date of the file to.

Default if omitted : one day.

retention_count | rc : This is the maximum number of OLD cycles to keep around (excluding the highest). Any cycles beyond this count will be deleted; regardless of their expiration date - lowest cycle number first.

Default if omitted : four old cycles.

set_expiration_date | sed : This parameter inhibits the setting of an expiration date on the cycle of the file specified on the file parameter.

Default if omitted : set the expiration date.

status : see NOS/VE error handling.

4.2.14 COMBINE_SCU_LIBRARIES (COMSL)

The purpose of this procedure is to merge scu libraries onto a base library and then rewrite the base library.

```
combine_scu_libraries
     base, b : file
     subset, s : file
     enforce_interloks, ei : boolean
     status : var of status
```

base | b : This is the file to have the subset library added to it.

subset | s : This is the library to add.

enforce_interlocks | ei : This controls whether interlocks are checked or not.

status : see NOS/VE error handling.

4.2.15 COMPRESS_JOB_LOG (COMJL)

This is a subroutine of BUILD_EXEC. It's purpose is to extract the most useful information ina job log created by a MAKE_BUILD_JOBS-generated batch job.

```
compress_job_log
     job_log, jl : file
     output, o : file
     status : var of status
```

job_log | jl : This is the file which contains the dayfile of a compile job produced by MAKE_BUILD_JOBS.

output | o : This is the file to write the results to.

status : see NOS/VE error handling.

4.2.16 COMPRESS_LINK_MAP (COMLM)

This procedure will look for link map in working, feature catalog and build level. It then output all compressed link map on one file and produces binary map if specified .

```
compress_link_map
     output, o : file = compressed_link_map
     display_options, do : list of key compressed, c,
     alphabetical_order, ao,
     compress_production_map, ..
     cpm : boolean = true
     compress_recovery_map, crm: boolean = false
     compress_job_template, cjt: boolean = true
```

```
     production_binary_map,       pbm:       file       =
     production_binary_map
     recovery_binary_map, rbm : file = recovery_binary_map
     development_base, db : file = .intve
     product_name, pn : name = os
     build_level, bl : name = $optional
     feature_catalog, fc : file or key none = none
     feature_build_level, fbl : name = object
     working_catalog, wc : file or key none = none
     working_build_level, wbl : name = object
     status : var of status = $optional
```

output | o : This is the file to put the compressed linkmap upon.

display_options | do : This selects the method of compression.

compress_production_map | cpm : This selects whither the linkmap for the production system is compressed.

compress_recovery_map | crm : This selects whither the linkmap for the recovery system is compressed.

NOTE: This is an obsolete parameter.

compress_job_template | cjt : This selects that the job template map be compressed along with the system core map.

production_binary_map | pbm : This is the file to receive the binary linkmap (used by QUERY_DEBUG_TABLE among others).

recovery_binary_map | rbm : This is the file to receive the binary linkmap (used by QUERY_DEBUG_TABLE among others) for the recovery system.

NOTE: This is an obsolete parameter.

development_base | db : specifies the catalog in which all products and build levels reside.
Default if omitted: .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build

level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

4.2.17 COPY_LABELLED_TAPES (COPLT COPY_LABELLED_TAPE)

This procedure copies labelled tapes on the 180 side by first requesting both the source tape and the target tape, then doing a COPY_FILE on them. If a list is specified for the TARGET_VSN, the procedure will continue to copying the remaining tapes if one of the tapes does not copy correctly. It will specify the external vsn of the bad tape and it will also return bad status.

```
copy_labelled_tapes
     source_vsn, svsn : name 1..6 = $required
     target_vsn, tvsn : list of name 1..6 = $required
     recorded_vsn, rvsn : name 1..6 = $required
     type, t : key mt9$800, mt9$1600, mt9$6250 = mt9$1600
     status : var of status = $optional
```

source_vsn | sv : This is the external vsn for the original copy of the tape.

target_vsn | tv : This is the external vsn for the new copy of the tape.

recorded_vsn | rv : This is the internal vsn on both copies of the tape.

type | t : Tape density.

status : See NOS/VE error handling.

4.2.18 COPY_MULTI_RECORD_TAPES (COPMRT COPY_MULTI_RECORD_TAPE)

The purpose of this procedure is to copy NOS/VE deadstart tapes.

```
copy_multi_record_tapes
     source_vsn, svsn : name 1..6 = $required
     target_vsn, tvsn : list of name 1..6 = $required
     type, t : key of, mt9$800, mt9$1600, mt9$6250 =
     mt9$6250
     status : var of status = $optional
```

source_vsn | sv : This is the external vsn for the original copy of the tape.

target_vsn | tv : This is the external vsn for the new copy of the tape.

type | t : Tape density.

status : See NOS/VE error handling.

4.2.19 COUNT_FEATURE_LINES (COUFL)

This is a subroutine used by MAKE_BUILD_REQUESTS to return statistics on the feature being requested; one deck at a time.

```
count_feature_lines
     deck, d : name
     feature, f : name
     lines_before_feature, lbf : var of integer
     lines_deleted, ld : var of integer
     lines_added, la : var of integer
     processor, p : var of string
     status : var of status
```

deck | d : This is the deck to return statistics on.

4.0 SUBROUTINES
4.2.19 COUNT_FEATURE_LINES (COUFL)

feature | f : This is the feature to return statistics on.

lines_before_feature | lbf : This returns the size of the deck (active lines) before the feature is added.

lines_deleted | ld : This is the number of INTEGRATED lines of code which are deleted.

lines_added | la : This the number of new lines introduced.

processor | p : This is the value in the deck's processor field

status : see NOS/VE error handling.

4.2.20 CREATE_EMPTY_FILE (CREEF)

This is a subroutine which creates a file with no contents.

```
create_empty_file
     file, f : file
     status : var of status
```

file | f : The file to create.

status : see NOS/VE error handling.

4.2.21 CREATE_INTEGRATION_DESCRIPTORS (CREID)

This procedure will create the program descriptors normally found in the tools library. The tying file will be used to determine where the program descriptors will get files, normally in the build catalog specified by wev$build_level. The target_build_level parameter can be used to override the tying file value and point to whatever target_build_level is specified.

4.0 SUBROUTINES
4.2.21 CREATE_INTEGRATION_DESCRIPTORS (CREID)

```
create_integration_descriptors
     command_library, cl : file
     target_build_level, tbl : name
     termination_error_level, tel : key warning, w, error,
     e, fatal, f = warning
     create_load_map, clm : boolean
     cybil_level, cybl : name
     build_level, bl : name
     working_catalog, wc : file or key none
     working_build_level, wbl : name
     feature_catalog, fc : file or key none
     feature_build_level, fbl : name
     status : var of status
```

command_library | cl : This is the file to receive the newly created program_descriptors.

Default if omitted : .INTVE.OS.<build_level>.

target_build_level | tbl : This is the build level to use for all os path names.

build_level | bl : specifies the OS build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : see NOS/VE error handling.

4.2.22 CREATE_MULTIPLE_SUBCATALOGS (CREMS)

This is a subroutine which is used to create subcatalogs to any desired depth, given a path name.

```
create_multiple_subcatalogs
     catalog, c : file
     status : var of status
```

catalog | c : This is the catalog to create.

status : see NOS/VE error handling.

4.2.23 CREATE_OPEN_SHOP_DI_OBJECT (CREOSDO)

The purpose of this request is to create the OPEN_SHOP_DI_OBJECT by combining all the found CDCNET TIPS with the DI_OBJECT for a given version level. If OPEN_SHOP_DI_OBJECT already exists the proc returns. If OPEN_SHOP_DI_OBJECT does not exist and there are no CDCNET TIPS found, the DI_OBJECT is copied to create OPEN_SHOP_DI_OBJECT anyway.

```
create_open_shop_di_object
     build_level, bl : name = $optional
     feature_catalog, fc : file or key none = none
     feature_build_level, fbl : name = object
     working_catalog, wc : file or key none = none
     working_build_level, wbl : name = object
     status : var of status = $optional
```

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

4.2.24 CREATE_SUBTEST_LISTS

This is a subroutine to CREATE_TESTNAMES_LISTS, which creates the list of subtest names. For more information on subtests; consult testing documentation.

```
create_subtest_lists
     status : var of status
```

status : see NOS/VE error handling.

4.2.25 EXAMINE_LINK_MAPS (EXALM)

This procedure will look in the working, feature and build_level catalogs for the specified link maps. 'OS' will use SYSTEM_CORE_LINK_MAP and JOB_TEMPLATE_LINK_MAP. 'EI' will use C170_EI_LINK_MAP. And, 'BOOT' will use the BOOT_LINK_MAP. It calls wem$search_link_map to find errors in the link maps and collects the errors found, if any, on the file $local.linker_errors. If the value of the DISPLAY_OPTIONS parameter is ERRORS, the link maps will be deleted and the performance summary file will not be generated. If PERFORMANCE_SUMMARY is specified then the summary file will be generated as long as there weren't linker errors. The link maps will be deleted in either case. If FULL is specified, the summary files will be kept as long as there aren't linker errors and the link maps are kept either way. Note, the performance summary is only generated for system_core_link_maps and job_template_link_maps.

```
examine_link_maps
     link_map_type, lmt: key
    boot, ei, os
  keyend = os
     display_options, do: key
    (errors, e),
    (full, f),
    (performance_summary, ps)
  keyend = performance_summary
     product_name, pn: name = wev$product_name, os
     build_level, bl: name = wev$build_level, $optional
```

```
     feature_catalog, fc: any of
    key
      none
    keyend
    file
  anyend = wev$feature_catalog, none
     feature_build_level,        fbl:        name        =
     wev$feature_build_level, object
     working_catalog, wc: file = $user
     working_build_level,        wbl:        name        =
     wev$working_build_level, object
     status)
```

link_map_type | lmt : This selects whether to check the map for the system boot, the system maps or the map for the environmental_interface.

display_options | do : This parameter is used to control the link map and the link map summary files. If ERRORS is specified the link maps are deleted and the summaries are not generated. If PERFORMANCE_SUMMARY is specified, and if there are no errors the link maps are deleted but the summary files are left alone. If FULL is specified, and if there are no errors the summary files are kept. If there are errors the summary files are deleted. The link maps are saved in either case. Note, the performance summary file is only generated for system_core_link_maps and job_template_link_maps.

