User Documentation

For

NOS/VE Integration

Revision Date: 11/13/87

1.0 INTRODUCTION

1.0 INTRODUCTION

In general, doing a full Integration build involves the same procedures and processes as a developer would use to do their own build. The working environment and its procedures are employed with the addition of a few extra tools geared specifically for Integration.

All build catalogs reside in the .INTVE catalog. However, in executing a build, the integrator does not run in the INTVE catalog but in their own catalog. A procedure is run to permit that user to write into the INTVE catalog, thus the user may establish the working catalog as if he/she were logged into INTVE. Build decks are created and transmitted at the beginning of build, so the integrator should set the working environment to the new build level. Features will be raised to state 2, indicating that they have been integrated, during the update job at night. The new features that comprise the new build are added to the WEF$FEATURE_LIST file. This allows Integration to build without updating the source libraries.

2.0 HOW TO DO BUILDS

2.0 HOW TO DO BUILDS

The process for building all products is similar except for the linking and how the product is made part of the system. The compilation is the same for each product and build decks must be created for each product.

A product build starts out with a list of features and build request forms. These are assembled by the DCT and given to integration when all features necessary for the build are in hand. This feature list is the building block of the product build.

There are several files in the product build catalog that are generated from the feature list. The build_request file contains the information from the product build request for the features going into this build. It is a selection criteria file for SCU to generate a subset library of all decks that need to be recompiled for the build. It also contains information concerning new and deleted decks going into this build. The TYING file is an important part of an OS build. It should not be in any other product catalog. The tying file contains information tying the various components (products) of an OS build together. It is important that this file is updated correctly before compiling. Information is contained in this file that is used by various procedures used in the build. WEF$FEATURE_LIST contains SCU include feature directives for all features going into the build.

2.1 BUILDING THE NOS/VE OPERATING SYSTEM

The process of doing a build involves raising new code to state 2 and recompiling any decks affected by this code, resulting in a new level of operating system or product. The tricky part is keeping track of what was raised to state 2 this time, as opposed to what was raised to state 2 the last time a build was done.

All work is done in the integrator's own catalog. Only the first procedure - create_integration_working_environment (creiwe) - needs to be executed while logged into INTVE.

Large compilation jobs should be run at night via batch jobs. Many other aspects of a build can be accomplished with batch jobs or asynchronous tasks, allowing a number of things to be accomplished

2.0 HOW TO DO BUILDS
2.1 BUILDING THE NOS/VE OPERATING SYSTEM

at the same time.

Log ALL problems you encounter, whether they are your own errors or caused by someone or something (machine crash) else. The procedure log_restart should be used to log these problems. This will allow you to go back and review what to avoid in the future, or enable you to answer questions when you may have forgotten the details of a problem, this is VERY important. There are so many details involved in doing a build that there is NO WAY you will remember what all has happened, no matter how good your memory is.

Don't keep extra cycles of files around, unless it is required. Don't keep garbage files around (like AAA or BACK), especially if they contain large listings, backup files or the like. This kind of stuff uses up file space and clutters up the build catalogs.

Let the integration project leader know the status of the build (progress made, problems encountered) before 9:30 a.m. each morning. The status must be reported at the DCT meeting.

2.1.1 STEPS IN DOING AN OS BUILD

For this example we will use build level 15005 and the development base INTVE.

1. Login to INTVE and run Create_Integration_Working Environment. This procedure creates the build catalog and permits the specified user to write into the specified build level catalog in INTVE. It also calls Move_Build_Files. Which moves the previous build level files into the new build level.

2. Logout of INTVE and into your user number.

3. Add to your user prolog the procedure call :

```
SETWE PN=OS WC=.INTVE.OS.BUILD_15005 WBL=CHANGES ..
A='your name' BL=BUILD_15005 DB=.INTVE
```

4. Copy the feature_list from the DCT catalog into the build catalog. Check the feature list against the pile of build_request forms you have been given. They should all be in the feature_list; there should be no extra features in the list.

2.0 HOW TO DO BUILDS
2.1.1 STEPS IN DOING AN OS BUILD

5. Proceed on to BUILD_SYSTEM (BUIS).
   This procedure will:

   a. update the build_request file
   b. update the wef$feature_list file
   c. create, update, and transmit the build decks
   d. run the generate_library_changes command
   e. spin off a task to run make_build_jobs

   This command can be called again to add more features to a build or to pull features from a build. Once the build_decks have been created the parameter creation_call should be set to false.

   Extreme care should be taken when specifying the build level parameters.

   It is important to verify that these special requests generated by generate_library_changes get executed completely and correctly (check with the person responsible for the Update Jobs the next morning). Build decks that are transmitted without the corresponding state changes have no affect.

6. The build jobs should be submitted at the end of the day (after 4 o'clock) unless it is a crisis build that must be done ASAP (usually there are only a few jobs in that case). If possible, it is a good idea to log in later in the evening and check to see if jobs have completed or aborted. The jobs write their job log to a file in the build_job catalog that was setup by MAKBJ and which can be examined interactively. If there is a problem and you can figure it out, correct it and resubmit the job(s). (see the section on PROBLEMS)

7. Cannot proceed beyond this point until all compilations have completed without error.

8. If OSF$NOSBINS, OSF$NVELIB or OSF$NVERELS was recompiled LINK_170 must be run. Specify file MAP_170 in the build catalog for the listing parameter, m=all, fb=true, and be sure to check MAP_170 for errors when the procedure has completed.

9. If LINK_170 has been run it is necessary to remake the NOS deadstart tape (see section on creating NOS tapes).

10. GENERATE_SYSTEM (GENS) should now be run. This procedure calls LINK_OPERATING_SYSTEM and if there are no linker errors, GENERATE_DEADSTART_FILE. You must provide it with a tape vsn (which you have provided to the operators upstairs in the lab)

to which the NOS/VE deadstart file will be written. You must also supply the OSI (operating_system_identifier) and PSI (product_set_identifier). This causes the system header on VE displays and the job log to be correct.

EXAMPLE: osi='15005' psi='3d3'

At release time this must be changed again to the correct release identifier (such as 617).

See section on PROBLEMS if there are linker errors.

When the procedure is complete it is always wise to check the 170 dayfile that is produced to make sure that the tape was written correctly on the NOS side. EDIT the dayfile and search for the LABEL string. This is the NOS tape request. Then list the dayfile until you find the string 'EOI ENCOUNTERED'. This means the copy to tape finished successfully. If there is a lot of weird stuff before you get to 'EOI ENCOUNTERED' it represents recovered tape errors. It could mean a questionable tape or questionable tape drive. As long as you find 'EOI ENCOUNTERED' you are ok; otherwise the tape was not written and you must rerun GENDF.

11. If SCU, OCU or one of the command libraries have been rebuilt, a product set tape must also be created (see appropriate section). With the installability feature, the product set tape has to be generated when a bunch of things change. This should be better defined.

12. Now you are ready to attempt to deadstart the new system and run 'THE BIG 3' or submit it for the operators to run.

13. While you are waiting for this to happen, the portion of BUILD_SYSTEM_2 that completes the build catalog can be run.

14. When the BIG 3 have successfully run and the object libraries are all merged the build report can be generated and distributed. This can be accomplished very simply by executing the GENERATE_BUILD_REPORT procedure. The resultant file should be saved in the build catalog.

15. Cleanup the build catalog after the big 3 have successfully run. All garbage files should be detected at this time and all extra file cycles too. There is a procedure, DELETE_EXTRA_FILE_CYCLES (DELEFC), to do this.

2.1.2 PSUDO BUILDS

The term "Psudo Build" is given to those integration builds which are not intended to be part of the main release sequence. These builds are identified by having the last two digits of the build level be greater than seventy. Build decks for psudo builds are not formed the way that they are for normal builds. Instead of consisting of sets of "EXCLUDE_FEATURE" commands; they consist of sets of "INCLUDE_FEATURE" commands. Also, psudo builds are always tied to a specific mainline build (called the primary build). Third; modifications are not raised to state 3 in psudo builds as they are in normal build (and any NEW_DECKS attributes are not removed either.) Some items of note which have caused problems in psudo builds in the past are:

1. Make sure that you have a set of procedures that match the system you are building (psudo builds are frequently used to test radical changes to the mainline systems).

2. Be careful to differentiate between predecessor_build_level and primary_build_level. For example; if 15582 was built upon 15581 which was built upon 15536, then both 15582 and 15581's primary build level value is 15536; however, 15582's predecessor build level is 15581 and 15581's level is 15536.

3. If you are building alternate (non-OS) products as psudo builds and they are based off of build level values which are not equal to that of the OS build level base; you will need a small portion of the TYING file for each of these levels.

2.1.3 DUAL STATE

Dual state is the name given to the interface between either NOS/VE and NOS or between NOS/VE and NOS/BE. The intefaces between the partner systems both look very similer - most of the differences are internal to the code decks and are not visible to the casual observer. The NOS and NOS/BE interfaces reside on ULIB format libraries with similer names. In this document; we will reference the NOS name like this: NVELIB; and the NOS/BE varient like this: (NVELIBB).

There are eight (nine) major components of the dual state interface. The first component is called DSMDST. This is the sequencing and control procedure for deadstarting VE. It resides on NVELIB (NVELIBB). The second componet is called DSMRUN. This is the sequencing and control procedure used while VE is executing. It

also resides on NVELIB (NVELIBB). The third component of the dual state interface is called DSMTRM. This is the sequencing and control procedure used when terminating a VE system. It also contains the mechanism used to produce dumps from system crashes. It resides on NVELIB (NVELIBB). The fourth component is called DSMDSTG. This is the NOS/VE deadstart tape generator. It also resides on NVELIB (NVELIBB).

The fifth component of the dual state interface is called RHAQEP. This is the interface for queue files. It is the mechanism by which files are routed from VE to NOS(BE) print queues and the mechanism by which batch jobs are submitted to VE from NOS or NOS/BE. It resides on NOSBINS (NBEBINS). The sixth component of dual state is RHMPFP. This is the permanent file interface used by GET_FILE and REPLACE_FILE. It also resides on NOSBINS (NBEBINS). The seventh component is called IIAPAS (also called PASSON). This is the interactive interface between NOS(BE) and VE. It is the interface used to route all terminal IO from VE to terminals connected via NOS NAM (or NOS/BE connect). The final component of dual state is called FASLAVE. This is used to execute NOS(BE) commands from VE - see the commands CREATE_INTERSTATE_CONNECTION, EXECUTE_INTERSTATE_COMMANDS and DELETE_INTERSTATE_CONNECTION on the product FMA for more details. It also resides on NOSBINS.

Each of the eight major components of a dual state system consists of a set of linked modules written in either CYBIL CC, or COMPASS. These modules are compiled to a file called OSF$NVERELS (OSF$NVERELB). After all modules have been compiled; they are linked to create the dual state components mentioned above.

In addition to the major components; there are many minor components, most of which exist on NVELIB (NVELIBB). These components are used to maintain the ULIB libraries; list their contents and do general housekeeping. Included in these minor components are CCL procedures which perform the linking needed to produce the major components from their constituant modules. These minor components are compiled to a file called OSF$NVELIB (OSF$NVELIBB) and moved to NVELIB (NVELIBB) when the major components are linked. (The major components are linked by the 180 procedure LINK_170 which merges integration, feature and working copies of the OSF$xxx files and then calls the CCL link procedures on NVELIB (NVELIBB) to relink the major components. See LINK_170 for more details.)

The procedure followed after the NOS and NOS/BE varients of dual state are linked now diverge. After the NOS version to the files are created; a new NOS deadstart tape must be generated. Given an old version of the NOS deadstart tape; the NVELIB file and the

2.0 HOW TO DO BUILDS
2.1.3 DUAL STATE

CONTENTS (without the ULIB header) of the NOSBINS file must be added using NOS's LIBEDIT. This may be done from the 180 via the procedure GENERATE_NOS_DEADSTART_TAPE.

The NOS/BE files differ somewhat from the NOS varient. For the NOS/BE version; there is a ninth major component called DSMCPYS. This is a specialized copy procedure which is written in FTN5 and which resides on (NVELIBB). After all of the major components have been linked and the minor components merged; a procedure called CREATE_NOS_BE_RELEASE_FILES is executed. This procedure will strip the ULIB header from (NVELIBB), (NOSBINS) and (OSF$NVERELB). It will also remove any CCL procedures from (NVELIBB). It will produce the files NOSBE_NVELIB, NOSBE_NVERELS and NOSBE_NOSBINS. These files are then used to modify an existing NOS/BE deadstart tape.

All of the 180 procedures which compile or build 170 files expect an environment to have been set up by integration at some time in the past. They expect this environment to exist in a 170 catalog, selected by the tying file variables WEV$NOS_BASE (WEV$NOSBE_BASE) and WEV$CYBIL_CC_BASE (WEV$NOSBE_CYBIL_CC_BASE). WEV$NOS_BASE (WEV$NOSBE_BASE) selects the version of the NOS(BE) system to compile for; and WEV$CYBIL_CC_BASE (WEV$NOSBE_CYBIL_CC_BASE) selects the version of the cybil compiler. The catalog selected by WEV$NOS_BASE (WEV$NOSBE_BASE) must contain the following files: PL170, BAMLIB, BAMPL, CETEXT, IOTEXT, IPTEXT, NETIO, NETIOD, NOSTEXT, PSSTEXT, SSYTEXT, SYMLIB and SYSLIB. (PL170 is a subset of the NOS OPL. NETIOD is the debug version of NETIO. NOS/BE versions of some of these files must be manufactured artificially.) The catalog selected by WEV$CYBIL_CC_BASE (WEV$NOSBE_CYBIL_CC_BASE) must contain the files CYBILC and CYBCLIB. Note: It is possible for a user to create his own copy of one of these catalogs without all of the files that an integration version needs. If he is planning on a limited set of decks to be recompiled; he only needs those files which will be referenced by his decks. He then creates the appropriate variable(s) at the job level; scope=xdcl; and with his own value. The integration procedures will then use the contents of his catalog.

2.1.4 NOS TAPES

There are two tapes, used by the VE project, which are still generated on NOS. The first is the NOS deadstart tape used by dual state; the second is the CIP (CYBER_INITIALIZATION_PACKAGE). Both of these are copies of the standard released tapes; with some minor additions and replacements.

2.0 HOW TO DO BUILDS
2.1.4 NOS TAPES

The NOS deadstart tape generated by NOS Integration is usually correct for VE use. This is accomplished by providing NOS with a copy of the DUAL_STATE_SOURCE library and text files produced by RAM$CONVERT_DSSL_TO_TEXT. If, for any reason, the tape produced by NOS must be changed; NVELIB, NOSBINS and SES must bee added - see "NEW LEVELS OF NOS" for details.

Modified CIP tapes are generated by VE Integration. We start with the release version of CIP. A copy of this is proceduced by using COPY_NOS_TAPE - the format is SI and the density PE. Previously generated CIP components are then added from the working, feature and integration build level catalogs via the procedure BUILD_CIP_TAPE.

2.1.5 NEW LEVELS OF NOS

From time to time; NOS/VE must be mated to a new level of NOS. This will occur as long as dual state is supported. When the DCT decides to upgrade to a new level of NOS; the following items occur.

1. A 170 catalog (DEV1,2,3,4 or INT1,2,3 or 4) is selected to contain the new system.
2. VE Integration receives a pair of tapes from NOS Integration. One of these is the NOS OPL tape; the other the NOS deadstart tape.
3. The procedure GENERATE_NEW_NOS is executed to extract the following files into the catalog selected at 1). Files extracted :

   NOSTEXT - Text required by all COMPASS programs at assemlbly time.

   PSSTEXT, SSYTEXT - Text required by CP COMPASS programs at assembly time.

   NETIO, BAMLIB - NOS user library required by link procedures when forming Interactive Facility absolute binaries.

   NETIOD - NOS user library required by link procedure when forming debug version of the Interactive Facility absolute binaries.

   SRVLIB, SYSLIB , SYMLIB - NOS user libraries required by link procedure when forming FASLAVEabsolute binaries (NOS partner for File Migration Aids product).

   PL170 - NOS MODIFY program library (OPLD) containing common

2.0 HOW TO DO BUILDS
2.1.5 NEW LEVELS OF NOS

decks (OPLC records) required by COMPASS XTEXT pseudo-ops at assembly time.

CETEXT, IPTEXT, IOTEXT, TXTCRM - Text required by 170 Processors at assemble time.

Note: GENERATE_NEW_NOS must be executed from the VE side of the user number selected. Also; a new nos deadstart tape may be generated at the same time.

4. The TYING file entry for WEV$NOS_BASE must be changed (in the level of VE that the new NOS is introduced) to point to the new catalog.
5. There is the possibility that the new level of NOS will require a new CYBIL CC compiler. This should be investigated.

2.1.6 PROBLEMS

Problems with the integration process can manifest themselves in several steps along the integration path. Each level of problem requires it's own type of corrective action. As a general rule, the earlier a problem is detected, the easier the correction is to introduce. All problems encountered should be logged so that they can be analyzed and correctable errors avoided in future builds.

2.1.6.1 BEFORE A BUILD

The first level of problems occur before a build has "officially" started. These tend to be problems with the environment which the integration procedures expect to exist at startup. Typical examples of this include missing features (from the build_request file), object libraries from the previous level which have not been merged, source libraries which have not been updated and unspecified dependencies in the list of new build requests.

2.1.6.1.1 MISSING BUILD_REQUEST DATA

Feature information may be lost from the build_request file in any one of several ways (lost in a way such that the paper portion still exists). One way is for the system upon which the build request was made to crash and lose permanent files. If the originator of the request does not realize that information is saved

by MAKE_BUILD_REQUESTS; he will turn in the paper build request like nothing has happended. Another condition is for the BUILD_REQUESTS file to be busy. If this happens an originator may terminate his MAKE_BUILD_REQUEST and print the form manually. In any case the corrective action to take is to execute MAKE_BUILD_REQUESTS for the appropriate feature.

2.1.6.1.2 UNMERGED OBJECT LIBRARIES

The cause of the object libraries not being merged is integration's not finishing the prior build before a new one is started. The integration procedures are in the process of being changed to delete the information from the CHANGES subcatalog as object libraries are merged with their MAINTENANCE counterparts. This condition will then be detectable by CREATE_INTEGRATION_WORKING_ENVIRONMENT. The correction is to execute MERGE_DESTINATION_LIBRARIES before attempting to execute CREIWE.

2.1.6.1.3 UNUPDATED SOURCE LIBRARIES

There are many possible reasons for the update jobs to not have been executed or to not have executed correctly. The key item in this type of problem is that a successor build is attempted before the code introduced by the parent build has been changed to state 2. This will occur when builds are done in rapid succession. The corrective action in this case is to include the parent build's WEF$FEATURE_LIST data along with that for the new code.

2.1.6.1.4 UNSPECIFIED DEPENDENCIES

One of the most difficult problems to detect early in the build process are code dependencies which are not specified on the MAKE_BUILD_REQUEST call. These missing dependencies are due to a developer either not being aware that he has a dependency or not being aware that he needs to inform integration of this fact. Dependencies come in varying degrees of "firmness", and their ease of detection reflects this.

The "hardest" dependency is that in which two modifications directly affect the same line(s) of code in a deck. This case typically results in compilation errors on the deck in question.

A softer dependency results when changes in common decks become uncoupled from changes in the decks which include them or, in the case of XREF (xxP$ decks), from the deck containing the

2.0 HOW TO DO BUILDS
2.1.6.1.4 UNSPECIFIED DEPENDENCIES

corresponding XDCL. This case tends to result in linker errors (unsatisfied externals, parameter verification mismatch, etc.).

The softest form of dependency is the purely logical dependency. This is the form where a feature expects to use another feature in order to execute correctly; but there are no interface changes anywhere. This form of dependency is only detectable at run-time by determining that the new feature does not work correctly.

The corrective action is the same for all cases of missing dependencies. In all cases a decision must be made by the DCT (and, possibly, the originators of the dependant features). Either the missing feature must be added to the build, the dependant feature removed or the dependant feature regenerated to remove the dependency.

2.1.6.2 DURING A BUILD

Problems which occur during a build are similer in symptom to those for missing dependencies. These include compile errors linker errors and problems with the system; such system crashes and tape errors. It also includes items which are not errors; but require some amount of rework, such as new code being added after a build has started.

2.1.6.2.1 COMPILE ERRORS

Most compile errors are the result of unspecified dependencies. The ones which remain tend to indicate that development has not tested their code adequetly before transmittal. There are several varients on this lack of adiquate testing. The simple varient is where code is simply wrong as it is entered. This should not happen. A second varient is code which has aged too much before being built. In this case, changes were made to an interface (procedure definition, type, variable, etc.) without making the change to the current deck's unintegrated code so that it will match. In any case a PSR should be written (CRITICAL) to the effect that there are errors in the deck. A decision is also required by the DCT as to whither the feature that caused the error needs to be pulled or a fix introduced.

2.1.6.2.2 LINKER ERRORS

As with compile errors, most linker errors which integrators will encounter are due to unspecified dependencies. The remainder are due to one of several causes: a) incorrectly adding code to the predecessor build level after the current level has started, b) missing decks which *copy's common decks. Neither of these conditions are a very common occurance. Condition a) usually occurs when doing parallel builds. Condition b) indicates a problem either with the feature list used as input or a problem with SCU. In either case, some additional decks need to be recompiled and the link reattempted. Condition b) may also imply that a PSR be written against SCU for not finding the additional *copying decks.

2.1.6.2.3 SYSTEM PROBLEMS

System problems in a build are things such as crashes or bad tapes. In general these are problems which an integrator cannot do much about - they simply have to be endured. The general fix for problems of this class is to simply start the process over from the point where it left off. (Note: in some cases; parts of the integration environment will have to be reset to values which equal those at the start of the build, before continuing.) The best that an integrator can do is to check the materials (build catalogs, tapes, etc.) that he produces to see if they have been corrupted and correct the problems found.

2.1.6.2.4 ADDING ADDITIONAL CODE

Adding code to a build which has already started is a "problem" in that it interfears with the timely completion of a build. Ideally no build should be started until all code which is to go into that build is in hand. Unfortunatly, this is not always true. Several mechanisms will introduce code into a system after it has started. One of these is the correction of errors encounted in the build, another is the addition of features which are urgently needed for closed shop stability. The best that probably can be hoped for is to keep the number if instances of added code to a minimum.

2.1.6.2.5 DECK HEADER ERRORS

These types of errors are due to incorrect information on a deck header. They generally affect new decks (which are going into this build for the first time); although they can also appear on decks which have the the object of major rewrites. The fields of the deck header which are subject to error are: the processor field, the

processor group-attribute(s) and the destination library field(s) Errors in the processor field will manifest themselves at compile time. Either the field will be blank, producing an informative message from MAKE_BUILD_JOBS; or it will have an incorrect compiler value and produce massive compile errors. Errors in the processor field are more subtle and may only show up as a deck not recompiled when a build involving a compiler change is done. (Note: It is NOT incorrect for a deck which has as it's processor value a pre-processor procedure to ALSO have the end compiler listed in it's group attribute list; e.g. MESSAGE_TEMPLATES also have CYBIL.) Errors in destination groups will manifest themselve in either a linker error or in some form of runtime error. (It is possible that an error in destination group would cause a deck to be placed on a library which would cause it to be a security breach without causing noticeable errors. I do not know of a way to prevent this other than vigilance.) Typical linker errors are undefined xxx (variable/entry point/etc.) or duplicate xxx (this is the one which shows up on an old deck which has been changed). The correction for any form of error in a deck header is to extract the deck; correct the error on the extracted copy, transmit the corrected copy and recompile. In some cases, the integrator may need to remove the deck's binary from an incorrect object library.

