SOFTWARE ENGINEERING SERVICES

Version 1.0

User's Handbook

REVISION DEFINITION SHEET

REV	DATE	DESCRIPTION
1	07/11/83	Update for SES Version 1.0 Level 23. Includes descriptions of new procedures, updates and corrections to old procedures.
2	12/13/83	Update for SES Version 1.0 Level 24. Includes descriptions of new procedures, updates and corrections to old procedures.
3	05/22/84	Update for SES Version 1.0 Level 25. Includes descriptions of new procedures, updates and corrections to old procedures.
4	08/14/84	Update for SES Version 1.0 Level 26. Includes descriptions of new procedures, updates and corrections to old procedures.
4A	12/18/84	This revision reflects SES Usability Tools at Level 617B incorporating the following features and changes: FTNTODD and FTNTOSCT tools added for generating structure charts from FORTRAN source code; newly named SCADG and SCADGEDIT replacing CADSG and CADSGEDIT; XREFMAP cross reference tool added; SCOOP file comparison tool added; SORT5 replacing all internal uses of SORT/MERGE4; EXPLAIN interface to new online Miscellaneous Routines Interface Reference Manual; minor enhancements to GOF, GOL, COM, DEOM, DIOM, BIND68K and TRANDILIB.

NOTICE:

The content of this manual obtained from DCS may differ from the on-line version. The on-line version reflects exactly the SES tool set installed on your development machine at your site, whereas the DCS version describes the full SES tool set and all available options.

1.0 PREFACE

1.0 PREFACE

This manual describes the procedural user interface to all of the Software Engineering System (SES) Usability Tools under the five product packages listed below:

1. NOS_Tools. This is the core set of SES tools and is required by the other four packages. This set of tools is sometimes referred to in this manual (and others) as the 170 SES tools, SES tools or simply SES. All SES procedures that are not included in the other four packages are included in this one.

2. CYBIL_170_Code_Generator. This product is also referred to as CYBIL-CC. The SES procedures in this package are CYBIL, CYBFORM, GETCOMN, GETCCDB and GETLIB.

3. CYBIL_P-Code_Generator. This product is also referred to as CYBIL-CP. The SES procedures listed for CYBIL-CC plus all of the procedures described in the SES Object Code Utilities chapter and the UCSD P-Code Environment chapter in this manual.

4. NOS_DEFINE. This product package consists of the data dictionary utilities described in the Structured Process Tools chapter. It is sometimes referred to as CADS DD.

5. NOS_GRAPH. This product package is also called CADSG or CADS Graphics and consists of the SES interfaces CADSG and CADSGEDIT described in the Structured Process Tools chapter.

6. NOS_CHART. This product package consists of the CYBIL to structure chart tools described in the Structured Process Tools chapter.

7. CDCNET_Support_Tools. This product is also referred to as Network Tools. These tools provide a Motorola MC68000 development environment on a CYBER 170.

Each of these product packages can be ordered separately from Control Data and can be installed in any combination which includes the NOS Tools package.

1.0 PREFACE
1.1 INTRODUCTION

1.1 INTRODUCTION

"... Software stands between the user and the machine ..."
Harlan D. Mills

The Software Engineering System (SES) provides task oriented or procedural access to a collection of Software Tools, running on the Network Operating System (NOS). An underlying philosophy of SES is : where there's a function or task to be done, provide a PROCEDURE to do it. Programs are generally not used directly, rather they are accessed via SES procedures that provide a consistent access method.

This SES User's Handbook is intended both as a tutorial introduction for new SES users (who should definitely read this chapter), and also as a reference manual for experienced SES users. Although this is not the only existing relevant SES document, it's the main trunk of the tree of knowledge of what SES PROCEDURES are available. This SES User's Handbook is usually updated only at SES Releases.

Software Production covers many areas other than the narrow fields of "assemblers, compilers and loaders". To cover the wider areas, we need good tools. SES provides the tools. This SES User's Handbook is arranged in chapters that "cluster" around related functional areas (such as document formatting, file management, text processing and so on). This chapter is a general introduction to the concepts of SES and its use. The remaining chapters of this handbook cover the functional areas. Below is a brief summary of the contents of the remaining chapters.

A major proportion of SES is devoted to the CYBER 180 SIMULATED ENVIRONMENT, running on the CYBER 170. The chapter called 'CYBER 180 Virtual Environment Creation and Simulation' contains information on building the virtual environment, simulating, and transferring text files back and forth between the CYBER 180 and CYBER 170. The chapter on 'CYBER 180 Object Code Utilities' describes the CYBER 180 OBJECT CODE UTILITY, used to manage CYBER 180 Object Text. One of CYBER 180's significant differences from CYBER 170 is the use of real eight-bit ASCII. NOS runs on a six-bit oriented system, and provides only partial support of ASCII. SES provides extensive support for ASCII. The CYBIL compiler is ASCII oriented, as is the CYBIL formatter, the documentation facilities, many of the text manipulation tools, and the source text maintenance utilities.

Of course, Compilers, Assemblers and Loaders do play a part in software production - the chapter called 'Compiling, Linking and

Debugging' covers a lot of useful tools in this area. Included in this chapter is the CYBER IMPLEMENTATION LANGUAGE - CYBIL. As well as a CYBIL compiler, this chapter also describes CYBIL-CC INTERACTIVE DEBUG, a symbolic debug facility, based on CYBER Interactive Debug but oriented towards CYBIL code.

Production of large (or small) software systems requires some means of MANAGING SOURCE TEXT to avoid a lot of detail work. SES supplies a comprehensive collection of source text maintenance procedures geared up for the ASCII version of MODIFY (known as MADIFY) to manage ASCII source text for you, or, in fact, any other ASCII text, such as documents, data dictionaries, process descriptions and so on. These facilities are described in the chapter called 'Source Text Maintenance. The chapter entitled 'Source Code Utility on NOS 170' describes the NOS-170 version of the CYBER 180 Source Code Utility (SCU). There's also a set of procedures to make UPDATE easier to live with, described in 'Handling Update Program Libraries'.

When you've compiled or assembled your source text of course there's the question of OBJECT CODE MANAGEMENT. The chapter on 'Library Or Multi Record File Management Using LIBEDIT' covers LIBRARY MANAGEMENT - a collection of procedures that interface to the NOS LIBEDIT utility. These library management procedures can in fact handle things other than object code.

One of the first hurdles to clear on most computer systems seems to involve the question "how do I print a file"? The chapter on 'Printing Files' shows that printing files really can be easy.

An important part of software production involves DOCUMENTATION. SES has a collection of tools that make a fun job of generating beautiful documents. Documentation aids revolve around the TXTCODE document formatter, used to produce this Handbook. The 'Document Formatting System' chapter covers document formatting.

Preparing beautifully formatted source text for high level language processors is something you like to have, but you'd rather not do it yourself. Various types of preprocessors are covered in the chapter on 'Source Text Preprocessors'.

In any time sharing system, MANAGING FILES is an important part of life, as is FINDING OUT WHAT'S GOING ON. 'Getting Information' relates to getting information, such as your files, limits, dayfile, while the chapter on 'Filespace Management' covers the aspect of dumping your files to tape, and hanging on to files you want to hang on to.