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used. This parameter defaults to $user.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

4.2.26 DUMP_CATALOG (DUMP_CATALOGS DUMC)

This is a utility procedure used to archive catalogs onto backup tapes. The procedure will submit a batch job to perform the actual backup.

```
dump_catalog
     catalog, catalogs, c : list of file
     vsn : list of string 1..6
     job_class, jc : key batch, local, maintenance
     list, l : file = $list
     password, pw : name = $name($job(user)//'x')
     status : var of status
```

catalog | catalogs | c : This is the list of catalogs to dump onto the tape(s).

vsn : This is name of the tape to do the backup to.

job_class | jc : This is the job class in which to submit the backup job.

list | l : This is the file to write the output from the backup onto.

password | pw : This is the user's password (needed for the batch job).

status : see NOS/VE error handling.

4.2.27 EXPAND_BUILD_DECKS (EXPAND_BUILD_DECK EXPBD)

This is a subroutine used by any procedure which needs to access the build decks maintained by integration. This procedure will expand the deck onto a local file, which can then be used as input to an scu criteria file parameter.

4.0 SUBROUTINES
4.2.27 EXPAND_BUILD_DECKS (EXPAND_BUILD_DECK EXPBD)

```
expand_build_decks
     expanded_build_decks, expanded_build_deck, ebd : file
     product_name, pn : name
     build_level, bl : name
     working_catalog, wc : file or key none
     feature_catalog, fc : file or key none
     status : var of status
```

expanded_build_decks | expanded_build_deck | ebd : This is the file to receive the expanded build level deck.

Default if omitted is the product name concatenated to the end of the build level; with the result truncated at 31 characters.

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

4.2.28 EXTRACT_SOURCE_MODULES (EXTRACT_SOURCE_MODULE EXTSM)

This is a subroutine used by EXTRACT_SOURCE. It is the procedure which does the actual extract.

```
     decks, deck, d : array of string or key all
     interlock, i : boolean
     selection_criteria, sc : file
     list, l : file = $local.log_list
     development_base, db : file
     product_name, pn : name
     build_level, bl : name or key none
     feature_catalog, fc : file or key none
     working_catalog, wc : file or key none
     status : var of status
```

decks | deck | d : This is the list of decks to extract.

interlock | i : This controls whether the interlocks are set during the extract process or not.

selection_criteria | sc : This is an scu selection criteria file which is used to select additional features when extracting without interlocks.

list | l : This is the file which receives the log data from the extract; and which EXTRACT_SOURCE will append to the logging file in the appropriate product catalog.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

4.2.29 EXTRACT_SUBSET_SOURCE_LIBRARY (EXTSSL)

This is a subroutine whose purpose is to extract a selected subset from each of the integration, feature and working libraries and then combine the pieces to form a new library.

```
extract_subset_source_library
     selection_criteria, sc : file
     result, r : file
     development_base, db : file
```

```
          product_name, pn : name
          build_level, bl : name
          feature_catalog, fc : file or key none
          working_catalog, wc : file or key none
          status : var of status

     selection_criteria | sc : This is the file  which  selects
          the contents of the new library.

     result |  r : This is the file to receive the new library.

     development_base | db : specifies the catalog in which all
          products and build levels reside.

          Default if omitted : .INTVE

     product_name | pn : specifies the system or product to  be
          used.

     build_level  |  bl : specifies the system or product build
          level to be used.

     feature_catalog | fc : specifies the feature catalog to be
          used (if any).

     working_catalog | wc : specifies the working catalog to be
          used.

     status : see NOS/VE error handling.
```

4.2.30 EXTRACT_VE_COMMON_DECKS (EXTVCD)

```
     The purpose of this procedure is to extract a canned  list
     of  OS common deck needed by the AV product; at a selected
     build level.

     extract_ve_common_decks
          result, r : file = $local.ve_common_decks
          development_base, db : file
          product_name, pn : name
          build_level, bl : name or key none
          status : var of status

     result  |  r  :  This is the file to receive the extracted
```

subset source library.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

status : see NOS/VE error handling.

4.2.31 FORMAT_HEADER_DECK (FORHD)

```
*****************************
DESCRIPTION NEEDED <<<<-------
*****************************
format_header_deck
     status : var of status = $optional
```

status : See NOS/VE error handling.

4.2.32 GENERATE_BINARIES (GENERATE_BINARY GENB)

The purpose of this procedure is to insulate COMPILE_SOURCE from any fatal compiler errors or compiler non-existance.

```
generate_binaries
     processor, p : string
     input, i : file
     binary, b : file = binary
     list, l : file = $list
     list_options, lo : list of name
     status : var of status
```

processor | p : The compiler name.

input | i : The file of source to be compiled.

binary | b : The location to place the result of the compile.

list | l : The file to receive the compile result list.

list_options | lo : This parameter selects which items the compiler is to report.

status : see NOS/VE error handling.

4.2.33 GENERATE_CDCNET_PRODUCT_TAPE (GENCPT)

This procedure is used to generate a CDCNET product tape. The procedure can pro from either the NOS/VE integration catalog or the CDCNET integration catalog. N Version_catalog should be equal the catalog path up to the version catalog.

```
generate_cdcnet_product_tape
     version_catalog, vc : file = $optional
     installation_catalog, ic : file = $user.installation_catalog
     save_installation_catalog, ..
     sic : boolean = false
     external_vsn, evsn : string 1..6 = $optional
     type, t : key mt9$800, mt9$1600, mt9$6250 = mt9$6250
     build_level, bl : name = $optional
     status : var of status = $optional
```

version_catalog | vc : This is the catalog which contains the CDCNET build.

installation_catalog | ic : This is the NOS/VE installation catalog to create.

save_installation_catalog | sic : This controls whither the installation catalog created will be saved or deleted.

external_vsn | ev : This is the external vsn for the new copy of the tape.

4.0 SUBROUTINES
4.2.33 GENERATE_CDCNET_PRODUCT_TAPE (GENCPT)

type | t : Tape density.

build_level | bl : specifies the system or product build level to be used.

status : See NOS/VE error handling.

4.2.34 GENERATE_DELETE_MODULE_COMMANDS (GENDMC)

This procedure creates the commands to add the group attribute 'DELETED_DECKS' to decks deleted in a build.

```
generate_delete_module_commands
     build_requests, br : file = $required
     command_file, cf : file = $output
     status : var of status = $optional
```

build_requests | br : This is the build requests for all of the new features added to a build.

command_file | cf : This is the file of commands to add the 'DELETED_DECKS' attributes. This file is appened to the product's SPECIAL_REQUESTS file.

status : See NOS/VE error handling.

4.2.35 GENERATE_FEATURE_CRITERIA (GENFC)

The purpose of this procedure is to generate include_modified_deck criteria directives for the features on the include_feature_list file. The format of the include_feature_list file is:

```
include_feature feature_name_1
include_feature feature_name_2
etc.
generate_feature_criteria
     include_feature_list, ifl : file = $required
     criteria_commands, cc : file = $required
     include_copying_decks, icd: boolean = true
     status : var of status = $optional
```

include_feature_list | ifl : This is the list of new features.

criteria_commands | cc : This is the file of 'INCLUDE_MODIFIED_DECKS' commands.

include_copying_decks | icd : This controls whither the ICD parameter of the INCLUDE_MODIFIED_DECKS is true or false.

status : See NOS/VE error handling.

4.2.36 GENERATE_HELP_MODULE (GENHM)

This is a language processor subroutine for COMPILE_SOURCE. It's purpose is to feed help screens into an object library.

```
generate_help_module
     input, i : file = $required
     binary, b : file = $local.lgo
     error, e : file = $errors
     list, l : file = $null
     list_options, list_option, ..
     lo : list of name = $optional
     status : var of status = $optional
```

input | i : This file contains the source text.

binary | b : This is the file where the result object library is to end up.

list | l : This is the file to put the listing onto. If compile_source's save_listing parameter is not none, the listing is massaged prior to output.

list_options | lo : This parameter exists only to provide compile_source with a uniform interface. It is ignored by this procedure.

status : See NOS/VE error handling.

4.2.37 GENERATE_INTEGRATION_PROCEDURES (GENIP)

The purpose of this procedure is to regenerate the library wev$development_base.COMMAND_LIBRARY and the files wev$development_base.BACKUP a wev$development_base.RESTORE

```
generate_integration_procedures
     development_base, db : file = .intve
     status : var of status = $optional
```

development_base | db : specifies the catalog in which all products and build levels reside.
Default if omitted: .INTVE

status : See NOS/VE error handling.