2.1.6.2.6 "BAD" MODSETS, MISSING MODSETS, MODSET HEADER ERRORS

The three items listed on the header for this section are all intertwined. A modset's header may have an incorrect feature name. This will cause it to be missed from a build and this may cause the build to not correct a problem that it should solve. Modset problems will appear as either a compilation error or a fix or new feature which does not work correctly when executed. If the modset is simply missing for any reason; it can be added and another cycle of the build process executed with the parameters set to the standard values for adding code to an existing build. Note: A modification may not be corrected in the same manner as a deck. The changes must be fed through the update jobs via the special requests file. See MAKE_SPECIAL_REQUEST and CHANGE_MODIFICATION_HEADERS for more details. If the modification contents are incorrect; either a new modification (a fix) is reqired from development or the modification must be removed from the build. This is a DCT decision.

2.1.6.2.7 COMPILER ERRORS

Infrequently, an integrator will encounter compiler errors, where correct code produces incorrect results. These will either manifest themselves in compile errors or runtime errors (which will require

very careful dump analysis). When a compiler error is encountered, a PSR should be written to the responsible project. A decision is required on whether to produce a workaround, back up to a compiler that works or await a new/fixed compiler.

2.1.6.2.8 WRONG COMPILER LEVELS

There are times when the wrong compiler level is selected. The version of 180-based compilers are determined by their program descriptors which exist in <development_base>.OS.<build level>.MAINTENANCE.COMMAND_LIBRARY. The version of the CYBIL_CC compiler is determined by the variable WEV$CYBIL_CC_BASE (or the variable WEV$NOSBE_CYBIL_CC_BASE for nosbe builds) from <development_base>.OS.<build_level>.TYING file. All other 170-based compilers and assemblers are taken from the nos system currently running. To correct the compiler level, change either the program_descriptor or the CYBIL_CC_BASE variable to the correct value. It is not possible to select other versions of the other 170 compilers.

2.1.6.2.9 PROCEDURE ERRORS

Since integration is infalible; there are no procedure errors, only randomly operating unanticipated features. These will manifest themselves in a variety of ways. Since we try to correct procedure problems as soon as possible; it is impossible to produce a current list of these problems in this document. These problems may be overt, such as a procedure aborting; or covert, such as 'losing' features from a build or failing to change code to the correct level. In all cases; a PSR should be written and a correction produced and used.

2.1.6.3 PROBLEMS DURING CHECKOUT

There are several things which may go wrong with a system after it has been built. The most obvious is not deadstarting. Other examples include test failures and missing components.

2.1.6.3.1 SYSTEM DOESNT DEADSTART

This is a condition which may be due to bad code, bad tapes, hardware problems or missing or defective system components. In all cases a dump should be taken at the point of failure to preserve the problem. A knowledge of the contents of the current build, the point of failure and of any unusual events encountered in the build process are invaluable in determining the exact problem.

If the problem is bad code (or is suspected to be) a dump analysis may be needed to determine the exact point of failure. A PSR should be written and a decision on whether to pull the offending feature or await a fix is needed from the DCT. A PSR should be written in any case.

If bad tapes are a problem; those tapes should be regenerated on fresh tapes, on a tape drive which is different from the one originally used and which has been recently cleaned.

Many problems with 2550's or CDCNET DI's can be cleared up by doing a master-clear. You should be careful that you do not interfear with other users on other machines. In other cases, disk drives may be write-protected, or logically down. There may be a very good reason for this - check with other users before changing these settings. In still other cases switches may be left in the wrong state - check the deadstart panels for correct settings. If all else fails; write a MAF for the CE's.

2.1.6.3.2 TESTS REPORT FAILURES

If a regression test reports a failure; it should preserve enough information to determine the cause of the problem. This information then needs to be analyzed to determine if there is a new problem with the system or if there is a test case error. a new system problem results in a PSR and determination on whether to remove, fix or ignore the problem. A test case error results in a PSR against the test; but are usually ignored.

2.1.6.3.3 MISSING SYSTEM COMPONENT

Entire products can sometimes be left off of a system. In one case; they are optional and simply forgotten. Other cases indicate either a problem with the installation table used to produce the product tapes or to build levels which have been deleted from .INTVE. (The latter will ideally be caught before the build is finished; since ASSEMBLE_RELEASE_MATERIALS will report missing elements.) In the first case; the tests should be rerun after

insuring the missing product has been loaded. In the latter two cases some of the product tapes will have to be regenerated (possibly after reloading the lost build level catalog).

2.1.7 COMPILE ERRORS

Decks encounter compile errors due to several causes. Among these are modification conflicts or missing dependencies, incomplete testing, compiler changes, and so on. When these occur; it is necessary to isolate the cause before correction can be attempted. To isolate the cause; a listing is usually required. The integrator can then look at the section of code returning the error. A large number of errors in a module is symptomatic of a type definition error. A small number is typical of an error in the module itself. (A small number of errors in several modules may also be a type definition problem.)

After the listing has been produced. Look at the area encountering the errors. If the module appears misformatted; this may be symptomatic of modification problems. The integrator should produce a scu listing showing both the active and inactive lines for the deck in question. This will allow the integrator to determine if there is a missing modification.

A variation on the missing modification problem can occur with every possible piece of data (state, feature, etc.) showing that the missing mod really should not be missing; but should be in the build. In this case, examine working and feature catalogs, alternate base or product catalogs, etc. - look for a mod with the same name; but with different states, features and so on. Also look for another version of the deck in question that may not have the mod applied to it.

Compiler changes are generally known in advance and the integrator will be aware that there is a new compiler being used with the current build. Any offending modules should be recompiled with an older compiler to see if this will eliminate the error. (A decision is then needed as to whither the compiler or the module must be corrected.) A variation of problems with the compiler may occur with new decks. A new deck may expose a compiler bug that never surfaced before. (This tends to be unlikly in a well used version of the compiler.)

Another avenue to try is to look at all new code being added to the current build. Look for code which either affects the module or to code in common decks which might be included in the module. If

it can be determined which modification might be causing the problem; the originator of the mod can be contacted for additional help.

Correcting compile problems usually requires adding another mod to the build. This mod may be a newly generated mod or one that was inadvertantly excluded. Another option is the removal of an offending modification. One of these options must be chosen before the build can continue. All modules included in a build must compile correctly for the build to work.

2.1.8 LINK ERRORS

The linker errors that integrators see come in three flavors. The first of these is "parameter verification" errors. The second error type is "declaration mismatch" and the third error type is "unsatisfied externals".

Parameter verification errors and declaration mismatch errors are very similer in cause and scope. They indication that the description which one module has of an entry point in another module is not correct. Parameter verification errors indicate that the type definition of one (or more) of the parameters to an interface have changed. Declaration mismatch indicates that difference that is other than a type definition mismatch. The first step in attempting to correct this type of problem is to recompile all modules reporting problems and attempt to relink. Inmany (all parameter verification, expanded,) this will resolve the dispute. If recompiling does not eliminate the problem; get listings of both the module containing the XDCL and the one attempting the XREF, att the current level. (Uncompiled listings are better than compiler listings because inteface declarations are suppressed by pragmats in many modules.) Compare the interface from both listings, looking for differences. After the difference is discovered; find the mod which introduced it and ask the mod creator for a fix. (Alternativly the offending mod can be dropped from the build.)

Unsatisfied external references indicates that a module (or modules) containing an XDCLed interface is missing from the current build. The first step is to determine the module that is missing. The best source for this information is the project responsible for the area identified by the missing interface. They can also tell you if the module is a new one that has been missed or if it is an existing one which has been removed (the second alternative is the most common one). A decision is now needed on whither to correct

the problem or remove the mod(s) which caused it.

2.2 BUILDING OCU

Since the OCU product is kept on the OS source library; building OCU is almost automatic. A compile job will be produced whenever any change affecting OCU is included in an OS build. Whenever a build job is produced for OCF$OBJECT_CODE_UTILITIES; you will have to produce a new OCU to go with the OS. There are only two additional steps needed to make the new version of OCU. The first of these is BIND_OCU which is documented in the manual "User Documentation for the Development Environment of NOS/VE on NOS/VE" (PROCDOC). BIND_OCU should be run at about the same time as LINK_OPERATING_SYSTEM is done on OS. The final operation is to do a GENERATE_PRODUCT_TAPE to produce a product deadstart tape (this is also documented in PROCDOC) containing the new OCU and any other new products.

2.3 BUILDING SCU

SCU is also built in a manner similer to the OS. All of the build steps 1-7 are identical except that the value of the matching OS level must be specified on the OS_BUILD_LEVEL parameter of BUILD_SYSTEM. After all compile jobs have completed sucessfully; The procedure BIND_SCU must be run. This procedure is documented in the manual "User Documentation for the Development Environment of NOS/VE on NOS/VE" (PROCDOC). If there are no linker errors in either SCU or the Editor; the bound SCU is included with the other products in producing a new product deadstart tape.

2.4 BUILDING EI

2.5 BUILDING SCD

2.6 BUILDING PROCS

The working environment procs are build somewhat differently from all other products. Their production is the same for steps 1-7 of the OS build. After step 7; the processes begin to diverge. Instead of being linked and put onto a product set tape as other products are; the libraries produced by PROCS compile jobs are merged with the existing tools libraries which exist in the master catalog. Each of the compile jobs merges with the existing tools library like this:

```
          wef$command_library -> <d-b>.command_library
          wef$integration_command_library -> <d-b>.misc_files.
     integration_command_library
          wef$update_intve_command_lib -> <d-b>.update_jobs.
     wef$update_intve_command_lib
```

After these new libraries have been tested and all of the components appear to work correctly; they are distributed to all of the closed shop machines which integration supports.

2.7 INCORPORATING THE SUNNYVALE PRODUCTS

3.0 GENERATING MICROFICHE

3.0 GENERATING MICROFICHE

3.1 MICROFICHE GENERATION PROCEDURES

3.1.1 COMBINE_FICHE_LISTING_LIBRARY (COMFLL)

The purpose of this procedure is to take a set of listing libraries (produced by compile_source) which are usually sorted by destination object library and transform them into a second set of libraries which are sorted alphabetically. For example, decks aaa through bbb from libraries x1, x2, etc.; are combined and put onto library y1, ccc through ddd on y2, etc.

```
combine_fiche_listing_library
     input_listing_library, ill: list of file = $require
     listing_library_boundaries, llb : list 1..10, 1..2 of  name  or
     key all = all
     output_listing_library, oll : list of name 1..7 = $required
     working_catalog, wc : file or key none = none
     delete_input_listing_library, dill : boolean = true
     status
```

input_listing_library | ill : This is the set of listing libraries produced by compile_source.

listing_library_boundaries | llb : These pairs of names specify the starting and ending limits of each of the new libraries.

output_listing_library | oll : This is the prefix for each of the new libraries. Information about the boundaries is appended to create a unique name.

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

delete_input_listing_library | dill : This parameter will cause the cleanup of the input libraries. The purpose is to conserve

3.0 GENERATING MICROFICHE
3.1.1 COMBINE_FICHE_LISTING_LIBRARY (COMFLL)

disk space.

status : See NOS/VE error handling.

3.1.2 FICHE_MAKE_BUILD_JOBS (FICHE_MAKE_BUILD_JOB FICMBJ)

This procedure is a copy of the make_build_jobs procedure with some additional options needed for making micro fiche added to each job it produces.

```
fiche_make_build_jobs
     selection_criteria, sc : file = $required
     save_listings, save_listing, sl : key all, none, good, bad =
     all
     build_job_catalog, bjc : file = $user.build_jobs
     submit_jobs, submit_job, sj : key immediate, i, delay, d,
     no_submit, ns = immediate
     os_build_level, obl : name = $optional
     development_base, db : file = .intve
     product_name, pn : name = os
     build_level, bl : name = $optional
     working_catalog, wc : file or key none = none
     listing_library_boundaries, llb : list 1..10, 1..2 of name or
     key all = all
     password, pw : name = $required
     status
```

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

3.0 GENERATING MICROFICHE
3.1.2 FICHE_MAKE_BUILD_JOBS (FICHE_MAKE_BUILD_JOB FICMBJ)

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).

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.

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

listing_library_boundaries | llb : These pairs of names specify the starting and ending limits of each of the new libraries. The values specified on this command are passed to the procedure combine_fiche_listing_libraries.

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

status : See NOS/VE error handling.

3.1.3 FICHE_TITLINGS (FICHE_TITLING FICT)

****
DESCRIPTION NEEDED <<<<-------
****

fiche_titlings
-->> PARAMETERS NEEDED <<--
-->> PARAMETER DESCRIPTIONS NEEDED <<--

```
exitnest
```

3.1.4 GENERATE_FICHE (GENF)

This procedure generates a file containing listings with headers suitable for writing to a tape from which microfiche can be produced. The input is one or two SCU libraries, with each module on a separate deck. These libraries are produced by PUT_LISTINGS_ON_SOURCE_LIBRARY. Other input is provided by the SECTION_MAP and OTHER_FILES parameters. The output is written to a file specified by the NOS_FILE_NAME parameter. This file is replaced to NOS.

Titling information is provided by the SMD_PART_NUMBER parameter, which is a string 8 characters long assigned to the release by SMD; the SYSTEM_LEVEL_IDENTIFIER, which is a string of eight characters identifying the NOS/VE level; the PRODUCT_LEVEL_IDENTIFIER, which is a string of up to 18 characters placed on the banner pages; the OTHER_NAMES parameter, which gives titles to the files specified by the OTHER_FILES parameter.

```
generate_fiche
     listing_library, listings_library, ll : list .. of file =
     $required
     smd_part_number, smdpn, spn : string .. = $required
     system_level_identifier, sli : string .. = $required
     product_level_identifier, pli : string .. = $required
     nos_file_name, nfn : name .. = $required
     section_map, sm : file = $optional
     other_files, of : list of file = $optional
     other_names, on : list of string = $optional
     status
```

listing_library | listings_library | ll : This parameter is the file which contains all of the deck compilation listings.

smd_part_number | smdpn | spn : This is the number assiged by smd to the particular product. It is used in the microfiche titling.

system_level_identifier | sli : This is the build_level or release level of the system part of the microfiche. It is used in the microfiche titling.

3.0 GENERATING MICROFICHE
3.1.4 GENERATE_FICHE (GENF)

product_level_identifier | pli : This is the build_level or release level of the product set part of the microfiche. It is used in the microfiche titling.

nos_file_name | nfn : This is the file that the listings are replaced to, prior to being dumped to a 170 tape.

section_maps | sm : This is a file containing the linkmaps for the product. They are included with the compilation listings.

other_files | of : This is a hook to allow additional, unusual information to be included.

other_name | on : This is the list of names of the items specified on the other_files parameter.

status : See NOS/VE error handling.

3.1.5 GENERATE_MICRO_FICHE_DECKS (GENMFD)

The purpose of this procedure is to generate a cross reference deck and other miscellaneous decks on a micro fiche listing library. The types of miscellaneous decks depend on the product name specified. If 'OS' is specified then system_core_link_map, job_template_link_map, and link_170_map decks are create on the listing library. If 'SCU' is specified then a map deck is create on the listing library. And if any other product is specified no miscellaneous decks are added.

```
generate_micro_fiche_decks
     output_library, ol : name = $required
     selection_criteria, sc : file = $optional
     working_catalog, wc : file = $optional
     product_name, pn : name = os
     build_level, bl : name = $optional
     status
```

output_library | ol : This is the listing_library upon which the decks which are created are put.

selection_criteria | sc : This is a scu criteria file, which is used to modify the version of the product selected by the build_level

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

3.0 GENERATING MICROFICHE
3.1.5 GENERATE_MICRO_FICHE_DECKS (GENMFD)

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.

3.1.6 GENERATE_MICRO_FICHE_TAPES (GENMFT)

This procedure calls GENERATE_FICHE to create a file of microfiche data on NOS and then creates a job to copy that file to tape.

```
generate_micro_fiche_tapes
     listing_library, ll : list of file = $required
     vsn : list of name ..  = $required
     smd_part_number, smdpn, spn : string ..  = $optional
     system_level_identifier, sli : string ..  = $optional
     product_level_identifier, pli : string ..  = $optional
     move_file_to_170, mft1 : boolean = true
     status, s : status = $optional
```

listing_library | ll : This parameter is the file which contains all of the deck compilation listings.

vsn | v : This is the vsn of the tape to copy the microfiche listings onto.

smd_part_number | smdpn | spn : This is the number assiged by smd to the particular product. It is used in the microfiche titling.

system_level_identifier | sli : This is the build_level or release level of the system part of the microfiche. It is used in the microfiche titling.

product_level_identifier | pli : This is the build_level or release level of the product set part of the microfiche. It is used in the microfiche titling.

status : See NOS/VE error handling.

3.1.7 MICRO_FICHE_BUILD_JOBS (MICFBJ)

This procedure completes the first step in making micro fiche. A micro fiche catalog is created with a selection criteria file and then the procedure fiche_make_build_jobs is excuted local or batch. This procedure causes the recompilation of the entire product, minus some undesired object libraries. It also generates batch jobs to produce linkmap listing decks.

```
micro_fiche_build_jobs
     build_level, bl : name = $optional
     product_name, pn : name or key os, scu, tdu = os
     fiche_catalog, fc : file = $fname('$user.micro_fiche')
     job_class, mjc : key batch, local, maintenance = batch
     password, pw : name = $required
     status
```

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

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

fiche_catalog | fc : This is the catalog to use in creating the microfiche.

job_class | jc : This is the job class that is passed on to fiche_make_build_jobs.

password | pw : This is the current usernumber's password.

status : See NOS/VE error handling.

4.0 RELEASE PROCESS

4.0 RELEASE PROCESS

4.1 HOW TO GENERATE RELEASE MATERIALS

Everytime a new build is going to CLSH or to SMD release materials are needed. The release materials, or tapes, that integration needs to make are whatever materials may have changed since the last set of release materials were made.

4.1.1 UPDATE_RELEASE_MATERIALS (UPDRM UPDATE_RELEASE_MATERIAL)

The purpose of this procedure is to figure out which tapes need to be made for the new release level by comparing the previous release level with the current (new) release level. The procedure calls FIND_CHANGED_PRODUCTS to determine which products have changed. It then calls GENERATE_TAPE_LIST giving the list of changed products. After it knows which tapes need to be made it calls GENERATE_RELEASE_MATERIALS to make the tapes.

```
update_release_materials
  current_release_level, crl: name = $required
  previous_release_level, ..
  prl : name = $required
  development_base, db : file = .intve
  build_level, bl : name = $optional
  status : var of status = $optional
```

current_build_level | cbl : The end build to check.

previous_build_level | pbl : The starting build to check.

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

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

4.0 RELEASE PROCESS
4.1.1 UPDATE_RELEASE_MATERIALS (UPDRM UPDATE_RELEASE_MATERIAL)

status : See NOS/VE error handling.

After the release materials are generated, and if they need to be sent to SMD, they need to be copied because integration can not send the master copy to SMD.

4.1.2 COPY_RELEASE_MATERIALS (COPRM)

This procedure copies release materials for a specified build_level. It goes into EDIT_TAPE_LIBRARY and returns the vsn's for the I-tapes that are associated with that build_level. It also uses the function $PRODUCT_VSN, inside of PREPARE_RELEASE_MATERIALS, to optain the SMD-standard vsns/product-names for the release tapes. After it has the needed vsn's it uses the procedure COPY_LABELLED_TAPES to do the actual copying. It will continue copying if any of the tapes do not copy correctly, but it will return bad status.

```
copy_release_materials
  build_level, bl : name = $required
  exclude_products, ..
  exclude_product, ep : list of name = $optional
  status : var of status = $optional
```

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

exclude_products | ep : This is a list of release tape product names (eg. BASE_1, COBOL, etc.) which are not to be copied.

status : See NOS/VE error handling.

4.2 VERIFYING RELEASE MATERIALS

SMD will make copies of the tapes it received from integration and send them back to integration for verification. Integration is then suppose to verify each tape against the master tape in the tape library.

4.2.1 VERIFY_RELEASE_MATERIALS (VERRM)

This procedure verifies release materials for a specified build_level. It goes into EDIT_TAPE_LIBRARY and returns the vsn's for the I-tapes that are associated with that build_level. It also uses the function $PRODUCT_VSN, inside of PREPARE_RELEASE_MATERIALS, to optain the vsn's for the release tapes. After it has the needed vsn's it uses the procedure VERIFY_LABELLED_TAPES to do the actual verifying. It will continue verifying if any of the tapes do not verify correctly, but it will return bad status.

```
verify_release_materials
  build_level, bl : name = $required
  exclude_products, ..
  exclude_product, ep : list of name = $optional
  status : var of status = $optional
```

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

exclude_products | exclude_product | ep : A list of the standard product vsns which are not to be verified at this build level.

status : See NOS/VE error handling.

4.3 RELEASING SOURCE MATERIALS

It is easiest to make materials for CSERV first, then generate the the SMD materials from them.

4.3.1 GENERATING SOURCE MATERIAL FOR CSERV.

The major key to success is STARTING EARLY. Also it useful to recompile the entire system late in the final build cycle. This will help detect any state or group inconsistancies.

Customer Service needs source materials for each release that is supported. They need to start from a clean environment. A clean environment no state 1 code or extraneous build decks.

Steps in generate source materials.

1. Transmit and build all procs used by integration for the final

4.0 RELEASE PROCESS
4.3.1 GENERATING SOURCE MATERIAL FOR CSERV.

build. This should be done during or before the final build.

2. Backup to tape all the lastest product build catalogs supported by CSERV. skp$delfc Also backup catalogs needed for building (ep: AAM, COMMON, COMMON_1 and command libraries. Give the tapes to CSERV so they can begin installation.

3. Create the catalog $user.release_XXX, where XXX is the release level. (ep: create_catalog $user.release_678) This catalog will be used as the development base.

4. Move the COMMON, COMMON_170, CIP, FMA, FMU, and WORKBENCH catalogs from INTV to the new developement base catalog.

5. Run the procedure GENERATE_SOURCE_RELEASE for each AHPD product supported by CSERV. This procedure will produce clean source_libraries and move the build catalogs to the new development base.

6. Compile the entire system with the development base set to $user.release_XXX

7. Compare the compilation files with those in INTVE. Investigate the differen there could be missing or extra code. This is usually caused by wrong groups or in integration source_libraries. Correct the errors and recompile. Some decks won't compare because dates are embedded within the code.

8. Once the compilation files match a closely as possible, backup the source_l to tape and give them to CSERV. A memo stating what was handed off and which de didn't compare should be included with the hand off to CSERV. The integration project leader should have an example memo.

4.3.2 GENERATING SOURCE MATERIALS FOR SMD.

1. Delete all the subcatalogs in $user.release_XXX EXCEPT AAM, AV, PROCS, OS, S and WORKBENCH.

2. Delete all files associated with tests.
(eg3. $user.release_XXX. os.build_YYYYY. maintenance.osf$f_test_data)

4. Rerun generate_source_release for OS and SCU. Specify a

selection criteria exclude the test groups.

5. Update the deck RELEASE_DOCUMENTATION. Print current version of WEM$INTDOC

6. Write the tape NVES01, NVES02, and NVES03. If possible, match the format of File sizes could make this impossible. If so, the documentation will have to be Print the backup listings.

7. Hand off the tapes NVES01, NVES02, and NVES03 to SMD. Include the release d PROCDOC and INTDOC, and backup listings.

4.3.3 GENERATING TDU SOURCE MATERIALS FOR SMD.

1. Create the catalog $user.tdu_release_XXX and subcatalogs OS, SCU, and TDU.

2. Copy the file $user.release_XXX. tdu.source_library to $User. tdu_release_XX To create source_libraries to the OS and SCU catalogs talk to TOM LITTLE. He is groups on the needed decks. Use extract_source_library, and extract from the s used for CSERV.

3. The tdu project will supply a file called RUNIT. It contains instruction fo TDU. Someone outside TDU should implement the instructions in the runit file.