An amazing amount of the daily job of software production involves massaging text in some way or another. 'Text Manipulation

and Conversion Utilities' covers many aspects of TEXT MANIPULATION. Conversion of character sets tends to be a way of life rather than an infrequent exercise, and there are utilities to work well with the NOS version of ASCII.

Software Production really is a community activity, and 'SES Communications' describes facilities for users to talk to each other via a mail system.

SES PROCEDURES take care of the details of using software, such as locating, returning and rewinding files, obtaining required memory, obtaining required utility programs to perform tasks, and so on, leaving you free to specify what you want done. To state it briefly :

YOU decide WHICH SES PROCEDURE you want to use for the job.

SES generates the JOB CONTROL STATEMENTS to do the job.

Using SES, you can run a procedure directly as part of your interactive session, or you can make many of the procedures run independently as batch jobs.

Experienced or curious users wishing to create their own SES procedures can read the SES Procedure Writer's Guide.

At the end of this chapter there's a list of documents relevant to peaceful and productive coexistence with NOS.

As you read this document, you'll notice that although many SES procedures have lots of parameters, most parameters are optional. For example, the PRINT and FORMAT procedures have, respectively, ten and eighteen parameters; 90% of PRINT and FORMAT usage requires only one parameter. Finally, the usability of SES results from people using it and feeding back their (constructive) criticisms, ideas for enhancements to existing facilities and ideas for new facilities. We hope you'll do the same.

Note : We strongly recommend that you read all of this introductory chapter before reading other chapters. As you read this introduction, try using some of the SES procedures described in the examples as you go. Finally :

When all else fails, read the documentation!

1.1.1 USING AN SES PROCEDURE

Whether you use SES interactively, or run it in a batch job, the usage is the same, as shown here :

ses.procedure_name list_of_parameters

the ses before the period starts the SES processor. The period (.) following ses is required to keep NOS happy. The statement directs the SES processor to locate the procedure given by procedure_name, and when it has been located, to process that procedure and substitute the (optional) parameters given by list_of_parameters, and finally, execute the procedure either directly ("while you wait") as part of your interactive session, or SUBMIT the procedure as an independent job. For example :

ses.print blivet

PRINTs file blivet on the ASCII printer, and :

ses.count lines in grundge

COUNTs the number of lines in file grundge.

1.1.2 PARAMETERS FOR SES PROCEDURES

As mentioned above, you use an SES procedure by typing a statement of the form :

ses.procedure_name list_of_parameters

where list_of_parameters is the parameters required for that particular procedure.

Parameters may be specified by **keyword**, for example :

ses.print f=myfile

In this example, f is the parameter **keyword** or parameter **name**, and myfile is the parameter **value**. Generally, the name on the left of the equal sign is the parameter **keyword** or parameter **name**, while the thing on the right of the equal sign is the parameter **value**.

1.0 PREFACE
1.1.2 PARAMETERS FOR SES PROCEDURES

While parameters may be specified by keyword, it's also possible (and usually more convenient) to leave out the keyword and the equal sign and specify parameters **positionally**. This means that you don't have to type the keyword or the equal sign. For example :

```
ses.copyacr infile, outfile, 3..79
```

When keywords are used, it doesn't matter in what order you code the parameters. The use of COPYACR above could have been coded :

```
ses.copyacr o=outfile, 3..79, i=infile
```

however if you want to code parameters without using keywords, you must get them in the right order.

When you use an SES procedure you can code parameters in a mixture of both positional and keyword forms, whatever suits you.

Every SES procedure expects its parameters in a specific order. When you code a parameter by keyword, you effectively "tab" the parameter number to that "position". That's why, in the example above, we were able to code :

```
o=outfile, 3..79
```

by coding o=outfile we "tabbed" the parameter to the second "position". The next parameter (coded without the keyword) was already in the third "position". But when we wanted to code the input file name we had to specify it by keyword in order to "tab" back to the first parameter position.

Use of positional parameters sometimes leads to minor problems when what you think is a file name is in fact the keyword for a parameter. For example, many SES procedures have a parameter called group, which is the name of a group (or multi record) file. If you use a procedure such as COLLECT, for instance, like this :

```
ses.collect (tom, dick, harry) group
**E  CL  11001:  EXPECTING  "value(s)  for  parameter  GROUP" ON
COMMAND STATEMENT
```

in this case, the error message actually means : you have coded a parameter keyword, and not supplied a value for that parameter. In such cases, you have to either code group=group, or code the positional value in string quotes : 'group'.

Most SES procedures use short names like i and f for parameter

keywords, so it's best if you avoid single letter filenames. For example :

```
ses.uptolow f
```

will cause an error, because f is one of the keywords for the first parameter of UPTOLOW. You can either code the parameter as f=f, or code it as 'f', or use a name like grab or junk for the filename.

1.1.2.1 Types_of_Parameter_Values

Values supplied for parameters may be names, long_names, numbers, or character strings. These are described in detail here.

1.1.2.1.1 NAMES_AS_PARAMETER_VALUES

A **NAME** is from one to seven characters long. It must start with a letter, and may contain only letters or digits. For example :

A, ZQ91AP6, and FILEA are valid names, whereas : 1AJ, input_file, and oversizedname are invalid names according to the rules above. An example of using names in an SES procedure is :

```
ses.uptolow i=hibrow, o=lowbrow
```

for converting file hibrow from upper case to lower case with the result appearing on file lowbrow.

1.1.2.1.2 LONG_NAMES_AS_PARAMETER_VALUES

A **LONG_NAME** is from one to 31 characters long. It must start with a letter, and may contain letters, digits and the special characters : #, $, _, @. Any valid NAME is also a valid LONG_NAME, as are input_file and oversidedname.

1.1.2.1.3 NUMBERS_AS_PARAMETER_VALUES

A **NUMBER** contains only digits. The number may be followed by an optional BASE. For example :

579(10)	1493	are DECIMAL NUMBERS, or numbers to the base 10.
677(8)	1754(8)	are OCTAL NUMBERS, or numbers to the base 8.
0AF6(16)	3E8(16)	are HEXADECIMAL NUMBERS, or numbers to the base 16. Note that hexadecimal numbers must begin with a decimal digit, even if you have to start them with a zero. This avoids a potential confusion of a hexadecimal number with a name.

1.0 PREFACE
1.1.2.1.3 NUMBERS AS PARAMETER VALUES

Numbers such as 6789(8) and 3A5(10) are disliked by SES. Use of numeric parameters occur in procedures like :

```
ses.print copies=3, f=yinyang
```

to print three (3) copies of file yinyang on the ASCII printer.

1.1.2.1.4 CHARACTER STRINGS AS PARAMETER VALUES

A **CHARACTER STRING** is any arbitrary string of characters enclosed in single quote signs ('). If you wish to include a single quote sign in a string, you must represent it by two quote signs in a row. For example :

```
'this is an example of a character string'

'quote signs must be represented by two '' quote signs in a row'
```

You use strings in SES procedures when you have to supply strange names such as SYM$SM, and when supplying sentences to procedures :

```
ses.genupcf d='mcs$c30'
```

when generating an update compile file, and :

```
ses.link170 binfile, xld='ldset(usep=swisys,subst=swisy-$swisy.1$)'
```

to give extra loader directives to LINK170.