4.2.38 GET_170_JOB_RESULTS (GET1JR)

This procedure is a subroutine whose purpose is to retrieve and interpret a file created by a 170 job which contains the creation command for a 180 status variable that will contain the status of the 170 job. The procedure will also retrieve the 170 job's dayfile.

```
get_170_job_results
     results_file_170, rf7 : name 1..7
     results_file_180, rf8 : file = $local.nos_results
     dayfile_180, d1 : file = $local.dayfile_170
     status : var of status
```

results_file_170 | rf7 : This is the name of the 170 file containing the status variable.

results_file_180 | rf8 : This is the 180 file to retrieve the 170 file to.

dayfile_180 | d1 : This is the file to retrieve the job's dayfile to.

Default if omitted : DAYFILE_170.

status : see NOS/VE error handling.

4.2.39 GET_BUILD_LEVEL (GETBL)

The purpose of this procedure is to retrieve a default value for a product's build level. This default value is stored in the version field of the product source_library.

```
get_build_level
     source_library, sl : file
     build_level, bl : string
     status : var of status
```

source_library | sl : The library to reference for the version.

build_level | bl : specifies the system or product build level to be returned.

status : see NOS/VE error handling.

4.2.40 GET_NEXT_INTVE_PRODUCT (GETNIP)

The purpose of this procedure is to return, one-by-one, the names of the product level catalogs in a master catalog. A subcatalog is considered a product only if it contains both a source_library and latest_changes - accessable to the caller in read mode.

```
get_next_intve_product
     development_base, db : file
     development_base_contents, dbc : string
     product_value : var of string
     status : var of status
```

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

development_base_contents | dbc : The name of a file containing the DISPLAY_CATALOG of the development_base level catalog. This file must not be rewound before calls to GETNIP.

product_value | pv : The variable to receive the next product name.

status : see NOS/VE error handling.

4.2.41 GET_TAPE_VSN (GETTV)

This procedure retrieves tape vsns from the tape library for tapes reserved with the reserve_build_tape procedure.

```
get_tape_vsn
     group, g : name
     tape, t : key s051cip, s052cip, s1cip, s2cip, s3cip,
     s4cip, s5cip, nos_deadstart, ve_deadstart,
     full_product, short_product, test, backup_build_1,
     backup_build_2, backup_build_3, backup_build_4,
     backup_build_5, ic_backup_1, ic_backup_2, nos_base
     volume_serial_number, vsn : var of string
     status : var of status
```

group | g : This is the group classification to assign to the tape. Current standard is the build_level.

tape | t : This is the tape useage classification to assign to the tape.

volume_serial_number | vsn : This i the variable to receive the assigned tape's vsn.

status : see NOS/VE error handling.

4.2.42 INSTALL_TESTS (INST)

This procedure moves files from the wev$development_base catalog to the TESTVE c then backs up the TESTVE master

catalog to tape. The backing up of files is done if the tape_vsn parameter is specified, and files are moved to TESTVE only if their corresponding parameter was specified; i.e., if parameter os_build_leve was specified, those test files in the wev$development_base.OS subcatalog are mo TESTVE prior to the backup to tape. This allows the TESTVE catalog to be updated with any particular product's test files. This procedure SHOULD NOT BE EXECUTED when there is either a load on NOS/VE (prime time) or when the TESTVE catalog is being used due to the use of tape for backup and the potential impact to users of TESTVE files.

BE AWARE: Parameter TTBL when invoked, creates a high cycle of the file .TESTVE.TEST_TOOLS.BOUND_PRODUCT. However, the low cycle cannot be deleted by this procedure's batch job, since this file lives in RING 7, and the batch job i permitted only to delete in RING 11. Interactively you can reach RING 7 (Using T commands). This is most advisable since each invoking of the TTBL parameter fill considerably more & more memory disk space. Also, in RING 7 you will have to iss a CHAFA .TESTVE.TEST_TOOLS.BOUND_PRODUCT RA=(7,13,13) command to assist correct WRITE/READ/EXECUTE ring level permits to this file.

```
install_tests
     os_build_level, obl : name = $optional
     scu_build_level, sbl : name = $optional
     cybil_build_level, cbl : name = $optional
     ocu_build_level, ocbl : name = $optional
     system_test_build_level, ..
     stbl : name = $optional
     test_tool_build_level, ..
     ttbl : name = $optional
     tape_vsn, tv : string 1..6 = $optional
     development_base, db : file = .intve
     status : var of status = $optional
```

os_build_level | obl : The build catalog to get the os test set from.

scu_build_level | sbl : The build catalog to get the scu test set from.

cybil_build_level | cbl : The build catalog to get the cybil test set from.

ocu_build_level | ocbl : The build catalog to get the ocu test set from.

system_test_build_level | stbl : The build catalog to get the system_tests test set from.

test_tool_build_level | ttbl : The build catalog to get the test_tools test set from.

tape_vsn | tv : The tape to write the tests onto.

development_base | db : specifies the catalog in which all products and build levels reside.
Default if omitted: .INTVE

status : See NOS/VE error handling.

4.2.43 LIST_INTERLOCKS (LISI)

The purpose of this procedure is to list all decks with interlocks, and the values of the interlocks. It expects to be called from inside of a scu source library.

```
list_interlocks
     list_file : file = $OUTPUT
     status : var of status
```

list_file : This is the file to display to information to.

status : see NOS/VE error handling.

4.2.44 LIST_VALID_PROCESSORS (LISVP)

The purpose of this procedure is to return the list of all of the language types accepted as "valid" by transmit_to_integration.

```
list_valid_processors
     status : var of status = $optional
```

status : See NOS/VE error handling.

4.2.45 MAKE_LOG_ENTRY (MAKLE)

The purpose of this procedure is to log extracted and transmitted decks. It will append the contents of log_file to the file LOGGING that is contained under each product catalog.

```
make_log_entry
     log_file, lf : file
     development_base, db : file
     product_name, pn : name
     status : var of status
```

log_file | lf : This is the file of information to save.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

status : see NOS/VE error handling.

4.2.46 MERGE_OBJECT_LIBRARIES (MEROL)

The purpose of this procedure is to merge object libraries from the integration, feature and working catalog levels to produce a aggregate library. This procedure expects that the working envionment variables be declared externally.

```
merge_object_libraries
     delete_modules, dm : file
     object_library, ol : name
     result_object_library, rol : file
     omit_library, oml : name or key none
```

4.0 SUBROUTINES
4.2.46 MERGE_OBJECT_LIBRARIES (MEROL)

no_libraries_merged, nlm : var of boolean
status : var of status

delete_modules | dm : This is a list of modules to delete after all of the libraries have been merged.

object_library | ol : This is the name of the file to merge.

result_object_library | rol : This is the aggregate library.

omit_library | ol : The allows one of the libraries to be skipped.

no_libraries_merged | This variable is set true if there are no feature or working catalog versions of the object library; otherwise it is false.

status : see NOS/VE error handling.

4.2.47 MODIFY_TEST_LIST_GROUPS_DECK (MODTLGD)

This proc adds an entry to a table in the deck EVI$TEST_LIST_GROUPS so that a new list will become part of the .TESTVE testname lists. These lists are created by CREATE_TESTNAMES_LISTS which uses the table to determine:
(1) the file paths of the lists to be created
(2) the scu groups that are associated with each list
(3) the product that the list belongs to The intent of this procedure is to add a new list entry to the table, or by setting NEW_LIST to false, add an entry which will cause tests to be appended to the end of an existing list in the table. If you wish to to modfiy an existing entry you must do it manually.

In order to use this proc you must first perform the following:
(1) extract deck EVI$TEST_LIST_GROUPS from the PROCS product library
with interlocks using EXTS
(2) assign a modification using ASSM
(3) enter your working environment (**this proc can not be called while

outside the working environment**)

After these steps are completed you can call this proc to modify the deck.

```
modify_test_list_groups_deck
     modification, m : name = $optional
     product_name, pn : key os, ps_quicklooks, ocu, scu,
     system_tests, av =
     test_list_file_path, tlfp : string = $optional
     groups, group, g : list 1..3 of name = $optional
     new_list, nl : boolean = TRUE
     help, h : file = $null
     status : var of status = $optional
```

modification | m : The modification which was created using ASSM.

product_name | pn : Product that the list will belong to.

test_list_file_path | tlfp : The partial file path of the list that that will be created.

groups | group | g : Any test decks having these will be associated with the list.

new_list | nl : If FALSE then the file given for TEST_LIST_FILE_PATH refers to one that is already in the table. The intent is that tests with the specified groups will be appended to that list. This allows tests with different groupnames to all be associated with one list.

help | h : This will produce some online help information.

status : See NOS/VE error handling.

4.2.48 OMIT_LIBRARY (OMIL)

The purpose of this procedure is to omit the specified library references from the library that is being created. NOTE : Must be in the create_object_library utility to call this procedure, all load modules in the current library will have the specified library reference

omitted.

```
omit_library
     library_to_omit, lto : name = cyf$run_time_library
     status : var of status
```

library_to_omit | lto : This is the library reference to remove.

status : see NOS/VE error handling.