5.0 VERIFYING A BUILD

5.0 VERIFYING A BUILD

5.1 EXECUTING "BIG 3" TESTS

5.2 BYOPS FORMS

6.0 PRODUCT SOURCE LIBRARY CLEANUP

6.0 PRODUCT SOURCE LIBRARY CLEANUP

Typically, after a system is released, the product source libraries are cleaned up. Cleaning up the libraries consist of removing deleted decks and resequencing all modifications that make up the released system. In addition to this, there could be deck name changes and reformatting of CYBIL decks.

Each of these operations can be performed independently of the others. Reformatting cybil decks is something that could be done by integration but as yet, is not being done. Deck name changes require review and can take quite awhile. As naming conventions get better defined, there may be a need for this again. As new decks are transmitted, they should be verified for correctness.

To remove deleted decks, one must first generate two lists, a list of unreferenced decks, decks which are not copied (*copy or *copyc) by any other deck and which are not module decks, via GENERATE_NON_REFERENCED_DECK_LIST, and a list of the current members of the scu group DELETED_DECKS via GENERATE_DELETED_DECK_LIST. These two lists are then passed to development to verify that all decks being deleted should be. After development has responded with any changes to the delete list and these changes have been incorporated into the list, a copy of the list is produced without the 'DELETE_DECK' commands on each line. This list is input for two cross checking procedures, CHECK_REFERENCING_DECKS and CHECK_DELETED_DECK_LIST. CHECK_REFERENCING_DECKS checks the deck list against product source libraries for references to the decks being deleted. It requires the list of unreferenced decks (produced by GENERATE_NON_REF_DECK_LIST) and uses this list to determine whether a deck is referenced or not. If either of these procedures produces an anomoly, it must be resolved in concert with development and the checks repeated until no problems result. Once we have a clean list of decks to delete, the list (with delete_deck commands on each line) can be added to the special_requests file for the product.

To change deck names we must first generate a list of those decks whose names violate our naming conventions. There is a procedure to check naming conventions: VERIFY_NAMING_CONVENTIONS, which works off of a list of deck names or an scu library. The output of this procedures is a list of names which do not match conventions. This list must be checked to determine the scope of change - how many decks copy the deck being changed. This is determined by runing CHECK_REFERENCING_DECKS on the list produced by VERNC. This

6.0 PRODUCT SOURCE LIBRARY CLEANUP

procedure must also be run against other products if deck names are being changed on the program interface. This will produce output showing how large the change is. This information, along with the change list, is passed to development so that they can provide the new deck names. After all new names have been decided upon, a file using the scu CHADN command format (OLD_NAME=xxx NEW_NAME=yyy) must be produced. After this file exists, CHADN commands must be added to the SPECIAL_REQUESTS file for every product affected by the changes to complete the operation.

Resequencing modifications compresses all released modifications applying to a deck into the deck's creation modification. This is a two part process, raising the state of all released modifications to state 4 and then resequencing all decks. GENERATE_MOD_STATE_CHANGE_LIST generates a file of commands to change all state 2 and 3 mods, up to and including the specified build level, to state 4. This file must be copied to a file named SEQUENCE_LIST and appended to special requests, both of which reside in the product catalog being updated. When the update job is run, the commands in special_requests are executed and the states of the modifications should all be at state 4. To sequence the product library, the scu command SEQUENCE_DECK is added to special requests. Special care should be taken to insure that modifications which have been 'pulled' from a build, and which should not be compressed with the other modifications, are not. The file SEQUENCE_LIST is used by the update intve procedures to resequence latest changes before it is combined with the product source library. This is necessary to resequence decks that were extracted when the product source library was sequenced.

To reformat cybil decks; it is necessary to make a special request to execute the procedure FORMAT_LIBRARY. This procedure will CYBFORM the requested number of cybil decks, print out the starting point for the next run and stop. This procedure must be repeated until all cybil decks have been formatted.

6.1 PRODUCT SOURCE LIBRARY CLEANUP PROCEDURES

6.1.1 CHECK_DELETED_DECK_LIST (CHEDDL)

The purpose of this procedure is to check the list of deleted decks against the list of non-referenced decks and list the deleted decks that are not in the non-referenced deck list. These are referenced decks scheduled for deletion. This procedure can only be used on the products from which the decks are being deleted.

```
CHECK_DELETED_DECK_LIST
     delete_list, dl : file = $required
     non_referenced_list, nrl : file = $required
     referenced_deleted_list, rdl : file = referenced_decks
     development_base | db : file = .INTVE
     product, p : name = $optional
     referenced_deleted_xref, rdx : file = referenced_xref
     status
```

delete_list | dl : This is the file that contains the list of decks to delete.

Required parameter.

non_referenced_list | nrl : This is the list of non-referenced decks.

Required parameter.

referenced_deleted_list | rdl : This is the output produced by check_deleted_deck_list of all referenced decks which are being deleted.

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

Default if omitted: .INTVE

product_name | pn : This is the product name containing the source_library whose decks are being deleted.

If this parameter is specified, the procedure will produce a cross referenece of all decks on rdl.

referenced_deleted_xref | rdx : This is the cross references of all decks on rdl.

status : see NOS/VE error handling.

6.1.2 CHECK_REFERENCING_DECKS (CHERD)

The purpose of this procedure is to find all decks which have '*copy's of decks on one of the input lists. This procedure can take as input either a list of decks, or an SCU name change file.

```
CHECK_REFERENCING_DECKS
     source_library, sl : file = $required
     development_base | db : file = .INTVE
     deck_list, dl : file = $optional
     name_change_list, ncl : file = $optional
     referencing_decks_list : file = $local.referencing_decks_list
     status
```

source_library | sl : This is the source library which is to be searched for referencing decks.

Required parameter.

deck_list | dl : This is the list of deck names to check for references. There is one name per line.

name_change_list | ncl : File that contains name change directives for SCU. The names on the old name parameter are the ones that are checked.

referencing_decks_list | rdl : This is the list of decks which reference those on the input files.

status : see NOS/VE error handling.

6.1.3 FORMAT_LIBRARY (FORL)

The purpose of this procedure is to cybform all cybil decks on a source library in groups small enough to be run during one of the update intve jobs. This procedure is designed to work only within the SPECIAL_REQUESTS file.

```
FORMAT_LIBRARY
     starting_deck, sd : name = $optional
     number_of_decks, nod : integer = 500
     modification, m : name 1..7 = $required
     status
```

starting_deck | sd : This parameter specifies the deck on the source

library with which to start the current cybform group.

Omission causes first deck on the library to be used.

number_of_decks| nod : This parameter specifies the number of decks to cybform before terminating.

Omssion causes 500 to be used.

modification | m : The modification used for the cybform changes.

status : ses NOS/VE error handling.

6.1.4 GENERATE_DELETED_DECK_LIST (GENDDL)

The purpose of this procedure is to generate the list of decks which are currently candidates for deletion.

```
GENERATE_DELETED_DECK_LIST
     source_library, sl : file = $required
     development_base | db : file = .INTVE
     product_name, pn : name = os
     build_level, bl : name = $optional
     deleted_deck_list, ddl : file = $local.deleted_deck_list
     status : var of status = $optional
```

source_library | sl : This parameter specifies the source library from which the group 'DELETED_DECKS' is to be derived.

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 to generate the list for.

Omission causes OS to be used.

build_level | bl : The build level for the product at which to generate list.

deleted_deck_list | ddl : This is the list of deleted decks. This information is derived from the group DELETED_DECKS.

Omission causes $local.deleted_deck_list to be used.

6.0 PRODUCT SOURCE LIBRARY CLEANUP
6.1.4 GENERATE_DELETED_DECK_LIST (GENDDL)

status : see NOS/VE error handling.

6.1.5 GENERATE_MOD_STATE_CHANGE_LIST (GENMSCL)

The purpose of this procedure is to generate the commands needed to change all state 2 and state 3 modifications, which are members of a specified build level, to state 4.

```
GENERATE_MOD_STATE_CHANGE_LIST
     development_base | db : file = .INTVE
     product_name, pn : name = os
     build_level, bl : name = $optional
     state_change_list, scl : file = $local.state_change_list
     status
```

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

Default if omitted: .INTVE

product_name | pn : This specifies the product to generate the state change file for.

Omission causes OS to be used.

build_level | bl : This specifies the build level to generate the state changes for.

state_change_list | scl : This is the file containing the change_modification commands.

Omission causes $local.state_change_list to be used.

status : see NOS/VE error handling.

6.1.6 GENERATE_NON_REFERENCED_DECK_LIST (GENNRDL)

The purpose of this procedure is to generate a list of decks which are not referenced by any other deck on the source library.

```
GENERATE_NON_REFERENCED_DECK_LIST
     source_library, sl : file = $required
     list, l : file = non_referenced_decks
```

6.0 PRODUCT SOURCE LIBRARY CLEANUP
6.1.6 GENERATE_NON_REFERENCED_DECK_LIST (GENNRDL)

```
     status

source_library | sl : This is the source library whose decks are  to
     be checked.

     Required parameter.

list | l : This is the list of unreferenced decks.

     Omission causes $local.non_referenced_decks to be used.

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

6.1.7 VERIFY_NAMING_CONVENTIONS (VERNC)

This procedure enforces nosve naming conventions against either a list of deck names (one per line) or all decks on a source library.

```
VERIFY_NAMING_CONVENTIONS
     input_list, il : file = $required
     source_library, sl : file = $required
     non_conforming_decks, ncd : file = $local.non_conforming_decks
     status

input_list | il : The list of deck names to check.

     Required parameter.

source_library  |  sl  :  The  source library to check.  Either this
     parameter or the INPUT_LIST parameter is required.

     Required parameter.

non_conforming_decks | ncd : A list of any decks  which  don't  meet
     conventions.

     Omission causes $local.non_conforming_decks to be used.

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

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE

Integration maintains source code in scu libraries. Since code can be extracted and transmitted on more than one machine, the product source libraries must be reconciled daily. Each machine is working with its own source library, all source libraries are identical however. One machine is designated as the master machine and the others are slave machines. The machine that integration runs on is designated the master machine. Some would say this is because integration is the master. The master machine also is usually available more (all) of the time than the slave machine.

On each machine on which integration accepts code, there exists a set of 'product' catalogs containing a source_library, latest_changes and build_requests files. There also exists a special subcatalog called UPDATE_JOBS which contains files used by these procedures to determine what the designation of the current machine is. One of these files is SN_MID_CORRELATIONS. This file determines which machine in the network is the 'master' and which are 'slaves'. An example of this file is:

```
"This file contains the correlations between a machine's
"processor/serial number and it's RHF machine-id.
"This correlation is in the following format:
"  <processor/model number as a variable> = '<machine id>'
"This file also initializes a variable which contains the
"master machine's machine-id.
"This file also initializes a variable which contains a
"count of all of the slave machines in the list as an
" XDCLed variable.
"NOTE: The procedure update_intve will only accept as
"a slave machine initialization variable those
"lines which are less than 20 characters long. All
"processor/model number lines should be left justified
"all other lines (such as these comments) should be
"longer than 20 char or have 20 char of blanks in front
"of them.
P3S0002 = 'M02'
P3S006D = 'M09'
MASTER_MACHINE = 'M02'
create_variable slave_machine_count kind=integer ..
value=2 scope=sdcl
```

The critical elements in this file are the "P3S0002='M02'" and the "MASTER_MACHINE='M02'". When the values of these are equal it

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE

indicates the master machine, any other machines are slave machines. Another critical file is LAST_INTVE_UPDATE_DATE_TIME which contains a variable whose value is the date (ordinal) and time that the last update job was run on this machine and (only on slave machines) the time the master update was run as well. Other files which exist in this catalog are used to enforce the correct execution sequence for the procedures. This sequence is:

1. REMOTE_UPDATE_INTVE: run on all slave machines which have been active since the last update.
2. UPDATE_INTVE: run only on the master machine.
3. REMOTE_REINITIALIZE_INTVE: run on all slave machines.

These three procedures enforce the following algorithm for each product:

1. Merge latest_changes with the source library, remove decks whose interlocks do not match. The decks with nonmatching interlocks are saved. This is done on all slave machines and the master machine.

2. Merge the latest_changes from each slave machine and the master machine, removing duplicate decks. This detects extracting the same deck on two machines. This merging is done on the master machine.

3. Update the master source_library with the combined latest_changes from above step. This is also done on the master machine. The updated source library is transported to each slave machine.

4. Replace the slave source_library with the updated master source_library from above step. Make an empty latest_changes for use with the next cycle.

In addition to the source code, these procedures also handle the moving of the information for slave machine build_requests to the master machine.

These update jobs are designed to be run by the system operators after each day's activity is complete. Therefore these procedures must communicate with the operators to tell them of certain error conditions. This is done by refering the operators to a series of notes which give detailed information on the error and what to do about it. This list of notes looks like this:

NOTE 1

An error has occured during an attempt to backup the catalog

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE

:NVE.INTVE.UPDATE_JOBS.SLAVE_DATA_CATALOG to a tape. If this error message has appeard more than three times tonight, please terminate this job and call GEW at 3657(w) or 755-4841(h). If this error message has not appeared more than three times, please either move the tape being requested to another drive or clean the drive that the tape is on and then make a response to the request on the screen. The update job will then re-request the tape and try again.

NOTE 2

An error has occured in attempting a replace_file. There may be an error on either the 170 or 180 part of IRHF. Please check these items and then attempt:

```
  REPF :NVE.INTVE.UPDATE_JOBS.M09NAME TO=M09NAME
```

This must be done from the .INTVE catalog.
If this still doesn't work, please call GEW at 3657(w) or 755-4841(h).

NOTE 3

An unexpected error has occured during the slave_update job. Please call GEW at 3657(w) or 755-4841(h).

NOTE 4

The master_update job cannot find a file which it uses to communicate with the slave_update job. It is asking whether the machine, indicated by the machine_id, has been running closed shop today. You should answer either YES or NO as your reply.

NOTE 5

You responded YES to the request in note 4. Since the master_update job cannot find the communications file it is seeking, it is assuming that the slave_update job was not run the machine, indicated by the machine_id, and it is telling you that you must run that job. If that job has been run, please call GEW at 3657(w) or 755-4841(h).

NOTE 6

An error has occured during an attempt to load the catalog :NVE.INTVE.UPDATE_JOBS.SLAVE_DATA_CATALOG from a tape. If this error message has appeard more than three times tonight, please terminate this job and call GEW at 3657(w) or 755-4841(h). If this error message has not appeared more than three times, please either move the tape being requested to another drive or

clean the drive that the tape is on and then make a response to the request on the screen. The update job will then re-request the tape and try again.

NOTE 7

The master_update job could not replace some source_library from 180 to 170. This means that there is either a problem with replace_file (170 or 180 side) or a 170 disk full has occured. If you can, please correct the problem and give this request a go. If you cannot correct the problem, please terminate this job and call GEW at 3657(w) or 755-4841(h).

NOTE 8

An error has occured in attempting a replace_file. There may be an error on either the 170 or 180 part of IRHF. Please check these items and then attempt:

```
  REPF :NVE.INTVE.UPDATE_JOBS.MMNAME TO=MMNAME
```

This must be done from the .INTVE catalog.
If this still doesn't work, please call GEW at 3657(w) or 755-4841(h).

NOTE 9

An error has occured during an attempt to backup one of the intve source_library files updated today onto a tape. If this error message has appeard more than three times tonight, please terminate this job and call GEW at 3657(w) or 755-4841(h). If this error message has not appeared more than three times, please either move the tape being requested to another drive or clean the drive that the tape is on and then make a response to the request on the screen. The update job will then re-request the tape and try again.

NOTE 10

An unexpected error has occured during the master_update job. Please call GEW at 3657(w) or 755-4841(h).

NOTE 11

An unexpected error has occured during the slave_reload job. Please call GEW at 3657(w) or 755-4841(h).

NOTE 12

An unexpected error has occured during the resubmit_slave_update job. Please call GEW at 3657(w) or

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE

755-4841(h).

NOTE 13

An unexpected error has occured during the resubmit_slave_reload job. Please call GEW at 3657(w) or 755-4841(h).

NOTE 14

A file which contains the current password for .INTVE is messed up. Please call GEW at 3657(w) or 755-4841(h).

7.1 PARALLEL SOURCE LIBRARY MAINTENANCE PROCEDURES

7.1.1 COMBINE_LATEST_CHANGES (COMLC)

The purpose of this procedure is to reject non_matching interlock decks via lc_combine_reject and then to merge all latest_changes files for the current product and reject duplicated decks. This merged latest_changes is then combined with the source_library. This should NOT be used as a standalone procedure.

```
COMBINE_LATEST_CHANGES
     log_file, lf : string = $required
     development_base | db : file = .INTVE
     product_name, pn : name = $required
     working_catalog, wc : file or key none = $required
     status
```

log_file | lf : This is the file to put log information on.

Required parameter.

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 to be updated.

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE
7.1.1 COMBINE_LATEST_CHANGES (COMLC)

Required parameter.

working_catalog | wc : This is the catalog level pathname for files for the current product, eg .INTVE.OS.

Required parameter.

status : see NOS/VE error handling.

7.1.2 EXECUTE_UPDATE_JOBS (EXEUJ)

The purpose of this procedure is to submit a batch job to intve which will contain the invocation of the appropriate update procedures.

```
EXECUTE_UPDATE_JOBS
     job_class, jc : name
     force_resubmit_slave_update, frsu : boolean = false
     force_resubmit_slave_reload, frsr : boolean = false
     development_base, db : file = $fname(':'//$string($default_famil
<*
       y)//'.intve')
     development_base_user, dbu : name = intve
     development_base_family, dbf : $default_family
     status
```

job_class | jc : This parameter specifies the job class of the submitted job. Any system job class is appropriate.

force_resubmit_slave_update | frsu : This parmeter overrides the interlocking files and forces the remote_update_intve job to be run another time.

Omission causes FALSE to be used.

force_resubmit_slave_reload | frsr : This parmeter overrides the interlocking files and forces the remote_reintialize_intve job to be run another time.

Omission causes FALSE to be used.

development_base | db : This is the catalog that contains the integration libraries to update.

development_base_user | dbu : This is the user number portion of the development base path.

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE
7.1.2 EXECUTE_UPDATE_JOBS (EXEUJ)

development_base_family | dbf : This is the family portion of the development base path.

status : ses NOS/VE error handling.

7.1.3 LC_COMBINE_REJECT

The purpose of this procedure is to remove from latest_changes all decks whose interlocks do not match those of the source_library. This procedure is called by both combine_latest_changes and remote_update_intve. This should NOT be used as a standalone procedure.

```
LC_COMBINE_REJECT
     log_file, lf : string = $required
     development_base | db : file = .INTVE
     product_name, pn : name = OS
     working_catalog, wc : file or key none = .intve.os
     status
```

log_file | lf : This is the file to put log information on.

Required parameter.

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 to be updated.

Omission causes OS to be used.

working_catalog | wc : This is the catalog level pathname for files for the current product, eg .INTVE.OS.

Omission causes .INTVE.OS to be used.

status : see NOS/VE error handling.

7.1.4 MASTER_MISC_TRANSFERS (MASMT)

The purpose of this procedure is to enable the update jobs to easily move things from the master machine to the slave machine. This procedure works on the master machine in concert with another procedure on the slave machine.

This procedure will backup a set of items specified on a file called the transfer_file to a set of tapes specified on the vsns parameter. The items in the transfer file fall into one of three catagories: a file, a subcatalog, or a procedure or command line. The first two items follow a rigidly defined requirement in that they are preceeded by one of the key- words FILE or CATALOG. This keyword must be capitalized and followed by a single space which is followed by a file path - including family specifications. If a line does not start with one of these two prefixes; it is assumed to be a procedure call or command line and is executed by doing an include_line. A procedure must be capable of executing on both the master and slave machines. To make the determination of which machine the procedure is on; a special XDCLed variable is created in this procedure and in the SLAVE_MISC_TRANFERS procedure. This variable, named MACHINE, is of type string and has a value of either 'MASTER' or 'SLAVE'.

Along with the items being transfered is a copy of the transfer file itself. This is the first item on the transfer tape. It is used by the SLAVE_MISC_TRANSFERS procedure to determine the identity of the items to be reloaded.

See also: SLAVE_MISC_TRANSFERS

```
MASTER_MISC_TRANSFERS
     vsns, v: list of name 1 ..  6 = (mmxfr1 mmxfr2)
     transfer_file, tf: file
     development_base, db: file
     status
```

vsns | v : This parameter specifies the tape vsns used to transport the materials.

Omission causes MMXFR1 and MMXFR2 to be used.

transfer_file | tf : This is the file of items to be transfered.

Omission causes wev$development_base.update_jobs. transfer_file to be used.

development_base : specifies the catalog in which the transfer file

resides if it is not specified.

Default if omitted : INTVE.

status : see NOS/VE error handline.

7.1.5 REMOTE_REINITIALIZE_INTVE (REMRI)

The purpose of this procedure is load updated source libraries into the correct product catalogs and cleanup latest_changes and build_request files. The source libraries can come from 180 tape.

```
REMOTE_REINITIALIZE_INTVE
     development_base | db : file = .INTVE
     products, product, p : list of name = os
     repermit_files, rf : boolean = true
     set_remri_lock, srl : boolean = true
     expect_remui_lock, erl : boolean = true
     allow_cleanup_of_latest_changes, acolc, a : boolean = true
     cleanup_build_request_files, cbrf : boolean = true
     cleanup_special_request_files, csrf : boolean = true
     compare_commuication_files, ccf : boolean = true
     status
```

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

Default if omitted: .INTVE

products | product | p : This parameter specifies the product to be updated.

Omission causes ALL to be used.

repermit_files | rf : This parameter allows the readdition of the public access permits to be overridden (any public access other than read is normally denied while an update is going on).

Omission causes TRUE to be used.

WARNING: USE OF THIS PARAMETER IS DANGEROUS AND NOT RECOMMENDED.

set_remri_lock | srl : This parameter allows the creation of the file which indicates that a product has been reloaded and should not be updated again, to not be created.

Omission cuases TRUE to be used.

WARNING: USE OF THIS PARAMETER IS DANGEROUS AND NOT RECOMMENDED.

expect_remui_lock | erl : This parameter allows the check of the lock files created by remote_reupdate_intve to be skipped. This check is used to ensure that the source_library was updated and can now be cleaned up.

Omission cuases TRUE to be used.

WARNING: USE OF THIS PARAMETER IS DANGEROUS AND NOT RECOMMENDED.

allow_cleanup_of_latest_changes | acolc | a : This parameter allows the cleanup of information in latest_changes to be prevented. Note: if information is left in latest_changes and another update cycle is run, it will be rejected for non-matching interlocks.

Omission causes TRUE to be used.

cleanup_build_request_files | cbrf : This parameter allows the cleanup of information in the build request files to be prevented. Note: if information is left in a build request file and another update cycle is run, it will be added to the master machine's file twice.

Omission causes TRUE to be used.

cleanup_special_request_files | csrf : This parameter allows the cleanup of information in the special request files to be prevented. Note: if information is left in a special request file and another update cycle is run, it will be executed again.

Omission causes TRUE to be used.

compare_communication_files | ccf : This parameter causes the communication files passed via tapes and via nos to be compared to ensure that the correct tapes have been hung.

Omission causes TRUE to be used.

NOTE: USE OF THIS PARAMETER IS DANGEROUS AND NOT RECOMMENDED.

status : ses NOS/VE error handling.

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE
7.1.6 REMOTE_UPDATE_INTVE (REMUI)

7.1.6 REMOTE_UPDATE_INTVE (REMUI)

This procedure merges the latest_changes file onto a source library and rejects non-matching-interlock (nmi) decks (this is done by LC_COMBINE_REJECT) and positions the files (latest_changes, build_requests and logging) for pickup by UPDATE_INTVE by dumping to 180 tape.

```
REMOTE_UPDATE_INTVE
     development_base | db : file = .INTVE
     products, product, p : list of name = os
     delete_file_permits, dfp : boolean = true
     set_remui_lock, srl : boolean = true
     expect_remri_lock, erl : boolean = true
     dump_tape_vsns, dtv : list of string 1..6 = $optional
     status
```

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

Default if omitted: .INTVE

products | product | p : This parameter specifies the product to be updated.