1.1.2.2 Ranges of Parameter Values

Some SES procedures handle parameter values in the form of a **RANGE**. For instance, procedure COPYACR has a parameter named **cols**, which represents a pair of column numbers. You can write the **cols** parameter in the form :

```
cols=3..65
```

the pair of periods is called an ELLIPSIS. The range specification consists of a **low side**, on the left of the ellipsis, and a **high side** on the right of the ellipsis. Another example of a procedure that accepts ranges is GENCOMP, which accepts ranges of module names, like this :

```
ses.gencomp almaden..bertero
```

which refers to all modules **almaden** through **bertero** inclusive, and :

```
ses.select lines 10..20 of galloop
```

to display lines 10 thru 20 of file **galloop** at the terminal.

1.1.2.3 Lists_of_Values_for_Parameters

Some SES procedures handle parameters consisting of many values. For instance, you can tell procedure PRINT to print a whole bunch of files, like this :

```
ses.print f=(atlas, orion, mercury, pegasus, sirius, hermes)
```

the value coded for the f parameter is called a **VALUE LIST**. A value list must be enclosed in parentheses, as shown in the example. The individual items in a value list are called **ELEMENTS** of the list, and they are separated from each other by a comma.

Elements in a value list can also be ranges :

```
ses.select lines (1, 2, 3, 12..17, 21..25) of widget
```

An element of a value list may itself be another value list. For instance, procedure GENCOMP has a parameter called ab, which can be coded in the form :

```
gencomp ab=((riesling, mondavi), gamay, (chenin, wente))
```

to get alternate bases from other user's catalogs. Note that the first and second elements in each of the sublists are the filename and the users catalog name, respectively. The description of the GENCOMP procedure talks more about the ab parameter. Also see the note below, at the end of this description on parameter lists. Procedure ASORT uses lists of lists to indicate the columns for the sort keys :

```
ses.asort i=ganip, o=ganop, keys=((1, 5), (4, 7, d))
```

Each sublist in the list above is a single sort key; the elements in the sublists are the start column, length, and direction of sort. See the description of the ASORT procedure for more information.

If you code only one value for a parameter, you can leave the parentheses off, like this :

```
ses.print f=myfile
```

but as soon as you want to supply more than one value you must supply the parentheses, like this :

```
ses.print (bowmore, jura, miltown)
```

Such are called **multi valued parameters**, or **value lists**, or just plain **lists**.

Referring to the discussion on the ab parameter of GENCOMP above, a few SES procedures handle parameters of that form. Those parameters are somewhat special forms for accessing files from other users' catalogs, for example, the LINK170 procedure uses this form for the list of libraries :

```
ses.link170 p=((lib_1,lib_2,user_1),lib_3,(lib_4,lib_5,user_2))
```

each **element** in the outer parenthesised list is itself a list (possibly containing only one element) of libraries and the places to get them from. So, in this example, the **first** sublist says : obtain lib_1 and lib_2 from the catalog of user_1; the **second** sublist has only **one** element, which means : obtain lib_3 from the catalog of the current user; the **third** sublist says : obtain lib_4 and lib_5 from the catalog of user_2. For procedures that use this form of parameter, the last element in each of the sublists is the user name of the catalog from whence to obtain the files. Conversely :

```
ses.link170 p=(lib_1, lib_2, lib_3, lib_4, lib_5)
```

means : obtain all five library files from the catalog of the current user.

1.1.2.4 Parameter_Lists_as_Parameter_Values

Some SES procedures handle multi valued parameters which are themselves parameter lists to be passed on to other procedures. For instance, procedure FORMAT has a parameter called print, which is actually the parameters for the PRINT procedure, used by FORMAT in the course of its run. You code the print parameter of FORMAT like this :

```
ses.format myfile, print=c=2

ses.format hisfile, print=(c=5,h='final ers')
```

1.1.2.5 Parameter_Keywords_as_Procedure_Options

Some SES procedures have parameters that don't want any values at all. In these cases, the parameter keyword is used to indicate OPTIONS for the procedure. For instance, procedure PRINT has a shift key (no pun intended), which if coded, indicates that the file to be printed has not been formatted for a line printer and must therefore be "shifted" to the right by 1 column and have carriage control characters added in (the new) first column. You use PRINT in such a case like :

```
ses.print f=anyfile, shift
```

and another example of options is GENCOMP (GENerate COMPile file), where you can type :

```
ses.gencomp all, b=cidbase, cf=comfile
```

in this case, the all key indicates that all modules are to appear on the resulting "compile file".

Throughout the procedure descriptions, we use the word key to mean a keyword used to provide options to a procedure.

1.1.3 USING MORE THAN ONE SES PROCEDURE AT A TIME

It's possible to use many SES procedures, each one following on right after the previous ones with no execution breaks between them. You do this by coding them all on the same line, each procedure terminated by a semicolon, like this :

```
ses.print myfile, noshift; print hisfile shift; catlist
```

This example PRINTs myfile and doesn't shift it (the noshift key), PRINTs hisfile with carriage control added (the shift key), and finally does a CATLIST of your files for you.

1.1.4 ORDINARY CONTROL STATEMENTS ON A CALL TO SES

It's also possible to include regular NOS control statements on the same line as SES procedure calls, all mixed together, like this :

```
ses.print myfile; 'return,myfile'; 'get,hisfile'; format hisfile
```

As you can see, the ordinary NOS control statements are enclosed in string delimiters just like any other character string. When SES processes one of these control statement strings, it follows the rule that :

if the string doesn't contain a period or a close parenthesis, a period is appended to the string, otherwise the string is left alone.

This means that you can place NOS control statements such as GTR and MODIFY (which use the control statement comment field for directives) with no problems. For example :

```
ses.'attach,plocrib'; 'gtr,plocrib,fred.*'; print fred
```

In this example, the second NOS control statement, namely the GTR, has a period in it, so SES doesn't place any period at the end of the string, whereas SES does place a period at the end of the first string (the ATTACH statement).

1.1.5 CONTINUING SES STATEMENTS OVER MORE THAN ONE LINE

SES can handle continuation lines (even when you're using SES interactively), for those cases where you can't specify all you want to on one line. You indicate continuation by placing two or more periods at the end of the line. SES then asks for continuation lines to be typed in. For example :

```
/ses.print f=(snark, shark, charmed, quarks, ..
..?  larks, boojums, grogs, bander, snatch)
```

After you've signalled the incomplete line by placing an ellipsis at the end of the line, SES prompts for a continuation line with a ..? at the start of each continuation line. The slash (/) before the ses on the first line indicates where NOS outputs its prompt for commands.

When you use SES in a batch job stream you simply code the continuation lines one after the other, each incomplete line having the ellipsis at the end, for example :

```
batch
   job
      statements
SES.COLLECT (CHENAS, FLEURIE, MORGON, CORNAS, TAVEL, ..
GHEMME, SOAVE, ORVIETO, CAREMA, PARRINA, BAROLO)
          and more
                batch
                   job
                      statements
```

1.1.6 PROFILE - SETTING UP YOUR LOCAL ENVIRONMENT

A PROFILE is a file where you can place information about you and the things you're currently working on. Many SES procedures use information out of the PROFILE to set up defaults for frequently used things, such as the name of an object library. Having a PROFILE can save you much time since you don't have to code parameters for information that's already in the profile.

