INTEGRATION AND DEVELOPMENT GUIDELINES AND CONVENTIONS

for the NOS/VE product set

1.0 INTRODUCTION

1.0 INTRODUCTION

This document describes the process for integration and development of the NOS/VE product set. It is directed mainly toward the role of development in their interaction with integration.

An overview of our catalog structure is discussed first. This includes the residence of all the NOS/VE product's source and the structure of a completed, installed product set.

The source for each product must be in NOS/VE Source Code Utility format. The conventions for these SCU libraries are given in section 3.

Section 4 will prove useful as a guide for all of our available special procedures and tools.

If a developer is transmitting a new product to integration, he or she must provide a short verification job and a quicklook test which are run after the completion of each build cycle. The instructions for this process are given in section 5.

Sections 6 and 7 describe our development procedures. These procedures were written in Arden Hills for their operating system development and we have adopted them in Sunnyvale for the products. Appendix A gives a summary of each command and its parameters.

2.0 CATALOG STRUCTURE

2.0 CATALOG_STRUCTURE

2.1 SOURCE_MAINTENANCE_STRUCTURE

The SCU library for each product is kept in the INTVE catalog. Once integration receives a new product, the source library is placed into this catalog. From this moment on, this will be the only copy of the source and the developers use this as their base for extracting and replacing changed decks.

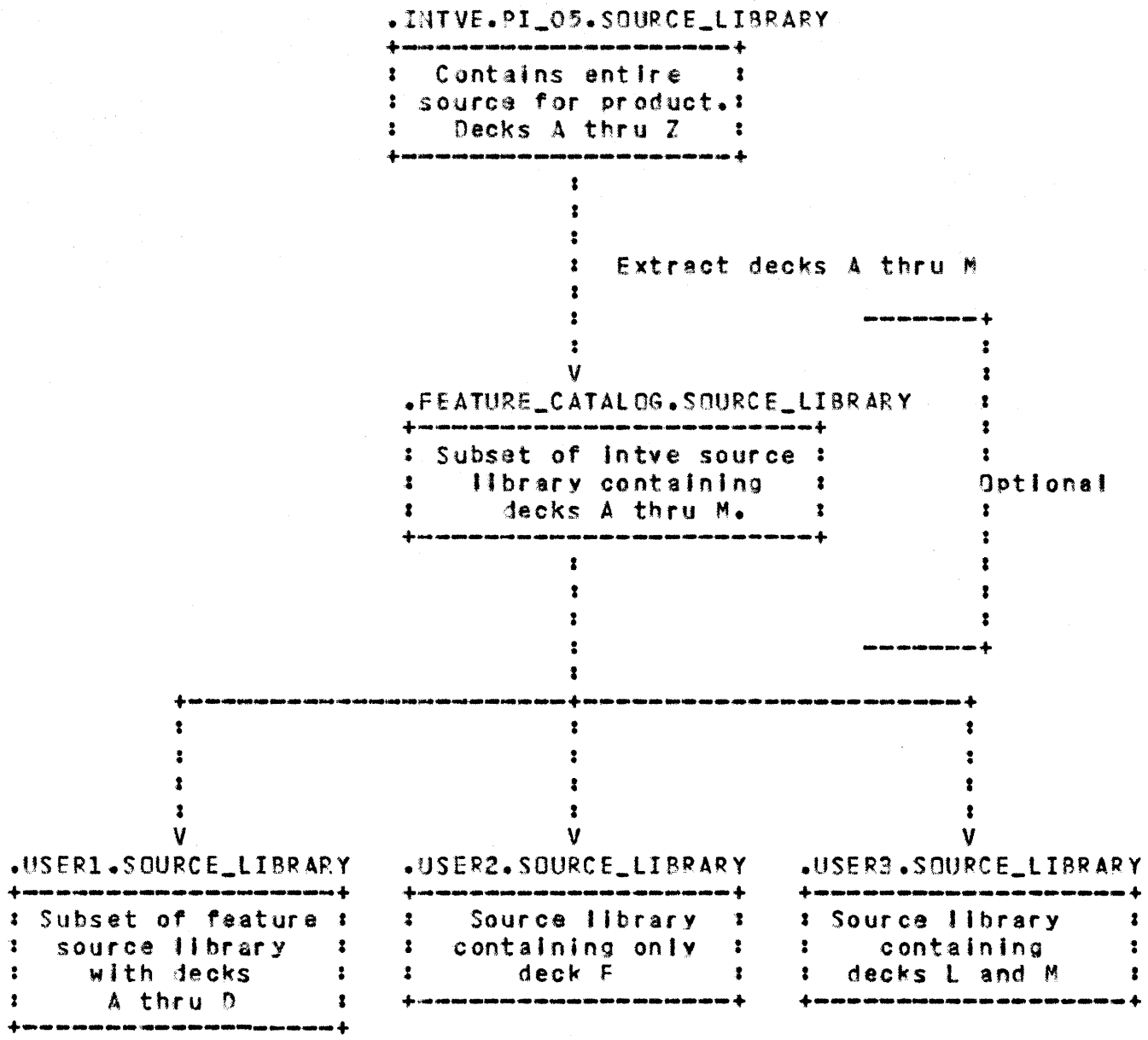
When a developer extracts decks from their INTVE source, this subset of decks will be in SCU format and reside on their "working catalog". This is usually a person's main catalog.

If a number of individual developers are working on the same feature or project for a product, they can set up a "feature catalog". The feature catalog, i.e. another main catalog or a sub-catalog, would contain all the decks pertaining to the project. Each developer would extract decks from the source in the feature catalog into their individual working catalogs. See diagram 1. Section 6 discusses the procedures which work with this catalog structure.

Decks are interlocked when they are extracted to prevent updates getting lost by being overwritten by an older version of the deck. Interlocks are discussed more in section 3.

2.0 CATALOG STRUCTURE
2.1 SOURCE MAINTENANCE STRUCTURE

DIAGRAM 1



Notes: PI is the two-letter product identifier

2.0 CATALOG STRUCTURE
2.1.1 THE INTVE CATALOG

2.1.1 THE INTVE CATALOG

Each product will have a subcatalog in the main integration catalog. Each product subcatalog will contain all the data necessary to maintain that product, specifically the SCU source library and build procedures.

All the source libraries in the INTVE catalog are kept in a subcatalog identified by the product's 2-letter identifier followed by the release sequence number.

For example, the FORTRAN product's source files corresponded with the following releases:

FILE_	RELEASE_	DATE
.intve.fc_01.source_library	R1.0.2	April, 1984
.intve.fc_02.source_library	R1.1.1	August, 1984
.intve.fc_03.source_library	R1.1.2	est. March, 1985
.intve.fc_04.source_library	R1.1.3	est. June, 1985
.intve.fc_05.source_library	R1.2.1	est. December, 1985

Of course, not all the files are on our disk packs at one time in order to conserve disk space. Normally, one would find two source libraries for each product on the INTVE catalog. The previous libraries are stored onto magnetic tapes.

The following is a table showing each product with its corresponding two letter identifier:

2.0 CATALOG STRUCTURE
2.1.1 THE INTVE CATALOG

IDENTIFIER	PRODUCT
AA	AAM - Advanced Access Method
AP	APL - A Programming Language
CB	COBOL compiler
CC	CCM - Common Compiler Modules
CG	CCG - Common Code Generator
DB	DEBUG utility
FA	FMA - File Migration Aid
FC	FORTRAN compiler
FL	FRTL - Fortran Runtime Library
FM	FMU - File Management Utility
HU	HELP - Help Utility for On-line Manuals
LI	LISP - Lisp Interpreter
ML	CMML - Mathematics Library
OM	MANUALS - On Line Manuals
PA	PASCAL compiler
PE	PROGRAMMING ENVIRONMENTS
PR	PROLOG interpreter
SM	SORT/MERGE utility
ST	EDIT CATALOG utility
TT	TEST TOOL LIBRARY - Testing Environment Utility
ZS	ZEUS - A Report Generator

Each source library in the INTVE catalog must have a "latest changes" and a "logging" file associated with it. For example, for fortran's third release, the INTVE catalog would contain .INTVE.FC_03.SOURCE_LIBRARY, .INTVE.FC_03.LATEST_CHANGES and .INTVE.FC_03.LOGGING.

The latest changes file is an SCU source library containing one day's worth of updated decks located in the source library. When a developer wants to update code on the source library in INTVE via the transmit_to_integration procedure, the changes are located only on the latest changes file until the following day. Each morning, all of our NOS/VE product's latest changes files are combined into the main source library.

The logging file contains information about each extract of code or transmission of code between source libraries.

2.0 CATALOG STRUCTURE
2.2 CATALOG STRUCTURE OF A COMPLETED PRODUCT SET

2.2 CATALOG STRUCTURE OF A COMPLETED PRODUCT SET

A completed product set built by integration is usually installed into the $SYSTEM catalog. If there are concurrent builds related to different releases, the closest release is installed in $SYSTEM and the future release is installed into the INT180 catalog. The $SYSTEM products are automatically available to anyone using NOS/VE. To use products from the INT180 catalog, see the SETPC command in the Sunnyvale Tools Library, described in DCS document S4702.

The $SYSTEM and INT180 catalog structure consists of full product name sub-catalogs containing run time libraries and bound products. A further breakdown in structure is the maintenance catalog which contains files such as unbound libraries and maps.

For example, a FORTRAN product installed into the $SYSTEM catalog would consist of these files:

FILE	DESCRIPTION
$system.fortran.bound_product	Fortran compiler with all references to other product's satisfied
$system.fortran.flf$library	Fortran library used during run time or execution
$system.fortran.maintenance.unbound_component	Fortran library created solely from cybil and assemble binaries
$system.fortran.maintenance.bound_component	Bound fortran module
$system.fortran.maintenance.map	Loadmap created during the binding process
$system.fortran.maintenance.command_library	Command library containing fortran's program descriptions and error templates

Not all products have all these related files. Some common products are not bound.

The catalog structure is the same in the INT180 catalog. The only difference is the main catalog name.