Omission causes ALL to be used.

delete_file_permits | dfp : This parameter allows the removal of the public access permits to be overridden (any public access other than read is normally denied while an update is going on).

Omission causes TRUE to be used.

WARNING: USE OF THIS PARAMETER IS DANGEROUS AND NOT RECOMMENDED.

set_remui_lock | srl : This parameter allows the creation of the files (one per product) which indicate that a product has been updated and should not be updated again, to not be created.

Omission causes TRUE to be used.

WARNING: USE OF THIS PARAMETER IS NOT RECOMMENDED.

expect_remri_lock | erl : This parameter allows the check of the lock file created by remote_reinitialize_intve to be skipped. This check is used to ensure that the source_library was reloaded from the prevous cycle.

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE
7.1.6 REMOTE_UPDATE_INTVE (REMUI)

Omission causes TRUE to be used.

WARNING: USE OF THIS PARAMETER IS DANGEROUS AND NOT RECOMMENDED.

dump_tape_vsns | dtv : This parameter specifies the vsn's of the tapes to dump the files on for transfer to the master machine.

Omission causes files to be transferred via the 170 using get_file and replace_file.

status : NOS/VE status conventions.

7.1.7 SLAVE_MISC_TRANSFERS (SLAMT)

This procedure works with MASTER_MISC_TRANSFERS to complete the moving of miscellanious files from the master machine to the slave machines each time the update jobs are run.

This procedure will restore all items listed on the transfer file and substitute the current machine's development base for the one specified on the master machine. This procedure will also execute as commands any lines not starting with either FILE or CATALOG - WITHOUT CHANGE. If a procedure is expected to work on a file specified as a parameter; it's slave version should take care of the conversions itself.

Also see MASTER_MISC_TRANSFERS.

```
SLAVE_MISC_TRANSFERS
     vsns, v: list of name 1 ..  6 = mmxfr1
     master_transfer_file, mtf: file = $required
     master_development_base, mdb: file
     development_base, db: file
     status
```

vsns | v : This parameter specifies the tape vsns used to transport the materials.

Omission causes MMXFR1 and MMXFR2 to be used.

master_transfer_file | mtf : This is the file of items to be transfered. This is the name of the file backed up onto the tape by the master machine job.

Omission causes master_development_base.update_jobs.

transfer_file to be used.

master_development_base : This is the development_base parameter of the master machine. Any files or catalogs with this base path will be reloaded into corresponding files or catalogs under the regular development base.

Default if omitted: the normal development base.

development_base : This is the new file path which to which items dumped from the master machine will be reloaded.

Default if omitted : INTVE.

status : see NOS/VE error handline.

7.1.8 UPDATE_INTVE (UPDI)

This is the master update job. It will call UPDATE_INTVE_PRODUCT for each product in the intve catalog and then call UPDATE_INTVE_TAPE_DUMP to dump all of the updated source_libraries to tape (if specified).

```
UPDATE_INTVE
     development_base | db : file = .INTVE
     products, product, p : list of name = os
     include_special_request_updates, isru : boolean = true
     include_manual_updates, imu : boolean = false
     dump_tape_vsns, dtv : list of string 1..6 = $optional
     compare_communication_files, ccf : boolean = true
     status
```

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

Default if omitted: .INTVE

products | product | p : This parameter specifies the product to be updated.

Omission causes ALL to be used.

include_special_requests_updates | isru : This parameter allows the special_requests file of changes to be excluded.

Omission causes TRUE to be used.

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE
7.1.8 UPDATE_INTVE (UPDI)

include_manual_updates | imu : This parameter can cause the procedure to put the user into an ENTWE/EXIWE type environment on the product source_library. This parameter is NOT recommended. The special_requests mode is the prefered method of making changes to the source_library.

Omission causes FALSE to be used.

dump_tape_vsns | dtv : This parameter specifies the vsn's of the tapes to dump the source_libraries on for transfer to the slave machines.

Omission cuases files to be transferred via the 170 using get_file and replace_file.

compare_communication_files | ccf : This parameter causes the communication files passed via tapes and via nos to be compared to ensure that the correct tapes have been hung.

Omission causes TRUE to be used.

NOTE: USE OF THIS PARAMETER IS DANGEROUS AND NOT RECOMMENDED.

status : see NOS/VE error handling.

7.1.9 UPDATE_INTVE_PRODUCT (UPDIP)

This procedure updates one product. It will call combine_latest_changes to merge latest_changes and source_library and will handle build_requests and logging files for this product. This procedure is not normally a standalone procedure, but is a subroutine to UPDATE_INTVE.

```
UPDATE_INTVE_PRODUCT
     product_index, pi : integer = $required
     development_base | db : file = .INTVE
     product_name, pn : name = $required
     working_catalog, wc : file = $required
     include_special_request_updates, isru : boolean = true
     include_manual_updates, imu : boolean = false
     dump_to_tape, dtt : boolean = false
     status
```

product_index | pi : This parameter specifies the index of the current product in the .intve catalog (ie. is it the 1st, 2nd or 100th product catalog).

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE
7.1.9 UPDATE_INTVE_PRODUCT (UPDIP)

Required parameter.

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 to be updated.

Required parameter.

working_catalog | wc : This is the catalog level pathname for files for the current product, eg .INTVE.OS.

Required parameter.

include_special_requests_updates | isru : This parameter allows the special_requests file of changes to be excluded.

Omission causes TRUE to be used.

include_manual_updates | imu : This parameter can cause the procedure to put the user into an ENTWE/EXIWE type environment on the product source_library. This parameter is NOT recommended. The special_requests mode is the prefered method of making changes to the source_library.

Omission causes FALSE to be used.

dump_to_tape | dtt : This parameter is used to tell UPDATE_INTVE_PRODUCT not to replace the source library to the 170; but to leave it for later dumping onto 180 tape.

Omission causes FALSE to be used.

status : ses NOS/VE error handling.

7.1.10 UPDATE_INTVE_TAPE_DUMP (UPDITD)

The purpose of this procedure is to dump all updated source libraries and then to add public file permits to them. This procedure is called by UPDATE_INTVE. This procedure CAN be used independantly.

UPDATE_INTVE_TAPE_DUMP

7.0 PARALLEL SOURCE LIBRARY MAINTENANCE
7.1.10 UPDATE_INTVE_TAPE_DUMP (UPDITD)

dump_tape_vsns, dtv : list of string 1..6 = $optional
development_base | db : file = .INTVE
status

dump_tape_vsns | dtv : This parameter specifies the vsn's of the tapes to dump the source_libraries on for transfer to the slave machines.

Omission causes files to be transferred via the 170 using get_file and replace_file.

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

Default if omitted: .INTVE

status : see NOS/VE error handling.

8.0 USE OF TAPE LIBRARY UTILITY

8.0 USE OF TAPE LIBRARY UTILITY

8.1 EDIT_TAPE_LIBRARY UTILITY

This utility will maintain information for a collection of magnetic tapes.

The following information is stored for every tape:

VOLUME_SERIAL_NUMBER: This is a one to six character name that must be unique within a specific tape_library. All routines will search through the list of tapes in ascending order according to the VSN.

GROUP: This name is used to collect several tapes with some common factor in to a unit.

CONTENTS: This name should be used to uniquely describe the contents of an individual tape within a group. The main purpose of this field is to provide the capability of determining the VSN of a specific tape given the group and contents fields.

DENSITY: This name should describe the density of the tape.
EXAMPLE: MT9$6250
MT9$1600
GE

FORMAT: This name should describe the format of the tape.
EXAMPLE: NOS_INTERNAL
NOS_SYSTEM_INTERNAL
NOSVE_BACKUP

USER: This name shows the user name of the last job that modified this tape. The following commands will update this field: reserve_tape, release_tape and change_tape.

LABELLED: This boolean field determines if there is a physical label on the magnetic tape.

RECORDED_VSN: This field contains the value of the physical label on a labelled magnetic tape.

8.0 USE OF TAPE LIBRARY UTILITY

8.1 EDIT_TAPE_LIBRARY UTILITY

CREATION_DATE_TIME: This field contains the date and time the tape was initially created or converted from a version_1 tape_library.

MODIFICATION_DATE_TIME: This field contains the date and time a tape was last modified. The following commands will update this field: create_tape, reserve_tape, release_tape and change_tape.

TAPE_DESCRIPTION: This field should be used to describe the contents of a tape.

8.2 EDIT_TAPE_LIBRARY UTILITY COMMANDS

8.2.1 EDIT_TAPE_LIBRARY

The edit_tape_library utility is entered with the EDIT_TAPE_LIBRARY command. While within the utility the input prompt is "ETL".

```
EDIT_TAPE_LIBRARY, EDITL
     status : status_variable
```

status : see NOS/VE error handling.

8.2.2 ADD_LIBRARY

Loads a tape library into the utility.

```
ADD_LIBRARY, ADDL
     library, l : file = $required
     result, r : file = $optional
     status : var of status = $optional
```

library | l : Specifies the file name of the tape library.

Required parameter.

result | r : Specifies the file name of the new tape library.

status : see NOS/VE error handling.

8.2.3 CHANGE_TAPE

Changes the contents of one or more tape library entries.

```
CHANGE_TAPE, CHANGE_TAPES, CHAT
     volume_serial_number, vsn : list range of name 6 = $required
     group, g : name = $optional
     contents, c : name = $optional
     format, f : name = $optional
     density, d : name = $optional
     labelled, l : boolean = no
     recorded_vsn, rvsn : name 1..6 or key none
     tape_description, td : string = $optional
     status : var of status = $optional
```

volume_serial_number | vsn: Tapes whose entries are changed. You can specify a list of one or more names or a list of one or more ranges.

Required parameter.

group | g : New group.

If GROUP is omitted, the group field is not changed.

contents | c : New contents.

If CONTENTS is omitted, the contents field is not changed.

format | f : New format.

If FORMAT is omitted, the format field is not changed.

density | d : New density.

If DENSITY is omitted, the density field is not changed.

labelled | l : New label presence value.

If LABEL is omitted, the label field is not changed.

recorded_vsn | rvsn : New recorded_vsn.

If RECORDED_VSN is omitted, the recorded_vsn field is not changed.

tape_description | td : New tape description.

8.0 USE OF TAPE LIBRARY UTILITY
8.2.3 CHANGE_TAPE

If TAPE_DESCRIPTION is omitted, the tape description field is not changed.

status : see NOS/VE error handling.

8.2.4 CREATE_TAPE

Creates one or more tape_library entries.

CREATE_TAPE, CREATE_TAPES, CRET
volume_serial_number, vsn : list range of name 6 = $required
status : var of status = $optional

volume_serial_number | vsn : Tapes whose entries are created. You can specify a list of one or more names or a list of one or more ranges.

Required parameter.

status : see NOS/VE error handling.

8.2.5 DELETE_TAPE

Delete one or more tapes from the tape library. Only tapes with the group AVAILABLE can be deleted. You must release a tape before deleting it.

DELETE_TAPE, DELETE_TAPES, DELT
volume_serial_number, vsn : list range of name 6 = $required
status : var of status = $optional

volume_serial_number | vsn : Tapes whose entries are deleted. You can specify a list of one or more names or a list of one or more ranges.

Required parameter.

status : see NOS/VE error handling.

8.2.6 DISPLAY_GROUP

Display all tapes associated with the specified group or groups.

```
DISPLAY_GROUP, DISPLAY_GROUPS, DISG
     group, g : list of name = $required
     display_option, do : key brief, b, full, f = brief
     output, o : file = $output
     status : var of status = $optional
```

group | g : List of one or more group names.

display_option | do : Specified the information listed.

brief | b : Lists only the volume_serial_number, group and tape_description.

full | f : Lists all information known about the tapes.

Omission causes brief to be used.

output | o : specifies file to which the output will be written.

Omission causes $output to be used.

status : see NOS/VE error handling.

8.2.7 DISPLAY_GROUP_LIST

List the groups in the library.

```
DISPLAY_GROUP_LIST, DISGL
     output, o : file = $output
     status : var of status = $optional
```

output | o : specifies file to which the output will be written.

Omission causes $output to be used.

status : see NOS/VE error handling.

8.2.8 DISPLAY_LIBRARY

Displays the library header of the working library.

```
DISPLAY_LIBRARY, DISL
     output, o : file = $output
     status : var of status = $optional
```

output | o : specifies file to which the output will be written.

Omission causes $output to be used.

status : see NOS/VE error handling.

8.2.9 DISPLAY_TAPE

Displays one or more tapes.

```
DISPLAY_TAPE, DISPLAY_TAPES, DIST
     volume_serial_number, vsn : list range of name 1..6 = $required
     display_option, do : key brief, b, full, f = brief
     output, o : file = $output
     status : var of status = $optional
```

volume_serial_number | vsn : Tapes that are displayed. You can specify a list of one or more names or a list of one or more ranges.

Required parameter.

display_option | do : Specified the information listed.

brief | b : Lists only the volume_serial_number, group and tape_description.

full | f : Lists all information known about the tapes.

Omission causes brief to be used.

output | o : specifies file to which the output will be written.

Omission causes $output to be used.

status : see NOS/VE error handling.

8.2.10 DISPLAY_TAPE_LIST

List the tapes on the working library.

```
DISPLAY_TAPE_LIST, DISTL
     output, o : file = $output
     status : var of status = $optional
```

output | o : specifies file to which the output will be written.

Omission causes $output to be used.

status : see NOS/VE error handling.

8.2.11 RELEASE_TAPE

Cancel the current reservation of the specified tapes and make the tapes available for future reservation.

```
RELEASE_TAPE, RELEASE_TAPES, RELT
     volume_serial_number, vsn : list range of name 6 = $optional
     group, g : list of name = $optional
     status : var of status = $optional
```

volume_serial_number | vsn : Tapes that are released. You can specify a list of one or more names or a list of one or more ranges.

group | g : Groups that are released. You can specify a list of one or more names.

status : see NOS/VE error handling.

8.2.12 RESERVE_TAPE

Reserves tapes on the working library.

```
RESERVE_TAPE, RESERVE_TAPES, REST
     group, g : name = $required
     contents, c : name = $optional
     format, f : name = $optional
     density, d : name = $optional
```

8.0 USE OF TAPE LIBRARY UTILITY
8.2.12 RESERVE_TAPE

```
     labelled, l : boolean = no
     recorded_vsn, rvsn : name 1..6 or key none = none
     tape_description, td : string = $optional
     number, n : integer 1..65535 = 1
     volume_serial_number, vsn : var of string = $optional
     status : var of status = $optional
```

group | g : The group name associated with the tape(s).

Required parameter.

format | f : Optional format description.

density | d : Optional density description.

labelled | l : Label type on tape requested.

If LABELLED is omitted, an unlabelled tape will be reserved.

recorded_vsn | rvsn : The value found on the physical label on a labelled magnetic tape.

Only labelled tapes are allowed to have a RECORDED_VSN. If the RECORDED_VSN is not specified, but the tape is reserved as a labelled tape, the recorded_vsn will be set to the value of the external_vsn.

tape_description | td : Optional tape description.

number | n : Number of tapes to reserve.

If NUMBER is omitted, one tape will be reserved.

volume_serial_number | vsn : The volume serial number of the first tape reserved will be returned to the SCL string variable specified.

status : see NOS/VE error handling.

8.2.13 QUIT

Write the working library to the specified file (if a result file was specified on the ADD_LIBRARY command) and exit the utility. This is the recommended method of rewriting the library.

QUIT

8.0 USE OF TAPE LIBRARY UTILITY
8.2.13 QUIT

```
     write_library, wl : boolean
     status : var of status = $optional
```

write_library | wl : This parameter is used to inhibit the generation of the result file if desired. (Attempting to do WL=TRUE without a result file will cause an error.) The default is whither the result library has been specified or not.

status : see NOS/VE error handling.

8.2.14 WRITE_LIBRARY

Write the working library to the specified file. NOTE: This command is not recommended. A RESULT file should be specified on the ADD_LIBRARY command. This will cause the quit command to write the new library.

```
WRITE_LIBRARY, WRIL
     result, r : file = $required
     status : var of status = $optional
```

result | r : File on which to write the working library.

Required Parameter.

status : see NOS/VE error handling.

8.3 EDIT_TAPE_LIBRARY UTILITY FUNCTIONS

8.3.1 $GROUP_LIST

Returns an array of strings listing the names of groups on the working library.

```
$GROUP_LIST, $GL
```

No Parameters.

8.3.2 $GROUP_MEMBERS

Returns an array of strings listing the volume serial numbers of the tapes that are members of the specified group on the working library.

```
$GROUP_MEMBERS, $GM
     name
     parameter 1 : name
```

parameter 1 : Name of the requested group.

Required Parameter.

8.3.3 $TAPE_DATA

Returns a string containing the contents of any field for the specified tape.

```
$TAPE_DATA, $TD
     tape name, field name
     parameter 1 : name = $required
     parameter 2 : key group, g, contents, c, format, f, density,
              d, labeled, l, user, u, creation_date, cd,
              creation_time, ct, modification_date, md,
              modification_time, mt, recorded_vsn, rvsn,
              tape_description, td = $required
```

parameter 1 : Name of the tape whose field is returned.

Required Parameter.

parameter 2 : Name of field to return.

group | g : Group associated with the specified tape.

contents | c : Tape contents.

format | f : Tape format.

density | d : Tape density.

labelled | l : Tape label presence.

user | u : Last user number to modify tape.

8.0 USE OF TAPE LIBRARY UTILITY
8.3.3 $TAPE_DATA

creation_date | cd : Date tape was created.

creation_time | ct : Time tape was created.

modification_date | md :Date tape was last modified.

modification_time | mt : Time tape was last modified.

recorded_vsn | rvsn : The recorded_vsn of the tape.

tape_description | td : Description of contents of tape.

Required Parameter.

8.3.4 $TAPE_LIST

Returns an array of strings listing the tapes on the working library.

$TAPE_LIST, $TL

No Parameters.

8.3.5 $VOLUME_SERIAL_NUMBER

Returns a string containing the volume serial number for the tape with the specified GROUP and CONTENTS fields.

$VOLUME_SERIAL_NUMBER, $VSN
group name, contents name
parameter 1 : name
paremeter 2 : name

parameter 1 : Name of group on tape to be found.

Required Parameter.

parameter 2 : Name of contents on tape to be found.

Required Parameter.

9.0 DESCRIPTION OF PROCEDURES

9.0 DESCRIPTION OF PROCEDURES

9.1 SYSTEM BUILD PROCEDURES

The following are procedures that are used for building the operating system and products.

9.1.1 ASSIGN_AND_RETURN_MODIFICATION (ASSARM)

This procedure is used exactly like assign_modification with the following execptions.

1. The count parameter is eliminated. Only one modification can be created.
2. The created modification is returned as a string in 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)
     modificaton, m: var of string = $required
     status
```

feature | f : specifies the feature with which the modification is to be associated.

Required parameter.

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

Required parameter.

output | o : specifies the file that the created modification is written to.

9.0 DESCRIPTION OF PROCEDURES
9.1.1 ASSIGN_AND_RETURN_MODIFICATION (ASSARM)

Omission causes $output to be used.

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

Omission causes $JOB(USER) to be used.

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

modification | m : specifies the variable in which the modification name is returned.

Required parameter.

status : see NOS/VE error handling.

9.1.2 BACKUP_OS_BUILD_LEVEL (BACOBL)

The purpose of this procedure is to provide a mechanism for making a backup file of an "extended" build level. An extended build level is an os level along with the matching product levels from a selected set of products.

```
backup_os_build_level
     backup_file, bf : file = $local.bacobl
     associated_products_list, apl : list of name or key all =
     $optional
     output, o : file = $output
     development_base, db : file = .intve
     product_name, pn : name = os
     build_level, bl : name = $optional
     status
```

backup_file | bf : This is the file upon which the backup will be written. If associated_products_list is ALL, this is required to be a tape file.

associated_products_list | apl : This is a list of product names whose builds are to be included in the backup. The appropriate level is selected from the tying file in the os build selected.

output | o : This is the file to recieve the listing of what is backed up.

9.0 DESCRIPTION OF PROCEDURES
9.1.2 BACKUP_OS_BUILD_LEVEL (BACOBL)

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

product_name | pn : specifies product name from which the build_level is derived. This feature is included to accomodate paralled os catalogs within one development base catalog.

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

status : See NOS/VE error handling.

9.1.3 BACKUP_PRODUCT_LEVELS (BACPL)

The purpose of this procedure is to backup the build subcatalogs of all products in base_catalog and who have a specified product_level. This procedure will skip the changes and installation subcatalogs.

```
backup_product_levels
     base_catalog, bc : file = .intve
     product_levels, product_level, pl : list of name = $optional
     backup_file, bf : file = $local.backup
     output, o : file = $output
     status
```

base_catalog | bc : This is the catalog in which all of the candidate product catalogs reside.

product_level | pl : This is the level of each product to dump (if it exists).

backup_file | bf : This is the file upon which the backup will be written. If associated_products_list is ALL, this is

output | o : This is the file to recieve the listing of what is backed up.

status : See NOS/VE error handling.

9.1.4 BUILD_SYSTEM (BUIS)

Build_system builds both primary and secondary builds. With secondary builds the groups and modifications are not updated. The build decks are backward driven include_feature commands. Secondary builds are a tool to allow pre-integration of features.

This command can be called again to add or pull features in a build.

The following conventions must be followed:

1. A build level name must be of the form build_xxxxx.
2. If the last 2 characters are greater than 70, then the build is considered to be a secondary build. Otherwise is a primary build.
3. Primary builds must have a previous build level decks.
4. Creation_call parameter must be set to false once the build_decks have been created.

This procedure completes the build up to compilation. The following steps are performed:

1. The build_request file is updated.
2. The wef$feature_list file is updated.
3. Update_build_decks is run for features and then for decks.
4. Generate_library_changes is run to update the groups for new and deleted_decks, and change the states of the modifications.
5. Transmit the build decks.
6. Start a task to run make_build_jobs. Do not logout before the task returns its status.