If you want to have a profile, you should create a file called PROFILE which has the word PROFILE as the first line of the file. The most important and useful pieces of information to place in your profile are your password, charge number and project number. Thus, your initial profile should have (at least) the following in it:

```
PROFILE
\  passwor = 'your_password'
\  charge  = 'your_charge_number'
\  project = 'your_project_number'
```

An easy way to set up an initial PROFILE for yourself is to call the BuILD PROFile (**BLDPROF**) procedure like this :

```
ses.bldprof
```

BLDPROF prompts you for the most common items to be placed in your profile. In addition, it creates a MAILBOX for you. For information on the "mail" facilities of SES, see the chapter on "USER TO USER COMMUNICATIONS".

SES supplies a procedure called IAF, which sets up defaults for particular terminal types. BLDPROF prompts for your terminal type, and places that information in your profile. For further information on IAF, see the IAF procedure description in the "miscellaneous useful goodies" chapter.

BLDPROF will also ask you for your graphics terminal type which may be different than your usual terminal. Even if you use the same terminal for graphic (CADSG) and non-graphic work, you should answer both requests. BLDPROF will ask additional questions about your graphics terminal - baud rate, hard copy available, and for TEKTRONIX 4014 Extended Graphics Module and synchronous or asynchronous. For further information on CADSG, see the CADSG procedure description in the Structured Process Tools chapter.

There are many other things that may go into your profile. These are mainly concerned with source code maintenance and library management functions and so on. These other aspects of profiles are

discussed in the chapters dealing with the specific subjects.

Another useful thing that can go into a profile is SES directives that specify which procedure libraries, and where, to search for procedures. The SEARCH directive is discussed in the appendices.

1.1.7 SES PROCEDURES RUN AS BATCH JOBS

Instead of running interactively, or "while you wait" at your terminal, many SES procedures can be submitted to run in the background as batch jobs. You do this by coding a key (batch or batchn or defer) which causes the framework of a batch job to be built around the specified procedure, then SUBMITs the file so built to NOS as a batch job.

Using SES in this way is often very effective, especially where the job to be done uses lots of resources. By running the job in batch, you can proceed with your interactive session instead of having to wait.

For such procedures, the list of parameters described below are those which affect the running of the batch job. Note that these parameters are only applicable if you run the procedure in batch mode (that is, non-local).

jobun :
(optional) JOB User Name to be used on the USER statement of the batch job. If you don't code the **jobun** parameter, the job is built with the user name of the current user. Note : that in some NOS sites, it is not possible to SUBMIT a job to run under another user's account. The so called "secondary user" statment is disabled. If secondary user statements are disabled, and you try to submit a job for another account, you get logged off!

jobpw :
(optional) JOB PassWord to be used on the USER statement of the batch job. If you don't code the **jobpw** parameter, and there's no **passwor** variable defined in your profile (see the section on profiles), SES asks you for your password in a similar manner to that of NOS.

jobfmly :
(optional) JOB FaMiLY to be used on the USER statement of the batch job. If you don't code the **jobfmly** parameter, and there's no **family** variable defined in your profile (see the section on profiles), the family name for the current user is used.

jobcn :
(optional) JOB Charge Number to be used on the CHARGE

statement of the batch job. If you don't code the **jobcn** parameter, and there's no **charge** variable defined in your profile (see the section on profiles), SES asks you for your charge number.

jobpn :

(optional) JOB Project Number to be used on the CHARGE statement of the batch job. If you don't code the **jobpn** parameter, and there's no **project** variable defined in your profile (see the section on profiles), SES asks you for your project number.

jobfl :

(optional) specifies the JOB Field Length to be used for running the job. You don't normally need to code the **jobfl** parameter, since all the procedures which run in batch set the FL to the required value. If you want to specify the job field length as an octal number, you must add the (B) suffix to the number, because SES treats numbers as decimal unless told otherwise.

jobtl :

(optional) specifies the JOB Time Limit to be used for the job. This is always the same for all procedures, namely 2000(B). If you need more time for the job, you must code some value for the **jobtl** parameter. If you want to specify the job time limit as an octal number, you must add the (B) suffix to the number, because SES treats numbers as decimal unless told otherwise.

jobpr :

(optional) specifies the JOB PRiority (that is, the P parameter on the NOS job statement). If you want to specify the job priority as an octal number, you must add the (B) suffix to the number, because SES treats numbers as decimal unless told otherwise.

bin :

(optional) BIN number parameter used at sites allowing bin references to indicate which "bin" the job output should go in. The **bin** parameter is coded in the form of a string, for example : **bin='7h'**. You can also define a **userbin** variable in your PROFILE (see the section on profiles), so that the bin number is picked up automatically. SES generates a large print banner of your bin number on the

front of the job output, and also places the bin number in the job name field of the job statement. If you don't code the **bin** parameter, and there's no **userbin** variable defined in your profile, SES generates a bin number of NO-ID, which almost certainly guarantees that your job output ends up in the NOS/WO BIN NUMBER bin.

local or batch or batchn or defer :
coding one of these (optional) keys determines the mode in which the procedure or job is to be run. Coding the **local** key runs the procedure LOCAL to your terminal, or while you wait. Coding the **batch** key issues a SUBMIT with option B for the SUBMIT statement. Coding the **batchn** key issues a SUBMIT with the N option for the SUBMIT statement. Coding the **defer** key DEFERS the job for after hours running. You can find all of the details of the B and N parameters for SUBMIT in the NOS reference manual.

nodayf or dayfile or df :
this (optional) parameter applies to some of the procedures that can be run as batch jobs. Each procedure description (for procedures which can run as batch jobs) specifically mentions this parameter only if the procedure makes use of it. For those procedures that do use it, this parameter applies only if the procedure is run as a batch job (that is, non-local). If you don't code the parameter at all, or if you simply code **dayfile** or **df**, if the job hits an EXIT statement due to errors, the job dayfile is dumped to a file called **dayfile**, which is placed in your catalog when the job terminates. If you code **dayfile=filename** or **df=filename**, the dayfile is placed in your catalog in a file of name **filename**. If you code the **nodayf** option, no dayfile is produced at job end. The diagram below provides a more graphical explanation of the interactions of the **dayfile** parameter.

1.0 PREFACE
1.1.7 SES PROCEDURES RUN AS BATCH JOBS

Coded On Procedure	Action Taken
Nothing	Job dayfile is dumped to file dayfile if any errors occur during the job.
dayfile or df key only	Job dayfile dumped to file dayfile if any errors occur during the job.
dayfile=file_name or df=file_name	Job dayfile dumped to file file_name if any errors occur during the job.
nodayf key	Job dayfile is not dumped even if errors occur during the job.

Note : that only those procedures whose descriptions contain a reference to the parameters described above can be run as batch jobs.

Example of Batch Usage

A fairly widely used SES procedure is LISTMOD (which supplies a cross reference and a printout of the modules in a base - see the chapter relating to source text maintenance). LISTMOD runs "while you wait" by default, and since it grinds, it's much better to run it batch, like this :

```
ses.listmod  b=newdata, batch
 15.00.19. SUBMIT COMPLETE.  JOBNAME IS AFTQBXT
REVERT.    JOB LISTMOD SUBMITTED
```

This example shows a simple usage of LISTMOD run in batch by coding the batch key. The example assumes that your password, charge number, project number and all are defined in your PROFILE as described earlier. The message is returned by NOS : it's the time of day that your job was submitted, and the so called job name. The last message about job submitted comes from SES.