3.0 SCU LIBRARY ORGANIZATION

3.0 SCU_LIBRARY_ORGANIZATION

3.1 DECK_ORGANIZATION

3.1.1 USAGE OF GROUPS

Critical to the structure of the product's source libraries, and to the efficiency of the source maintenance procedures, is the association of SCU "group names" with each deck on the library. These groups may be used to manipulate "blocks" or "groups" of decks, such as all decks in a job template or system core library, fairly easily.

The different types of groups are identified by the conventions used to name them. Except for the "processor" and "generic" groups (described below), the format of all group names is:

xxy$aaaaaa

where 'xx' is the product identifier, "aaaaaa" is a descriptive name for the particular group, and "y" is one of the following:

S Specifies a "Source" group, i.e. a subdivision of the source library into component libraries. Several examples of groups of this type are:

oss$program_interface (osf$program_interface)
fcs$front_end
cbs$cobol_source

F Specifies a "destination File" group (i.e. a file onto which a deck is to be placed after processing). Several examples of groups of this type are:

aaf$44D_library
osf$monitor
osf$object_code_utilities

3.0 SCU LIBRARY ORGANIZATION
3.1.1 USAGE OF GROUPS

G Specifies a "Group", i.e. a collection of decks that have been decided would be useful to be able to refer to en masse. Several examples of groups of this type are:

```
cbg$procedure_common_decks
cbg$run_time_procedures
fcg$bridge_modules
```

NOTE: All decks on the source library which belong in the library psf$external_interface_source should be associated with the group name "psf$external_interface_source". Keypoint decks and error codes should also belong to this group.

Most other group names will follow the conventions described in the previous text for the product source libraries. However, there are two classes of groups for which this is inappropriate: "processor" groups and "generic" groups.

A "processor" group will be given the same name as the corresponding processor, e.g. the group name for decks to be compiled by CYBIL will be "CYBIL". Other processor group names are:

```
assemble
fortran
cobol
pp_compass
cp_compass
cybil_cc
```

Note that some of these must obviously be processed on the NOS side of the machine.

A "generic" group is used in those cases where knowing the processor for a deck is insufficient for some purpose. The generic groups that have been identified and are required, if applicable, are:

```
common
program_descriptions
message_templates
scl_procedures
ccl_procedures
scu_selection_criteria
build_procs
deleted_decks
```

All decks which are called by other decks will belong to the group "common". When a deck needs to be deleted, it should belong

to a group "deleted decks". At this point in time, the deck is not really deleted, so the build procedure must be specifically excluding this deck until someone in integration can physically delete the deck. This will only be done between releases to insure our ability to back up to a previous level.

Some of these generic groups overlap to some extent with the processor groups.

A deck may have up to 255 group names associated with it. At this point in time, a group cannot be associated to another group. A DAP has been written to allow this capability. For the time being, issue the change_deck SCU command to establish all deck to group associations.

3.1.2 INTERLOCKS

Among the information in an SCU deck header are the original interlock and sub-interlock fields. When a deck is extracted from a source library, the sub-interlock field is set to contain the name of the user which asked for the deck and the date and time of the extraction. After this sub-interlock field is set, the deck cannot be extracted by anyone.

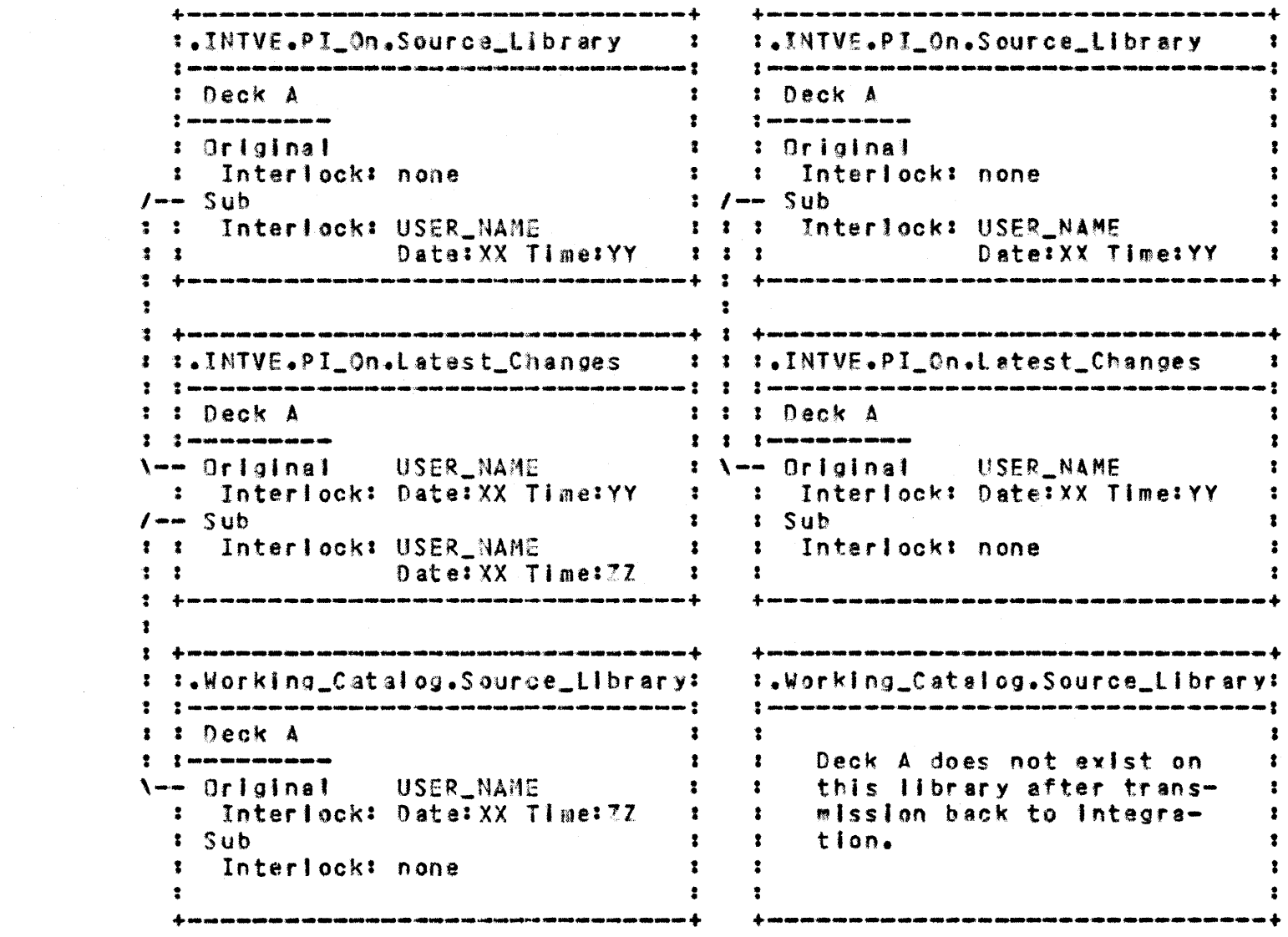
When the sub-interlock field is set on the library where the deck resides, the deck now also resides on the source library in the user's catalog. The original interlock field is set for this deck on the user's source library to the same values of the main library's sub-interlock field, i.e. the user name and date and time of extraction.

After changes are made to this deck and it is transmitted back to the main library, the transmit command used checks that these sub-interlock and original interlocks match between the two source libraries. After the transmit is complete, the two interlock fields are cleared.

The following diagrams explain the interlock settings when a deck is extracted from the integration catalog. The first example shows an extraction into the working catalog only. The second example shows the result of using a feature catalog.

3.0 SCU LIBRARY ORGANIZATION
3.1.2 INTERLOCKS

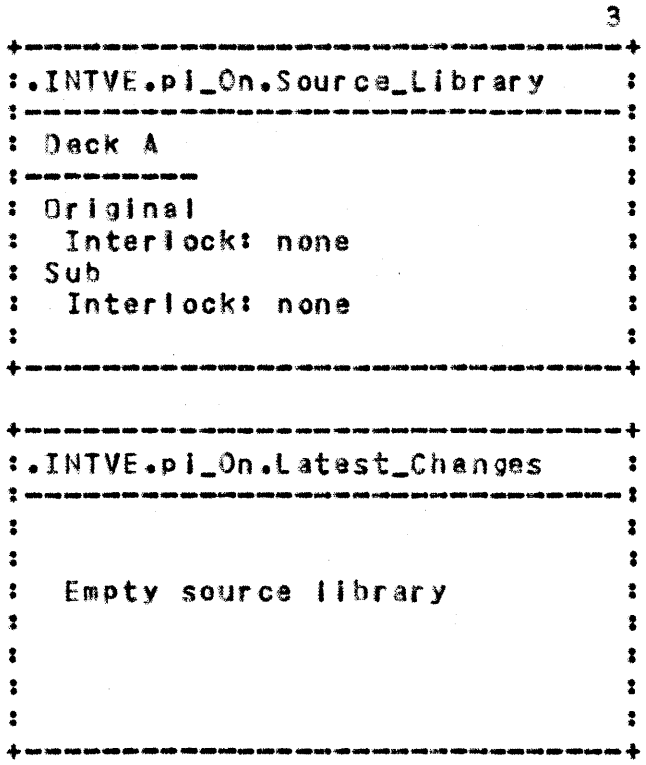
Interlocks: Usage of a working catalog only, i.e. no feature catalog.



Deck A has been extracted into the working catalog and all interlocks have been set. This deck can now be modified in the working catalog. Deck A cannot be extracted for the purposes of making changes at this time by anyone else.

The deck has been transmitted back to the integration catalog. Even though an interlock still exists between the intve latest changes and main source library, this deck can be re-extracted by another or the same user.