```
build_system
     creation_call, cc: boolean = true
     action, a: key building, pulling = building
     feature_list, fl: file = $null
     predecessor_build_level, pbl: name = $required
     current_build_level, cbl: name = $required
     successor_build_level, sbl: name = none
     selection_criteria, sc: file = $null
     build_job_catalog, bjc: file = $user.current_build_level.product
<*
     _name
     submit_jobs, submit_job, sj: key immediate, i, delay, d,
     no_submit, ns = immediate
     password, pw: name = $required
     product_name, pn: name = os
     working_catalog, wc: file = none
```

9.0 DESCRIPTION OF PROCEDURES
9.1.4 BUILD_SYSTEM (BUIS)

os_build_level, obl: name = $required
status

creation_call | cc : specifies rather this the first or a subsequent call to build_system. Creation call should alway be false once the build_decks have been created.

action | a : specifies the action to be performed.

predecessor_build_level | plb : specifies the previous build level to the current build.

current_build_level | cbl : specifies the build level that is being built.

successor_build_level | sbl : specifies the successor build level to the current build.

selection_criteria | sc : specifies a criteria file that is appended to the criteria file for the call to make_build_jobs.

build_job_catalog | bjc : specifies the value for the corresponding parameter for the call to make_build_jobs.

submit_jobs | sj : specifies the value for the corresponding parameter for the call to make_build_jobs.

password | pw : specifies the value for the corresponding parameter for the call to make_build_jobs.

product_name | pn : specifies the product to be built.

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

os_build_level | obl : specifies the value for the corresponding parameter for the call to make_build_jobs. The default value is the same as the current build level.

status : see NOS/VE error handling.

9.1.5 BUILD_SYSTEM_2 (BUIS2)

This procedure is designed to build the NOS/VE operating system and all the products whose source is maintained by the Arden Hills NOS/VE integration group.

9.0 DESCRIPTION OF PROCEDURES
9.1.5 BUILD_SYSTEM_2 (BUIS2)

The following steps should be completed before running this procedure.

1) Update the tying_file.
2) Execute Update_integration_descriptors
3) Execute Update_build_information
4) Run build_system on all the products being built and
   wait until the compilations have completed with no errors.

This procedure can be started and stopped at any of the starting_positions defin The starting_position parameter indicates which is the first operation build_sys will attempt and the exit_position parameter indicates which is the last operati build_system_2 will attempt. The display_starting_position parameter will displa the possible starting_positions along with an integer index. Either the name of starting_position or the index may be used for the starting_position and exit_po parameters.
The output from this procedure and all the procedures it calls is collected ont is not specified then the output is placed on file buis2_output.$eoi in the OS b

The list of operations which this procedure will perform is:

1. - RESERVE_TAPES
2. - GENERATE_DELETE_MODULE_CMDS
3. - LINK_EI
4. - BUILD_NAMVE
5. - LINK_OPERATING_SYSTEM
6. - GENERATE_CIP_COMPONENTS
7. - GENERATE_DEADSTART_FILE
8. - LINK_170
9. - GENERATE_NOS_DEADSTART_TAPE
10. - BUILD_OCU
11. - BUILD_TDU
12. - BUILD_SCU
13. - BUILD_AV
14. - BUILD_CML
15. - BUILD_DESKTOP_ENVIRONMENT
16. - BUILD_DUMP_ANALYSIS_PROCS
17. - BUILD_HPA
18. - BUILD_KERMIT
19. - BUILD_MAILVE
20. - BUILD_MAILVE_V2
21. - BUILD_MALET
22. - BUILD_MANUALS
23. - BUILD_MENU_VE
24. - BUILD_PERF_TOOL
25. - BUILD_PFTF
26. - BUILD_PROCS
27. - BUILD_SVS

9.0 DESCRIPTION OF PROCEDURES
9.1.5 BUILD_SYSTEM_2 (BUIS2)

28. - BUILD_SYSTEM_DOC
29. - BUILD_SYSTEM_TESTS
30. - BUILD_TEST_TOOLS
31. - GATHER_OS_BUILD_LIBRARIES
32. - GENERATE_RELEASE_SOURCE_LIBRA
33. - GENERATE_FULL_PRODUCT_TAPE
34. - GENERATE_SHORT_PRODUCT_TAPE
35. - BUILD_TEST_TAPE
36. - BUILD_CIP_TAPES
37. - MERGE_DESTINATION_LIBRARIES
38. - BUILD_ANAD_COMMAND_LIBRARY
39. - GENERATE_BUILD_REPORT
40. - STANDARDIZE_LIBRARIES
41. - GENERATE_DUAL_STATE_FILE
42. - VERIFY_BUILD_COMPLETION
43. - BACKUP_SYSTEM_TO_TAPE

```
build_system_2
  starting_position, sp : any = reserve_tapes
  exit_position, ep : any = move_linker_files
  display_starting_positions, dsp : boolean = false
  output, o : file = $optional
  product_name, pn : name = os
  build_level, bl : name = $optional
  working_catalog, wc : file or key none = none
  working_build_level, wbl : name = object
  status : var of status = $optional
```

starting_position | sp : This is point in the menu to pick up the build from.

exit_position | ep : This is the point to end the build run on.

display_starting_positions | dsp : This prints the menu above out.

output | o : This is the file that the build results are printed to.

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.

9.0 DESCRIPTION OF PROCEDURES
9.1.5 BUILD_SYSTEM_2 (BUIS2)

status : See NOS/VE error handling.

9.1.6 CHECK_ALTERNATE_PRODUCTS (CHEAP)

This procedure is a subroutine of BUILD_SYSTEM. It's purpose is to check a list of alternate products to see if any decks on any of their source libraries should be recompiled because of new features on the product on which BUILD_SYSTEM is being run.

```
check_alternate_products
  build_requests, br : file = $required
  output, o : file = $response
  product_name, pn : name = $optional
  status : var of status = $optional
```

build_requests | br : This is the subset of the main build requests file which contains the information for the new features being added.

output | o : This is the file to write the list of products that must be recompiled to.

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

status : See NOS/VE error handling.

9.1.7 CHECK_CYBIL_COMMON_COMPILATION (CHECCC)

If check_compile_in_combination is indicated, all cybil common decks will be expanded with a module/modend block around them and a compilation attempted. If check_compile_in_isolation is selected, each deck is expanded individually (with MODULE/MODEND) and compiled. Isolation tests that each deck is 'complete' in that it has *copy's of all common decks it references. Combination tests for conflicts between deck defintions.

```
check_cybil_common_compilation
  library, l : file = $optional
  check_compile_in_combination, ..
  ccic : boolean = true
  check_compile_in_isolation, ..
```

9.0 DESCRIPTION OF PROCEDURES
9.1.7 CHECK_CYBIL_COMMON_COMPILATION (CHECCC)

```
  ccii : boolean = true
  status : var of status = $optional
```

library | l : This is the library of common decks to check.

check_compile_in_combination | ccic : See procedure description.

check_compile_in_isolation | ccii : See procedure description.

status : See NOS/VE error handling.

9.1.8 CHECK_FEATURE_LIST (CHEFL)

This procedure verifies that all modifications associated with a feature in the feature list are at the specified state or all are at the same state. Modifications not at the specified state are listed on $output.

```
check_feature_list
     feature_list, fl: file = $required
     state: integer 0 ..  4 or key same_state = 2
     product_name, pn: name = os
     status
```

feature_list | fl : specifies the file containing the feature names to check, one feature name per line.

Required parameter.

state | specifies the state all modifications are expected to be at or the key SAME_STATE. If the key SAME_STATE is specified the state of all modifications must be at the state of the first modification checked.

Omission causes state 2 to be used.

product_name | pn : specifies the product to which the features to be checked apply.

Omission causes OS to be used.

status | see NOS/VE error handling.

9.1.9 CHECK_TRANSMITTAL (CHET)

This procedure checks the code submitted with a transmittal. Pertinant information about the features is listed on $output to be checked visually. Information listed is the associated modification headers, new deck headers and modified decks. If specified the modified decks and optionally copying decks can be compiled. If the compile option is selected the product_name and target_build_level parameters are used to form the file path to a feature list that will be used when compiling. The path name for the feature list is $user.target_build_level.product_name.feature_list. This is the feature list of features already selected for this build, features that are accepted for a build have to be manually moved to this file.

```
check_transmittal
     feature_list, fl: file = $required
     include_copying_decks, icd: boolean = true
     compile, c: boolean = false
     selection_criteria, sc: file = $null
     development_base | db : file = .INTVE
     product_name, pn: name = os
     base_build_level, bbl: name = $required
     target_build_level, tbl: name = $required
     status
```

feature_list | fl : specifies the file containing the feature names to check, one feature name per line.

Required parameter.

include_copying_decks | icd : specifies whether copying decks should be compiled if the compile option is selected.

Omission causes false to be used.

compile | c : specifies whether to compile decks that are modified or affected by the specified features.

Omission causes false to be used.

selection_criteria | sc : specifies selection criteria to be used when compiling.

Omission causes $null to be used.

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

9.0 DESCRIPTION OF PROCEDURES
9.1.9 CHECK_TRANSMITTAL (CHET)

Default if omitted: .INTVE

product_name | pn : specifies the product to check.

Omission causes OS to be used.

base_build_level | bbl : specifies what build level to use for compiling.

Required parameter.

target_build_level | tbl : specifies the target build for the features being checked.

Required parameter.

status : see NOS/VE error handling.

9.1.10 CLEAN_OUT_BUILD_LEVEL (CLEOBL)

The purpose of this procedure is to remove all files from an os build level catalog except linkmap-type files, the tying file and the tools command libraries.

```
clean_out_build_level
  build_level, bl : name = $optional
  status : var of status = $optional
```

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

status : See NOS/VE error handling.

9.1.11 CLEAN_UP_PARTIAL_BUILDS (CLEUPB)

The purpose of this procedure is to delete the partial build catalogs that are no longer being used by development. Partial builds need to be accessed within a months time or they are considered unused by development. The procedure first finds the list of partial builds by using the procedure RETURN_BUILD_LIST.

9.0 DESCRIPTION OF PROCEDURES
9.1.11 CLEAN_UP_PARTIAL_BUILDS (CLEUPB)

Then it loops through the partial build catalogs and uses the commands INCLUDE_CYCLES and BACKUP_CATALOG, inside of the BACKUP_PERMANENT_FILES utility, to backup anything in the partial build catalog that has been accessed within the past month. It then determines if anything was backed up or not by searching for 'NUMBER OF CYCLES BACKED UP: 0'. If this string is found then the partial build has not been accessed and is deleted.

```
clean_up_partial_builds
  development_base, db : file = .intve
  product_name, pn : name = os
  status : var of status = $optional
```

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

Omission causes .INTVE to be used

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

Omission causes OS to be used

status : See NOS/VE error handling.

9.1.12 CLEANUP_BUILD_ENVIRONMENT (CLEBE)

The purpose of this procedure is to delete OS build catalogs and their associated products. Delete_os_build_level parameter is a list of OS build catalog names. Two types of deletion can be specified by type_of_deletion parameter. 'ALL' means that the whole OS build catalog will be deleted and it's associated products. If 'PARTIAL' is specified all of the files, except system_ debug_table, tying, build_report, boot_debug_table, and command_library are deleted from the OS build catalog. OS associated products are also deleted when 'PARTIAL' is specified.

```
cleanup_build_environment
  dobl : list of name = $optional
  type_of_deletion, tod : key all, partial = partial
  status : var of status = $optional
```

dobl : The os build levels to clean up.

type_of_deletion | tod : See procedure description

9.0 DESCRIPTION OF PROCEDURES
9.1.12 CLEANUP_BUILD_ENVIRONMENT (CLEBE)

status : See NOS/VE error handling.

9.1.13 BACKUP_BUILD_ENVIRONMENT (BACBE)

The purpose of this procedure is to capture a subset build environment for a spe build level. This environment gives the following capabilities.

a) Transmitting to and extracting from the os source_library
b) Compiling 180 decks.
c) Linking the boot and the OS.
d) Generating OS Deadstart tapes.
e) Generating Cip tapes. Plus all the commands used to build a system are available.

```
backup_build_environment
  backup_file, bf : file = $required
  list, l : file = $output
  build_level, bl : name = $optional
  status : var of status = $optional
```

backup_file | bf : This is the file to back up the build environment onto.

list | l : This is file that receives the output from the backup.

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

status : See NOS/VE error handling.

9.1.14 COMBINE_CATALOGS (COMC)

The purpose of this procedure is to combine one catalog onto another, such that file cycles which are duplicated in both catalogs are replaced.

```
combine_catalogs
     from, f : file = $required
     to, t : file = $required
     combine_subcatalogs, cs : boolean = no
     output, o : file = $output
```

status

from | f : This parameter identifies the catalog which is to be added to the "to" catalog.

to | t : This parameter specifies the catalog which is being changed.

combine_subcatalogs | cs : This parameter controls whither the procedure recurses or not.

output | o : This specifies the file which is to recieve the status messages from the combine.

status : See NOS/VE error handling.

9.1.15 COMBINE_PRODUCT_MAINT_LIBRARIES (COMPML)

The purpose of this procedure is to combine selected files which have components that originate from several products. An example of this is osf$command_library, which contains the program descriptors and message templates from every product which produces them.

```
combine_product_maint_libraries
     library, l : name = $required
     base_search_catalog, bsc : file = .intve
     os_build_level, obl : name = $optional
     include_products, include_product, ip : list of name or key all
     = all
     exclude_products, exclude_product, ep : list of name or key
     none = none
     library_name_and_product, lnap : list 1..$max_value_sets, 1..2
     of any = $optional
     result_library, rl : file = $fname(wev$working_catalog//
     '.maintenance.library.$next')
     development_base, db : file = .intve
     feature_catalog, fc : file = none
     feature_build_level, fbl : name = object
     working_catalog, wc : file = none
     working_build_level, wbl : name = object
     status
```

library | l : This is the file to search for.

base_search_catalog | bsc : This is the catalog in which all

products reside.

os_build_level | obl : This is the os build level which is the basis for the search. All product levels are derived from the tying file contained in this build level.

include_products | include_product | ip : This is the list of product catalogs to search.

exclude_products | exclude_product | ep : If IP=ALL; then this is the list of products to skip.

library_name_and_product | lnap : This is a list of additional libraries to combine onto the set.

result_library | rl : This is the place to put the merged library.

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

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.

9.1.16 COMPILE_LIBRARY (COML)

The purpose of this procedure is to compile the specified destination object libraries with the specified processors on a product source library. It uses the feature names in the file wef$feature_list (from working and feature catalogs) to select which decks in the libraries are compiled. A job is created and submitted for each object library compiling.

compile_library

9.0 DESCRIPTION OF PROCEDURES
9.1.16 COMPILE_LIBRARY (COML)

```
     processor, p: list of name or key all = cybil
     object_library, ol: list of name or key all, none = none
     job_class, jc: name = express_batch
     save_compile_jobs, scj: boolean = false
     processor_parameters, pp: list of string = 'cybil rc=(r,s)
     opt=low'
     include_copying_decks, icd: boolean = true
     features, feature, f: list of name
     list, l: file = $null
     list_options, list_option, lo: list of name
     save_listings, save_listing, sl: key all, good, fatal, warning,
     none = none
     save_binaries, sb: boolean = true
     alternate_products, alternate_product, ap: list 1 ..
     $max_value_sets 1 .. 2 of
     alternate_bases, alternate_base, ab: list of file or key none =
     none
     target_operating_system, tos: key nos nosbe = nos
     os_build_level, obl: name
     development_base, db: file = .intve
     product_name, pn: name = os
     build_level, bl: 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
```

processor | p : This selects the processor types for the decks to be compiled. Only decks with this processor will be compiled.

object_library | ol : This selects the destination library attributes for the decks which are to be compiled. Only decks which would end up on the specified object libraries are selected.

job_class | jc : This is the system job class to be used for all compile jobs.

save_compile_jobs | scj : This selects whither compile jobs are to be retained after they are submitted.

processor_parameters | pp : These are the parameters to be used with each processor type.

include_copying_decks | icd : This selects whither those decks which *copy the decks selected by the processor and object_library parameters are to be included, also.

9.0 DESCRIPTION OF PROCEDURES
9.1.16 COMPILE_LIBRARY (COML)

features | feature | f : These are the features to include for the selected decks - beyond those selected by the build_level values.

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

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.

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.

save_binaries | sb : This specifies whither the binaries produced by the compile are to be preserved.

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 3 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.

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

9.0 DESCRIPTION OF PROCEDURES
9.1.16 COMPILE_LIBRARY (COML)

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.

9.1.17 CREATE_INTEGRATION_WORKING_ENV (CREIWE)

This procedure creates the catalog structure in the integration catalog with the proper permits to allow building from another catalog. This requires the creation of the build subcatalog structure used by the build procedures so that files can be created in the integration catalog running from the permitted user. Only one user is given permission to do this.

This procedure also call move_build_files.

NOTE: This procedure must be run from the integration user number.

```
create_integration_working_env
     permitted_user, pu: name = $required
     predecessor_build_level, pbl: name = $required
     current_build_level, cbl: name = $required
     development_base | db : file = .INTVE
     product_name, pn: name = os
     status)
```

permitted_user | pu : specifies the user that is given permission to create and delete files in the integration product build catalog being created.

predecessor_build_level | pbl : specifies from where the build files are moved.

9.0 DESCRIPTION OF PROCEDURES
9.1.17 CREATE_INTEGRATION_WORKING_ENV (CREIWE)

current_build_level | cbl : specifies the name of the build catalog that is to be created for the specified product.

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

Default if omitted: .INTVE

product_name | pn : specifies the product for which the build catalog is being created.

Omission causes OS to be used.

Required parameter.

status : see NOS/VE error handling.

9.1.18 CREATE_NEW_PRODUCT_CATALOG (CRENPC)

This procedure will perform the following functions:
1) Create a new product catalog in the specified development base
2) Create all the files integration requires to transmit and extract in the product catalog with the correct permits.
   a) source_library
   b) latest_changes
   c) build_requests
   d) special_requests
   e) logging
3) Create the build_00000 build deck and transmit to the product catalog.

To run this procedure you must be logged into the development bases user number.

```
create_new_product_catalog
  product_name, pn : name = $required
  create_build_decks, cbd : boolean = true
  development_base, db : file = .intve
  status : var of status = $optional
```

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

create_build_decks | cbd : This will cause the creation of empty build decks for build '00000'.

development_base | db : specifies the catalog in which all products

and build levels reside.
Default if omitted: .INTVE

status : See NOS/VE error handling.

9.1.19 CREATE_OPEN_SHOP_TAPES (CREOST)

This procedure creates the tape sets used by development when testing their own systems.

```
create_open_shop_tapes
     build_level, bl : name
     tape_type, tt : list of key  s051cip,  s052cip,  s1cip,  s2cip,
     s3cip, s4cip, s5cip, ve_deadstart, nos_deadstart, full_product,
     short_product, test, all
     status
```

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

tape_type | tt : This specifies the tape or tape set to be created.
Default if omitted: Create a copy of all tapes.

status : see NOS/VE error handling.

9.1.20 DELETE_ASSOCIATED_PRODUCTS (DELAP DELETE_ASSOCIATED_PRODUCT)

The purpose of this procedure is to delete associated products that are not being used by OS build catalogs.

```
delete_associated_products
  product_name, pn : name = os
  build_level, bl : name = $optional
  status : var of status = $optional
```

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.

9.1.21 DETERMINE_AP_CHANGES (DETAC)

This procedure is a subroutine of CHECK_ALTERNATE_PRODUCTS. It uses the scu criteria file produced by GENERATE_AP_CRITERIA to check if a single alternate product has decks which need to be recompiled.

```
determine_ap_changes
  criteria_file, cf : file = $required
  output, o : file = $response
  product_name, pn : name = os
  status : var of status = $optional
```

criteria_file | cf : This is criteria file which selects the decks affected by the current build.

output | o : This is the file to write the list of products that must be recompiled to.

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

status : See NOS/VE error handling.

9.1.22 DETERMINE_BUILD_CHANGES (DETBC)

The purpose of this procedure is to check a file of build request information to determine if a particular type of change is included.

```
determine_build_changes
     changes_made, cm : var of boolean = $required
     type_changes, tc : key nos, n, program_interface, pi, tests, t,
     test_data, td subsystem_interface si = tests
     build_requests, br : file = $null
     status
```

changes_made | cm : This is the parameter that returns the result of the search.

type_changes | tc : This is the type of change to look for.

9.0 DESCRIPTION OF PROCEDURES
9.1.22 DETERMINE_BUILD_CHANGES (DETBC)

build_requests | br : This is the file of requests.

status : See NOS/VE error handling.

9.1.23 DISPLAY_BUILD_ENVIRONMENT (DISBE)

The purpose of this procedure is to display the value of the build environment global variables.

```
display_build_environment
     display_options, display_option, do : list of name or key all =
     all
     output, o : file = $output
     status
```

display_options | display_option | do : This is the information to display. Currently the only valid option is OS_BUILD_LEVEL.

output | o : This is the file to display onto.

status : See NOS/VE error handling.

9.1.24 DISPLAY_UNUSED_PRODUCTS (DISUP DISPLAY_UNUSED_PRODUCT)

The purpose of this procedure is to list all the products that are not being used by the full OS build catalogs. First the procedure finds all of the full OS build catalogs. Using the tying file and finding all of the products that have build catalogs, the procedure generates a list of products and its build catalogs. This list is then compared with all of the full OS build catalog's ty files to determine which build catalogs, in each product catalog, can be deleted

```
display_unused_products
  development_base, db : file = .intve
  product_name, pn : name = os
  output, o : file = $response
  status : var of status = $optional
```

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.

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

status : See NOS/VE error handling.

9.1.25 DUMP_CATALOG (DUMC)

This command provides automatic backup of specified catalog(s) to specified tape(s). This command can be performed in one of three job classes: local, batch or maintenance.

The format of DUMP_CATALOG is as follows:

```
DUMP_CATALOG, DUMP_CATALOGS
     catalog, catalogs, c: list of file = $required
     vsn : list of string 1..6 = $required
     job_class, jc: key batch local maintenance = maintenance
     list, l: file = $list
     password, pw: name = $name($job(user)//'X')
     status
```

catalog: catalogs: c: Specifies the catalog(s) to be backed up.

Required parameter.

vsn: Specifies the external tape(s) that the catalog(s) is/are to be backed up on. Each external tape name must be a 1 to 6 character string.

Required parameter.

job_class: jc: Specifies which of the three classes the job will run in. The keywords are:

LOCAL - Runs interactive.
BATCH - Submitted as batch job.
MAINTENANCE - Submitted as a deferred batch job.

Omission causes job to run in maintenance.

list: l: Specifies where related output should be written.

Omission causes $LIST to be used.

9.0 DESCRIPTION OF PROCEDURES
9.1.25 DUMP_CATALOG (DUMC)

password: pw: Specifies a user's password for logging in during a batch or maintenance job run.

Omission causes use of the standard (user's name//'x') password to be used.

status: See NOS/VE error handling.

9.1.26 FIND_CHANGED_PRODUCTS (FINCP FIND_CHANGED_PRODUCT)

The purpose of this procedure is to determine which products have changed between two given build levels. It compares the values of the tying file variables to determine which products have changed.

```
find_changed_products
  current_build_level, cbl : name = $required
  previous_build_level, pbl : name = $required
  output, o : file = $response
  status : var of status = $optional
```

current_build_level | cbl : The end build to check.

previous_build_level | pbl : The starting build to check.

output | o : The file to receive the list of changed products.

status : See NOS/VE error handling.

9.1.27 GATHER_OS_BUILD_LIBRARIES (GATOBL)

This procedure is used to call combine_product_maint_libraries for the os build_level.maintenance libraries that need to be combined. It automates the process of merging the library_name_and_product parameter. This procedure contains a canned list of the object libraries which must be merged across products and therefore must be updated whenever a new object library is added to that list.

```
gather_os_build_libraries
     development_base, db : file = .intve
     os_build_level, obl : name = $optional
     feature_catalog, fc : file = none
     feature_build_level, fbl : name = object
```

9.0 DESCRIPTION OF PROCEDURES
9.1.27 GATHER_OS_BUILD_LIBRARIES (GATOBL)

```
     working_catalog, wc : file = none
     working_build_level, wbl : name = object
     status
```

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

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.

status : See NOS/VE error handling.

9.1.28 GENERATE_AP_CRITERIA (GENAC)

This procedure is a subroutine of CHECK_ALTERNATE_PRODUCTS. It uses the build request subset file produced by BUILD_SYSTEM to generate a scu selection criteria file that - when applied to alternate products - will determine whither any decks on the alternate product need to be recompiled.

```
generate_ap_criteria
  build_requests, br : file = $required
  criteria_file, cf : file = $output
  status : var of status = $optional
```

build_requests | br : This is the subset build request file produced by build_system, which contains the features going into the current build.

criteria_file | cf : This is the file which will select the affected decks set and which is used to determine if alternate products need recompilation.

status : See NOS/VE error handling.

9.1.29 GENERATE_BUILD_DECK_COMMANDS (GENBDC)

This commands generates a file of exclude_feature commands and exclude_deck commands for the build. The file contains commands that exclude the features in the feature_list and excludes the decks identified as new_decks in the build_request file. Later this file will be inserted in the previous build level build deck.