4.2.49 PARSE_CATALOG (PARC)

This procedure will separate the files and subcatalogs that exist within the specified catalog. A list of the subcatalogs are written to the file specified for the catalog_list parameter and a list of the files are written to the file specified for the file_list parameter. The number_of_catalogs and number_of_files parameters return the number of elements in the respective lists.

```
parse_catalog
     catalog, c : file = $catalog
     catalog_list, cl : file = $output
     file_list, fl : file = $output
     number_of_catalogs, noc : var of integer = $optional
     number_of_files, nof : var of integer = $optional
     status : var of status = $optional
```

catalog | c : The catalog to parse.

catalog_list | cl : The file to receive the list of subcatalogs.

file_list | fl : The file to receive the list of files.

number_of_catalogs | noc : The number of entries in CATALOG_LIST.

number_of_files | nof : The number of entries in FILE_LIST.

status : See NOS/VE error handling.

4.2.50 PRINT_LASER_FILE (PRILF)

The purpose of this procedure is to print a file on the laser printer; using 8 1/2 x 11 paper. This procedure will generate formatting directives for the file.

```
print_laser_file
     file, f : file
     format, fmt : key of, document, d, listing, l :
     listing
     number_of_sides, nos : integer 1..2 = 2
     pages_per_side, pps : integer 1..2 = 1
     copies, c : integer 1..32 = 1
     shift_left_margin, slm : boolean
     status : var of status
```

file | f : The file to print.

format | fmt : This controls whether the file is printed with the lines horizontal normal mode (DOCUMENT) or turned sideways so that the lines run vertically from bottom to top (LISTING).

number_of_sides | nos : This controls single or double sided printing.

pages_per_side | pps : This controls the number of printed pages that are to be put onto each side of each piece of paper.

copes | c : The number of copies to make.

shift_left_margin | slm : This shifts the page slightly towards the right siade of the page to facilitate binding the document.

status : see NOS/VE error handling.

4.2.51 PROCESS_TEST_DECKS (PROTD)

```
******************************
DESCRIPTION NEEDED <<<<-------
******************************
```

4.0 SUBROUTINES
4.2.51 PROCESS_TEST_DECKS (PROTD)

```
process_test_decks
     deck_list, dl : file = $required
     scu_base, sb : file = $required
     product_name, pn : name = os
     catalog, c : string = $required
     status : var of status = $optional

-->> PARAMETER DESCRIPTIONS NEEDED <<--
product_name | pn : specifies the system or product to  be
     used.

status : See NOS/VE error handling.
```

4.2.52 QUERY_DEBUG_TABLE (QUEDT)

This procedure invokes the utility which will produce the address, the segment, display the address and other attributes for each section in the module. It ca a procedure name and output the address, and the module name from the system_deb

IF a file is specified for the debug_table parameter this proc will use that bin If the keyword SYSTEM is specified, it will search the working_catalog, feature_ and build_level catalog for a file called: system_debug_table . If the keyword R is specified, the NOS/VE system currently running on the computer is searched fo

```
query_debug_table
     debug_table,   dt   :   file   or   key   system,   s,
     running_system, rs, boot, b
     input, i : file = $input
     output, o : file = $output
     build_level, bl : name = $optional
     feature_catalog, fc : file or key none = none
     feature_build_level, fbl : name = object
     working_catalog, wc : file or key none = none
     working_build_level, wbl : name = object
     status : var of status = $optional

debug_table | dt : This selects the reference source  that
     QUEDT  will use to look up references.  System is the
     one   from   the   working   environment    settings;
     running_system  is the one in the currently exicuting
```

system and boot is the boot debug table from the working environment.

input | i : This file to take the queries from.

output | o : This is the file to write the response data to.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

4.2.53 REPLACE_170_FILE (REP170F)

The purpose of this procedure is to replace one of the 170 files NVELIB, NVEBINS, NOSBINS or RH170 to the 170 side using REPLACE_MULTI_RECORD_FILE. The file must come from the appropriate position in the working environment structure.

```
replace_170_file
     file_name, fn : name = $required
     development_base, db : file = .intve
     product_name, pn : name = os
     build_level, bl : name = $optional
     feature_catalog, fc : file or key none = none
     feature_build_level, fbl : name = object
     working_catalog, wc : file or key none = none
     working_build_level, wbl : name = object
     status : var of status = $optional
```

4.0 SUBROUTINES
4.2.53 REPLACE_170_FILE (REP170F)

file_name | fn : The file to replace.

development_base | db : specifies the catalog in which all products and build levels reside.
Default if omitted: .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

4.2.54 RETURN_ENVIRONMENT_FILE_PATH (RETEFP)

Procedure to search working_catalog, feature_catalog and build_level_catalog and for environmental files, then return the path as a string if found.

```
return_environment_file_path
     file_name, fn : name = $required
     file_type, ft : key cip, compiled, linked, general,
     pp, z80 = gener
     path, p : var of string = $optional
     development_base : file = .intve
     product_name : name = os
     build_level : name
     feature_catalog : file or key none = none
     feature_build_level : name = object
     working_catalog : file or key none = none
     working_build_level : name = object
```

status : var of status = $optional

file_name | fn : The file name to search for.

file_type | ft : The place to search.
cip : The CIP subcatalog
compiled : The MAINTENANCE subcatalog.
linked : The build level subcatalog.
genereral : Either the build level or MAINTENANCE level subcatalog.
pp : The OSF$PP subcatalog.
z80 : The OSF$Z80 subcatalog.
*block,i0,h5

path | p : This is returned path to the appropriate catalog containing the file.

development_base | db : specifies the catalog in which all products and build levels reside.

Default if omitted : .INTVE

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

feature_catalog | fc : specifies the feature catalog to be used (if any).

feature_build_level | fbl : specifies the feature build level to be used.

working_catalog | wc : specifies the working catalog to be used.

working_build_level | wbl : specifies the working build level to be used.

status : See NOS/VE error handling.

4.2.55 RETURN_FILE_LENGTH (RETFL)

The purpose of this procedure is to count the number of lines on a text file.

```
return_file_length
     file, f : file = $required
     length, l : var of integer = $required
     status : var of status = $optional
```

file : The file whose length is to be returned.

length : The variable to which the line count is returned.

status : See NOS/VE error handling.

4.2.56 RUN_LINK_JOB (RUNLJ)

The purpose of this procedure is to execute a NOS/170 batch job from the 180 side.

```
run_link_job
     file, f : file
     wait, w : boolean
     recursive_call, rc : boolean
     status : var of status
```

file | f : This contains the NOS batch job.

wait | w : This controls whether the 180 and 170 jobs are to be run syncronously or asynchronously.

recursive_call | rc : This parameter is not for external use.

status : see NOS/VE error handling.

4.2.57 SAVE_BUILD_JOB_STATUS (SAVBJS)

This is a procedure used in jobs created by MAKE_BUILD_JOBS. It reports the job's status to a file that is interpreted by BUILD_EXEC.

```
save_build_job_status
     status_file, sf : file
     job_name, jn : name
     job_status, js : name
     product_name, pn : name
     build_level, bl : name
     status : var of status
```

status_file | sf : This is the file to report to.

job_name | jn : This is the name of the current job. (MAKBJ assigns each job a name equal to the object library it is compiling to.)

job_status | js : This is the status value to report.

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

status : see NOS/VE error handling.

4.2.58 SELECT_DECKS_BY_MODIFICATION

This is a subroutine of RE_EXTRACT_SOURCE. It's purpose is to determine which decks are affected by a list of modifications.

```
select_decks_by_modification
     modifications, m : array of string
     modification_index : integer
     selected_decks, sd : file
     selected_deck_count : var of integer
     status : var of status
```

4.0 SUBROUTINES
4.2.58 SELECT_DECKS_BY_MODIFICATION

modifications | m : This is the list of modifications for which we want modified decks.

modification_index | mi : This is the starting index in the list of modifications.

selected_decks | sd : This is the file to append the newly selected decks to.

selected_deck_count | sdc : This is the number of decks added to the selected_decks file.

status : see NOS/VE error handling.

4.2.59 SELECT_MODS_BY_FEATURE

This procedure insures that all modifications in a list have a given feature name.

```
select_mods_by_feature
     modifications : array of string
     features : array of string
     selected_modifications : file
     selected_modification_count : var of integer
     status : var of status
```

modifications | m : This is the list of modifications for which we want to check states.

feature | f : This is the feature under which all selected modifications must exist.

selected_modifications | sm : This is the file to append the newly selected modificaitons to.

selected_modification_count | smc : This is the number of modifications added to the selected_modifications file.

status : see NOS/VE error handling.

4.2.60 SELECT_MODS_BY_STATE (SELMBS)

This procedure insures that all modifications in a list have a given state.

```
select_mods_by_state
     modifications, m : array of string
     state : integer 0..1
     selected_modifications, sm : file
     selected_modification_count : var of integer
     status : var of status
```

modifications | m : This is the list of modifications for which we want to check states.

state | s : This is the state in which all selected modifications must exist.

selected_modifications | sm : This is the file to append the newly selected modificaitons to.

selected_modification_count | smc : This is the number of modifications added to the selected_modifications file.

status : see NOS/VE error handling.

4.2.61 SELECT_UNIQUE_NAMES (SELUN)

This is a subroutine of RE_EXTRACT_SOURCE. It's purpose is to add to a file of names any names not already there.

```
select_unique_names
     new_names : array of string
     new_names_ub : integer
     selected_names : file
     selected_names_count : var of integer
     status : var of status
```

new_names : This is the list of candidate names to add to the file. for which we want modified decks.

new_names_ub : This is the upper bound of the new_names array.

selected_names : This is the file to append the newly selected names to.

selected_names_count : This is the number of names added to the selected_names file.

status : see NOS/VE error handling.