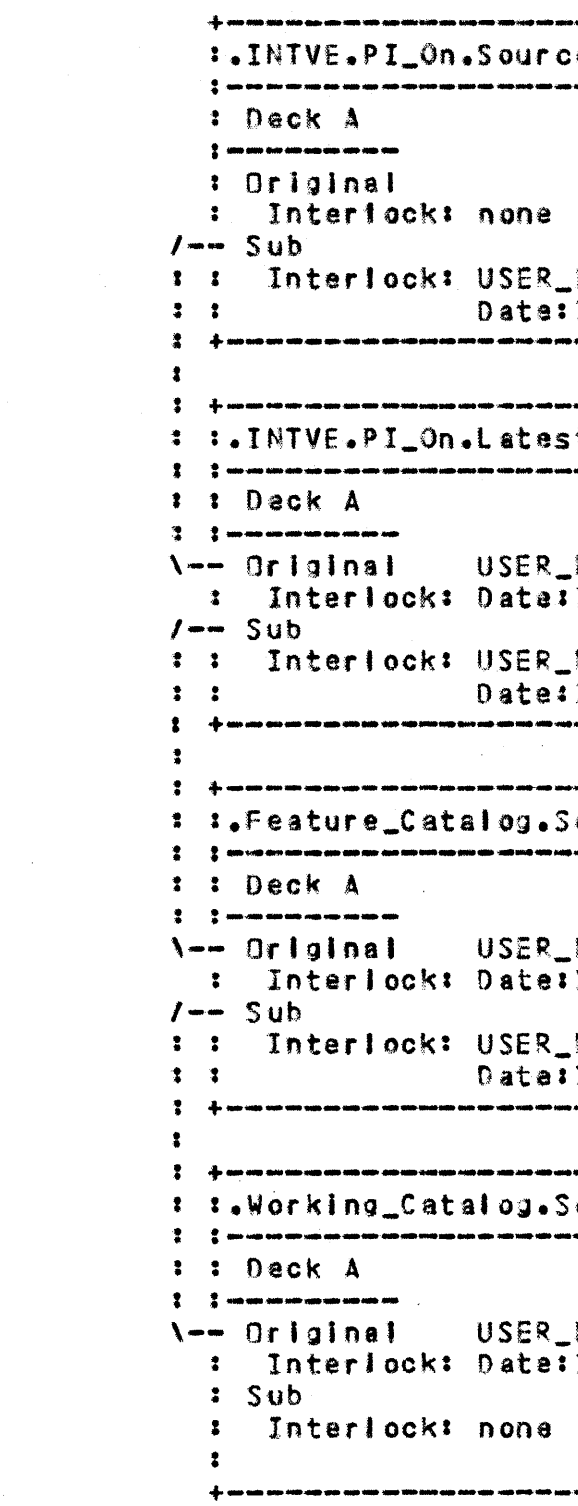
3.0 SCU LIBRARY ORGANIZATION
3.1.2 INTERLOCKS



This diagram shows the result of the latest changes file having been combined with the main source library. This automatically occurs every morning and is the responsibility of the integration group.

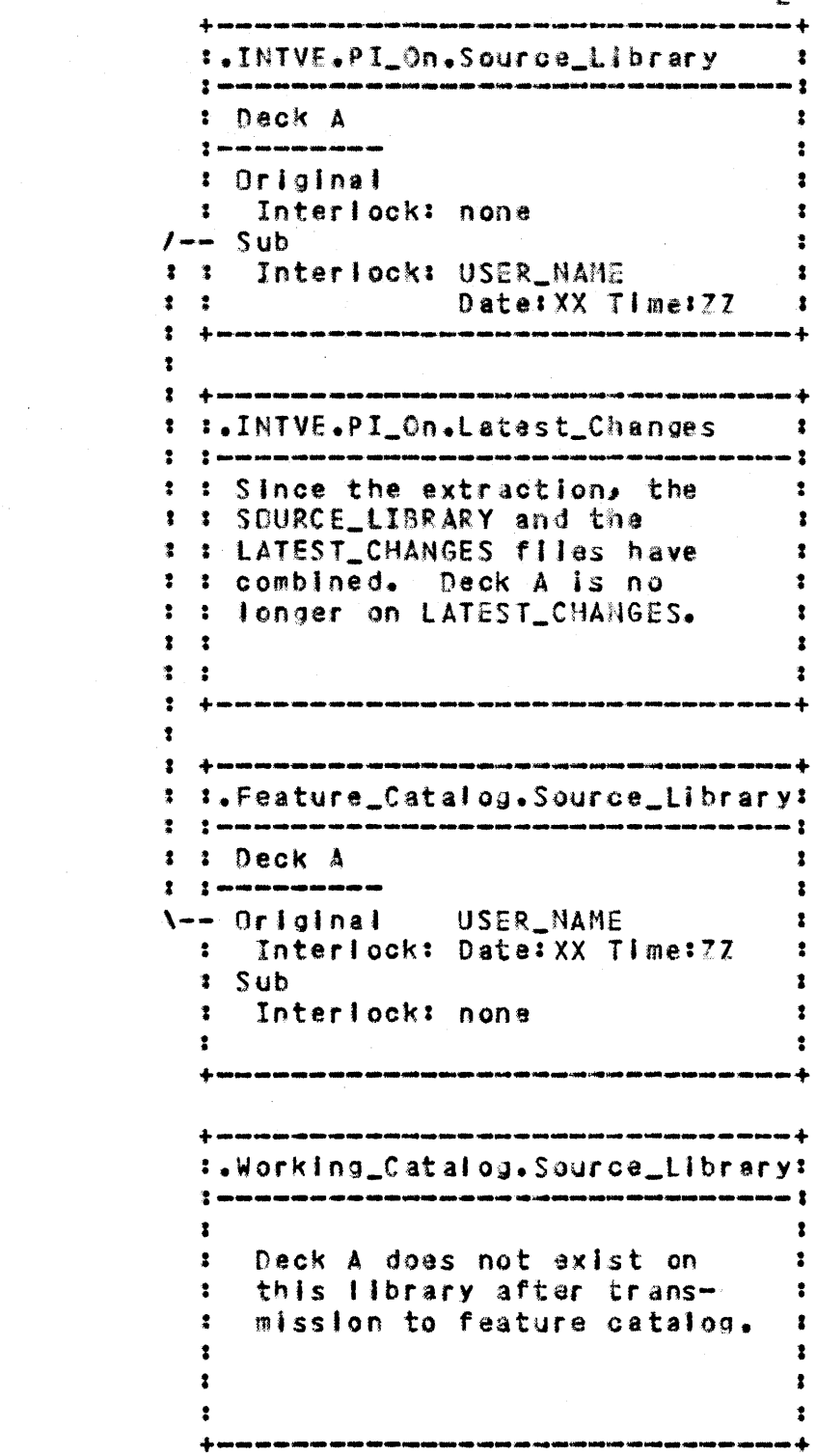
3.0 SCU LIBRARY ORGANIZATION
3.1.2 INTERLOCKS

Interlocks: Usage of a feature catalog.



Extraction of Deck A through the feature catalog to the working catalog sets deck interlocks. Changes to the deck are made at this time in the working catalog. Deck A cannot be extracted now by anyone.

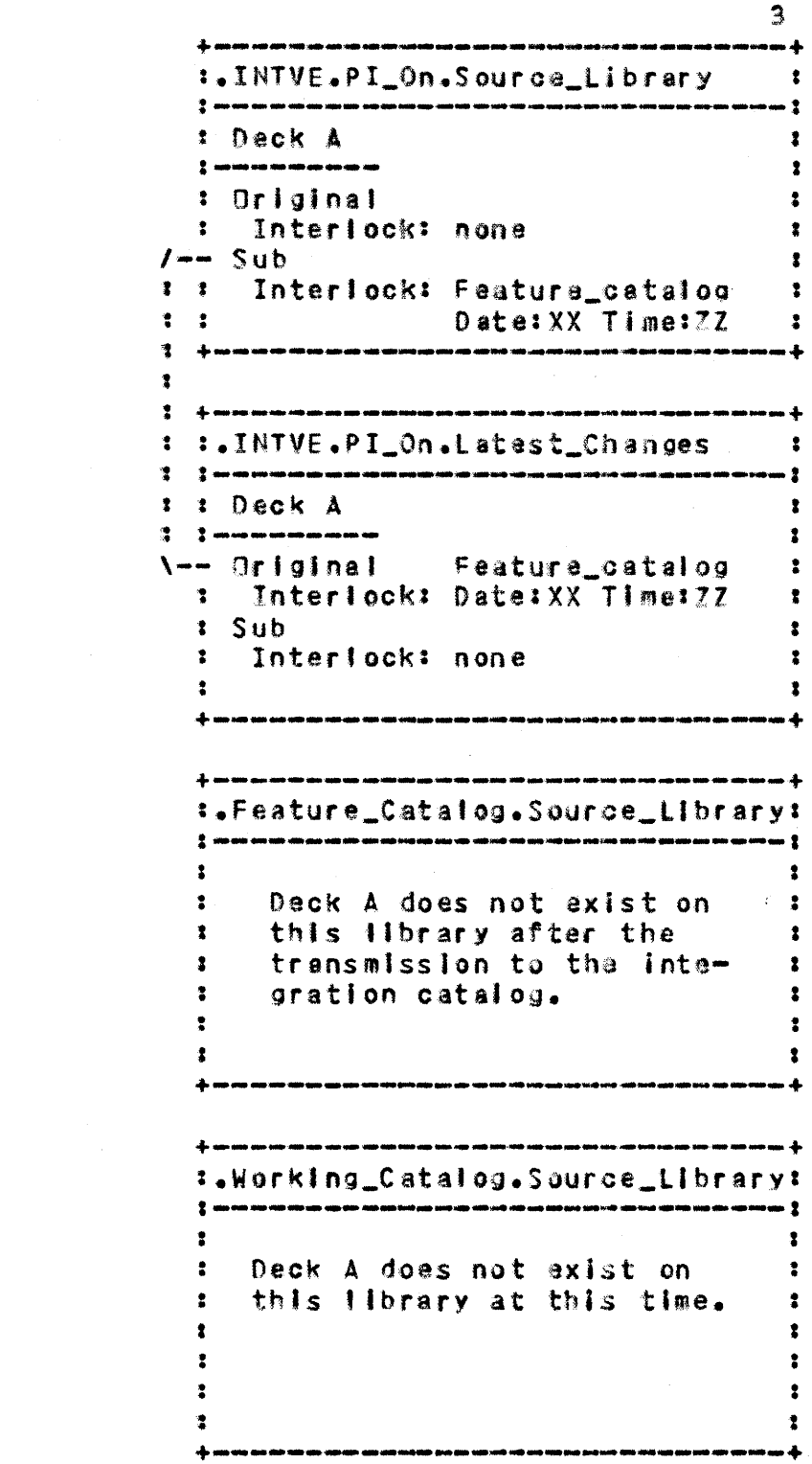
3.0 SCU LIBRARY ORGANIZATION
3.1.2 INTERLOCKS



Deck A was transmitted back to the feature catalog. The deck can now be re-extracted into another working catalog or given back to the INTVE catalog.