1.1.7.1 PROFILE Variables for Controlling Batch Mode

SES procedures that can optionally run in batch mode may be controlled by the settings of certain variables in your PROFILE. This makes it easier to use SES all round, since you don't then have to supply the information every time you use a procedure. The set of directives below can be coded into your profile if you wish.

```
PROFILE
\ passwor = 'your_password'
\ charge  = 'your_charge_number'
\ project = 'your_project_number'
\ family  = 'your_family'
```

passwor defines your password

charge defines your charge number.

project defines your project number.

family defines your family name.

Note : that if your validation on the system is such that you do not require a CHARGE statement in batch jobs or when you log in at a terminal, you should set the charge and project variables to null (empty) strings. This inhibits the generation of the CHARGE statement by SES for batch jobs.

Examples of PROFILES

```
PROFILE
\ passwor = 'ASTERIX'
\ charge  = 'OBELIX'
\ project = 'GETAFIX'
```

That is an example of a minimal PROFILE. Here's a typical PROFILE as it might be defined for a user :

```
PROFILE
\ passwor = 'WABBIT'
\ charge  = ''
\ project = ''
\ myname  = 'Lorenzo Desmondo O''Hare'
```

Note that the charge and project variables are set to empty strings - this would apply to sites that don't require CHARGE statements on NOS.

1.1.8 INFORMATIVE MESSAGES FROM SES PROCEDURES

Many SES procedures are set up such that when they run "while you wait" at the terminal, they output informative messages to tell you what they're doing. The informative messages are controlled by **msg** and **nomsg** keys on the appropriate procedures. The normal default, in the absence of any other information is for the procedures to output informative messages. Coding the nomsg key suppresses the messages. However, it is possible to establish your own default by defining a profile variable called msgctrl. The msgctrl variable can be defined in one of two ways :

\ msgctrl = '*' this sets the normal default for informative messages to no messages. In order to obtain messages from procedures the msg key must be coded on the procedure call.

\ msgctrl = 'SESMSG.*' this sets the normal default to output informative messages. To turn messages off, the nomsg key must be coded on the procedure call.

The table below should give a more graphical idea of the interactions of the msgctrl variable and the msg and nomsg keys on the procedures.

	DEFINED IN PROFILE		
Coded On Procedure	Nothing	\ MSGCTRL='SESMSG.*'	\ MSGCTRL='*'
Nothing	Messages	Messages	No Messages
MSG	Messages	Messages	Messages
NOMSG	No Messages	No Messages	No Messages

1.1.9 GENERAL NOTES ON SES PROCEDURES

SES procedures are, in general, written to do the detail work for you, so you can concentrate on the job at hand. Let's look at the workings of a typical SES procedure - COPY ASCII Coded Record (COPYACR), to see what it does :

```
ses.copyacr i=input_file, o=output_file
REVERT.    END COPYACR  INPUT_FILE -> OUTPUT_FILE
```

The COPYACR procedure emits NOS control statements to do the following :

```
acquire the input_file (see the note on ACQUIRE below)
acquire the program (tool) that does the COPYACR process
return the output_file
COPYACR from input_file to output_file
return the COPYACR program
rewind both the input_file and the output_file
```

you may if you wish, use procedures such as COPYACR like this :

```
ses.copyacr  file_to_be_processed
REVERT.    END COPYACR  FILE_TO_BE_PROCESSED
```

In such cases, where does the output of the procedure go? Well it certainly doesn't go to file output (the terminal). In such cases, the output file eventually replaces the input file :

```
acquire the file_to_be_processed
acquire the COPYACR program (tool)
COPYACR file_to_be_processed to a unique_named_temp_file
return the COPYACR program
* rename file_to_be_processed = unique_named_temp_file
return unique_named_temp_file
```

There are many SES procedures that function this way - they are often referred to as FILTERS - a process that copies an input to an output with some transformation in between.

ACQUIRING Files in SES Procedures

When you call up an SES procedure to :

```
ses.do_some_process i=input_file, o=output_file
```

one of the control statements emitted by the procedure is a statement to ACQUIRE the input_file, which means :

1. if there's a LOCAL file of the required name, REWIND that file and use it as the input_file.

2. otherwise ATTACH or GET a permanent file of the required name from the current user's catalog.

All this means you don't have to care whether or not files are local - SES procedures grab whatever is available under that name - that is what ACQUIRE is all about. There is a full description of the ACQUIRE utility in the Appendix.

The foregoing has some interesting effects which you should be aware of however. Consider the following sequence :

```
ses.files; catlist
 WORK... ...12 PT.
         CATLIST OF WCN        ( 9 FILES, 2685 PRUS )
 MAILBOX I SP A ...1  OLDUSER D PU R 1127  PNTSORS D PU R .105
 PROCLIB D SP R .124  PROFILE I PR R ...1  PROGLIB D PU R .835
 SEVEN.. I PU R ...2  STATSES D PU W .485  USERDIR I PU R ...5
REVERT.    END CATLIST

ses.pack  i=statses
REVERT.    END PACK  STATSES

ses.files
 WORK... ...12 PT.     STATSES ..481 LO.
REVERT.    END FILES
```

Note that we PACKed file STATSES. Note that originally, STATSES was a PERMANENT DIRECT ACCESS file. By the time the PACK process finished however, the PACK procedure (because of the RENAME process described earlier) has now left a LOCAL file of the same name. SES procedures don't, in general, rewrite permanent files. Those that do say so explicitly.

Using SES Procedures in Batch

If you followed the description of the behavior of SES procedures above, you may take note that SES is designed primarily as an INTERACTIVE tool. You can use SES in batch job streams if you wish, but you should be aware that SES procedures don't, in general, make a special case of files INPUT and OUTPUT, whereas NOS does make a special case of files INPUT and OUTPUT. Usually, when using SES procedures interactively, you don't use OUTPUT as the output file from procedures, those procedures that do use INPUT or OUTPUT say so, and take special note of those files. All this means that if you run SES in batch streams, and use files INPUT or OUTPUT as input or output to procedures, you're likely to lose a lot of output because SES procedures REWIND or RETURN things. So, to avoid grief, uses files other than INPUT and OUTPUT.

Unique Names in SES procedures

When SES builds the JCL stream from an SES procedure, programs, scratch files and so on are usually given unique names, so that you don't have to care which names are reserved by each particular procedure. These SES generated unique names are always seven characters long, always begin with the letters ZQ, and are always guaranteed to be different from the name of any file currently assigned to the running job. This note is for your information, just in case you ever look at a dayfile where SES procedures have been used, and see lots of funny looking names. The example below shows the JCL generated by the **ses.limits** procedure.

```
$RFL(20000)
$IFE(FILE(OUTPUT,AS),LABEL1)
$RENAME(ZQWT8Q6=OUTPUT)
$ENDIF(LABEL1)
$ASSIGN(MS,OUTPUT)
$LIMITS.
$RENAME(ZQWT8QQ=OUTPUT)
$IFE(FILE(ZQWT8Q6,AS)LABEL2)
$RENAME(OUTPUT=ZQWT8Q6)
$ENDIF(LABEL2)
$REWIND(ZQWT8RA)
EDT(ZQWT8QQ,ZQWT8RA,ZQWT8RW)
$RETURN(ZQWT8RA,ZQWT8RW)
$COPYBR(ZQWT8QQ,OUTPUT)
$RETURN(ZQWT8QQ)
```

```
$RFL(0)
REVERT.    END LIMITS
```

1.1.10 PROCEDURE DESCRIPTION CONVENTIONS

The remaining chapters of this handbook describe SES procedures that are currently available. Each procedure description contains a list of parameter descriptions; and the order in which the parameters are described specifies their positional significance within the procedure.