4.2.62 STANDARDIZE_LIBRARIES (STAL)

The purpose of this procedure is to massage the object libraries in a given product to put the modules in the right order and delete unsed modules. For os libraries external modules will be added and the references to the cybil run time library wil be omitted.

```
standardize_libraries
     library_and_format,  laf : list 1..32, 1..2 of name =
     $optional
     delete_modules, dm : file = $null
     product_name, pn : name = $optional
     build_level, bl : name = $optional
     status : var of status = $optional
```

library_and_format | laf : The list of files to standardize.

delete_modules | dm : A list of modules to remove.

product_name | pn : specifies the system or product to be used.

build_level | bl : specifies the system or product build level to be used.

status : See NOS/VE error handling.

4.2.63 STANDARDIZE_NAME_LIST (STANL)

This procedure standardize list of names by:
1) convert each line to a name and translate
2) eliminating leading blanks
3) sorting
4) eliminating duplicate entries

```
standardize_name_list
     input, i : file = $required
     output, o : file = $optional
     translate,    t    :    key    upper_to_lower,   utl,
     lower_to_upper, ltu = uppe
     status : var of status = $optional
```

input | i : This file contains names to standardize.

output | o : This is the file to receive the standardized list.

Default if omitted : input

translate | t : This controls the character conversion.

status : See NOS/VE error handling.

4.2.64 TRANSMIT_SOURCE (TRAS)

This is the procedure which performs the transmit operations done by both TRANSMIT_TO_INTEGRATION and TRANSMIT_TO_FEATURE_CATALOG.

```
transmit_source
     from_base, fb : string
     to_base, tb : string
     selection_criteria : file
     last_to_base : string
     verify_states : file
     change_states, change_state, cs : boolean
     status : var of status
```

from_base | fb : This is the library to transmit from.

4.0 SUBROUTINES
4.2.64 TRANSMIT_SOURCE (TRAS)

to_base | tb : This is the library to transmit to.

selection_criteria : This is the information to transmit.

last_to_base : This is used to pass the product_name source_library to TRAS from TRATI (to_base= LATEST_CHANGES).

verify_states : This will cause the procedure to check that states match between the two libraries (on the mods transmitted).

change_states | change_state | cs : This will cause the procedure to change all transmitted state 0 mods to state 1.

status : see NOS/VE error handling.

4.2.65 UNBUNDLE_LISTINGS (UNBUNDLE_LISTING UNBL)

This is a subroutine of COMPILE_SOURCE. It will take a compiler's listings, seperate them by module, create a deck on a listing library for each module's listings and place group attributes on the deck giving the compiler's name and the severity of errors on the module listing.

```
unbundle_listings
     list_file, lf : file
     processor, p : name
     list_library, ll : name : listing_library
     save_listings, sl : key all, good, fatal, warning,
     none = all
     working_catalog, wc : file or key none
     status : var of status
```

list_file | lf : This is the file containing the compiler listings.

processor | p : The compiler name.

list_library | ll : This is the scu source library to place the listings upon.

save_listings | sl : This selects the severity of the

listings to save. (Note : saving warning listings saves fatals as well.)

working_catalog | wc : specifies the working catalog to be used.

status : see NOS/VE error handling.

4.2.66 VALIDATE_DECK (VALD)

The purpose of this procedure is to insure that all required fields in the deck header contain valid values.

```
validate_deck
     deck, d : name = $optional
     output, o : file = $output
     status : var of status = $optional
```

deck | d : The deck to validate.

output | o : The file to write errors onto.

status : See NOS/VE error handling.

A1.0 GLOSSARY OF TERMS

A1.0 GLOSSARY OF TERMS

BACKWARD CHAINING - A method of building characteristic of secondary builds. The *copys in each build deck point to it's predecessor. Build decks are also characterized by the use of include_feature commands pulling in unintegrated code.

BIG 3 - A set of regression tests which include the OS quicklooks, product set quicklooks and performance quicklooks.

CBL - Commercial Benchmark Library. A set of benchmark tests designed to simulate business enviornments. This set is characterized by an emphasis on COBOL and by relatively small amounts of arithmatic computation.

DCT - The Development Coordinating Team. A group responsible for planning the contents of each build and each release.

FORWARD CHAINING - A method of building characterists of primary builds. The *copys in each build deck pint to it's successor. Build decks are also characterized by the user of exclude_feature commands excluding integrated code.

IBL - Interactive Benchmark Library. A set of benchmark tests designed to test interactive response times.

MASTER MACHINE - The machine upon which integration does it's work and upon which the master portion of the update jobs are executed.

OS QUICKLOOKS - A set of regression tests that are designed to quickly spot major flaws in a new level of OS.

PERFORMANCE QUICKLOOKS - A set of benchmark tests that are designed to quickly spot major degradations in response to a new level of the system.

A1.0 GLOSSARY OF TERMS

PRIMARY BUILDS - System builds done by integration which are part of the system release path.

PRODUCT SET QUICKLOOKS - A set of regression tests that are designed to quickly spot major flaws in a new level of the Sunnyvale product set or in the product set's interface with the system.

SBL - Scientific Benchmark Library. A set of benchmark tests designed to simulate engineering environments. This set is characterized by an emphasis on FORTRAN and by relatively large amounts of arithmatic computation.

SECONDARY BUILDS - Systems builds done by integration which are not part of the system release path. These builds are, typically, done to aid a particular group. These builds are usually discarded once the need for them is ended, instead of being used as the basis for the next build level.

SLAVE MACHINE - Any machine used by the NOS/VE department closed shop that is not a master machine.

SN_MID_CORRELATION - A file used by the update jobs to determine which machine is the master machine and which are slave machines. It also contains information about the method of communicating between the machines.

SPECIAL_REQUESTS - A file which exists in each product catalog on the master machine. This file forms a repository for all library change directives that will be executed on the particular product during the next execution of the update job.

TYING - A file which exists in each OS build level subcatalog. The purpose of this file is to link (or tie) the OS build level with the correct level of each of the other available products. It also will tie the OS level to the 170 catalogs containing the NOS and NOS/BE levels and to the set of tapes which are used to test the system.

UPDATE JOBS - A set of procedures designed to be run as batch jobs during off shift time. The purpose

A1.0 GLOSSARY OF TERMS

of the updat jobs is to maintain the source libraries for all products across all machines. The update jobs also provide a mechanism for introducing and distributing the changes integration makes to the libraries during builds and other operations.

WEF$FEATURE_LIST - A file which exists in a user's feature or working catalogs. It is used by many working enviornment procedures to override the build_level value a user has selected, adding the code that the user has developed back into the included set.

A2.0 REPRESENTATION OF MULTIPLE BUILD LEVELS

A2.0 REPRESENTATION OF MULTIPLE BUILD LEVELS

One of the obvious requirements of this environment is the ability to hold more than one version of the operating system, products, etc.

Each build level (version) of the operating system or product will have a name of the form BUILD_xxxxx, where "x" designates the level. Since a name in NOS/VE cannot contain characters such as "/" and ".", such characters should be "mapped to" the "_" character when composing a build level name. Build levels are generally restricted to eleven characters or less because of the need to add descriptive information to the decks which reside on the source libraries, and which control the build contents. It is possible to use names longer than eleven characters, but they may be truncated by procedures which need to utilize the descriptive information.

All source (or text) data for all build levels will be stored in the same SCU library. SCU modifications and features will be used to represent the incremental changes that transform one version of the SCU library into the next.

In addition to source data, there may be certain tools (compiler, linker, etc.) which are unique to one or more build_levels of a product. If such is the case, a tools command library will exist for each such build level of the product. When specified by the tools_library_build_level parameter of the SET_WORKING_ENVIRONMENT command, the tools command library for the build_level will be added to the user's command list.

A2.0 REPRESENTATION OF MULTIPLE BUILD LEVELS
A2.1 BUILD SUBCATALOGS

A2.1 BUILD SUBCATALOGS

There will be a subcatalog for each operating system build. The name of this subcatalog will be that of the build level. This subcatalog will contain the object libraries that together form the operating system and the files that result from linking the operating system.

These files are used both by integration and by developers who are working on additions or changes and need to put together a system for checkout. When they link a system their changes are combined with these full libraries prior to the actual linking taking place.

In addition to the files for the system itself, there is a subcatalog for the feature test list. The files in this catalog are subsequently moved to the TESTVE catalog.

System tests are considered to be a "product" and as such are maintained in their own "product subcatalog". Likewise, test tools are considered to be a "product" and are maintained in their own "product subcatalog".

A2.2 SCU REPRESENTATION OF A BUILD LEVEL

Corresponding to each build level on a SCU library there will be a deck named according to the build level.

The contents of each such deck will specify what was added to the library in the subsequent build. Thus, when build "n" is added to the library, an empty deck called BUILD_n is created. Then when build "n+1" is added, deck BUILD_n is updated to list those things that were included in the new build level preceded by a "*COPY,BUILD_n+1".

The "subsequent build content" is represented by SCU selection criteria commands such as "EXCLUDE_FEATURE,f" where "f" identifies a

"feature" introduced in the next build.

In order to permit easy backward and forward tracing of build levels, the deck_descriptions of the "build decks" will be used as well as the library_description. The deck description for build "n" will have the form :

Predecessor Build = BUILD_n-1 Successor Build = BU

in which the build level names will be left justified, space filled in a 31 character field. This permits a fixed method of access to the predecessor and successor build level names. If there is no predecessor or successor the corresponding field of the deck description should have the value "NONE".