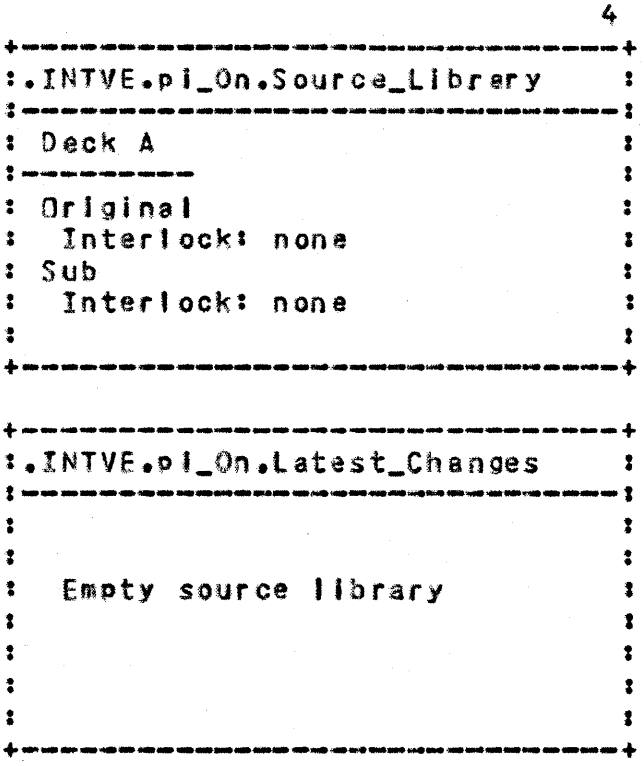
Note the change in the interlock on SOURCE_LIBRARY due to the combining with LATEST_CHANGES.

3.0 SCU LIBRARY ORGANIZATION
3.1.2 INTERLOCKS



Deck A was transmitted back to the integration catalog. The deck could be re-extracted at this time. Note that the date and time associated with the interlocks here are the same as those associated between the source library and feature catalog source libraries in the previous diagram.

3.0 SCU LIBRARY ORGANIZATION
3.1.2 INTERLOCKS



This diagram shows the result of the latest changes file having been combined with the main source library. This automatically occurs every morning and is the responsibility of the integration group.

3.2 MODIFICATION ORGANIZATION

3.2.1 MODIFICATION CONVENTIONS

Each change made to a deck or a group of decks on an SCU library is associated with a modification name. Every modification header should have information containing the author's name and a description of the purpose for the change.

The following conventions for modifications are proposed:

- Modifications for PSR's must be named by the PSR number wherever possible. If this isn't possible, or if the mod answers more than one PSR, the PSR number(s) must be in the description. If the name is the number, it should still be in the description, where space permits. If a BSL is associated with the PSR, the DCS number must be in the description.

- Mods for a specific DCS document such as a DAP should be named by a form of the DCS number. If the project has another convention, or if there are many mods associated with the document, the description must contain the DCS number.

- If there are several mods for a feature, the description of the last mod transmitted before Evaluation begins testing of the feature must begin with the word "FINAL". A memo or note should also be sent to the responsible evaluator.

- Descriptions of modifications which are not PSR's or external features must begin with an exclamation point. Following the exclamation point must be a keyword indicating the type of modification. We propose the following keywords:

  PERF - This signals that the modification is intended to improve performance. Performance enhancements which are PSR responses must follow the convention for other PSR's.

  DOC - This indicates that the modifications are to internal documentation.

3.0 SCU LIBRARY ORGANIZATION
3.2.1 MODIFICATION CONVENTIONS

FMT - This applies to all modifications made to improve readability of the source code. Cybforming, addition of titles and page ejects, and many deck name changes would be made by this type of modification.

PROJ - This would identify changes to code which will never be released. Such code would include debugging statements and project utilities such as table generators.

Other keywords will no doubt be added to the list.

- Bug fixes which are not PSR'd will not be tagged by an exclamation mark. Wherever possible, however, PSR's should be written.
- The build content report will list the decks modified, so the description does not have to do this.
- The description, beyond the tag described above, should be a clear description of the changes made. Complete sentences are recommended.
- The author field must be filled in. Since the MADE_MODIFICATION procedure sets this to the value specified by the SETWE procedure, this is no hardship. The full last name is preferred to initials or user name only.

3.2.2 USAGE OF FEATURES

If a project wishes to group certain modifications, each modification can be assigned to the same feature. Conventions for feature names have not been set up, so the project can choose their own names.

If a modification does not have a feature associated with it by the time it is going through an integration build, the integration group will give it a feature name BUILD_XXXX where XXXX is the current product set build level.

3.2.3 STATES

SCU states will be used to indicate the degree to which a modification has been tested. Integration, being the owner of INTVE, will have sole authority in upgrading the state codes higher than state 1.

State 0 - 'Under Development'

Code with state 0 modifications resides in the feature and working source libraries. State 0 is used on an integration source library to designate a modification that has been rejected. (This rejection process needs to be defined by Evaluation and Integration.)

State 1 - 'Testable'

Code and test modifications will be transmitted to the system source library at state 1. This code has successfully passed the unit tests and is now ready to be included on the development system.

State 2 - 'Integrated'

Code with state 2 modifications has been formally integrated into the base system by Integration. Quicklooks are executed. Evaluation executes the regression and feature testbases.

State 3 - 'Release Candidate'

Code with state 3 modifications has passed a set of criteria (e.g. stability, performance, documentation criteria) specified in the test plans. (This approval process needs to be defined by Evaluation and Integration.) The system is installed in closed shop.

State 4 - 'Released'

Code with state 4 modifications has been formally released.

There will be no state 0 modifications on the INTVE source library with the possible exception of those that have achieved a higher state and then been rejected for some reason.

3.3 VERSION_REPRESENTATION_OF_AN_SCU_LIBRARY

Each source library in the INTVE catalog of the product set will contain a version representing the build level of which the library was last built for the closed shop system. The level will be contained in the header field of the library. These level numbers are changed by the integration group.

3.4 COPYRIGHT_CODE

Copyright notices are required to be part of every CDC software product. The source code for each product must include a copyright notice as comments within the first twenty lines such that the notice appears with the name of the product. The comments should be placed within the module which is initiated first (where the starting procedure resides). For products where this is not applicable, place the copyright comments in the first alphabetic module. This information must also be contained in the resultant binary of the product. The Cyber 170 convention is applicable, i.e.

```
COPYRIGHT CONTROL DATA CORPORATION 19xx
```

It is the development project's responsibility for generating and transmitting code to include the copyright information in the source. Annual changes to update to the current year is not required. Integration places the copyright information into the product's bound binary module comment field.

For further information, refer to the CDC procedure on copyright ownership notices, policy number 10:08:01:011.

4.0 TOOLS

4.0 TOOLS

There are a few libraries available which contain helpful procedures for everyone to access. The following is a list of these libraries, how they are accessed and a list of the tools they provide.

4.1 SVE$TOOLS_LIBRARY

LIBRARY: svf$tools_library
ACCESS: Library is automatically in the command list when logged onto NOS/VE. (Available only in Sunnyvale)

More information about the tools listed below can be obtained from the DCS document, S4702.

TOOLS

BANNER
BURST_COMPILATION_LISTINGS
CHOOSE
COUNT_LINES
CYBXREF
FORMAT_STRING
SCOOP
SELECT
SENATOR
SET_MAXIMUM_STACK_SIZE
XEDIT
ALARM
CHANGE_FEATURE
CHANGE_GLOBAL
CHANGE_GROUP_NAME
COUNT_ACTIVE_LINES
DETACH_UNIQUE_FILE
DISPLAY_ALL_PERMITS
DISPLAY_CATALOG_CONTENTS
DISPLAY_MODIFICATIONS_BY_STATE
DISPLAY_NEW_MODIFICATIONS
EDIT_DESCRIPTION
EXPAND_SOURCE_FILE
EXTRACT_PROC

4.0 TOOLS
4.1 SVF$TOOLS_LIBRARY

GENERATE_BUILD_CONTENT_REPORT
GENERATE_SELECTION_CRITERIA
HARDCOPY
INTEGRATE_MODIFICATION
PRINT
PULL_MODIFICATIONS
PUT_LISTINGS_ON_SOURCE_LIBRARY
REPORT_MODIFICATION
SET_PRODUCT_CATALOG
UPDATE_PRODUCT
DISPLAY_USER_COUNT
LIST_BASE
PLOT_USERS
DUMP_CATALOGS
DISPLAY_FILE_USERS
COMPARE_DATES
COMPRESS_DELETE_LISTING
SCREEN

4.2 SES/VE_TOOLS_LIBRARY_=_RELEASE_1

LIBRARY: SES/VE tools library
ACCESS: Set_command_list Add=.sesve.stf$tools

TOOL_	DESCRIPTION
CHANGE_REMOTE_FILE	Change NOS file access parameters
CHECK_MAIL	Checks NOS SES and NOS/VE mailbox files
CREATE_NEW_MAILBOX	Reinitializes your NOS/VE mailbox
CREATE_REMOTE_FILE_PERMITS	Sets NOS file access permissions
DELETE_REMOTE_FILE	Purges NOS files
DISPLAY_ACTIVE_USERS	Displays currently logged on NOS/VE users
DISPLAY_NOS_CATALOG	Displays a NOS catlist
DISPLAY_TOOL_HELP	Displays parameters and

	explanation of a tool
DISPLAY_TOOL_STATUS	Displays a selected tool's current status
EXTRACT_LINES	Writes lines from an input file to an output file if the line contains the chosen string
EXTRACT_STRING	Writes strings only from an input file to the output file if the string is found in the input file
FORMAT_TEXT	Runs the NOS TEXTPRO package
GET_MAIL	Gets entries from NOS SES and NOS/VE mailboxes
GET_REMOTE_FILE	Acquires a NOS permanent file
GENERATE_LARGE_BANNNER	Puts a large letter banner at the beginning of a file
HELP	Gives access to the SES/VE tools library online user manual, the NOS/VE messages manual, etc.
PRINT_BANNERED_FILE	Prints a file which has banner information included
PUT_LINE_USING	Produces formatted SCL output
REDO_COMMAND	Re-executes previous SCL command, editting allowed
REPLACE_REMOTE_FILE	Replaces a NOS permanent file
RETRY_COMMANDS	Re-executes a sequence of commands, editting allowed
RUN_CADS_GRAPHICS	Executes the NOS/VE CADSG package. (This cannot currently be accessed because of a Fortran compiler problem.)
RUN_LINK_JOB	Causes a file to be transferred

4.0 TOOLS
4.2 SES/VE TOOLS LIBRARY - RELEASE 1

	to NOS to be executed
RUN_NOS_JOB	Causes a file to transferred to NOS to be executed, or to any other linked NOS machine

In order to access more information regarding the tools listed above, enter the HELP command after doing the SETCL A=.SESVE.STF$TOOLS.