Only those procedures that make reference, in their parameter lists, to the **"batch job parameters"** (see the section called "SES PROCEDURES RUN AS BATCH JOBS") can in fact be run as batch jobs.

If a procedure description does not contain a description for a parameter, that procedure does not have such a parameter.

If a parameter description does not explicitly specify that a parameter is optional then the parameter must be supplied when the procedure is called.

In order to reduce redundancy within procedure descriptions, some of the features provided by a procedure are discussed in the descriptions of particular parameters; therefore, we **strongly** recommend that when you read a procedure description, you read the whole thing.

Before you read a particular procedure description, we **strongly** recommend that you read the introductory subsection of the chapter containing the procedure description. Such things as general parameter descriptions, profile usage, and naming conventions are discussed in those subsections.

1.1.11 PARAMETERS TO THE SES PROCESSOR

You have just read about parameters for SES procedures in the first sections of this chapter. You can also provide parameters to the SES processor to direct it's actions. These parameters come after the SES, and before the .procedure_name. These parameters may be specified in order dependent or keyword manner.

1.1.11.1 UN

The UN parameter specifies the catalog containing the requested procedure. For example, if a procedure is in the SESPLIB library on catalog TEAM, just type SES,UN=TEAM.procedure_name to use it. See also Appendix A.

1.1.11.2 LPFN

The LPFN parameter specifies the library permanent file name of the SESPLIB containing the requested procedure. If the library is on another catalog, both the UN and the LPFN parameters are needed. For example,

```
SES,UN=TEAM,LPFN=PROJLIB.procedure_name
```

gets the procedure from library PROJLIB on catalog TEAM. See Appendix A for more information about accessing procedures.

1.1.11.3 PUN

The PUN parameter specifies the catalog containing the PROFILE you wish to use. See Appendix A for additional details.

1.1.11.4 P

The P parameter gives the name of the PROFILE file you wish to use. See also Appendix A.

1.1.11.5 DUN

The DUN parameter is a default catalog that contains the user files needed by the procedure being called. For example,

```
SES,DUN=TEAM.PRINT TESTFIL
```

will print file TESTFIL from catalog TEAM. This parameter is generally used with the DPN parameter.

1.1.11.6 DPN

The DPN parameter is a default packname that contains the user files needed by the procedure being called. This parameter allows you to maintain your files on a private pack and still use them with SES.

1.1.11.7 Processor Operating Mode Parameter

The next parameter determines the operating mode of the SES processor.

The SES processor processes procedures in one of four modes :

RUN — This is the normal mode. The procedure is processed, presumably generating control statements, and then these control statements are executed.

TEST — In this mode the procedure is processed in the normal manner, but the generated control statements are not executed, instead they are placed on a designated file for possible inspection by the user. This mode is meant as an aid in debugging new procedures.

HELP — This mode is similar to test mode, however instead of generating control statements, a procedure set up for HELP mode produces some documentation on it's purpose and usage.

STATUS — This mode is similar to test mode, however instead of generating control statements, a procedure set up for STATUS mode produces any change information available about the procedure.

Refer to Appendix B for additional information on these modes.

1.1.11.7.1 RUN

This is the normal operating mode of the processor and is the default mode. The specified procedure is processed and executed. It is normally not necessary to specify run as the mode.

1.1.11.7.2 TEST

The TEST parameter tells the SES processor to execute in TEST mode. In this mode, the procedure is processed in the normal manner, but the generated control statements are not executed. Instead, they are placed on a designated file for possible inspection by the user.

```
SES,TEST=myfile.procedure_name
```

will place the generated control statements on file myfile. If no file is specified, SESTEST will contain the control statements.

1.1.11.7.3 HELP_OR_H

The HELP parameter tells the SES processor to execute in HELP mode. SES provides a mode of operation of procedures that produces descriptive information about the specified procedure. The information includes a one or two sentence description of what the procedure does plus a list of the parameters, their defaults, and the permissable values. In addition, required parameters are "flagged" with (R) and defaults that are controllable by PROFILE variables are "flagged" with an *. To get the help documentation for a procedure listed out on your terminal, you simply type :

```
ses,help.procedure_name
```

Notice that no parameters are given. SES responds by displaying the help documentation for the named procedure (if any is available). For example :

```
ses,help.getmod
```

produces :

```
GETMOD extracts 1 or more modules (decks) from a MADIFY base
library and places them, in MADIFY source format, on a group file.
Parameters are:
---PARAMETER-----DEFAULT---------ALLOWABLE VALUE(S)-----------------
(R) m,all          none           MADIFY deckname(s) (ranges allowed)
    g,group      *group           filename
    b,l          *base            filename
    un           *current user    username
    cors,c        none            filename
    width,w       madify          numeric maximum width of retrieved deck
    status,sts    none            keyword
    msg,nomsg    *msg             keyword
-HELP FOR GETMOD ON FILE OUTPUT-
```

SES,HELP=myfile.procedure_name will place the help documentation on file myfile.

Note : that help mode is not intended as a training mechanism or a tutorial for new SES users - that's what this SES User's Handbook is for - help mode is there as a "memory jogger" for more experienced SES users.

1.1.11.7.4 STATUS_OR_S

SES provides a STATUS mode of operation that lists current status information about the specified procedure. Changes to SES procedures that potentially affect users are communicated in this manner. To get status information on a particular procedure, you simply type :

ses,status.procedure_name

Status items are listed in reverse chronological order. All changes affecting the procedure since the previous SES Release are listed. For example :

ses,status.gencomp

could produce :

02/09/82

The field length value used by GENCOMP has been increased to 77700(8) to accommodate larger program libraries.
STATUS OF GENCOMP ON FILE OUTPUT

SES,STATUS=myfile.procedure_name will place the status information on file myfile.

1.1.12 PRE-RELEASE VERSIONS OF TOOLS

SES provides a convenient and consistent method for accessing pre-release versions of SES supplied tools. To access the pre-release version of a procedure, you simply type :

```
ses,sss.procedure_name list_of_parameters
```

If there is a pre-release available for the specified tool, it executes. If there is no pre-release version, you get the standard release version.

If you always want pre-release versions of tools that are available, you can place in your PROFILE the following directive :

```
\ SEARCH, USER, SSS, SESUNAM
```

This tells SES to first look in the current user's catalog for the specified procedure (just as normally happens), then look in the pre-release catalog (SSS), and finally look in the standard release catalog (specified by the SESUNAM variable).

A more detailed description of the SEARCH directive can be found in the Appendix.