The "version" in the header of the SCU library will have the form :

BUILD_x

providing a pointer to the most recent build.

A2.2.1 REPRESENTATION OF A DELETED DECK

Because more than one build level is maintained on the source library, it is not feasible to actually delete decks from the library that become obsolete at a certain build, since such decks are still needed by previous builds.

To delete a deck, the developer should generate a modification which deletes all lines from the deck, leaving 0 active lines on the deck. This modification should be transmitted with the appropriate increment. The group DELETED_DECKS is added to the deck when the feature deleting the deck is incorporated into a system by integration. The DELETED_DECKS group is used for informational purposes. When the source library is resequenced, if the modification is at state 4, and the group attributes equal DELETED_DECK, the deck will be deleted from the source library.

A2.2.2 REPRESENTATION OF A NEW DECK

When create_source creates a new deck, it has the "NEW_DECKS" group assigned to it. This prevents a new deck from being included in a system until it is added to a build.

Once the deck has been added to a build level, it is deleted from the group NEW_DECKS.

To prevent a deck added at a certain build level from becoming part of all preceding builds, an EXCLUDE_DECK directive is added to the appropriate "build deck" in a manner analogous to that described for deleting a deck in the preceding section.

A2.3 PRODUCT SUBCATALOGS

Each product will have a subcatalog in the main integration catalog. The structure of these subcatalogs will, to as great an extent as possible, parallel the structure of the operating system subcatalog. Specifically this means that the subcatalog will contain a source library and a subcatalog for individual builds of the product.

The source library will contain, besides the source code for the product itself, the tests for the product and procedures to build and maintain the product and its tests.

The product build subcatalogs will parallel the structure of the product subcatalogs located in the $SYSTEM catalog with the addition of a "tests" subcatalog similar in structure and content to the feature tests subcatalog for the operating system.

All non-source (or non-text) data for an individual build level will be stored in a subcatalog named according to the build level.

B1.0 USE OF "GROUPS"

B1.0 USE OF "GROUPS"

The use to be made of SCU groups for the operating system source library is described in a section below. Similar conventions will be followed for each product source library as appropriate.

SCU "groups" will be used on the system source library for a number of different purposes. The different types of groups are identified by the conventions used to name them. Except for the "processor" and "generic" groups (described above), the format of all group names is :

xxy$aaaaaa

where "xx" is the product identifier associated with the group (usually OS), "aaaaaa" is a descriptive name for the particular group, and "y" is one of the following :

S Specifies a "Source" group, i.e. a subdivision of the source library into component libraries. The groups of this type that have been identified are :

oss$program_interface
oss$source
oss$real_state_source
oss$feature_tests
oss$process_commands

F Specifies a "destination File" group (i.e. a file onto which a deck is to be placed after processing). The groups of this type that have been identified are :

osf$job_template_rrr
osf$system_core_rrr
osf$monitor
osf$object_code_utilities
osf$tasks
osf$utilities
osf$commands
osf$operator_commands
osf$non_standard_commands

B1.0 USE OF "GROUPS"

G Specifies a "Group", i.e. a collection of decks that somebody has decided it would be useful to be able to refer to en masse (e.g. clg$command_list_management). Examples of groups which have been established are :

```
LLG$OBJECT_CODE_DEFINITIONS
PMG$ANALYZE_PROGRAM_DYNAMICS
IFG$INTERACTIVE
```

Most group names will follow the conventions described for the operating system source library. However, there are two classes of groups for which this is inappropriate : "processor" groups and "generic" groups.

A "processor" group will be given the same name as the corresponding processor, e.g. the group name for decks to be compiled by CYBIL will be "CYBIL". Other processor group names are :

```
assemble
fortran
cobol
pp_compass
cp_compass
cybil_cc
program_descriptions
scl_procedures
ccl_procedures
```

Note that some of these must obviously be processed on the Nos side of the machine.

A "generic" group is used in those cases where knowing the processor for a deck is insufficient for some purpose. The generic groups that have been identified are :

```
scu_selection_criteria
deleted_decks
build_decks
new_decks
```

Some of these generic groups overlap to some extent with the processor groups.

B1.0 USE OF "GROUPS"
B1.1 USE OF "STATES"

B1.1 USE OF "STATES"

SCU states will be used to indicate the degree to which a modification has been tested.

State 0 - 'Under Development'

Code with state 0 modifications resides in the feature and working source libraries. State 0 is used on an integration source library to designate a modification that has been rejected.

State 1 - 'Testable'

Code and test modifications will be transmitted to the system source library at state 1. State 1 modifications have been proven 'testable', i.e. a system with state 1 code modifications in it is able to deadstart and the tests with state 1 modifications for this code execute. This code is ready to be tested on the developing system.

State 2 - 'Integrated'

Code with state 2 modifications has been formally integrated into the base system, i.e. the base for the developing system, and may be included in the set of binary modules that goes into the release candidate system, depending on the release plan.

State 3 - 'Release Candidate'

Code with state 3 modifications has passed a set of criteria (e.g. stability, performance, documentation criteria) specified in the test direction and is being tested by system evaluation.

State 4 - 'Released'

Code with state 4 modifications has been formally released.

B1.0 USE OF "GROUPS"
B1.1 USE OF "STATES"

There will be no state 0 modifications on the system source library with the possible exception of those that have achieved a higher state and then been rejected for some reason. Having no state 0, i.e. experimental code, on the system source library also reduces contention difficulties.

C1.0 CONVERTING LIBRARIES TO NOS/VE FORMAT

C1.0 CONVERTING LIBRARIES TO NOS/VE FORMAT

The procedure to convert a NOS library to 180 SCU format is fairly simple. Integration maintains files that contain lists of the NOS deck names for NOSVEPL, OSLPI, and DSPIPL and their corresponding 31 character NOSVE names. SCU provides a utility to convert the library and substitute the long (31 character) names for the short (NOS) names so that calls to other decks remain compatible.

Example :

```
getf oldlib dc=b60
convert_madify_to_scu opl=oldlib  ..
 r=$user.new_library cs=ascii612
*(this assumes you are converting a madify libr
   the same files can be used for update librar
scu b=$user.new_library r=$user.long_names_libr
delete_file_connection $ERRORS output
* the following command will generate errors fo
  decks not found on your library - the above
  command avoids seeing all the warnings.
 change_deck_names  ..
  modification=<assigned modification> ..
  nl=<your name change list>
crefc $ERRORS output
end true
create_file_permissions $user.long_names_librar
 group=user am=(all cycle control) sm=none ai='
```

This last statement gives the owner of the file permission to extract with interlocks from the library and change the state of a modification within the library.

C2.0 CONVERTING A BINARY TO NOSVE FORMAT

C2.0 CONVERTING A BINARY TO NOSVE FORMAT

To convert an existing binary file to a format usable on NOS/VE, the user need only utilize Convert_Object_File.

Example :

```
conof to=new_bin from=oldbin
```

C3.0 CONVERTING A TEXT FILE TO NOS/VE FORMAT

C3.0 CONVERTING A TEXT FILE TO NOS/VE FORMAT

Conversion is not needed for text files; they can simply be "gotten" and used.

Example :

```
getf to=text from=nostext
```

D1.0 HOW TO DO BUSINESS ON NOS/VE

D1.0 HOW TO DO BUSINESS ON NOS/VE

Following are some examples of some of the processes a developer may typically go through to get work done.

D1.1 INITIAL PREPARATION

1) Edit or create a user prolog
2) Add to it a Set_Command_List of the command library (.INTVE.COMMAND_LIBRARY).
3) Add to it a Set_Working_Environment specifying the desired parameters. Should specify product name, build level, author and working catalog as a minimum.
4) Add to it the Modification_Index and Modification_Initials variables needed to generate modification names. (Note - the capital letters are necessary for the modification_index - be sure to use this format.)

Example :

```
edif $user.prolog

insert

set_command_list add=.intve.command_library

set_working_environment author='C.A.Sheats'
 bl=build_1_11_18 wc=.cas

create_variable WEV$MODIFICATION_INDEX
 kind=integer scope=job value=1

create_variable wev$modification_initials
 kind=string scope=job value='cas_'

(these create_variable lines should be one line -
 not continued via '..'.  It was necessary to show
 the lines this way for the benefit of Textform).
```

D1.0 HOW TO DO BUSINESS ON NOS/VE
D1.1 INITIAL PREPARATION

```
**

end

detach_file $user.prolog
```

The following examples assume that the above Set_Working_Environment command is in effect.

D1.2 CREATING A NEW DECK

1. Get a modification name (via assign_modification) to use in the creation of the desired deck. Use existing modification name associated with feature if have one.
2. Execute Create_Source to create the deck on the working (and/or feature) library.
3. Call Edit_Source to add text.

Example :

```
assign_modification f=source_maintenance ..
 md='Original source for new_deck'

create_source d=new_deck m=cas_27 e=true ..
 dg=osf$commands p='scl' sg=oss$process_commands

edit_source m=cas_27 d=new_deck
```

If there are no decks on the source_library, assign_modification will not display the modification (due to a bug in SCU). To find out the modification assigned to you, do a display_modification_list of the source_library.