4.3 OTHER_USEFUL_TOOL_LIBRARIES_AVAILABLE

Another useful library which contains tool procedures is the $system.osf$site_command_library. This is automatically available to anyone who is logged onto the NOS/VE operating system. The commands will not be listed here because there are so many and they are used by several different groups or projects. The user can display the commands and access them if needed.

The library which contains the procedures for development to maintain their product's source library is .INTVE.COMMAND_LIBRARY. To access this library, the command, "Set_program_attributes Add=.INTVE.COMMAND_LIBRARY" is executed. Usually, this command is added to the user's prolog file.

Section 6 discusses these procedures in more detail and includes several examples.

5.0 NEW PRODUCT CONSIDERATIONS

5.0 NEW PRODUCT CONSIDERATIONS

5.1 PRODUCT HEADER INFORMATION

The following information about a product's level and version number represents Section 3.3.1.6 of the Cyber 180 System Interface Standard, DCS document S2196. Product's which are run interactively should also conform to these standards.

5.1.1 PRODUCT VERSION NUMBER

This number is in the form 9.99 and represents the product's release version. For release 1.1.3, the version number for each product are as follows:

Product	Version Number
AAM	1.1
APL	1.0
ASSEMBLE	1.1
BASIC	1.0
COBOL	1.2
DEBUG	1.2
EDIT CATALOG	1.0
FMA	1.1
FMU	1.1
FORTRAN	1.1
HELP UTILITY	1.2
LISP	1.1
PASCAL	1.0
PROGRAMMING ENVIRONMENT	1.0
PROLOG	1.0
SORT	1.1
ZEUS	1.0

5.0 NEW PRODUCT CONSIDERATIONS
5.1.2 PRODUCT LEVEL

5.1.2 PRODUCT LEVEL

Each product which calls CCM to generate a listing must set the ccm variables, ccv$install_date and ccv$product_level, to the julian date of compilation of the product. To acquire the julian date, the product needs a module which stores the current julian date in their own variable. The value of this variable is then given to the ccm variables. The module can either reside on the product's source library or can be created and compiled within the build procedure. An example from a build procedure is:

```
COLLECT_TEXT install_date_compile until='**' substitution_mark='%'
MODULE pim$build_date;
  VAR
    piv$build_date: [XDCL] string (5) :=
          '%$sustr($date(ordinal), 3, 5)%';
MODEND pim$build_date;
**
```

where pi is the product identifier.

Products which do not call CCM, but produce a header line when executed, must include the build date in the header line. Other products should put the build date to the job log.

5.2 QUICKLOOK_TEST_TRANSMITTAL_GUIDELINES

If a product is going through the process of transmitting their code to integration for the first time and the evaluation group is not prepared to test the product, the project must supply a small test called a quicklook.

If the new product can run batch jobs, there is a file available which is a template for a quicklook test. The file name is .sviol.quicklook_template and this can be copied into your own catalog and editted accordingly for your product.

The file is listed below.

```
**************************************************************

JOB Job_name=xxx
"XXX is the job name.  This is usually the same as the test name."

  create_variable name=stats kind=status
```

5.0 NEW PRODUCT CONSIDERATIONS
5.2 QUICKLOOK TEST TRANSMITTAL GUIDELINES

```
  create_variable trouble_flag kind=boolean value=false
  IF $variable(ttv#action, declared) = 'UNKNOWN' THEN
    create_variable ttv#action kind=(string, $max_name) scope=job ..
          value='KILL'
  IFEND
  IF $variable(ttv#test_tool_library, declared) = 'UNKNOWN' THEN
    create_variable ttv#test_tool_library kind=string scope=job ..
          value=':$SYSTEM.$SYSTEM.VETOOL.TTF$TEST_TOOL_LIBRARY'
  IFEND
" ------------------------------------------------------------------
" acquire test tool library "
"   First, remove any prior usage.
  set_program_attribute delete_library=$local.ttf$test_tool_library..
        status=stats
  set_command_list delete=$local.ttf$test_tool_library status=stats
  detach_file file=$local.ttf$test_tool_library status=stats
  attach_file f=$fname(ttv#test_tool_library) ..
        ifn=ttf$test_tool_library am=(read execute) ..
        sm=(read execute) wait=true
  set_command_list add=$local.ttf$test_tool_library
  set_program_attribute add_library=$local.ttf$test_tool_library
" ------------------------------------------------------------------
  automated_checking_routine pi=zzz status=stats
"zzz is the three-letter product identifier, i.e. AA8 is the identifi
"for the AAM product."
  display_message message='CONTROL DATA COMPANY PRIVATE' ..
        status=stats
  automated_checking_routine tn=xxx status=stats
"xxx is the test name."

COLLECT_TEXT output=textinp until='END_COLLECT_TEXT' status=stats

     program   "This is the program you provide to test your product.

END_COLLECT_TEXT
  WHEN any_fault DO
    trouble_flag = true
    display_message message='AN ERROR HAS OCCURED' status=stats
    display_catalog catalog=$local
    display_value value=osv$status output=$output
    display_program_attributes output=$output
    automated_checking_routine state=fail status=stats
  WHENEND

"The next line would contain the compilation command for your program
" i.e. cobol,i=textinp,l=output,lo=s,audit=true,b=binary1

"The next line would contain the execute statement, i.e. an lgo or
  execute_task binary1 lm=$output lmo=(s b ep)
```

5.0 NEW PRODUCT CONSIDERATIONS
5.2 QUICKLOOK TEST TRANSMITTAL GUIDELINES

```
COLLECT_TEXT output=out until='DESCRIPTION_END' status=stats
TEST_NAME: xxx
PRODUCT_NAME: xyz
ABSTRACT:
DESCRIPTION:
FEATURE:
DESCRIPTION_END
  automated_checking_routine action=$name(ttv#action)
JOBEND
```

****

The program provided must be easy to analyze and test a few features of the product. The output from this program must contain a version number. See section 3.3 for more information on version numbers.

If the new product is normally only run interactively, then a simple interactive job must be supplied which will display the product's version number and will display a distinct passing or failing status.

Once you have prepared a quicklook job for your product, this should be given to the evaluation group to include the test in the quicklook library.

The purpose of the quicklook tests is to verify that the products have been installed correctly. The tests are run by integration following the completion of product set builds.

5.3 VERIFICATION_JOB_TRANSMITTAL

Every product must provide a job to quickly verify the product is installed. This job differs from the quicklook test in that it is an extremely short test to verify that the product is callable. It is not designed to test any of the product's features.

The following is a list of guidelines for a verification job and an example of an already existing job for the cobol product.

Guidelines :

- Put the job into procedure form which will be called interactively.
- Do not use any other products.

5.0 NEW PRODUCT CONSIDERATIONS
5.3 VERIFICATION JOB TRANSMITTAL

- Do not use any local tools.
- Keep it "short and sweet". Just the bare minimum to verify the existence of the product.
- Make the test results obvious. Put comments in the job log to make tha test self-explanatory.
- Do not include error processing (WHEN loops). Let the job abort at error. (This is the current direction; it may change later.)
- Name as follows: INSTALLATION_VERIFICATION_xx8 where xx=product identifier.

Example :

```
**************************************************************************

Job Job_name=installation_verification_cb8
" This is the installation verification procedure for COBOL
" Method ==>
"   1. Create a tiny COBOL program using collect_text command
"   2. Call COBOL compile statement to compile this program
"   3. Execute the binary of this program
"   4. 'COBOL is Installed !' should be displayed to $OUTPUT.

  create_variable (file_connection_status, verify_status) kind=status

COLLECT_TEXT output=cobol_program until='COBOL_PROGRAM_END'
       identification division.
       program-id. call-cobol.
       environment division.
       data division.
       procedure division.
       the-only-paragraph.
           display "COBOL is Installed !"
           stop run.
COBOL_PROGRAM_END

  delete_file_connection standard_file=$errors file=output ..
    status=file_connection_status
  cobol, i=cobol_program, b=cobol_igo, l=:$local.listing.$eol

  IF file_connection_status.normal = true THEN
    create_file_connection standard_file=$errors file=output
  IFEND

  execute_task cobol_igo termination_error_level=f status=verify_stat
  detach_file file=(cobol_igo, cobol_program)

  EXIT_PROC WITH verify_status
Jobend
```

5.0 NEW PRODUCT CONSIDERATIONS
5.3 VERIFICATION JOB TRANSMITTAL

****

The verification job should be given to integration when the product is transmitted for the first time.

6.0 A GUIDE TO USING THE DEVELOPMENT ENVIRONMENT

6.0 A GUIDE TO USING THE DEVELOPMENT ENVIRONMENT

The following is a guide to using the Development Environment procedures. It describes the most commonly used features, which should be sufficient for most situations.