```
generate_build_deck_commands
     build_request, br: file = $required
     feature_list, fl: file = $required
     build_deck_commands, bdc: file=$local.build_deck_commands
     status_variable, sv: name = $required
     development_base | db : file = .INTVE
     product_name, pn: name = os
     status
```

build_request | br : specifies the file of build request information for the features in the feature list.

Required parameter.

feature_list | fl : specifies the file containing the feature names going into a build.

Required parameter.

build_deck_commands | bdc : specifies the file on which the commands are written.

Omission causes $local.build_deck_commands to be used.

status_variable | sv : specifies the status variable used on the exclude_deck and exclude_feature commands.

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

Default if omitted: .INTVE

status : see NOS/VE error handling.

9.1.30 GENERATE_BUILD_REPORT (GENBR)

This procedure uses the feature_list file and the build_requests file in the product catalogs to generate a build report. The build_report lists all features that make up a particular build_level for the specified product. It also provides descriptions of modifications that make up each feature. The procedure leaves a copy of the generated build report in the file specified by the parameter build_report. If the parameter is not specified, the procedure will leave it in wev$development_base.os.<build_level>.build_report. or simply print the resultant build report file.

```
generate_build_report
     build_report, br: file = $optional
     products, p: list of name = os
     test_failures, tf: list of name = none
     comment_file, cf: file = $optional
     copy, copies, c: integer 0 ..  32 = 1
     printer_type, pt: key line laser = line
     build_level, bl: name = $optional
     status
```

build_report | br : specifies the name of the file in which a copy of the report will be saved.

products | p : specifies the products the report is to be generated for.

test_failures | tf : This is a list of failing tests to be printed with the build report. This allows readers to determine if they have a new failure condition or not.

comments_file | cf : This is a file of text to be printed with the report.

copy | copies | c : specifies the number of printed copies. It is possible to specify 0 here.

printer_type | pt : This is the type of printer to route the report to.

build_level | bl : specifies the build_level the report is to be generated for.

status : see NOS/VE error handling.

9.1.31 GENERATE_CIP_MICRO_FICHE (GENCMF)

The purpose of this procedure is to generate micro fiche listings for EI, DFT, M and SCD. The micro fiche is generated by first, finding the decks associated wi group EI or DFT or MDD or SCD. Next the decks are compiled to produce listings then headers are added to the listings. Finally the listings are copyed to tape 170 side.

```
generate_cip_micro_fiche
  micro_fiche_type, mft : key ei, dft, sci = ei
  part_number, pn : integer 1..100000000 = $required
  revision_letter, rl : string 1 = $required
  external_vsn, evsn : string = $required
  build_level, bl : name = $optional
  status : var of status = $optional
```

micro_fiche_type | mft : This is the set of decks to compile to create the fiche listings.

part_number | pn : This is the SMD part number used by customers to order microfiche.

revistion_level | rl : This is the version of the fiche being generated.

external_vsn | ev : This is the tape to write the fiche info onto, to send to micrographics.

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

status : See NOS/VE error handling.

9.1.32 GENERATE_DECK_COMMANDS (GENDC)

This command creates a file of change_deck commands to delete the group new_decks, add the group deleted_decks or both. This file must be appended to the product_name.special_requests file for the commands to be executed. Changes to decks that are interlocked are lost when the deck is transmitted, so a file of interlocked decks is produced for special handling. Currently this special handling consists of having the person with the deck interlocked make this change. In the future, TRATI can be made to make this change.

```
generate_deck_commands
```

9.0 DESCRIPTION OF PROCEDURES
9.1.32 GENERATE_DECK_COMMANDS (GENDC)

```
     action, a: key building, pulling = building
     build_requests, build_request, br: file = $required
     deck_selector, ds: key deleted_decks new_decks all = all
     deck_commands, dc: file = $local.deck_commands
     interlocked_decks, id: file = $local.interlocked_decks
     status_variable, sv: name = $required
     development_base | db : file = .INTVE
     product_name, pn: name = os
     status
```

action | a : specifies the type of commands to generate.

build_requests | build_request | br : specifies the file that contains the build request information that is scanned for new and deleted decks.

Required parameter.

deck_selector | ds : specifies the type of commands to generate.

Omission causes ALL to be used.

deck_commands | dc : specifies the file that the change_deck commands are written to.

Omission causes $local.deck_commands to be used.

interlocked_decks | id : specifies a file that the interlocked decks are written to.

Omission causes $local.interlocked_decks to be used.

status_variable | sv : specifies the name of the status variable to put on each command.

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

Default if omitted: .INTVE

product_name | pn : specifies the product name to be used.

Omission causes OS to be used.

status : see NOS/VE error handling.

9.1.33 GENERATE_DECK_LIST (GENDL)

This procedure generates a file of deck names that have the group new_decks or deleted_decks. Build request files contain comments indicating new or deleted decks. So the list is generated by editing the build request file. A message is output when there are not any new or deleted decks on the build request file.

```
generate_deck_list
     build_requests, build_request, br: file = $required
     deck_selector, ds: key deleted_decks new_decks = new_decks
     deck_list, dl: file = $local.deck_list
     status
```

build_requests | build_request | br : specifies the file that contains the build request information that is scanned for the selected deck type.

Required parameter.

deck_selector | ds : specifies the deck type to search for.

Omission causes NEW_DECKS to be used.

deck_list | dl : specifies the file that the deck list is put on.

Omisison causes $local.deck_list to be used.

status : see NOS/VE error handling.

9.1.34 GENERATE_DELETE_TYING_VARIABLES (GENDTV GENERATE_DELETE_TYING_

The purpose of this procedure is to take a specified tying file and produce a file that contains delete_variable of the tying file variables.

```
generate_delete_tying_variables
  tying_file, tf : file = $required
  delete_variable_file, dvf : file = $output
  status : var of status = $optional
```

delete_variable_file | dvf : This is the file to write the delete_variable commands onto.

status : See NOS/VE error handling.

9.1.35 GENERATE_DUAL_STATE_FILE (GENDSF)

This procedure generates the nos and nos/be dual state file. It puts the result, on the NOS side, in the file name that is given as the NOS_FILE_NAME parameter.

```
generate_dual_state_file
  build_level, bl : name = $required
  nos_file_name, nfn : name 1..7 = $required
  target_operating_system, ..
  tos : key nos, nosbe = nos
  status : var of status = $optional
```

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

nos_file_name | nfn : This is the 170 file to put the dual state file onto.

target_operating_system | tos : This is the partner system to generate the dual state file for.

status : See NOS/VE error handling.

9.1.36 GENERATE_LIBRARY_CHANGES (GENLC)

This procedure generates a file of commands to raise the state of the modifications belonging to the features in the feature_list to state 2. And to add the group deleted_decks and remove the group new_decks from the decks designated in the build_request file when the action parameter equals building. When the action parameter equals pulling, then the state are lowered to state 1. And the group deleted_decks is removed and the group new_decks is reinstated.

```
generate_library_changes
     action, a: key building, pulling = building
     current_build_level, cbl: name = $required
     feature_list, fl: file = $null
     build_requests, build_request, br: file = $null
     library_changes, lc: file = $catalog.library_changes
```

9.0 DESCRIPTION OF PROCEDURES
9.1.36 GENERATE_LIBRARY_CHANGES (GENLC)

```
     append_to_special_requests, atsr: boolean = true
     development_base, db: file = .INTVE
     product_name, pn: name = os
     working_catalog, wc: file = $optional
     status
```

action | a : specifies the action to be performed.

current_build_level | cbl : specifies the build level that is being built.

feature_list | fl : specifies the file containing a list of feature names.

build_requests | br : specifies the file of build request information for the feature list.

library_changes | lc : specifies the output file.

append_to_special_requests | atsr : specifies rather to append the library_changes to the special request file for the specified product.

development_base | db : specifies the catalog containing all products.

product_name | pn : specifies the product that the changes are to be make to.

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

status : see NOS/VE error handling.

9.1.37 GENERATE_LSDD_CIP_FILE (GENLCF)

********** DESCRIPTION NEEDED <<<<-------
****

```
generate_lsdd_cip_file
  output, o : file = $response
  build_level, bl : name = $optional
  working_catalog, wc : file = $optional
  working_build_level, wbl : name = object
  feature_catalog, fc : file = $optional
  feature_build_level, fbl : name = object
  status : var of status = $optional
```

9.0 DESCRIPTION OF PROCEDURES
9.1.37 GENERATE_LSDD_CIP_FILE (GENLCF)

-->> PARAMETER DESCRIPTIONS NEEDED <<--

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.

9.1.38 GENERATE_NEW_NOS (GENNN)

The purpose of this procedure is to generate a customized NOS deadstart tape and to save the 170 files required to recompile the 170 interfaces.

```
generate_new_nos
     dst_tape_vsn, dtv : name ..  = $required
     opl_tape_vsn, otv : name ..  = $optional
     new_nos_tape_vsn, nntv : name ..  = $optional
     make_new_nos_tape, mnnt : boolean = true
     make_new_nos_catalog, mnnc: boolean = true
     status
```

dst_tape_vsn | dtv : This is the vsn of the standard NOS deadstart tape.

opl_tape_vsn | otv : This is the vsn of a tape which contains the NOS OPL that matches the dtv tape.

new_nos_tape_vsn | nntv : This is the vsn of the customized deadstart_tape.

make_new_nos_tape | mnnt : This controls whither the new tape is generated or not.

make_new_nos_catalog | mnnc : This controls whither the files (SYSTEXT, NOSTEXT, PSSTEXT, etc.) are saved in the current

user catalog.

status : See NOS/VE error handling.

9.1.39 GENERATE_NVE001_TAPE (GENNT)

This procedure will generate a NOS/VE NVE001 (VE deadstart) release tape. The deadstart file placed on the NVE001 tape can be obtained from a previously created NOS/VE deadstart tape by specifing the vsn or, if no vsn is specified, the deadstart file will be generated using Generate_deadstart_file.

```
generate_nve001_tape
     nve_vsn, nv : name ..  = $required
     deadstart_file_vsn, dfv : name ..  = $required
     target_operating_system, tos : key nos, nosbe = nos
     product_name, pn : name = os
     build_level, bl : name = $optional
     working_catalog, wc : file or key none = none
     status
```

nve_vsn | nv : This is the vsn of the new tape to generate.

deadstart_file_vsn | dfv : This is the vsn of an old NOS/VE deadstart tape of the level being created here.

target_operating_system | tos : This parameter is used to select whither to generate a tape for a NOS or NOS/BE dual state system.

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.

status : See NOS/VE error handling.

9.1.40 GENERATE_PRODUCT_LIST (GENPL)

This procedure will create a file containing a list of all products on the specified installation table. It is designed to be called from ascl proc that has already entered the prepare_release_materials utility and executed a set_installation_table command.

```
generate_product_list
     installation_table, it : file = $required
     product_list, pl : file = $required
     number_of_products, nop : var of integer = $optional
     status
```

installation_table | it : This is the installation table to have it's product list displayed.

product_list | pl : This is the file which recieves the listing of all of the products on the installation table.

number_of_products | nop : This is the number of items in product_list.

status : See NOS/VE error handling.

9.1.41 GENERATE_PRODUCT_LIST_FILE (GENPLF)

This procedure generates a product list file for the specified build level. The file contains all of the products and the files associated with those products. It also gives the size of the files along with the total size for each product by calling generate_product_list_size. The open shop tapes full_product and short_product are also given along with their files.

```
generate_product_list_file
  build_level, bl : name = $required
  product_list_file, plf : file = $required
  status : var of status = $optional
```

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

product_list_file | plf : This is the file with the list of products which were modified at the specified build level.

9.0 DESCRIPTION OF PROCEDURES
9.1.41 GENERATE_PRODUCT_LIST_FILE (GENPLF)

status : See NOS/VE error handling.

9.1.42 GENERATE_PARALLEL_CRITERIA_FILE (GENPCF)

This procedure is designed to generate a selection criteria file for parallel builds. The reason this selection criteria file needs to be generated is to make sure all the decks that need to be recompiled are recompiled. This is accomplished by expanding the previous (predecessor) build deck to obtain the needed features. Those features are then fed to GET_BUILD_REQUEST_INFORMATION, the result is the RETURNED_SELECTION_CRITERIA file.

```
generate_parallel_build_criteria
 previous_build_level, pbl : name = $required
 returned_selection_criteria, rsc : file = $required
 product_name, pn : name = os
 development_base, db : file = .intve
 status : var of status = $optional
```

previous_build_level | pbl : specifies the previous build level to the current build.

Required Parameter.

returned_selection_criteria | rsc : file where the output is placed.

Required Parameter.

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

Omission causes OS to be used.

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

Omission causes .INTVE to be used.

status : See NOS/VE error handling.

9.1.43 GENERATE_PRODUCT_LIST_SIZE (GENPLS)

This procedure must be used from within the Prepare_Release_Materials utility. It generates the size of the files that are associated with the products that are listed in the product_list parameter. If 'all' is specified then all of the products in the specified installation_table are used. It also returns the total byte size for the products. The vsn's for the product tapes are also given. If there isn't a tape associated with a product the 'SCRTCH' is returned for the vsn.

```
generate_product_list_size
  installation_table, it : file = $required
  product_list, pl : list of name or key of, all = all
  output, o : file = $output
  status : var of status = $optional
```

installation_table | it : This is file with all of the files associated with every released product on it.

product_list | pl : This is the list of products for which the information is to be generated.

output | o : This is the file to which the data is written.

status : See NOS/VE error handling.

9.1.44 GENERATE_RELEASE_MATERIALS (GENRM)

This procedure will generate release tapes given an installation table and catalog. If the installation catalog does not contain an installation table, it will be regenerated. The tapes will be reserved automatically on the integration tape_library. The release_level parameter specifies the group on the tape_library.

```
generate_release_materials
     installation_table, it : file = $required
     installation_catalog, ic : file = $required
     release_level, rl : name = $required
     products, p : list of name = all
     output, o : file = $output
     build_level, bl : name = $optional
     status
```

9.0 DESCRIPTION OF PROCEDURES
9.1.44 GENERATE_RELEASE_MATERIALS (GENRM)

installation_table | it : This is the installation table to be used in generating the release materials.

installation_catalog | ic : This is the catalog used to gather the materials in.

release_level | rl : This is the name to be assigned to the the set of materials produced.

products | p : Specifies the list of system or products for which materials are to be generated.

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

status : See NOS/VE error handling.

9.1.45 GENERATE_RELEASE_SOURCE_LIBRARY (GENRSL)

This procedure is used to generate scu format source libraries that are to be released to customers. Elements are extracted from the product source library, latest changes, feature and working libraries and then compiled. If a working catalog is not specified, the resultant files are left in the $local catalog. The version and description in the library header of the release source library are updated to reflect the release level. All author fields on all decks are changed to "CONTROL DATA CORPORATION". A cross reference may be generated to ensure that there are no references to decks not on the library. The cross reference listing should be checked to ensure this.

The extracted subset source library is written to a file in the working catalog if working catalog defined, if not it is written to the local catalog. The release source library parameter is used for the file name. The same is true for the cross reference if specified.

```
generate_release_source_library
     development_base | db : file = .INTVE
     build_level, bl: name = $required
     release_level, rl: integer = $required
     cross_reference, cr: name or key none = cross_reference
     release_source_library,     rsl:     key     program_interface,
     svl_program_interface, dual_state_source, subsystem_interface =
     program_interfac
     selection_criteria, sc: file
```

feature_catalog, fc: file or key none = none
working_catalog, wc: file or key none = none
status

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

Default if omitted: .INTVE

build_level | bl: specifies the build level for which the source library is extracted.

Required parameter.

release_level | rl: specifies the value to put in the library header of the extracted source library.

Required parameter.

cross_reference | cr: specifies the file name to which the cross reference of the extracted library is written. If 'none' is specified no cross referencing is done.

Omission causes cross_reference to be used.

release_source_library | rsl: keyword that specifies the group for which the subset library is extracted.

Omission causes program_interface to be used.

selection_criteria | sc: specifies file on which additional selection criteria may reside. These selection criteria directives are the last on the criteria file used to extract the subset library.

Omission causes no additional selection criteria directives to be used.

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.

9.1.46 GENERATE_TEST_DATA_SOURCE_LIB (GENTDSL)

The purpose of this procedure is to extract the subset test data source library from the specified product library. Latest changes, feature and working libraries are combined with the product source library before extracting. A cross reference of the result library may be generated to ensure that there are no references to decks not on the library. The cross reference listing should be checked to ensure this.

The extracted test data source library is written to a file in the working catalog if working catalog is defined, if not it is written to the local catalog. The file name is specified by the result parameter. The same is true for the cross reference if specified.

NOTE: The selection criteria for the test data source library is assumed to be on deck 'TDI$BUILD_DATA' of the product source library. This is a convention followed by all products at Arden Hills, if it is deviated from it should be corrected.

```
generate_test_data_source_lib
     selection_criteria, sc: file
     cross_reference, cr: name or key none = cross_reference
     result, r: name = $required
     development_base | db : file = .INTVE
     product_name, pn: name = os
     build_level, bl: name = $required
     feature_catalog, fc: file or key none = none
     working_catalog, wc: file or key none = none
     status
```

selection_criteria | sc: specifies file on which additional selection criteria may reside. These selection criteria directives are the last on the criteria file used to extract the subset library.

Omission causes no additional selection criteria directives to be used.

cross_reference | cr: specifies the file name to which the cross reference of the extracted library is written. If 'none' is specified no cross referencing is done.

Omission causes cross_reference to be used.

result | r: specifies file name for the subset library extracted.

Required parameter.

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

Default if omitted: .INTVE

product_name | pn: specifies the product from which the test data is extracted.

Omission causes OS to be used.

build_level | bl: specifies the build level for which the subset library is extracted.

Required parameter.

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.

9.1.47 GEN_EXCLUDE_FEATURE_COMMANDS (GENEFC)

This procedure generates the SCU exclude_feature commands from the feature list. The exclude_feature commands are appended to the exclude_feature_file.

NOTE: This procedure must be called from inside the SCU utility.

```
gen_exclude_feature_commands
     feature_list, fl: file = $required
     exclude_feature_file, eff: file = $required
     status_variable, sv: name = $required
     status
```

feature_list | fl : specifies the file name of the file containing feature names, one feature name per line.

Required parameter.

exclude_feature_file | eff : specifies the file to which the exclude_feature commands are appended.

Required parameter.

status_variable | sv : specifies the name of the status variable

that will be used on the exclude_feature commands.

Required parameter.

status : see NOS/VE error handling.

9.1.48 GENERATE_RESEQUENCE_COMMANDS (GENRC)

This procedure will generate three files in the working_catalog:
1) cham_commands: A file containing the change_modification commands
to change the correct modifications to state 4.
2) deld_commands: A file containing the delete_deck commands to delete
all decks with no active lines at the specified build_level.
3) empty_decks_without_group: This file contains the list of decks that
have no active lines and therefore are deleted by the
deld_commands file but do not have the group: deleted_decks.

These files are created as follows: Extract_subset_source_library is run to obtain a source_library that contains on the code active at the specified build_level. Every modification in this source_ above state 2 should be changed to state 4. Every deck with no active lines in t source_library should deleted.

The result files will be placed in the specified working_catalog.product_name ca

```
generate_resequence_commands
  product_name, pn : name = $optional
  build_level, bl : name = $optional
  working_catalog, wc : file or key none = none
  status : var of status = $optional
```

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.

9.0 DESCRIPTION OF PROCEDURES
9.1.48 GENERATE_RESEQUENCE_COMMANDS (GENRC)

status : See NOS/VE error handling.

9.1.49 GENERATE_SOURCE_RELEASE (GENSR)

The purpose of this procedure is to make the 'first cut' of a product intended for source release. It will clean up the main library and copy the appropriate build catalog into a new catalog under the release catalog.

```
generate_source_release
  release_catalog, rc : file = $required
  os_build_level, obl : name = $required
  product_names, ..
  product_name, pn : list of name = $required
  selection_criteria, sc : file = $optional
  status : var of status = $optional
```

release_catalog | rc : This is the catalog to write the source release product into.

os_build_level | obl : This is the os build level for the source release. Non-os products will be selected from the tying file.

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

selection_criteria | sc : This is a file of additional criteria directives to apply to the main library when cracking off the source release subset.

status : See NOS/VE error handling.

9.1.50 GENERATE_TAPE_LIST (GENTL)

The purpose of this procedure is to generate a list that contains the names of the release tapes that need to be generated.

```
generate_tape_list
  changed_products, cp : file = $required
  installation_table, it : file = $required
  output, o : file = $response
  status : var of status = $optional
```

9.0 DESCRIPTION OF PROCEDURES
9.1.50 GENERATE_TAPE_LIST (GENTL)

changed_products | cp : This is the list of products which have been modified.

installation_table | it : This is the table which contains information about all released product files.

output | o : This is the file to receive the list of changed release tapes.

status : See NOS/VE error handling.

9.1.51 GENERATE_TYING_FILE (GENTF)

This procedure is used to generate a tying file in a build_catalog. When creating a new tying file both the current_build_level and previous_build_level parameters should be specified. When updating an existing tying file only the current_build_level should be specified. The product_name parameter is included for use only when creating tying files in secondary_build catalogs. The all_products parameter will update every sunnyvale product entry with the specified value.