D1.3 MODIFYING AN EXISTING DECK

1. If the deck is not on the working library, execute Extract Source
2. Get a modification name via Assign_Modifications if one does not exist for feature working on.
3. Call Edit_Source to make changes

D1.0 HOW TO DO BUSINESS ON NOS/VE
D1.3 MODIFYING AN EXISTING DECK

Example :

```
extract_source d=old_deck

assign_modification f=NV0Z999 ..
 md='Problems in handling output.'

edit_source m=cas_28 d=old_deck
```

D1.4 COMPILING A DECK

1 Execute Compile_Source

Example 1 :

```
compile_source d=dmm$logger p='cybil d=nt' ..
 ol=osf$system_core_113 l=listing
```

If processor (P) and object library (OL) not specifi
it is figured out by the procedure as in example 2.

Example 2 :

```
compile_source dmm$logger l=listing
```

D1.5 USING ENTER_WORKING_ENVIRONMENT

The Enter_Working_Environment command enables the user to save time and avoid creating unnecessary cycles of the working library.

Example :

```
enter_working_environment

edit_source cas_29 tmm$dispatcher

compile_source tmm$dispatcher l=listing

edit_source cas_29 tmm$dispatcher
```

```
coms tmm$dispatcher
   .
   .
   .
  etc.

exit_working_environment
```

E1.0 WHAT TO DO IF INTERLOCK CONFLICTS

E1.0 WHAT TO DO IF INTERLOCK CONFLICTS

This describes what to do in the case of interlock conflicts between users on separate machines. Interlock conflicts occur when someone on SN002 and someone on SN109 extract the same deck on the same day. When this happens, the update procedures will detect this and the copy of the deck from the machine with the lowest priority is rejected. A deck which is rejected ends up on a file called .INTVE.product.XMIT_REJECTIONS.SDUP_<date>_<time>. There may be more than one deck upon this file depending on the number of rejected decks. There may also be more than one file depending on the number of slave machines each of whose name differs slightly in the date-and-time stamp.

Whenever integration deects an interlock conflict, it is the job of the person responsible for the update jobs to resolve the problem. This resolution ranges from ignoring the rejected mod to running re_extract_source on it or even to the point of asking the originator to recreate it if there are serious conflicts.

F1.0 GUIDELINES FOR SOURCE LIBRARY USAGE

F1.0 GUIDELINES FOR SOURCE LIBRARY USAGE

The procedure of transmitting code to the source library "long" ("long" generally means more than one week) before it is ready to be built into a system causes severe problems for people who need to modify the same deck. They would like to modify the deck without any conflicting code, but are forced to deal with it because of the way SCU operates. This requires building and testing with the conflicting feature - which may or may not be possible.

To avoid these problems, some guidelines are presented here. They are not meant to apply to every case, and they do require some cooperation between the parties involved.

1. If a feature being developed will take a "long" time, then the decks being modified should be extracted without interlocks, and the modifications made should not be transmitted to the source library (see #3 below).

2. If a conflict arises where one person is ready to transmit code and get it built, but cannot do this because of another person's code being present in the source library, then, after discussion with the owner of the conflicting code, the conflicting code may be deleted. Some arrangement should be made to save the deleted modset(s) for future use.

3. The procedure re_extract_source can be used to upgrade decks from one build to another. If conflicts arise during this process, they must be dealt with by hand. It is the feature developer's job to upgrade his mods from system to system while under development.

4. When a feature is ready to be transmitted, all decks should be extracted with interlocks and the accumulated modsets applied (see #3). Conflicts should be dealt with as detailed in #2. Once the feature is transmitted and a build request has been made, no conflict with this code is possible - its place in the system is established with the build request. Any conflicts after this point must adapt

F1.0 GUIDELINES FOR SOURCE LIBRARY USAGE

to this code.

Table of Contents

1.0 INTRODUCTION 1-1

2.0 CATALOG STRUCTURE FOR SOURCE MAINTENANCE 2-1
2.1 MAIN INTEGRATION CATALOG (INTVE) 2-3
2.2 OPERATING SYSTEM SUBCATALOG (OS) 2-6
2.3 SYSTEM SOURCE LIBRARY 2-6
2.3.1 GROUPS ON THE SOURCE LIBRARY 2-7
2.3.2 USE OF "FEATURES" AND "MODIFICATIONS" 2-7
2.3.3 USE OF INTERLOCKS 2-8

3.0 COMMANDS TO USE THE NOS/VE MAINTENANCE ENVIRONMENT . . . 3-1
3.1 ENVIRONMENT COMMANDS 3-2
3.1.1 DISPLAY_WORKING_ENVIRONMENT (DISWE) 3-2
3.1.2 ENTER_WORKING_ENVIRONMENT (ENTWE) 3-3
3.1.3 EXIT_WORKING_ENVIRONMENT (EXIWE) 3-5
3.1.4 SET_WORKING_ENVIRONMENT (SETWE) 3-6
3.2 COMMANDS FOR CODE DEVELOPMENT AND INTEGRATION 3-9
3.2.1 ASSIGN_MODIFICATION(S) (ASSM) 3-9
3.2.2 ASSIGN_AND_RETURN_MODIFICATION (ASSARM) 3-11
3.2.3 BUILD_EXEC (BUIE) 3-12
3.2.4 CHANGE_MODIFICATION_HEADERS (CHAMH) 3-13
3.2.5 CLEAR_INTERLOCK_REQUEST (CLEIR) 3-14
3.2.6 COMPILE_SOURCE (COMS) 3-15
3.2.7 CREATE_SOURCE (CRES) 3-22
3.2.8 CREATE_TESTNAMES_LISTS (CREATE_TESTNAMES_LIST . . . 3-24
3.2.9 DISPLAY_BUILD_INFORMATION (DISBI) 3-25
3.2.10 DISPLAY_FEATURE_BUILD_LEVEL (DISFBL) 3-27
3.2.11 DISPLAY_SYMBOL_TABLE (DISST) 3-27
3.2.12 EDIT_SOURCE (EDIS) 3-29
3.2.13 EXTRACT_SOURCE (EXTS) 3-29
3.2.14 FORMAT_SOURCE (FORS) 3-31
3.2.15 GENERATE_PRODUCT_SET_TAPE (GENPST) 3-32
3.2.16 GET_SOURCE (GETS) 3-34
3.2.17 MAKE_BUILD_JOBS (MAKBJ) 3-38
3.2.18 MAKE_BUILD_REQUEST (MAKBR) 3-41
3.2.19 QUERY_DEBUG_TABLE (QUEDT) 3-44
3.2.20 RE_EXTRACT_SOURCE (REES) 3-45
3.2.21 REPLACE_SOURCE (REPS) 3-47
3.2.22 TRANSMIT_TO_FEATURE_CATALOG (TRATFC) 3-49
3.2.23 TRANSMIT_TO_INTEGRATION (TRATI) 3-50
3.3 PRODUCT SPECIFIC COMMANDS 3-51
3.3.1 BIND_AV (BINA) 3-51
3.3.2 BIND_DE (BIND) 3-53
3.3.3 BIND_HPA (BINH) 3-54
3.3.4 BIND_KERMIT (BINK) 3-55
3.3.5 BIND_MAILVE 3-57
3.3.6 BIND_MALET (BINM) 3-58
3.3.7 BIND_MENU_VE (BINMV) 3-60
3.3.8 BIND_NAMVE (BINN) 3-61

3.3.9 BIND_OCU (BINO) 3-62
3.3.10 BIND_PERFORMANCE_TOOLS (BINPT) 3-64
3.3.11 BIND_PFTF (BINP) 3-65
3.3.12 BIND_SCU (BINS) 3-67
3.3.13 BIND_SVS (BINS) 3-69
3.3.14 BIND_TDU (BINT) 3-70
3.3.15 BIND_TDU_POST_PROCESSOR (BINTPP) 3-72
3.3.16 BIND_TEST_TOOLS (BINTT) 3-73
3.3.17 BUILD_CIP_TAPE (BUICT) 3-74
3.3.18 CREATE_TEST_TAPE (CRETT) 3-76
3.3.19 GENERATE_CIP_COMPONENTS (GENCC) 3-76
3.3.20 GENERATE_DEADSTART_FILE (GENDF) 3-77
3.3.21 GENERATE_NOS_DEADSTART_TAPE (GENNDT) 3-79
3.3.22 LINK_170 3-80
3.3.23 LINK_BOOT (LINB) 3-82
3.3.24 LINK_ENVIRONMENT_INTERFACE (LINEI) 3-83
3.3.25 LINK_OPERATING_SYSTEM (LINOS) 3-84
3.4 EDITOR ORIENTED COMMANDS 3-87
3.4.1 COPY_BOX (COPB CB) 3-87
3.4.2 DELETE_BOX (DELB DB) 3-88
3.4.3 MOVE_BOX (MOVB MB) 3-88
3.5 MISCELLANIOUS USEFUL PROCEDURES 3-88
3.5.1 BACKUP_USER_CATALOG (BACUC) 3-88
3.5.2 CATALOG_MULTI_RECORD_TAPE (CATMRT) 3-89
3.5.3 COMPARE_CATALOG_CONTENTS (COMCC) 3-89
3.5.4 COMPILE_DECKS_IN_ISOLATION (COMDII
COMPILE_DECK_IN_ISOLATION) 3-90
3.5.5 COPY_NOS_TAPE (COPNT) 3-90
3.5.6 COPY_NOS_VE_TAPE (COPNVT) 3-91
3.5.7 COPY_UNLABELLED_TAPES (COPUT COPY_UNLABELLED_TAPE) 3-92
3.5.8 DELETE_JOB (DELJ) 3-92
3.5.9 DISPLAY_CATALOG_CONTENTS (DISCC) 3-93
3.5.10 GET_MULTI_RECORD_FILE (GETMRF) 3-93
3.5.11 GENERATE_SCL_LISTINGS (GENSL
GENERATE_SCL_LISTING) 3-94
3.5.12 REPLACE_MULTI_RECORD_FILE (REPMRF) 3-94
3.5.13 RESTORE_USER_CATALOG (RESUC) 3-95
3.5.14 SEQUENCE_LIBRARY (SEQL) 3-95