6.1 WHY USE THESE PROCEDURES?

The purpose in using these procedures is to provide a consistent mechanism for Development and Integration to create and control modification of code. Conventions are enforced by the design of the procedures. The manual overhead of using NOS/VE and SCU is reduced by automatic creation of new cycles and the use of SCL variables to define user controllable defaults. You are required to use the EXTRACT_SOURCE and TRANSMIT_TO_INTEGRATION procedures, since these enforce deck interlocking. It is your option to use the other procedures.

6.2 INITIAL PREPARATION

In your prolog, you should have two commands which set up the development environment. The first makes the procedures available. This command is:

```
set_command_library add=.INTVE.COMMAND_LIBRARY
```

The second is the SET_WORKING_ENVIRONMENT command. It creates the SCL variables which define your environment. The following command is an example.

```
set_working_environment author='V. E. User'..
   product_name=PI_05 ..
   build_level=NONE working_catalog=$USER
```

6.0 A GUIDE TO USING THE DEVELOPMENT ENVIRONMENT
6.2 INITIAL PREPARATION

In the example, the master library for the product will be acquired from .INTVE.PI_05.SOURCE_LIBRARY. Your working library will be acquired from $USER.SOURCE_LIBRARY. If it does not exist, it will be created the first time you execute a procedure that needs to write to it. If you create it yourself, make sure that you leave cycle one unused. This cycle is used for temporary holding of the result library. You must give yourself additional permission to your own library. This can be done by the following command:

```
create_file_permit $USER.SOURCE_LIBRARY am=(all cycle ..
   control)  sm=none application_information='I4'
```

This additional file permission should also be given to your feature catalog source library if you have one. Be aware that sometimes the master catalog permissions override the file permissions and this will cause problems. For example, if you have a master catalog permit for a particular user with restricted permissions, this will override a file permission for the group public.

The APPLICATION_INFORMATION parameter is the additional permission. It tells SCU that you can set interlocks on this library and manipulate mods at state 4. The permit on the master library in INTVE should be 'I1'. Integration will insure the correct permits in the INTVE catalog.

6.3 CHECKING OUT A DECK TO MAKE CHANGES

You use the EXTRACT_SOURCE command to get copies of decks from the master library to your feature library. If another user has a deck checked out, you will not be able to acquire it. If you are using a feature catalog, a copy of the deck will be put there as well.

The procedure looks first on the feature library, if any, then on the LATEST_CHANGES file, and then finally on the master library, for the deck. (The LATEST_CHANGES file is regularly combined with the master library. For EXTRACT_SOURCE you can think of the two files as one.) The deck is then extracted from the library in which it was found, and combined with the feature library, if any. It is then in turn extracted from the feature library and combined with the working library. A new cycle is created of the working and feature libraries.

6.0 A GUIDE TO USING THE DEVELOPMENT ENVIRONMENT
6.3 CHECKING OUT A DECK TO MAKE CHANGES

Before concluding, the proc logs the names of the decks and the files on which they were found and put to a file called LOGGING in the INTVE product subcatalog.

6.4 ENTERING THE WORKING ENVIRONMENT

One easy way to use SCU to make changes is through the working environment. Executing the ENTER_WORKING_ENVIRONMENT command puts you in SCU command mode, with the high cycle of your working library as the BASE and cycle one as the RESULT. You can now enter any SCU command and most Development Environment commands.

When you are finished, and you want to save your changes, enter the EXIT_WORKING_ENVIRONMENT command. This will write the library to cycle one, end SCU, and make cycle one the new highest cycle. The retention period of the old high cycle will be set to 1, to keep file space from overflowing.

If you do not want to save the changes, enter END NO. This will end SCU and cycle one will be deleted. Be careful to enter EXIWE if you want to save your changes. To "checkpoint" your library, use the WRITE_LIBRARY command. ENTER_WORKING_ENVIRONMENT will output a message if cycle one already exists.

6.5 CREATING A MODIFICATION

You should use the MAKE_MODIFICATION procedure to create a new modification. This procedure will use the Author value specified by SETWE, and requires a feature name and description. If you are in the working environment, the modification will be added to the working library. If you are not in the working environment, a new cycle of your SOURCE_LIBRARY will be created.

6.6 RETURNING A MODIFIED DECK

When you want to return the decks you checked out with EXTRACT_SOURCE, you use the TRANSMIT_TO_INTEGRATION procedure. If

you are using a feature catalog, you first have to use the TRANSMIT_TO_FEATURE_CATALOG procedure. This procedure combines the decks specified with the existing feature library, and removes them from your working library.

TRANSMIT_TO_INTEGRATION removes the decks from your feature library if you have one, or your working library if not. It combines them with a file called LATEST_CHANGES, which is in the same catalog as your master SOURCE_LIBRARY. LATEST_CHANGES is combined with SOURCE_LIBRARY automatically after deadstart.

You can specify decks, modifications, or features to be transmitted, but the effect is always that affected decks are transmitted. If you specify a modification, all decks changed by that mod are transmitted. In addition, new mods not specified will be included. All mods affecting the transmitted decks which are at state 0 will be changed to state 1 by TRATI (but not by TRATFC).

A list of the decks transmitted, with the mods changed in state, is written to the LOGGING file.

6.7 EXAMPLE

The following example shows a simple use of the development environment procedures. It is assumed that the .INTVE.COMMAND_LIBRARY file is in the command list, and that a SET_WORKING_ENVIRONMENT command has been executed.

```
extract_source pim$a_deck_of_cybil_source

entwe  " From here on, you will get a 'we/' prompt. "
makm m=PI8A011 feature=build_04C0 md='PSR fix for abort in ..
listing generation when cross reference selected.'

edil m=pi8a011 d=pim$a_deck_of_cybil_source
                  .
                  .
                  .
      Editor commands to modify the deck.
                  .
                  .
                  .
      END " or function key in screen mode "
dism pi8a011
       "SCU commands can be executed inside the environment"
```

6.0 A GUIDE TO USING THE DEVELOPMENT ENVIRONMENT
6.7 EXAMPLE

```
exiwe  " creates a new high cycle with the modification

format_cybil_source m=pl8a011 d=plm$a_deck_of_cybil_source
       " creates another cycle, since outside the environment
       " This command is optional.

tratl d=plm$a_deck_of_cybil_source
```

Of course, you would compile the deck you modified and test it before transmitting it.

6.8 MAKING CHANGES WITHOUT USING THE SCU EDITOR

You can make changes to a deck in other ways than through the SCU editor by using the GET_SOURCE and REPLACE_SOURCE procedures. GET_SOURCE extracts a deck and writes it out in legible format. REPLACE_SOURCE compares a legible file with a deck on the working library and generates a correction set.

GET_SOURCE can be used for many other purposes. For this purpose, you must specify FEATURES=ALL and BUILD_LEVEL=NONE to get all active lines. Otherwise, GET_SOURCE may exclude some modifications.

This method of making changes is most useful when using tools like GENERATE_PDT and GENERATE_COMMAND_TABLE_MODULE. GENPDT, for example, accepts as input a PDT definition, which is very similar to an SCL procedure header. Its output is a file containing CYBIL declarations for the parameter descriptor table. The original input is listed as commentary.

The following example shows the use of GET_SOURCE and REPLACE_SOURCE to modify a deck which contains GENPDT output. As above, it assumes that a SET_WORKING_ENVIRONMENT command has been executed.

```
extract_source pid$parameter_descriptor_table

make_modification S5999_pdt f=new_parameters ..
md='Add new parameters for DAP S5999.'

get_source pid$parameter_descriptor_table ..
  source=original_pdt feature=all bl=none
```

6.0 A GUIDE TO USING THE DEVELOPMENT ENVIRONMENT
6.8 MAKING CHANGES WITHOUT USING THE SCU EDITOR

```
edit_file original_pdt
  " extract the PDT declaration from the commentary

generate_pdt i=original_pdt o=new_pdt

replace_source source=new_pdt ..
  deck=pid$parameter_descriptor_table modification=S5999_pdt

transmit_to_integration pid$parameter_descriptor_table
```

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT

7.0 COMMANDS_TO_USE_THE_NOS/VE_DEVELOPMENT_ENVIRONMENT

A library of commands (mostly SCL procedures) provides access to the SCU source libraries. These commands use standard facilities of NOS/VE and its product set.

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

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

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

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

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

The deleting of old cycles of source libraries from feature and working catalogs is the responsibility of the associated project/user. It is strongly recommended that cycles be monitored closely and purged regularly, as cycles can accumulate rapidly.

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.1 ENVIRONMENT COMMANDS

7.1 ENVIRONMENT_COMMANDS

The commands described in this section are used to specify the "environment" in which work will be done. NOTE : All keyword specifications and default parameter values are defined in the Command Summary (Appendix A).

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

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

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

> Although eventually all of the commands described in this document will be sensitive to whether the working environment has been "entered", initially the only commands that will operate both "inside" and "outside" the working environment are COMPILE_SOURCE, GET_SOURCE, REPLACE_SOURCE, ASSIGN_MODIFICATION, MAKE_MODIFICATION, EDIT_SOURCE, and FORMAT_CYBIL_SOURCE.

This is analogous to the current system of using profile variables as defaults in the 170 SES environment.

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.1.1 SET_WORKING_ENVIRONMENT (SETWE)

7.1.1 SET_WORKING_ENVIRONMENT (SETWE)

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

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

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

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

```
set_working_environment [author=<string>]
     [product_name=<name>]
     [build_level=<name>]
     [tools_library_build_level=<name>]
     [feature_catalog=<catalog>]
     [feature_build_level=<name>]
     [working_catalog=<catalog>]
     [working_build_level=<name>]
     [status=<status variable>]
```

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

The corresponding variable name is WEV$AUTHOR.

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

product_name ! pn : specifies the name of the product to be worked on. This is the name of the operating system or product

subcatalog in the main integration catalog.

The corresponding variable name is WEV$PRODUCT_NAME.

The default is OS.

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

The corresponding variable name is WEV$BUILD_LEVEL.