```
generate_tying_file
  product_name, pn : name = os
  previous_build_level, pbl : name = $optional
  current_build_level, cbl : name = $required
  applications_level, appl : name = $optional
  av_level, al : name = $optional
  base_os_level, bol : name = $optional
  cdcnet_level, cdcl : name = $optional
  cdcnet_version, cdcv : name = $optional
  cip_catalog, cc : name = $optional
  cip_level, cipl : name = $optional
  cml_level, cmll : name = $optional
  cybil_level, cybl : name = $optional
  cybil_cc_base, ccb : name = $optional
  dvs_level, dvsl : name = $optional
  format_cybil_source_level, ..
  fcsl : name = $optional
  hpa_level, hpal : name = $optional
  mailve_level, mvl : name = $optional
  malet_level, mal : name = $optional
  nos_base, nb : name = $optional
  nos_level, nl : name = $optional
  os_level, ol : name = $optional
  performance_tools_level, ..
```

9.0 DESCRIPTION OF PROCEDURES
9.1.51 GENERATE_TYING_FILE (GENTF)

```
  ptl : name = $optional
  pftf_level, pftfl : name = $optional
  primary_build_level, pribl: name = $optional
  procs_level, prl : name = $optional
  release_documentation_level, ..
  rdl : name = $optional
  scu_level, scul : name = $optional
  system_documentation_level, ..
  sdl : name = $optional
  system_tests_level, stl : name = $optional
  tdu_level, tdul : name = $optional
  test_tools_level, ttl : name = $optional
  z80_compass_base, zcb : name = $optional
  all_products_level, apl : name = $optional
  aam_level, aaml : name = $optional
  apl_level, apll : name = $optional
  assemble_level, asml : name = $optional
  basic_level, bl : name = $optional
  cobol_level, cobl : name = $optional
  common_level, coml : name = $optional
  debug_level, dl : name = $optional
  edit_catalog_level, ecl : name = $optional
  fma_level, fmal : name = $optional
  fmu_level, fmul : name = $optional
  help_level, hl : name = $optional
  imdm_level, il : name = $optional
  lisp_level, ll : name = $optional
  manuals_level, ml : name = $optional
  pascal_level, pasl : name = $optional
  pe_level, pel : name = $optional
  prologue_level, prol : name = $optional
  ps_quicklooks_level, pql : name = $optional
  quick_level, ql : name = $optional
  sort_level, sl : name = $optional
  vector_fortran_level, vfl : name = $optional
  status : var of status = $optional
```

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

previous_build_level | pbl : The value of the predecessor os build level.

current_build_level | cbl : The current os build level value.

applications_level | appl : The current sunnyvalue build level value.

av_level | al : The current value for this product variable.

9.0 DESCRIPTION OF PROCEDURES
9.1.51 GENERATE_TYING_FILE (GENTF)

base_os_level | bol : The previous standard os build level value. (Usually equal to PREVIOUS_BUILD_LEVEL, except for 'psudo' builds.)

cdcnet_level | cdcl : The current value for this product variable.

cdcnet_version | cdcv : CDCNET's build level value.

cip_catalog | cc : The current value for this product variable.

cip_level | cipl : The current value for this product variable.

cml_level | cmll : The current value for this product variable.

cybil_level | cybl : The current value for this product variable.

cybil_cc_base | ccb : The catalog containing the NOS-side cybil compiler.

dvs_level | dvsl : The current value for this product variable.

format_cybil_source_level | fcsl : The current value for this product variable.

hpa_level | hpal : The current value for this product variable.

mailve_level | mvl : The current value for this product variable.

malet_level | mal : The current value for this product variable.

nos_base | nb : The catalog containing the NOS OPL and XTEXT files.

nos_level | nl : Nos's current build level value.

os_level | ol : The current value for this product variable.

performance_tools_level | ptl : The current value for this product variable.

pftf_level | pftfl : The current value for this product variable.

primary_build_level | pribl: The current value for this product variable.

procs_level | prl : The current value for this product variable.

release_documentation_level | rdl : The current value for this product variable

scu_level | scul : The current value for this product variable.

system_documentation_level | sdl : The current value for this product variable.

system_tests_level | stl : The current value for this product variable.

tdu_level | tdul : The current value for this product variable.

test_tools_level | ttl : The current value for this product variable.

z80_compass_base | zcb : The current value for this product variable.

all_products_level | apl : The current value for this product variable.

aam_level | aaml : The current value for this product variable.

apl_level | apll : The current value for this product variable.

assemble_level | asml : The current value for this product variable.

basic_level | bl : The current value for this product variable.

cobol_level | cobl : The current value for this product variable.

common_level | coml : The current value for this product variable.

debug_level | dl : The current value for this product variable.

edit_catalog_level | ecl : The current value for this product variable.

fma_level | fmal : The current value for this product variable.

fmu_level | fmul : The current value for this product variable.

help_level | hl : The current value for this product variable.

imdm_level | il : The current value for this product variable.

lisp_level | ll : The current value for this product variable.

manuals_level | ml : The current value for this product variable.

pascal_level | pasl : The current value for this product variable.

pe_level | pel : The current value for this product variable.

prologue_level | prol : The current value for this product variable.

ps_quicklooks_level | pql : The current value for this product variable.

quick_level | ql : The current value for this product variable.

sort_level | sl : The current value for this product variable.

vector_fortran_level | vfl : The current value for this product variable.

status : See NOS/VE error handling.

9.1.52 GEN_EXCLUDE_NEW_DECK_COMMANDS (GENENDC)

This commands produces a file of exclude_deck commands for all new decks specified in the build request file. The deck names are extracted by using generate_deck_list.

Later this file will be inserted into the previous build level build deck.

```
gen_exclude_new_deck_commands
     build_requests, build_request, br: file = $required
     exclude_new_deck, end: file = $local.exclude_new_deck
     status_variable, sv: name = $required
     status
```

build_requests | build_request | br : specifies the file that contains the build request information that is scanned for new decks.

Required parameter.

exclude_new_deck | end : specifies the file on which the exclude_deck commands are written.

Omission causes $local.exclude_new_deck to be used.

status_variable | sv : specifies the name of the status variable

used on the exclude_deck commands.

Required parameter.

status : see NOS/VE error handling.

9.1.53 GEN_INCLUDE_FEATURE_COMMANDS (GENIFC)

This procedure generates the SCU include_feature commands from the feature list. The include_feature commands are appended to the include_feature_file.

NOTE: This procedure must be called from inside the SCU utility.

```
gen_include_feature_commands
     feature_list, fl: file = $required
     include_feature_file, iff: file = $required
     status_variable, sv: name
     status
```

feature_list | fl : specifies the file name of the file containing feature names, one feature name per line.

Required parameter.

include_feature_file | eff : specifies the file to which the include_feature commands are appended.

Required parameter.

status_variable | sv : specifies the name of the status variable that will be used on the include_feature commands.

Omission causes no status variable to be appended to the include_feature command.

status : see NOS/VE error handling.

9.1.54 GENERATE_STATE_CHANGE_COMMANDS (GENSCC)

This procedure generates the SCU change_modification commands to change the state of all modifications associated with features in the feature list. This procedure must be called from inside the SCU

utility.

NOTE: The change_modification commands are appended to the specified state change file.

```
generate_state_change_commands
     feature_list, fl: file = $required
     state_change_file, scf: file = $required
     state, s: integer 0..4 = $required
     status_variable, sv: name = $required
     status
```

feature_list | fl : specifies the file name of the feature list, one feature name per line, for whose modifications state change commands will be generated.

Required parameter.

state_change_file | scf : specifies the file name to which the state change commands are appended.

Required parameter.

state | s : specifies the state to which the associated modifications are to be changed.

Required parameter.

status_variable | sv : specifies the name of the status variable that will be used on the change_modification commands to change the state.

Required parameter.

status : see NOS/VE error handling.

9.1.55 GENERATE_SYSTEM (GENS)

This procedure combines two major steps in the integration process, link_operating_system and generate_deadstart_file, into one procedure.

This command can be used inside the working environment. The working environment variables must be initialized via set_working_environment before using this command.

9.0 DESCRIPTION OF PROCEDURES
9.1.55 GENERATE_SYSTEM (GENS)

```
generate_system
     volume_serial_number, vsn : name 1..6 = $required
     operating_system_identifier, osi : string 1..5 = $optional
     product_set_identifier, psi : string 1..3 = $optional
     link_system_core, lsc : boolean = true
     bind_operating_system, bos : boolean = true
     delete_modules, dm : file = $null
     display_options, do : key errors, e, full, f = errors
     cleanup_170, c7 : key start, end, both, none = both
     status
```

volume_serial_number | vsn : specifies the external vsn on the NOS/VE deadstart tape.

Required parameter.

operating_system_identifier | osi : specifies the NOS/VE system identifier displayed on the system header.

Omission causes the working environment variable, wev$build_level, to be used.

product_set_identifier | psi : specifies the NOS/VE product set identifier displayed on the system log.

Omission causes the tying file variable, wev$aam_level, to be used.

link_system_core | lsc : specifies if the system_core should be linked in addition to the job_template.

Omission causes the system core to be linked.

bind_operating_system | bos : specifies if the operating system will be bound.

Omission causes the operating system to be bound.

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_name status=ignore_status

The status must be specified as shown. The directives are executed for each library in the job_template and/or the system_core even though a module should exist in only one library. The linker will issue a warning each time a deletion directive is executed and the module is not on the library the

deletion directive is executed against.

display_options | do : specifies if the link maps should be saved or deleted.

full : link maps will always be saved

errors : link maps will be saved only if errors are located in the maps.

Omission causes ERRORS to be used.

cleanup_170 | c7 : determines how 170 files are transferred by generate_deadstart_file. Specifying START will move all files to the 170 side. Specifying END will purge all 170 files when the procedure completes and no files are moved to the 170 side. BOTH moves the files to the 170 side and purges them when the procedure completes. Specifying NONE will neither move the files to the 170 side or purge them when the procedure completes.

Omission causes BOTH to be used.

status : see NOS/VE error handling.

9.1.56 GET_BUILD_REQUEST_INFORMATION (GETBRI)

This procedure returns the build request information for the list of features on the feature list file, one feature name per line. The build request information is appended to the end of the selected_build_request_info file.

```
get_build_request_information
     feature_list, fl: file = $required
     build_requests, br: file = $required
     selected_build_request_info, sbri: file = $required
     status)
```

feature_list | fl : specifies the file name of the feature list for which build request information is to be returned.

Required parameter.

build_requests | br : specifies the file containing the product build requests from which the build requests for the specified features are extracted.

9.0 DESCRIPTION OF PROCEDURES
9.1.56 GET_BUILD_REQUEST_INFORMATION (GETBRI)

Required parameter.

selected_build_request_info | sbri : specifies the name of the file where the build requests for the specified features are appended.

Required parameter.

status : see NOS/VE error handling.

9.1.57 GET_FEATURE_BUILD_REQUEST (GETFBR)

This procedures scans the specified build request file and returns the build request information for the specified feature. If there are multiple occurrences of the feature name on the build request file the found_multiple_occurrences boolean is set to true, this indicates a possible error that should be checked into.

```
get_feature_build_request
     feature, f: name = $required
     build_requests, br: file = $required
     selected_build_request_info, sbri: file = $required
     found_multiple_occurrences, fmo: var of boolean
     status
```

feature | f : specifies the feature name for which build request information is to be returned.

Required parameter.

build_requests | br : specifies the file containing the product build requests from which the build request for the specified feature is extracted.

Required parameter.

selected_build_request_info | sbri : specifies the name of the file where the build request for the specified feature is appended.

Required parameter.

found_multiple_occurrences | fmo : specifies whether more than one build request was found for the feature. This indicates a possible error and should be checked into.

Required parameter.

9.0 DESCRIPTION OF PROCEDURES
9.1.57 GET_FEATURE_BUILD_REQUEST (GETFBR)

status : see NOS/VE error handling.

9.1.58 LIST_SUBCATALOGS (LISS)

This procedure returns a list of all subcatalog entries of a given catalog.

```
list_subcatalogs
     catalog, c : file = $user
     subcatalog_list, sl : file = $output
     count : var of integer = $required
     status
```

catalog | c : This is the catalog whose subcatalog's are to be listed.

subcatalog_list | sl : This is the file to recieve the listing.

count | c : This is a count of the number of items in subcatalog_list.

status : See NOS/VE error handling.

9.1.59 LOAD_CIP_FILES (LOACF)

This procedure will create a 170 job to acquire the cti supplied cip files from the user thezoo on sn907. The files are expected to follow the namin convention for the s1, s2, s3, and s4 files: CPxxxH?cip_level_key_letter?. Where 'xxx' is the model_number of the machine and 'H' indicates the HIVS version of cip materials which is the only version we use. For the s0_51 file the convention is S0688?cip_level_key_letter?M. And for the s0_52 file the convention is S0688?cip_level_key_letter?P. The cip_level_key_letter is expected to be the last character in the cip_level specified (i.e. 6K). The cip files are acquired from sn907 and placed in the 170 catalog of the user where the link attributes are pointing and then moved to the 180 side and placed the correct file in the integration catalog. The files on the 170 side are give public permission and the 180 version of the files have the file_access_procedur attached to them that PROCESS_NOS_FILES requires.

9.0 DESCRIPTION OF PROCEDURES
9.1.59 LOAD_CIP_FILES (LOACF)

```
load_cip_files
  cip_level, cl : string 1..3 = $required
  cip_files, cf : list of key s0_52, s0_51, s1, s2, s3, s4, s5,  all
  = all
  cip_subcatalog, cs : name = $optional
  status : var of status = $optional
```

cip_leve | cl : The version of the new cip to pick up.

cip_files | cf : The type of new files to pick up.

cip_subcatalog | cs : The place to put the new files.

status : See NOS/VE error handling.

9.1.60 LOG_RESTART (LOGR)

The purpose of this procedure is to gather data on the number of times that a build must be redone.

```
log_restart
     restart_file,   rf   :   file  =  $fname(wev$development_base//
     '.misc_files.restart_data.cycle_xx')
     build_level, bl : name = $optional
     development_base, bd : file = $optional
     status
```

restart_file | rf : This is the file to contain the gathered data.

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

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

status : See NOS/VE error handling.

9.1.61 MERGE_DESTINATION_LIBRARIES (MERDL)

This procedure merges the object libraries in the maintenance catalog of the working catalog with the object libraries in the maintenance catalog of the build catalog. The deck raf$destination_libraries on the product source library contains a list of the files to be moved and merged for that product.

```
merge_destination_libraries
     build_level_updating, blu: name = $required
     compare_object_libraries,  compare_object_library, col: boolean
       = true
     output, o: file = $output
     development_base | db : file = .INTVE
     product_name, pn: name = os
     working_catalog, wc: file or key none = none
     working_build_level, wbl: name = object
     status
```

build_level_updating | blu : specifies the product build level catalog to be used.

Required parameter.

compare_object_libraries | compare_object_library | col : specifies whether to compare the corresponding object libraries after the move.

Omission causes TRUE to be used.

output | o : specifies the file for the compare_object_library output.

Omission causes $output 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 name to be used.

Omission causes OS 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.

9.0 DESCRIPTION OF PROCEDURES
9.1.61 MERGE_DESTINATION_LIBRARIES (MERDL)

status : see NOS/VE error handling.

9.1.62 MAKE_SPECIAL_REQUESTS (MAKSR)

The purpose of this proc is to allow integration to save requests for changes to the source libraries which come in during the day for later execution during the daily updates.

The file the requests are saved on is SPECIAL_REQUESTS in the product catalog; and the proc which does the daily updates is UPDATE_INTVE.

The procedure will prompt for input interactivly. The interactive prompting should be terminated with a capital END command.

```
make_special_requests
  request_file, rf : file = $null
  development_base, db : file = .intve
  product_name, pn : name = os
  status : var of status = $optional
```

request_file | rf : A file of special requests. The data in the file is picked up first. The procedure will still prompt for interactive commands as well.

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.

9.1.63 MERGE_MULTI_RECORD_FILES (MERGE_MULTI_RECORD_FILE MERMRF)

The purpose of this procedure is to combine two NOS ULIB libraries into a single library.

```
merge_multi_record_files
     base_library, bl : file = $required
     combine_library, cl : list of file = $required
     result_library, rl : file = $optional
```

status

base_library | bl : This is the library to start with.

combine_library | cl : This is the library to add.

result_library | rl : This is the new library to generate.

status : See NOS/VE error handling.

9.1.64 MERGE_170_LIBRARIES (MER1L)

This command does four basic operations.

1. Merges all the pieces of NOSBINS, NVELIB, and NVEBINS using link_170.
2. Moves NOSBINS, NVELIB, NVEBINS from the working catalog to the build catalog.
3. Moves osf$prologs from working catalog to the build catalog.
4. Merges osf$nverels from working catalog with osf$nverels from build catalog. This merge is done on the 170 side.

This procedure only applies to the product OS.

```
merge_170_libraries
     build_level_updating, blu: name = $required
     working_catalog, wc: file or key none = none
     working_build_level, wbl: name = object
     status
```

build_level_updating | blu : specifies the product build level catalog to be used.

Required parameter.

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

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

status : ses NOS/VE error handling.

9.1.65 MOVE_BUILD_FILES (MOVBF)

This procedure moves the files that will serve as a base for a new build from a previous build level catalog to the new build level catalog. For all product builds the entire maintenance catalog is moved to the new build level catalog. For OS builds, the maintenance catalog, tying file, nvebins, nosbins, nvelib, and osf$prologs are all moved.

```
MOVE_BUILD_FILES
     previous_build_level, pbl : name = $required
     current_build_level, cbl : name = $required
     development_base | db : file = .INTVE
     product_name, pn : name = os
     status : var of status = $optional
```

previous_build_level | pbl : specifies the build catalog from which files are to be moved.

Required parameter.

current_build_level | cbl : specifies the build catalog to which the files are to be moved.

Required parameter.

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

Default if omitted: .INTVE

product_name | pn : specifies the product for which files are to be moved.

Default if omitted: OS

status : see NOS/VE error handling.

9.1.66 RESERVE_BUILD_TAPES (RESERVE_BUILD_TAPE RESBT)

The purpose of this procedure is to reserve the tapes needed to make a build.

```
reserve_build_tapes
```

9.0 DESCRIPTION OF PROCEDURES
9.1.66 RESERVE_BUILD_TAPES (RESERVE_BUILD_TAPE RESBT)

```
     group, g : name = $required
     tape,  t  : list of key cip_tapes, nos_deadstart, ve_deadstart,
     full_product,   short_product,    test_tape,backup_build_tapes,
     ic_backup_tapes, nos_base, normal = normal
     status
```

group | g : The build level of the tape set reserved.

tape | t : The purpose the tape is to be used for.

status : See NOS/VE error handling.

9.1.67 RESET_PRODUCT_LEVELS (RESPL)

The purpose of this procedure is to change a product's default build level. It will make a special_request to change the version field of a library header.

```
reset_product_levels
     development_base, db : file = .intve
     product_name, pn : name = os
     build_level, bl : name = $optional
     status
```

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.

9.1.68 RESTORE_OS_BUILD_LEVEL (RESOBL)

This is the partner procedure to backup_os_build_level. It will restore each build level backed up by bacobl.

```
restore_os_build_level
```

9.0 DESCRIPTION OF PROCEDURES
9.1.68 RESTORE_OS_BUILD_LEVEL (RESOBL)

```
     backup_file, bf : file = $local.bacobl
     associated_products_list, apl : list of name or key all = all
     output, o : file = $output
     development_base, db : file = .intve
     product_name, pn : name = os
     build_level, bl : name = $optional
     status
```

backup_file | bf : This is the file from which the backup data will be read. If associated_products_list is ALL, this is required to be a tape file.

associated_products_list | apl : This is a list of product names whose builds are to be included in the restore. The appropriate level is selected from the tying file in the os build selected.

output | o : This is the file to recieve the listing of what is restored.

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.

9.1.69 RETURN_BUILD_REQUEST_STATS (RETBRS)

The purpose of this procedure is to return counts of various elements in a file of build requests.

```
return_build_request_stats
     build_requests, build_request, br : file = $required
     new_decks, nd : var of integer = $optional
     deleted_decks, dd : var of integer = $optional
     lines_added, la : var of integer = $optional
     lines_deleted, ld : var of integer = $optional
     status
```

build_requests | build_request | br : This is the file of build

requests to count items in.

new_decks | nd : This is the sum of all decks being introduced by any build request.

deleted_decks | dd : This is the sum of all decks being removed by any build request.

lines_added | la : This is the sum of all of the lines added statistics from all of the build request forms.

lines_deleted | ld : This is the sum of all of the lines deleted statistics from all of the build request forms.

status : See NOS/VE error handling.

9.1.70 RETURN_BUILD_LIST (RETBL)

The purpose of this procedure is to return the full build catalog list and the partial build catalog list. It first finds all of the sub_catalogs in the catalog development_base.product_name. Then it loops through the sub_catalogs that begin with 'BUILD_' looking for the file SOURCE_LIBRARY. If it finds the file then it is a full build catalog otherwise it is a partial build catalog.

```
RETURN_BUILD_LIST, RETBL
     development_base, db : file = .intve
     product_name, pn : name = os
     full_build_list, fbl : file = $optional
     number_of_full_builds, nofb : var of integer = $optional
     partial_build_list, pbl : file = $optional
     number_of_partial_build_list, nopb : var of integer = $optional
     status : var of status = $optional
```

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

Omission causes .INTVE to be used.

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

Omission causes OS to be used.

full_build_list | fbl : The file to receive the list of full build

catalogs.

number_of_full_builds | nofb : The number of entries in FULL_BUILD_LIST.

partial_build_list | pbl : The file to receive the list of partial build catalogs.

number_of_partial_builds | nopb : The number of entries in PARTIAL_BUILD_LIST.

status : See NOS/VE error handling.

9.1.71 RETURN_TYING_FILE_PRODUCTS (RETTFP)

The purpose of this procedure is to return a file that contains a list of the products that are associated with a tying file.

```
return_tying_file_products
 build_level, bl : name = $optional
 output, o : file = $response
 status : var of status = $optional
```

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

output | o : This is the file to return the list to.

status : See NOS/VE error handling.

9.1.72 SELECT_MODIFIED_DECKS (SELMD)

This procedure returns a file containing the modified decks and the number of modified decks for a feature. This information is extracted from the build request information for the feature. If there is build request information for more than one feature on the build_request_information file, the information returned is for all features on the file.

```
select_modified_decks
     build_request_information, bri: file = $required
```

9.0 DESCRIPTION OF PROCEDURES
9.1.72 SELECT_MODIFIED_DECKS (SELMD)

```
     modified_decks, md: file = $required
     modified_deck_count, mdc: var of integer = $required
     status
```

build_request_information | bri : specifies the file which contains the build request information for one feature.

Required parameter.

modified_decks | md : specifies the file name to which decks modified by the feature are returned, one deck name per line.

Required parameter.

modified_deck_count | md : parameter where the number of modified decks on modified_decks file is returned.

Required parameter.

status : see NOS/VE error handling.

9.1.73 SET_BUILD_ENVIRONMENT (SETBE)

The purpose of this procedure is to initialize environment as needed by integration to perform a build. This procedure will set DEVELOPMENT_BASE, PRODUCT_NAME and BUILD_LEVEL the way the SETWE does. It will also set the WORKING_CATALOG to <development_base>.<product_name>.<build_level> and the WORKING_CATALOG to "CHANGES". It will also do a SET_WORKING_CATALOG to the <working_catalog> value defined above. It will add the correct versions of tools libraries to the user's command lists.

```
set_build_environment
     development_base, db : file = .intve
     product_name, pn : name = os
     build_level, bl : name = $optional
     status
```

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.

9.0 DESCRIPTION OF PROCEDURES
9.1.73 SET_BUILD_ENVIRONMENT (SETBE)

Required parameter.

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

status : See NOS/VE error handling.

9.1.74 STANDARDIZE_NAME_LIST (STANL)

This procedure standardize list of names by: 1) converting each line to a name and translating 2) eliminating leading blanks 3) sorting 4) eliminating duplicate entries

```
standardize_name_list
     input, i : file = $required
     output, o : file = $value(input)
     translate,  t  : key upper_to_lower, utl, lower_to_upper, ltu =
     upper_to_lower
     status
```

input | i : The source of the names to standardize.

output | o : The destination of the standardized list.

translate | t : The direction of character case substitution.

status : See NOS/VE error handling.

9.1.75 STANDARDIZE_OS_LIBRARIES (STAOL)

The purpose of this procedure is to massage the os object libraries to put the modules in the right order, add external modules in some cases and omit references to the cybil run time library.

```
standardize_os_libraries
     delete_modules, dm : file = $optional
     build_level, bl : name = $optional
     status
```

9.0 DESCRIPTION OF PROCEDURES
9.1.75 STANDARDIZE_OS_LIBRARIES (STAOL)

delete_modules | dm : This parameter allows users to delete modules which may be deleted in their version of the system; but are reintroduced from the main integration level.

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

status : See NOS/VE error handling.

9.1.76 TRANSMIT_GROUP (TRAG)

The purpose of this procedure is to act as a front-end for transmit_to_integration. This procedure will determine the members of a group and then supply them to trati.

```
transmit_group
     group, g : name = $optional
     development_base, db : file = .intve
     product_name, pn : name = os
     working_catalog, wc : file = none
     status
```

group | g : The name of the scu group whose members are 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.

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

status : See NOS/VE error handling.

9.1.77 UPDATE_BUILD_DECKS (UPDBD)

This procedure creates and updates the build decks for primary (standard) builds. It can be called again to add or pull features. But the creation_call parameter must be false once the build decks have been created.

9.0 DESCRIPTION OF PROCEDURES
9.1.77 UPDATE_BUILD_DECKS (UPDBD)

Extreme care should be taken when calling this procedure. The build deck descriptions are verified only on the creation_call. Specifying the wrong predecessor or current build level can create a mess.

NOTE: Build decks for features and decks are maintained individually. The build_request parameter should be $NULL when running UPDBD for the features. And the feature_list parameter should be $NULL when running UPDBD for the decks.