1.1.13 PROBLEM REPORTING

Problems encountered while using a released SES tool should be reported using a PSR (Programming System Report, CDC form AA1901).

Problems encountered while using a pre-released SES tool should also be reported using a PSR (Programming System Report, CDC form AA1901).

The completed form along with all support materials should be sent to the Arden Hills PSR Coordinator (AR4230).

1.1.14 APPLICABLE DOCUMENTS

SES provides a procedure called TOOLDOC which provides "on-line" access to many SES documents. If you call TOOLDOC without giving any parameters, you get a list of the available DCS documents.

The following CDC manuals for NOS/170 may be of help to users of SES.

60455280	CYBIL Reference Manual
60457280	CYBIL Language Specification
60457290	CYBIL Handbook
60457260	NOS DEFINE Data Dictionary Tools User's Handbook
60457270	NOS GRAPH Graphics User's Handbook
60435400	NOS Version 1 Reference Manual (Volume 1)
60445300	NOS Version 1 Reference Manual (Volume 2)
60455250	Network Products Interactive Facility Version 1 Reference Manual
60435500	NOS Version 1 Time-Sharing User's Reference Manual
60450100	NOS Version 1 Modify Reference Manual
60492600	COMPASS Version 3 Reference Manual
60328800	SYMPL Reference Manual
60497800	FORTRAN Extended Version 4 Reference Manual
60481300	FORTRAN Version 5 Reference Manual
60429800	CYBER Loader Version 1 Reference Manual
60481400	CYBER Interactive Debug Version 1 Reference Manual

2.0 SES PROCEDURE LIBRARY MANAGEMENT

2.0 SES_PROCEDURE_LIBRARY_MANAGEMENT

SES supplies a separate set of procedures to manage SES procedure libraries. These libraries are in a unique format that is incompatible with other library management procedures provided by SES; e.g., REPMEM. SES procedure libraries are always maintained in sorted order. The terminology that these library management procedures use are as follows :

LIBRARY A collection of SES procedure records on a file. All SES procedure library management procedures use an l parameter to refer to a Library.

PROCEDURE A single record in an SES procedure library file, or a file containing a procedure which is added to or replaced on a library. All SES procedure library management procedures use a p parameter to refer to PROCEDURES.

GROUP a collection of procedure(s) that occupy one record per procedure on the file.

2.0 SES PROCEDURE LIBRARY MANAGEMENT

Summary Of SES Procedure Library Management Procedures

GETPROC GET PROCedures out of a library onto a group file. The procedures appear one per logical record.

REPPROC REPlace or add PROCedures on a library. You also use REPPROC in a special way to create a brand new library from scratch.

CATPLIB CATalog a Procedure LIBrary to dislay the names and the types of procedures in the library, in short form output.

LISTPROC LIST PROCedures. LISTPROC generates a catalog of all the procedures on a library, and optionally prints every TEXT procedure from the library.

WIPEPROC WIPE (delete) PROCedures from a library.

2.0 SES PROCEDURE LIBRARY MANAGEMENT

Parameter Naming Convention for SES Procedure Library Management

This section introduces the parameter naming convention that's common to most of the procedures in the procedure library management system.

l or b	name of Library on which the operations are to be performed.
p	stands for Procedure(s).
g or group	name of file upon which a GROUP of procedures are accumulated when adding or replacing procedures on a library, or when procedures are extracted from a library. A group file contains one procedure per logical record.
nl or nb	name of a New Library or New Base, used in those procedures that perform operations that permanently alter a library, requiring that a new library be created. An explanation of the method used to update a library is contained in a subsequent section.

2.0 SES PROCEDURE LIBRARY MANAGEMENT

PROFILE Variables for SES Procedure Library Management

The variables described below may be declared in your PROFILE in order to establish default information for the parameters of the procedure library management procedures.

plib name of procedure library where procedures are to be found or replaced.

plibown user name in whose catalog the procedure library is to be found.

newplib name of new procedure library for those procedures which update a library, thereby creating a new one. The default name for newplib is the same name as the old library.

group used by all procedures to designate a group file, or a file containing one procedure per logical record. The default name for group is **group**.

sestmpp used by all of the procedures that update procedure libraries to specify the temporary library file during the update process. The default name for this file is **sestmpp**.

lokmode determines the interlocking action when updating a procedure library. The **lokmode** variable may be set to 'LOCK' (lock by default unless the **nolock** parameter is coded on the procedure) or 'NOLOCK' (no lock by default unless the **lock** parameter is coded on the procedure).

intrlok used by all of the procedures that update procedure libraries to specify the interlock file to be used during the update process. The default name for the interlock file is **intrlok**.

2.0 SES PROCEDURE LIBRARY MANAGEMENT

SES Procedure Library Updating Process

Some of the procedure library management procedures physically alter a library, requiring that a new library be created. The process is common to all such procedures, and is as follows :

If you code both the l and nl parameters, the new procedure library is created simply by performing the required operations on the old library, and writing the newly created library onto the file specified by the nl parameter.

If you code only the l parameter, the update takes place in two stages. The updating operation is performed and the new library written to a file called sestmpp (SES TeMPorary Procedure). sestmpp is then physically copied back over the file specified by the l parameter, and finally sestmpp is purged.

Note that if the file specified by the nl (or l if nb is omitted) does not already exist, a DIRECT access READ mode file is DEFINEd for the library (this, of course, can only be done if the owner of the library is the current user).

Note that if anything goes wrong during an updating run, the original library is always left intact, and only the new library or sestmpp is potentially incorrect. This means that you can recover by purging the new library or sestmpp and starting the run again. If something goes wrong before the library appears on sestmpp, original library is intact, and sestmpp may be purged, whereas if something goes wrong after the new library is on sestmpp, that is, during the "rewrite" phase of the update, it is sestmpp that is safe, and the old library that is wrong. In this case, you may use the REWRITE procedure to do the rewrite again, or alternatively, you may just purge the old library and then change the name of sestmpp to be the library name.

2.0 SES PROCEDURE LIBRARY MANAGEMENT

Interlock Process for Updating a Library

When updating a library you can interlock the update so that only one user at a time can update the library. Those procedures that update a library are set up so that if the library to be updated is in another user's catalog, the library is interlocked by default. You can override default actions by defining a lokmode variable in your profile, or by coding a lock or nolock key on the procedure, as shown in the diagram on the next page.

Interlocking of a library is done via an interlock file. Such a file **must** be a **DIRECT** access file **in the same user's catalog as the library being updated**. Naturally the interlock file must be a PUBLIC, WRITE MODE file if other users are likely to be using it. The default name used by the SES source code maintenance and library management procedures for the lock file is intrlok. You can have an interlock file of any name by defining the intrlok profile variable, or by coding a file name as a value for the lock parameter on the appropriate procedures.

2.0 SES PROCEDURE LIBRARY MANAGEMENT

Coded On Procedure	Action Taken
lock	library or base is locked regardless of contents of lokmode profile variable or other conditions.
nolock	library or base is not locked regardless of contents of lokmode profile variable or other conditions.

Nothing	Coded in Profile	Action Taken
	\ lokmode='LOCK'	library or base is locked unless overridden by nolock parameter on procedure.
	\ lokmode='NOLOCK'	library or base is not locked unless overridden by lock parameter on procedure.