The default is the latest system build level.

tools_library_build_level ! tlbl : used for OS development only. Sunnyvale uses should set this parameter to NONE.

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

The corresponding variable name is WEV$FEATURE_CATALOG.

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

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

The corresponding variable name is WEV$FEATURE_BUILD_LEVEL.

The default is OBJECT.

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

The corresponding variable name is WEV$WORKING_CATALOG.

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

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

The corresponding variable name is WEV$WORKING_BUILD_LEVEL.

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.1.1 SET_WORKING_ENVIRONMENT (SETWE)

The default is OBJECT.

status : see NOS/VE error handling.

7.1.2 DISPLAY_WORKING_ENVIRONMENT (DISWE)

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

```
display_working_environment [display_options=list of <name>]
     [output=<file>]
     [status=<status variable>]
```

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

Omission causes ALL to be used.

output ! o : specifies the file to receive the display.

Omission causes $OUTPUT to be used.

status : see NOS/VE error handling.

7.1.3 ENTER_WORKING_ENVIRONMENT (ENTWE)

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

This command provides a more efficient way of doing work than "popping in and out" of SCU to do editing, deck expansion, etc. For example, a user may enter the SCU utility, make some changes to a

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.1.3 ENTER_WORKING_ENVIRONMENT (ENTWE)

deck, compile it (without having to exit SCU and thus write a new cycle of the working library), fix whatever problems may show up in the compile, compile again, etc. Only when finished with that particular work session does the user need to leave SCU.

Checkpoints of updates to the source library can be made via SCU's WRITE_LIBRARY subcommand. Be sure to include the file name of the desired result_library on the Write_Library command, or the result will be a temporary scratch file.

The session of work can be terminated via a corresponding call to EXIT_WORKING_ENVIRONMENT which will write the new cycle of the working source library.

```
enter_working_environment [author=<string>]
     [product_name=<name>]
     [build_level=<name>]
     [feature_catalog=<catalog>]
     [feature_build_level=<name>]
     [working_catalog=<catalog>]
     [working_build_level=<name>]
     [status=<status variable>]
```

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

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.

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.1.4 EXIT_WORKING_ENVIRONMENT (EXIWE)

7.1.4 EXIT_WORKING_ENVIRONMENT (EXIWE)

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

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

exit_working_environment [status=<status variable>]

status : see NOS/VE error handling.

7.2 COMMANDS FOR CODE DEVELOPMENT AND INTEGRATION

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

NOTE : All keyword specifications and default parameter values are defined in the Command Summary (Appendix A).

7.2.1 ASSIGN_MODIFICATION(S) (ASSM)

This command can be used inside the Working_Environment. Remember that in SVL the MAKE_MODIFICATION command is recommended and the documentation follows this.

This command creates one or more unique modifications in the

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.2.1 ASSIGN_MODIFICATION(S) (ASSM)

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

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

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

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

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

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

```
assign_modification feature=<name>
     [count=<integer>]
     [modification_description=<string>]
     [output=<file>]
     [author=<string>]
     [working_catalog=<catalog>]
     [status=<status variable>]
```

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

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

Omission causes 1 to be used.

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

Omission causes no description to be used.

output ! o : specifies the file to which are written the names of

the modifications that have been created.

Omission causes $OUTPUT to be used.

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

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

status : see NOS/VE error handling.

7.2.2 MAKE_MODIFICATION (MAKM)

This command can be used inside the Working_Environment.

This command is similar to the ASSIGN_MODIFICATION command previously described EXCEPT that there is no "modification name seed". The modification name desired must be specified because it is not automatically created.

The command creates a unique modification in the working source library. As with assign_modificaton, do not use this to create a modification on the working library if the modification exists already on the base (.intve.) source_library from which the working library was extracted. An error will not be generated until "transmit-to-integration time" if this is the case.

Consistent with the standards for modifications, the description and feature fields are required.

```
make_modification modification=<name>
     feature=<name>
     modification_description= list of <string>
     [author=<string>]
     [working_catalog=<catalog>]
     [status=<status variable>]
```

modification ! m : modification name desired; should generally be the PSR number associated with the modificaton.

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

modification_description ! md : specifies a brief description of the

modification.

author ! a ! specifies the creator of the contents of the modifications; should contain the creator's first two initials and full last name. If this field is not supplied, the working environment specification is substituted.

working_catalog ! wc ! specifies the working catalog to be used.

status ! see NOS/VE error handling.

7.2.3 CREATE_SOURCE (CRES)

This command can NOT be used inside the Working_Environment.

This command creates one or more decks in all levels of the data base. Checks are made to insure that the decks don't already exist.

The effect of the command is to create the new decks in the system or product source library, then to extract the decks to the feature level and from there to the working level with appropriate interlocks set along the way. Once this is accomplished the text of the deck may be added via EDIT_LIBRARY or EDIT_SOURCE, or (indirectly) any other editor.

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

```
create_source deck=list of <name>
     modification=<name>
     source_group=<name>
     destination_group=list of <name>
     [group=list of <name>]
     [processor=<string>]
     [author=<string>]
     [deck_description=<string>]
     [expand=<boolean>]
     [same_as=<name>]
     [product_name=<name>]
     [feature_catalog=<catalog>]
     [working_catalog=<catalog>]
     [status=<status variable>]
```

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.2.3 CREATE_SOURCE (CRES)

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

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

source_group ! sg : specifies the group of decks to which the new decks are to be assigned, such as OSS$SOURCE. The name must follow the convention for source groups, i.e. it must have "S$" in the third and fourth positions. (See the section on "use of groups" above.)
This parameter is required

destination_group ! destination_groups ! dg : specifies the file(s) in which the "processed" (compiled, assembled, etc.) form of the modules is to be placed. (See Appendix B) If this is not applicable (in the case of a common deck) NONE should be specified. If specified, the name must have "F$" in the third and fourth positions.

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

Omission causes no "arbitrary" groups to be assigned.

processor ! p : specifies the processor for the decks.

Omission causes CYBIL to be used.

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

deck_description ! dd : specifies a brief description of the decks.

Omission causes no description to be used.

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

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

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

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.2.3 CREATE_SOURCE (CRES)

Omission causes the normal SCU defaults to be used.

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

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

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

status : see NOS/VE error handling.

7.2.4 EXTRACT_SOURCE (EXTS)

This command can NOT be used inside the Working_Environment.

This command is used to extract decks from the feature or system/product level to the working level in preparation for making changes. In addition to the decks themselves, the relevant "parts" of all associated modifications, features and groups are extracted (i.e. SCU's EXTRACT_SOURCE_LIBRARY facility is used).

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

This extraction is performed using deck interlocks, unless interlock=false. The interlock name used is always the user name. If a deck is extracted without interlocks it cannot be transmitted back to integration or a feature catalog.

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

At least one of the deck and selection_criteria parameters must be provided. If both are given the deck parameter is processed first. Currently the selection_criteria parameter is not implemented.

```
extract_source [deck=list of <name>]
     [interlock=<boolean>]
     [selection_criteria=<file>]
     [product_name=<name>]
     [build_level=<name>]
     [feature_catalog=<catalog>]
     [working_catalog=<catalog>]
     [status=<status variable>]
```

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.2.4 EXTRACT_SOURCE (EXTS)

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

interlock ¦ i : specifies whether decks are to be extracted with or without interlocks.

Omission causes TRUE to be used.

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

Omission causes $NULL to be used.

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

build_level ¦ bl : specifies the system or product build_level to be used when extracting decks without interlock. When decks are extracted with interlock, all active lines are included.

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

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

status : see NOS/VE error handling.

7.2.5 EDIT_SOURCE (EDIS)

This command can be used inside the Working_Environment.

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

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

```
edit_source modification=<name>
     [deck=<name>]
     [output=<file>]
     [working_catalog=<catalog>]
     [status=<status variable>]
```

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

working library.

deck ; d : specifies the deck to be edited.

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

output ; o : specifies the file to receive displays from the editor.

Omission causes $OUTPUT to be used.

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

status : see NOS/VE error handling.

7.2.6 GET_SOURCE (GETS)

This command can be used inside the Working_Environment.

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

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

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

NOTE: Refer to the description of compile_source for the deck selection and criteria file processing.

CAUTION: If you are using GET_SOURCE to extract a deck which will be modified and replaced with REPLACE_SOURCE, be sure to specify FEATURE=ALL and BUILD_LEVEL=NONE. Otherwise, some modifications may be omitted and not all active lines written to the source file.

```
get_source [deck=list of <name>]
     [source=<file>]
     [selection_criteria=<file>]
```

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.2.6 GET_SOURCE (GETS)

```
[include_latest_changes=<boolean>]
[expand=<boolean>]
[expansion_depth=<integer>]
[alternate_product=list of list of <name>]
[alternate_base=list of <file>]
[product_name=<name>]
[build_level=<name>]
[feature_catalog=<catalog>]
[feature_build_level=<name>]
[working_catalog=<catalog>]
[working_build_level=<name>]
[status=<status variable>]
```

deck ! decks ! d ! specifies the decks to be written.

source ! s ! specifies the file to receive the decks.

Omission causes a file named for the first specified deck to be written in the working catalog.

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

Omission causes $NULL to be used.

include_latest_changes ! ilc ! specifies whether code recently transmitted to integration by other developers should be included in the decks to be written.

Omission causes FALSE to be used.

expand ! e ! specifies whether the decks are to be expanded.

Omission causes FALSE to be used.

expansion_depth ! ed ! specifies the number of levels of COPY and COPYC directives to process.

Omission causes $max_integer to be used.

alternate_product ! 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 of 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 is specified, a build level of NONE is assumed.

Omission causes no alternate_products to be accessed.

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

Omission causes NONE to be used.

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.

7.2.7 REPLACE_SOURCE (REPS)

This command can be used inside the Working_Environment.

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

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

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

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.2.7 REPLACE_SOURCE (REPS)

CAUTION: If you used GET_SOURCE to obtain the source file, you should have specified FEATURE=ALL and BUILD_LEVEL=NONE. Otherwise, some modifications may be missing.

```
replace_source source=<file>
     deck=<name>
     modification=<name>
     [working_catalog=<catalog>]
     [status=<status variable>]
```

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

deck ; d : specifies the deck to be updated.