```
update_build_decks
     creation_call, cc: boolean = true
     action, a: key building, pulling = building
     predecessor_build_level, pbl: name = $required
     current_build_level, cbl: name = $required
     successor_build_level, sbl: name = none
     build_requests, build_request, br: file = $null
     feature_list, fl: file = $null
     modification, m: name = $required
     change_os_version, cov: boolean = false
     author: string = $job(user)
     development_base, db: file = .intve
     product_name, pn: name = os
     working_catalog, wc: file = $optional
     status
```

creation_call | cc : specifies rather this the first or a subsequent call to build_system. Creation call should alway be false once the build_decks have been created.

action | a : specifies the action to be performed.

predecessor_build_level | plb : specifies the previous build level to the current build.

current_build_level | cbl : specifies the build level that is being built.

successor_build_level | sbl : specifies the successor build level to the current build.

build_requests | br : specifies the file of build request information for the features in the build.

feature_list | fl : specifies the file containing the feature names going into the build.

modification | m : specifies the name a the modification used to edit or create the build decks.

9.0 DESCRIPTION OF PROCEDURES
9.1.77 UPDATE_BUILD_DECKS (UPDBD)

author: specifies the author when creating new decks.

development_base | db : specifies the development base to use.

change_os_version, cov: specifies rather to call the procedure changes_os_version.

product_name | pn : specifies the product to be built.

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

status : see NOS/VE error handling.

9.1.78 UPDATE_BUILD_DECK_DESCRIPTIONS (UPDBDD)

This command must be used inside the working environment and the predecessor, current, and successor build decks must be in the working library.

This command updates the deck descriptions of the build decks. The deck descriptions must be of the form 'PREDECESSOR BUILD = BUILD_XXXXX SUCCESSOR BUILD = BUILD_YYYYY' on the previous and successor build decks.

```
update_build_deck_descriptions
     predecessor_build_level, pbl: name = $required
     current_build_level, cbl: name = $required
     successor_build_level, sbl: name = none
     status
```

predecessor_build_level | pbl : specifies the build level previous to build level being built.

Required parameter.

current_build_level | cbl : specifies the build level being built.

Required parameter.

successor_build_level | sbl : specifies the build level subsequent to the build level being built.

Omission causes NONE to be used.

status : see NOS/VE error handling.

9.1.79 UPDATE_BUILD_INFORMATION (UPDBI)

This procedure updates the tying file in the appropriate OS build catalog. The tying file information is saved on the build information source library as a deck with the build_level as its name. The information stored in the decks is used by the procedure display_build_information. The source library is .INTVE.MISC_FILES. BUILD_INFORMATION_CATALOG.SOURCE_LIBRARY.

```
update_build_information
     development_base | db : file = .INTVE
     build_level, bl: name = $required
     nos_deadstart_tape, ndt: name
     product_tape, pt: name
     test_tape, tt: name
     ocu_level, ol: name
     cybil_level, cl: name
     scu_level, sl: name
     product_level, pl: name
     nos_level, nl: name
     assemble_level, al: name
     cybform_level, cf: name
     scl_tools_level, st: name
     system_tests_level, stl: name
     test_tools_level, ttl: name
     tdu_level, tl: name
     status
```

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

Default if omitted: .INTVE

build_level | bl : specifies the OS build level for which the changes are to be made. The build level is used as the deck name for the updated tying file information on the build information source library.

nos_deadstart_tape | ndt : specifies the external VSN of the associated NOS deadstart tape.

product_tape | pt : specifies the external VSN of the associated product tape.

test_tape | tt : specifies the external VSN of the associated test tape.

ocu_level | ol : specifies the build level of the associated OCU.

9.0 DESCRIPTION OF PROCEDURES
9.1.79 UPDATE_BUILD_INFORMATION (UPDBI)

cybil_level | cl : specifies the build level of the associated CYBIL.

scu_level | sl : specifies the build level of the associated SCU.

product_level | pl : specifies the build level of the associated Sunnyvale products.

nos_level | nl : specifies the build level of the associated NOS system.

assemble_level | al : specifies the build level of the associated 180 Assembler.

cybform_level | cl : specifies the build level of the associated CYBFORM.

scl_tools_level | scl : specifies the build level of the associated SCL_TOOLS.

system_tests | st : specifies the build level of the associated SYSTEM_TESTS.

test_tools_level | ttl : specifies the build level of the associated TEST_TOOLS.

status : see NOS/VE error handling.

9.1.80 UPDATE_INSTALLATION_CATALOG (UPDIC)

This procedure is used to update the installation catalog when a small number of files have changed and you do not wish to regenerate the entire installation catalog. You can specify which files to update individually using the update_entries parameter or update all files for a specific product by using the update_products parameter. The file_attribute user_information is set equal to the version field of the installation table entry just as assemble release_materials does.

```
update_installation_catalog
     installation_table, it : file = $required
     installation_catalog, ic : file = $required
     update_entries, ue : list of name = $optional
     update_products, up : list of name = $optional
     output, o : file = $output
```

9.0 DESCRIPTION OF PROCEDURES
9.1.80 UPDATE_INSTALLATION_CATALOG (UPDIC)

status

installation_table | it : The installation table with the modified files listed in it.

installtion_catalog | ic : The installation catalog to update.

update_entries | ue : This is the list of specific files to update.

update_products | up : This is the list of products whose members are to be replaced.

output | o : This is the file to recieve the listings of what has been changed.

status : See NOS/VE error handling.

9.1.81 UPDATE_INTEGRATION_DESCRIPTORS (UPDID)

The purpose of this procedure is to create and merge a new set of program descriptors (created by create_integration_descriptors), onto the

the existing tools command library.

```
target_build_level, tbl : name = $optional
product_name, pn : name = $optional
build_level, bl : name = $optional
working_catalog, wc : file or key none = none
working_build_level, wbl : name = object
status
```

target_build_level | tbl : This is the os level used in the path names of all program descriptors which point to os-based files. It also indicates the tying file to be used for other product levels. The default if omitted is BUILD_LEVEL.

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

9.0 DESCRIPTION OF PROCEDURES
9.1.81 UPDATE_INTEGRATION_DESCRIPTORS (UPDID)

used.

status : See NOS/VE error handling.

9.1.82 UPDATE_SECONDARY_BUILD_DECKS (UPDSBD)

This procedure performs the following steps when the value of creation_call is true. 1. Generate the include_feature commands (deck_commands). 2. Extract the predecessor. 3. Enter the working environment. 4. Create the current build deck. 5. Update the build deck descriptions. 6. Insert a *copyc to the predecessor deck in the current build deck. 7. Insert a *copyc to the primary build deck in the current build deck. 8. Insert an if test in each successor build decks, testing if the bev$product_level equals the build deck level. 9. Exit the working catalog.

"IF creation call if false, then the following step are performed. "1. Generate the include_feature commands. "2. Extract the current build deck. "3. Enter the working environment. "4. Edit in only new information from the deck commands when action = 'building'. "5. Edit out information from the deck commands when action = 'pulling'. "6. Exit the working catalog.

```
update_secondary_build_deck
     creation_call, cc : boolean = true
     action, a : key building, pulling = building
     predecessor_build_level, pbl : name = none
     current_build_level, cbl : name = $required
     feature_list, fl : file = $null
     modification, m : name = $required
     author : string = $job(user)
     development_base, db : file = .intve
     product_name, pn : name = os
     working_catalog, wc : file = $optional
     status
```

creation_call | cc : This indicates whither this is the first operation performed at the specified level or not.

action | a : This indicates whither the specified features are to be added or removed from the current build.

predecessor_build_level | pbl : This is the build level which is to be the ancestor of the current level.

9.0 DESCRIPTION OF PROCEDURES
9.1.82 UPDATE_SECONDARY_BUILD_DECKS (UPDSBD)

current_build_level | cbl : This is the level to be altered. If cc=true, then the build deck for this level cannot exist yet.

feature_list | fl : This is a file containing a list (one per line) of features to be added (or deleted) from the current_build_level.

author | a : This is the person making the 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.

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

status : See NOS/VE error handling.

9.1.83 UPDATE_WEF$FEATURE_LIST (UPDWL)

The purpose of this procedure is to alter the wef$feature_list in the current working_catalog such that any compile jobs produced by MAKE_BUILD_JOBS will either include or exclude a specified set of features.

```
update_wef$feature_list
     action, a : key building, pulling = building
     feature_list, fl : file = $required
     working_catalog, wc : file = $optional
     development_base, db : file = .intve
     product_name, pn : name = object
     status
```

action | a : This indicates whither the specified features are to be added or removed from the wef$feature_list file.

feature_list | fl : This is a file containing a list (one per line) of features to be added (or deleted) from the wef$feature_list file.

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

9.0 DESCRIPTION OF PROCEDURES
9.1.83 UPDATE_WEF$FEATURE_LIST (UPDWL)

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

status : See NOS/VE error handling.

9.1.84 VALIDATE_BUILD_DECK_DESCRIPTION (VALBDD)

The purpose of this procedure is to check to see if the information contained on a build deck's deck_description line is correct or not.

```
validate_build_deck_description
     predecessor_build_level, pbl : name = $required
     current_build_level, cbl : name = $required
     successor_build_level, sbl: name = none
     status
```

predecessor_build_level | pbl : This is the value to be checked against the deck's predecessor_build entry.

current_build_level | cbl : This is the deck to check.

successor_build_level | sbl : This is the value to be checked against the deck's predecessor_build entry.

status : See NOS/VE error handling.

9.1.85 VERIFY_BUILD_COMPLETION (VERBC)

This procedure verifies that no files exist in the working_build_level catalog or the working_build_level.maintenance catalog for a given product and build_lev If files exist in either of those catalogs it indicates that either the procedur to move the files was not run or ran incorrectly. Whatever the cause, problems are sure to arise if the files are not handled correctly.

```
verify_build_completion
  product_name, pn : name = $optional
  build_level, bl : name = $optional
  working_build_level, wbl : name = object
  status : var of status = $optional
```

9.0 DESCRIPTION OF PROCEDURES
9.1.85 VERIFY_BUILD_COMPLETION (VERBC)

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

status : See NOS/VE error handling.

9.1.86 VERIFY_INSTALLATION_TABLE (VERIT)

This procedure verifies that each file referenced by the intve_path field of the installation_table actually exists.

```
verify_installation_table
     installation_table, it : file = $required
     output, o : file = $output
     status
```

installation_table | it : The installation table whose contents are to be checked.

output | o : The file to recieve the results.

9.1.87 VERIFY_LABELLED_TAPES (VERLT VERIFY_LABELLED_TAPE)

This procedure verifies labelled tapes on the 180 side by first requesting both the source tape and the target tape, then doing a COMPARE_FILE on them. If a list is specified for the TARGET_VSN, the procedure will continue to verifying the remaining tapes if one of the tapes does not verify correctly. It will specify the external vsn of the bad tape and it will also return bad status.

```
verify_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 : The external vsn of one tape in the comparison.

9.0 DESCRIPTION OF PROCEDURES
9.1.87 VERIFY_LABELLED_TAPES (VERLT VERIFY_LABELLED_TAPE)

target_vsn | tv : The external vsn of the other tape in the comparison.

recorded_vsn | rv : The internal vsn of both tapes.

type | t : Tape density.

status : See NOS/VE error handling.

9.1.88 VERIFY_MULTI_RECORD_TAPES (VERMRT VERIFY_MULTI_RECORD_TAPE)

This procedure verifies multi record tapes on the 180 side by first requesting both the source tape and the target tape, then doing a COMPARE_FILE on them. If a list is specified for the TARGET_VSN, the procedure will continue verifying the remaining tapes if one of the tapes does not verify correctly. It will specify the external vsn of the bad tape and it will also return bad status.

```
verify_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 : The external vsn of one tape in the comparison.

target_vsn | tv : The external vsn of the other tape in the comparison.

type | t : Tape density.

status : See NOS/VE error handling.

9.1.89 VERIFY_OS_BUILD_CATALOG (VEROBC)

This procedure verifies a specified build catalog after the build catalog is moved from one machine to another. Verification is accomplished by doing a compile_source, link_operating_system, and generate_deadstart_file. This ensures that all the necessary components for a particular build are there.

```
VERIFY_OS_BUILD_CATALOG
     deck, d: name = bam$open
```

9.0 DESCRIPTION OF PROCEDURES
9.1.89 VERIFY_OS_BUILD_CATALOG (VEROBC)

```
     volume_serial_number, vsn: name 1 ..  6 = i00061
     build_level, bl: name
     job_class, jc: key batch local maintenance = batch
     password, pw: name = $name($job(user)//'x')
     status
```

deck | d : specifies the deck to be compiled.

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

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

job_class | jc : specifies the class for the job.

Omission causes BATCH to be used.

password | p : specifies the password for your user name.

Omission causes job name concatenated with an 'x'.

status : see NOS/VE error handling.

9.1.90 VERIFY_RELEASE_TAPES (VERRT)

This procedure requests the release tapes and does a DISPLAY_BACKUP_FILE on each tape and prints the listings produced. A visual inspection of the listings should be done for proper verification of the release tapes. Only the 180 BACPF tapes can be verified via this procedure.

```
verify_release_tapes
     release_tapes, rt : list ..  of name ..  = $required
     status
```

release_tapes | rt : The vsn's of the tapes to be displayed.
status : see NOS/VE error handling.

9.1.91 VERIFY_TAPES (VERT VERIFY_TAPE)

This procedure will verify NOS/VE and NOS tapes from the 180 side by submitting a 170 batch job to verify the tapes.

9.0 DESCRIPTION OF PROCEDURES
9.1.91 VERIFY_TAPES (VERT VERIFY_TAPE)

```
verify_tapes
  source_vsn, svsn : name 1..6 = $required
  target_vsn, tvsn : list of name 1..6 = $required
  source_density, sd : key of, pe, ge = pe
  target_density, td : key of, pe, ge = $optional
  format, f : key of, si, i, l = l
  status : var of status = $optional
```

source_vsn | sv : The external vsn of one tape in the comparison.

target_vsn | tv : The external vsn of the other tape in the comparison.

source_density | sd : The density of the source_vsn tape.

target_density | sd : The density of the target_vsn tape.

status : See NOS/VE error handling.

10.0 STATISTICS

10.0 STATISTICS

This section of the manual describes the procedures used to produce statistics for NOS/VE TQMP. These procedures produce a matrix of line counts broken down two ways. The first breakdown is by language. The second breakdown is by line type (active, inactive, code, blank, etc.). These statistics can be produced either on a library or on a functional-area basis. Currently, these statistics are produced for each released system; but this could change to each cycle or even to each build level.

An example of one of these matrices is: (12312 data)

PROCESSOR	CODE	COMMENTS	BLANK	OTHER
CYBIL	497060	85129	119727	42375
SCL_PROCEDURES	70840	16686	4530	2267
CP_COMPASS	11353	4001	969	499
PP_COMPASS	13868	3514	1839	618
Z80_COMPASS	1037	136	219	22
CCL_PROCEDURES	4357	0	160	74
SYMPL	925	37	204	9
PROGRAM_DESCRIPTIONS	52	0	1	0
CYBIL_CC	15890	2078	2525	533
ASSEMBLE	8193	4232	59	90
GENERATE_COMMAND_TAB	190	0	0	0
MESSAGE_TEMPLATES	2	0	0	37
META	41	8	0	1
PASCAL	116	0	0	0
TEXT	0	341	0	0
TEXT_170	0	0	0	0
TOTAL	623924	116162	130233	46525

10.0 STATISTICS

PROCESSOR	ACTIVE	INACTIVE	TOTAL
CYBIL	744291	126127	870418
SCL_PROCEDURES	94323	9857	104180
CP_COMPASS	16822	25991	42813
PP_COMPASS	19839	10411	30250
Z80_COMPASS	1414	592	2006
CCL_PROCEDURES	4591	2174	6765
SYMPL	1175	1	1176
PROGRAM_DESCRIPTION	53	6	59
CYBIL_CC	21026	14816	35842
ASSEMBLE	12574	1158	13732
GENERATE_COMMAND_TAB	190	11	201
MESSAGE_TEMPLATES	39	50	89
META	50	2	52
PASCAL	116	0	116
TEXT	341	0	341
TEXT_170	0	4792	4792
TOTAL	916844	195988	1112832

10.1 COMBINE_STATS_PIECES (COMSP)

The purpose of this procedure is to either add or subtract the statistics matrices produced by get GET_LANGUAGE_STATS and GET_AREA_STATS. It is also used to combine the matrices of the subset libraries produced by SPLIT_SOURCE_LIBRARY into a combined matrix.

The subset library matrices will be members of a set of files which differ only in a end number. To combine these files base_file is set to the common portion of the file name and number_of_pieces is set to the highest number (eg. for files XX1, XX2, XX3, XX4; bf='XX' nop=4).

To combine files with dissimiler names, specify them on the additional_files parameter. Additional_files and base_file/number_of_pieces may both be spcified similtaniously when adding.

To subtract, both files must be specified on the additional files parameter. The second file is subtracted from the first.

```
combine_stats_pieces
     base_file, bf: string
     number_of_pieces, nop: integer = 0
     additional_files, af: list of file
```

10.0 STATISTICS
10.1 COMBINE_STATS_PIECES (COMSP)

```
     subtract, s: boolean = false
     output, o: file = $output
     status
```

base_file | This parameter specifies the common portion of a set of files whose names differ only in a final integer value. It is used in conjunction with number_of_pieces to combine the subset library statistic files produced when the source_library being analyzed is too large to analyze as a whole.

number_of_pieces | This parameter is used in conjunction with base_file. It specifies the highest integer suffix on a set of files. (All files from xx_1 to xx_number_of_pieces must exist.)

additional_files | This parameter is used to add or subtract matrix files of dissimiler names.

subtract | This boolean flag tells the procedure whether to add or subtract (default=add). To do a subtract operation, both elements must be specified on the additional_files parameter (base_file and number_of_pieces are invalid for a subtract). The second element specified on additional_files is subtracted from the first.

output | The file to which the results are written.

10.2 CREATE_PRODUCT_STATS_JOBS (CREPSJ)

The purpose of this procedure is to produce statistics for a specified product at a specified build level. To do this, the procedure follows the following algorithm.

1. Extract a complete copy of the source_library as of the specified build level.
2. IF get_language_stats
   a. Split the source library (IF NEEDED)
   b. Create a job for each library piece
3. Create a job for each area specified.
4. IF submit=immediate; submit all jobs created

```
create_product_stats_jobs
     user_number, un: name = $user
     password, pw: name = $required
     get_language_stats, gls: boolean = true
     area_stats, as: list of name 1 ..  4 or key all none = none
```

```
     processor, p: name = all
     include_active_stats, ias: boolean = true
     include_inactive_stats, iis: boolean = true
     submit, s: key immediate, nosubmit = nosubmit
     stats_catalog, sc: file = $user.stats_catalog
     development_base, db: file = .intve
     product_name, pn: name = os
     build_level, bl: name
     status
```

user_number | This is the user number to run the statistics jobs on.

password | This is the password for the user number.

get_language_stats | This flag determines whether to produce jobs for the library-level statistics.

area_stats | This parameter allows the user to request statistics for a functional area or even a subset of a functional area.

processor | This allows the user to restrict statistics gathering to just one language.

include_active_stats | This flag controls whether statistics are produced for active lines of code.

include_inactive_Stats | This flag controls whether statistics are produced for inactive lines of code.

submit | This parameter allows automatic job submittal of the batch jobs produced by this procedure.

stats_catalog | This is a permanent file catalog used by this procedure to retain the following items: a copy of the product source_library at the specified build_level, a set of source_library subsets if the original copy needs to be split up, a copy of all batch jobs produced, and a copy of all output files from the execution of the batch jobs.

development_base | The master catalog which contains the product_name source_library. See W-E default hierarchy.

development_base | The subset catalog which contains the source_library. See W-E default hierarchy.

build_level | The level of the source_library for which statistics are desired. See W-E default hierarchy.

10.3 GET_AREA_STATS (GETAS)

The purpose of this procedure is to generate a statistics matrix for a functional area. A functional area is determined by the two character prefix of the deck name. This procedure permits up to 3 characters to be specified to allow a finer breakdown by deck type as well as area. This procedure operates by determining the starting and ending decks of the area in question on the source library, extracting the range of decks between them and then calling get_language_stats to produce the matrix. The procedure allows the keyword ALL to be specified to generate statistics for all available functional areas. Another feature of this procedure is the ability to specify a funtional area followed by the key ALL such as: "A=(BA, ALL)". This will result in the production of statistics for the area specified and all areas following it.

```
get_area_stats
     library, l: file = $required
     area, a: list of name 1 ..  3 or key all = all
     processor, p: name or key all total = all
     include_active_stats, ias: boolean = true
     include_inactive_stats, iis: boolean = true
     output, o: file = $output
     status
```

library | The source library file for which statistics are desired.

area | The functional area(s) for which statistics are desired.

processor | This allows the user to restrict statistics gathering to just one language.

include_active_stats | This flag controls whether statistics are produced for active lines of code.

include_inactive_Stats | This flag controls whether statistics are produced for inactive lines of code.

output | The file onto which the statistics are to be written.

10.4 GET_INACTIVE_STATS (GETIS)

The purpose of this procedure is to count all inactive lines on a scu library.

```
get_inactive_stats
     library, l: file = $required
     inactive: var of integer
     status
```

library | The source library file for which statistics are desired.

inactive | This is the variable which returns the count of the number of inactive lines on library.

10.5 GET_LANGUAGE_STATS (GETLS)

The purpose of this procedure is to produce a statistical matrix of all lines on a source library. The line count is broken into a matrix by language type and by line classification. The current line classifications consist of CODE, COMMENTS, BLANK, OTHER, TOTAL_ACTIVE, TOTAL_INACTIVE, and TOTAL.

```
get_language_stats
     library, l: file = $required
     processor, p: name or key all total = all
     include_active_stats, ias: boolean = true
     include_inactive_stats, iis: boolean = true
     output, o: file = $output
     status
```

library | The source library file for which statistics are desired.

processor | This allows the user to restrict statistics gathering to just one language.

include_active_stats | This flag controls whether statistics are produced for active lines of code.

include_inactive_Stats | This flag controls whether statistics are produced for inactive lines of code.

output | The file onto which the statistics are to be written.

10.6 GET_TEXT_STATS (GETTS)

The purpose of this procedure is to return the total number of lines in a text file along with some subclassifications of code, comments, blanks, and other. The basic algorithm for each subclassification is:

1. Delete all lines in the classification
2. Find how many lines are left in the file (using the bottom line as a reference).
3. Determine the difference between this number and the number of lines before the deletion.

```
get_text_stats
     text: file = $required
     language_type: key cybil, scl, compass, sympl, pascal,
     assemble, text, other = $required
     code: var of integer
     comments: var of integer
     blank: var of integer
     other: var of integer
     total: var of integer
     status
```

text | The file which contains the text to produce statistics about.

language_type | The language that the text consists of. This parameter is required to determine the difference between strings and comments, for example.

code | This parameter is returned with the count of the number of lines of actual code in the text file.

comments | This parameter is returned with the count of the number of lines of comments in the text file.

blank | This parameter is returned with the count of the number of blank lines in the text file.

other | This parameter is returned with the count of the number of other lines in the text file. (Other consists of things which are 'code-like'; such as scu directives, pragmats, etc.).

total | This parameter is returned with the count of the number of total lines in the text file.

10.7 SPLIT_SOURCE_LIBRARY (SPLSL)

The purpose of this procedure is to split a source_library into a number of subset libraries so that they can be processed independently.

```
split_source_library
     library_file: file = $required
     piece_size, ps: integer = 1000
     resultant_files, rf: file = $user.subset_sl
     resultant_count, rc: var of integer
     status
```

library | The source library file for which statistics are desired.

piece_size | This is the number of decks to break out into one subset library.

resultant_files | This is the common portion of the subset library file name. (E.g. if rf=XXX, then the subset libraries will be XXX_1, XXX_2, etc.)

resultant_count | This is a variable which returns the number of subset libraries produced.

Table of Contents