4.0 SUBROUTINES 4-1
4.1 LANGUAGE_PROCESSOR_SUBROUTINES 4-1
4.1.1 CCL_PROCEDURES 4-1
4.1.2 CONVERT_TEXT_TO_OBJECT 4-2
4.1.3 CP_COMPASS 4-3
4.1.4 CREATE_FASLAVE (CREFS) 4-3
4.1.5 CREATE_MANUAL (CREM) 4-4
4.1.6 CREATE_INSTALLATION_TABLE (CREIT) 4-5
4.1.7 CYBIL_CC 4-6
4.1.8 CYBIL_OBJECT_CHECKING 4-6
4.1.9 DEFINE_MULTI_TERMINAL (DEFMT) 4-7
4.1.10 DESKTOP_CONFIGURATION (DESCF) 4-8
4.1.11 DESKTOP_CONTEXT (DESC) 4-9
4.1.12 DESKTOP_TOOLBOX (DEST) 4-9
4.1.13 FORTRAN_CC 4-10

4.1.14 GENERATE_COMMAND_TABLE_MODULE (GENCTM) 4-11
4.1.15 GENERATE_MSG_TEMPLATE_MODULE (GENMTM) 4-11
4.1.16 META 4-13
4.1.17 PP_COMPASS 4-13
4.1.18 PROGRAM_DESCRIPTIONS (PROGRAM_DESCRIPTION, ... 4-14
4.1.19 SCL_PROCEDURES (SCL_PROCEDURE, SCLP, SCL) 4-15
4.1.20 SYMPL 4-16
4.1.21 TEXTCODE_TEXTFORM 4-16
4.1.22 TEXT_FILES 4-17
4.1.23 TEXT_170 4-18
4.1.24 Z80_COMPASS 4-18
4.2 MISCELLANIOUS SUBROUTINES 4-19
4.2.1 ACQUIRE_FILE (ACQF) 4-19
4.2.2 APPEND_TO_SPECIAL_REQUESTS (APPTSR) 4-20
4.2.3 BACKUP_USER_CATALOG_TO_TAPE (BACUCT) 4-20
4.2.4 BIND_OS_LIBRARY (BINOL) 4-21
4.2.5 BUILD_ANAD_COMMAND_LIBRARY (BUIACL) 4-22
4.2.6 BUILD_BUILTIN_LIBRARY (BUIBL) 4-22
4.2.7 BUILD_SYSTEM_IDENTIFIERS 4-23
4.2.8 BUILD_TASKS_LIBRARY (BUITL) 4-25
4.2.9 CHANGE_OBJECT_LIBRARY (CHAOL) 4-25
4.2.10 CHANGE_TEST_GROUPS 4-26
4.2.11 CHECK_PARAMS_FOR_TRAT 4-27
4.2.12 CHECK_SPECIFIED_DEPENDENCIES (CHESD) 4-28
4.2.13 CLEANUP_FILE_CYCLES (CLEANUP_FILE_CYCLE CLEFC) . 4-28
4.2.14 COMBINE_SCU_LIBRARIES (COMSL) 4-29
4.2.15 COMPRESS_JOB_LOG (COMJL) 4-30
4.2.16 COMPRESS_LINK_MAP (COMLM) 4-30
4.2.17 COPY_LABELLED_TAPES (COPLT COPY_LABELLED_TAPE) . . 4-32
4.2.18 COPY_MULTI_RECORD_TAPES (COPMRT
COPY_MULTI_RECORD_TAPE) 4-33
4.2.19 COUNT_FEATURE_LINES (COUFL) 4-33
4.2.20 CREATE_EMPTY_FILE (CREEF) 4-34
4.2.21 CREATE_INTEGRATION_DESCRIPTORS (CREID) 4-34
4.2.22 CREATE_MULTIPLE_SUBCATALOGS (CREMS) 4-35
4.2.23 CREATE_OPEN_SHOP_DI_OBJECT (CREOSDO) 4-36
4.2.24 CREATE_SUBTEST_LISTS 4-37
4.2.25 EXAMINE_LINK_MAPS (EXALM) 4-37
4.2.26 DUMP_CATALOG (DUMP_CATALOGS DUMC) 4-39
4.2.27 EXPAND_BUILD_DECKS (EXPAND_BUILD_DECK EXPBD) . . 4-39
4.2.28 EXTRACT_SOURCE_MODULES (EXTRACT_SOURCE_MODULE
EXTSM) 4-40
4.2.29 EXTRACT_SUBSET_SOURCE_LIBRARY (EXTSSL) 4-41
4.2.30 EXTRACT_VE_COMMON_DECKS (EXTVCD) 4-42
4.2.31 FORMAT_HEADER_DECK (FORHD) 4-43
4.2.32 GENERATE_BINARIES (GENERATE_BINARY GENB) 4-43
4.2.33 GENERATE_CDCNET_PRODUCT_TAPE (GENCPT) 4-44
4.2.34 GENERATE_DELETE_MODULE_COMMANDS (GENDMC) 4-45
4.2.35 GENERATE_FEATURE_CRITERIA (GENFC) 4-45
4.2.36 GENERATE_HELP_MODULE (GENHM) 4-46
4.2.37 GENERATE_INTEGRATION_PROCEDURES (GENIP) 4-47
4.2.38 GET_170_JOB_RESULTS (GET1JR) 4-47
4.2.39 GET_BUILD_LEVEL (GETBL) 4-48
4.2.40 GET_NEXT_INTVE_PRODUCT (GETNIP) 4-48

4.2.41 GET_TAPE_VSN (GETTV) 4-49
4.2.42 INSTALL_TESTS (INST) 4-49
4.2.43 LIST_INTERLOCKS (LISI) 4-51
4.2.44 LIST_VALID_PROCESSORS (LISVP) 4-51
4.2.45 MAKE_LOG_ENTRY (MAKLE) 4-52
4.2.46 MERGE_OBJECT_LIBRARIES (MEROL) 4-52
4.2.47 MODIFY_TEST_LIST_GROUPS_DECK (MODTLGD) 4-53
4.2.48 OMIT_LIBRARY (OMIL) 4-54
4.2.49 PARSE_CATALOG (PARC) 4-55
4.2.50 PRINT_LASER_FILE (PRILF) 4-56
4.2.51 PROCESS_TEST_DECKS (PROTD) 4-56
4.2.52 QUERY_DEBUG_TABLE (QUEDT) 4-57
4.2.53 REPLACE_170_FILE (REP170F) 4-58
4.2.54 RETURN_ENVIRONMENT_FILE_PATH (RETEFP) 4-59
4.2.55 RETURN_FILE_LENGTH (RETFL) 4-61
4.2.56 RUN_LINK_JOB (RUNLJ) 4-61
4.2.57 SAVE_BUILD_JOB_STATUS (SAVBJS) 4-62
4.2.58 SELECT_DECKS_BY_MODIFICATION 4-62
4.2.59 SELECT_MODS_BY_FEATURE 4-63
4.2.60 SELECT_MODS_BY_STATE (SELMBS) 4-64
4.2.61 SELECT_UNIQUE_NAMES (SELUN) 4-64
4.2.62 STANDARDIZE_LIBRARIES (STAL) 4-65
4.2.63 STANDARDIZE_NAME_LIST (STANL) 4-66
4.2.64 TRANSMIT_SOURCE (TRAS) 4-66
4.2.65 UNBUNDLE_LISTINGS (UNBUNDLE_LISTING UNBL) 4-67
4.2.66 VALIDATE_DECK (VALD) 4-68

Appendix A A1

A1.0 GLOSSARY OF TERMS A1-1

A2.0 REPRESENTATION OF MULTIPLE BUILD LEVELS A2-1
A2.1 BUILD SUBCATALOGS A2-2
A2.2 SCU REPRESENTATION OF A BUILD LEVEL A2-2
A2.2.1 REPRESENTATION OF A DELETED DECK A2-3
A2.2.2 REPRESENTATION OF A NEW DECK A2-4
A2.3 PRODUCT SUBCATALOGS A2-4

Appendix B B1

B1.0 USE OF "GROUPS" B1-1
B1.1 USE OF "STATES" B1-3

Appendix C C1

C1.0 CONVERTING LIBRARIES TO NOS/VE FORMAT C1-1

C2.0 CONVERTING A BINARY TO NOSVE FORMAT C2-1

C3.0 CONVERTING A TEXT FILE TO NOS/VE FORMAT C3-1

Appendix D D1

D1.0 HOW TO DO BUSINESS ON NOS/VE D1-1
D1.1 INITIAL PREPARATION D1-1
D1.2 CREATING A NEW DECK D1-2
D1.3 MODIFYING AN EXISTING DECK D1-2
D1.4 COMPILING A DECK D1-3
D1.5 USING ENTER_WORKING_ENVIRONMENT D1-3

Appendix E E1

E1.0 WHAT TO DO IF INTERLOCK CONFLICTS E1-1

Appendix F F1

F1.0 GUIDELINES FOR SOURCE LIBRARY USAGE F1-1