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

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

status : see NOS/VE error handling.

7.2.8 FORMAT_CYBIL_SOURCE (FORCS)

This command can be used inside the Working_Environment.

This command formats a CYBIL source deck.

It calls GET_SOURCE to retrieve the deck, CYBFORM to do the formatting, and REPLACE_SOURCE to incorporate the changes made by CYBFORM.

If CYBFORM detects an error the replacement is suppressed.

```
format_cybil_source deck=<name>
     modification=<name>
     [working_catalog=<catalog>]
     [status=<status variable>]
```

deck ; d : specifies the deck to be formatted.

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

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

status : see NOS/VE error handling.

7.2.9 COMPILE_SOURCE (COMS)

This command can be used inside the Working_Environment.

This command compiles/assembles one or more decks.

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

The SCU libraries included in the list to be searched will be done so in the following order:

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

At least one of the deck and selection_criteria parameters must be provided. If both are given the deck parameter is processed first. Currently, if just the selection_criteria parameter is given, the processor and object_library must also be supplied.

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

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

PROC, the following operations are performed:

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

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

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

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

5. Modifications associated with the features specified by the FEATURE parameter are now "included". If the FEATURE parameter is not specified, the features selected will be those in the WEF$FEATURE_LIST file of the working catalog. If this file does not exist or is empty and no value is specified for the FEATURE parameter, then no additional modifications will be included by this step. If the FEATURE parameter is given the keyword value of ALL, then step 3 is essentially undone - that is, all modifications with state 0 or 1 are included. If the FEATURE parameter is given the keyword value of NONE this step will result in no changes - that is, no modification introduced after the specified build level will be included.

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

7. The SCU selection criteria file specified by the SELECTION_CRITERIA parameter is now processed. The directives

in this file may - intentially or inadvertantly - undo some of what has been done before.

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

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

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

```
compile_source [deck=list of <name>]
     [selection_criteria=<file>]
     [include_latest_changes=<boolean>]
     [feature=list of <name>]
     [compile=<file>]
     [processor= <string>]
     [object_library=list of <name>]
     [list=<file>]
     [list_options=list of <list option>]
     [save_listings=list of <save option>]
     [alternate_product=list of list of<name>]
     [alternate_base=list of <file>]
     [product_name=<name>]
     feature_catalog=<catalog>]
     [feature_build_level=<name>]
     [working_catalog=<catalog>]
     [working_build_level=<name>]
     [status=<status variable>]
```

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

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

Omission causes $NULL to be used.

NOTE: The selection criteria file cannot be an interactive file. It must not contain an "END" directive.

include_latest_changes ! ilc : specifies whether code recently transmitted to integration by other developers should be included in the decks to be compiled.

Omission causes FALSE to be used.

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

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

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

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

processor ! p : specifies the compiler or assembler to be used to process each deck. Processor options may be selected by giving this parameter as a string (e.g. 'CYBIL OP=HIGH').

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

object_library ! object_libraries ! ol : specifies the object libraries in the working catalog into which the compiled/assembled modules will be placed. If an object library already exists the newly compiled modules are combined with it to form a new version of the library which effectively overwrites the original. If NONE is specified, no object_library or object_file will be written.

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

list ! l : specifies the file to which the compilation or assembly listings are to be written.

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

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

Omission causes (S,R) to be used, i.e. a listing of the source and a cross reference (excluding unreferenced identifiers).

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

Omission causes NONE to be used.

alternate_product ! alternate_products ! ap : specifies other product catalogs in which to find source_libraries to be used as alternate_bases in the expansion of the code. This parameter accepts either single names or pairs of 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 means all active lines at state 2 or higher, is assumed.

Omission causes no alternate_products to be accessed.

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

Omission causes NONE to be used.

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.

7.2.10 TRANSMIT_TO_FEATURE_CATALOG (TRATFC)

This command can NOT be used inside the Working_Environment.

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

Once the transmitted code has been written to its destination, it is removed from its source to free the interlock on the deck. The new version of an affected source library is written as the next cycle of the corresponding file. NOTE : The transmitted source is still available, however interlocked, in the previously-highest cycle of the working catalog.

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

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

```
transmit_to_feature_catalog [decks=list of <name>]
     [modification=list of <name>]
     [feature=list of <name>]
     [feature_catalog=<catalog>]
     [working_catalog=<catalog>]
     [status=<status variable>]
```

deck ! decks ! d ! specifies the decks to be transmitted.

modification ! modifications ! m ! specifies the modifications to be transmitted.

feature ! features ! f ! specifies the features to be transmitted.

feature_catalog ! fc ! specifies the feature catalog to be used.

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

status : see NOS/VE error handling.

7.2.11 TRANSMIT_TO_INTEGRATION (TRATI)

This command can NOT be used inside the Working_Environment.

This command transmits code (decks, modifications, features) from a feature or (if no feature catalog is being used) working catalog.

Once the transmitted code has been written to its destination, it is removed from its source to free the interlock on the deck. The new version of an affected source library is written as the next cycle of the corresponding file. NOTE : The transmitted source is still available, however interlocked, in the previously-highest cycle of the feature (or working) catalog.

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

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

```
transmit_to_integration [decks=list of <name>]
     [modification=list of <name>]
     [feature=list of <name>]
     [product_name=<name>]
     [feature_catalog=<catalog>]
     [working_catalog=<catalog>]
     [status=<status variable>]
```

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

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

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

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

7.0 COMMANDS TO USE THE NOS/VE DEVELOPMENT ENVIRONMENT
7.2.11 TRANSMIT_TO_INTEGRATION (TRATI)

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

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

status : see NOS/VE error handling.

A1.0 COMMAND SUMMARY

A1.0 COMMAND_SUMMARY

```
PROC assign_modifications, assign_modification, assm (
    feature, f : NAME
    count, c : INTEGER = 1
    modification_description, md : string = 'NO DESCRIPTION'
    output, o : file = $LOCAL.$OUTPUT
    author, a : string = $job(user)
    working_catalog, wc : file
    status)

PROC compile_source, coms (
    decks, deck, d : list of name or KEY ALL
    selection_criteria, sc : file=$null
    include_latest_changes, ilc : boolean = false
    feature, f : list of name or key ALL, NONE
    compile, c : file
    processor, p : string
    object_library, ol : list of name
    list, l : file = $list
    list_options, list_option, lo : list of key A F O R RA ..
            S X NONE DEFAULT = (S R)
    save_listings, sl : list of key ALL, NONE, GOOD, BAD = NONE
    alternate_products, alternate_product, ap : ..
            list of list of name
    alternate_bases, alternate_base, ab : list of file or ..
      KEY NONE = NONE
    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)

PROC create_source, cres (
     decks, deck, d : list of name
     modification, m : name = $required
     source_group, sg : name = $required
     destination_group, dg : list of name or KEY NONE = $required
     group, g : list of name
     processor, p : string = 'cybil'
     author, a : string = $job(user)
     deck_description, dd : string = $REQUIRED
```

A1.0 COMMAND SUMMARY

```
     expand, e : boolean
     same_as, sa : name
     output, o : file = $list
     product_name, pn : name = OS
     working_catalog, wc : file or KEY NONE = NONE
     feature_catalog, fc : file or KEY NONE = NONE
     status)

PROC display_working_environment, diswe (
    display_options, display_option, do : list of name or ..
      key all = all
    output, o : file = $OUTPUT
    status)

PROC edit_source, edis (
    modification, mod, m : NAME 1..9
    deck, d : NAME
    output, o : FILE = $OUTPUT
    working_catalog, wc : FILE or KEY NONE = NONE
    status)

PROC enter_working_environment, entwe (
    author, a : string = $job(user)
    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)

PROC exit_working_environment, exiwe(
    status)

PROC extract_source, exts (
    decks, deck, d : list of name or KEY ALL
    selection_criteria, sc : file = $NULL
    list, l : file = $LIST
    product_name, pn : name = OS
    feature_catalog, fc : file or KEY NONE = NONE
    working_catalog, wc : file or KEY NONE = NONE
    status)

PROC format_cybil_source, forcs (
     deck, d : name = $required
     modification, m : name = $required
     working_catalog, wc : file or KEY NONE = NONE
     status)
```

A1.0 COMMAND SUMMARY

```
PROC get_source, gets (
     decks, deck, d : list of name or KEY ALL
     source, s : file
     selection_criteria, sc : file=$null
     include_latest_code, ilc : boolean = false
     expand, e : boolean = false
     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)

PROC list_interlocks, lisi (
     list_file : file = $OUTPUT
     status)

PROC make_modification, makm (
     modification, m : NAME 1..9 = $REQUIRED
     feature, f : name
     modification_description, md : list of string = $REQUIRED
     author, a : string = $job(user)
     working_catalog, wc : file = $user
     status)

PROC replace_source, reps (
    source, s : file = $REQUIRED
    deck, d : NAME = $REQUIRED
    modification, mod, m : NAME 1..9 = $REQUIRED
    working_catalog, wc : FILE or KEY NONE = NONE
    status)

PROC set_working_environment, setwe (
    author, a : string = $job(user)
    product_name, pn : name = OS
    build_level, bl : name
    tool_library_build_level, tlbl : 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)

PROC transmit_to_feature_catalog, tratfc (
    decks, deck, d : list of name or key ALL
    modifications, modification, m : list of name or key ALL
    features, feature, f : list of name or key ALL
    selection_criteria, sc : file = $NULL
```

A1.0 COMMAND SUMMARY

```
    feature_catalog, fc : file or KEY NONE = NONE
    working_catalog, wc : file or KEY NONE = NONE
    status)

PROC transmit_to_integration, trati (
    decks, deck, d : list of name or key ALL
    modifications, modification, m : list of name or key ALL
    features, feature, f : list of name or key ALL
    product_name, pn : name = OS
    feature_catalog, fc : file or KEY NONE = NONE
    working_catalog, wc : file or KEY NONE = NONE
    status)
```

Table of Contents