Nothing	Owner Of Base Or Library	Action Taken
	Current User	library or base is not locked unless the lokmode profile variable is set to 'LOCK' or the lock parameter is coded on the procedure.
	Another User	library or base is locked unless the lokmode profile variable is set to 'NOLOCK' or the nolock parameter is coded on the procedure

2.0 SES PROCEDURE LIBRARY MANAGEMENT
2.1 GETPROC - EXTRACT PROCEDURE(S) FROM PROCEDURE LIBRARY

2.1 GETPROC - EXTRACT PROCEDURE(S) FROM PROCEDURE LIBRARY

GETPROC is intended for extracting any number of procedures from a SES procedure library onto a group file, with one procedure per logical record. Parameters to GETPROC are :

p or proc or all :
If you code the **all** keyword, GETPROC extracts **ALL** procedures from the procedure library specified by the l parameter. Otherwise, you code the p parameter as a list of name(s) of Procedure(s) to be extracted. Ranges of procedure names may also be coded, as shown in the examples at the end of this description.

g or group :
(optional) name of **group** file to receive the procedure(s) extracted from the procedure library. If you don't code the g parameter, GETPROC uses the value of profile variable **group**, and if there's no such variable defined, uses the name **group** for the group file.

l or b :
(optional) name of procedure Library from which the procedure(s) are to extracted. If you don't code the l parameter, GETPROC either uses the value of profile variable **plib**, or if that's undefined, GETPROC uses the name **plib** for the name of the procedure library.

un :
(optional) User Name in whose catalog the procedure library specified by l is to be found, if it isn't in the catalog of the current user. If you don't code the **un** parameter, GETPROC uses the value of profile variable **plibown**, and if there's no such variable defined, the GETPROC uses the user name of the current user.

status or sts :
status is an (optional) key used for those cases where GETPROC is being used as a building block of more sophisticated procedures or jobs. The **status** parameter causes GETPROC procedure to set the job control register EFG to the value zero if GETPROC successfully completed, and non zero if anything went wrong during the run of GETPROC. If **status** is not quoted and an abnormal

condition occurs, GETPROC will EXIT.

Examples of GETPROC Usage

```
ses.getproc format l=sesplib un=lr77
REVERT.    END GETPROC  GROUP <- SESPLIB
```

The example uses GETPROC to get one procedure called **format** from a library called **sesplib** in the catalog of user **lr77**.

```
ses.getproc (txtcode,txtform,txthead) binary l=prglib un=mv73
REVERT.    END GETPROC  BINARY <- PRGLIB
```

This example extracts a group of procedures from an SES procedure library called **prglib** in the catalog of user **mv73** and places them on a group file called **binary**.

```
ses.getproc (catlist..change, compare..dayweek) l=sesplib
REVERT.    END GETPROC  GROUP <- SESPLIB
```

This example illustrates how ranges may be coded for a parameter. The example extracts all procedures **catlist** through **change** inclusive, and all procedures **compare** through **dayweek** inclusive, from an SES procedure library called **sesplib**, and places the extracted procedures on the file called **group**.

2.2 REPPROC - ADD OR REPLACE PROCEDURE(S) ON PROC. LIB.

REPPROC is a procedure to add new procedure(s) to, or replace procedure(s) on an SES procedure library. Parameters to REPPROC are:

p or proc :
(optional) list of file(s) containing Procedure(s) to be added to or replaced on the procedure library. If you don't code the p parameter, REPPROC assumes that all the procedure(s) to be added or replaced are already on a file specified by the g parameter.

g or group :
(optional) name of a group file on which procedure(s) are accumulated as one procedure per logical record. If you don't code the g parameter, REPPROC uses the value of profile variable group. If there's no such profile variable defined, REPPROC uses filename group as the group file.

l or b :
(optional) name of procedure Library on which the procedure(s) are to be added or replaced. If you don't code the l parameter, REPPROC uses the value of profile variable plib as the name of the procedure library. If there's no such profile variable defined, REPPROC uses a library name of plib.

nl or nb :
(optional) name of New procedure Library to be created when REPPROC has completed its run. If you don't code the nl parameter, REPPROC uses the value of profile variable newplib, and if there's no such profile variable defined, REPPROC writes the new procedure library over the file specified by the l parameter.

un :
(optional) User Name in whose catalog the procedure library specified by l/nl is to be found, if l/nl is not in the catalog of the current user. If you don't code the un parameter, REPPROC uses the value of profile variable plibown as the user name from whose catalog the procedure library is to be obtained, and if there's no such variable

defined, REPPROC uses the current user's catalog.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of REPPROC Usage

```
ses.repproc (first, second, third) l=berklib
*    FIRST -> GROUP
*    SECOND -> GROUP
*    THIRD -> GROUP
*    REPLACING/ADDING MEMBERS ON BERKLIB
*    LIBRARY NOW ON SESTMPP
*    NEW LIBRARY NOW ON BERKLIB
*    SESTMPP PURGED
REVERT.    END REPPROC  GROUP -> BERKLIB
```

This example illustrates REPPROC used to collect three procedures onto a group file, and then place the contents of the group file on an SES procedure library called **berklib**. The example shows some of the informative messages that appear during a REPPROC run.

Creating a New Library with REPPROC

```
ses.repproc nl=sesplib, g=newproc
*    CREATING NEW LIBRARY PROCLIB
REVERT.    END REPPROC  NEWPROC -> PROCLIB
```

This example illustrates how to build a new SES procedure library. The new procedures to be added are on a group file called **newproc**.

2.3 CATPLIB - PRODUCE LIST OF PROCEDURES IN A PROC. LIB.

CATPLIB is used to display a procedure list of the procedures in an SES procedure library. CATPLIB outputs the procedure list in a condensed format, five procedures per line, showing only the procedure names. Parameters to CATPLIB are :

l or b :

(optional) name of procedure Library which is to have its procedure names displayed. If you don't code the l parameter, CATPLIB uses the value of profile variable **plib** as the name of the library, and if there's no such variable defined, CATPLIB uses a file name of **plib**.

un :

(optional) User Name in whose catalog the procedure library specified by the l parameter is to be found. If you don't code the un parameter, CATPLIB uses the value of profile variable **plibown** as the user name, and if there's no such variable defined, CATPLIB uses the current user's catalog.

o :

(optional) name of file to receive the Output from CATPLIB. If you don't code the o parameter, CATPLIB places the procedure list on file output.

Example of CATPLIB Usage

```
ses.catplib b=seshlib un=mv69
 PROCEDURES ON SESHLIB PROCEDURE LIBRARY
ANYMAILHLP      ASORTHLP        BANNERHLP       BELLHLP         BOVLAYHLP
CHANGEHLP       CHANGPFHLP      COLLECTHLP      COMPHLP         COMPAREHLP
DAYWEEKHLP      DDCREATHLP      DDDISHLP        DDHLP           DDMERGEHLP
DUMP999HLP      DUMPPFHLP       EDTHLP          F5FORMHLP       FILESHLP
GENCPFHLP       GENMARHLP       GENMODSHLP      GENMSGHLP       GENPDTHLP
GETMAILHLP      GETMEMHLP       GETMODSHLP      GETPFHLP        GETPROCHLP
LIBEDITHLP      LIMITSHLP       LINK170HLP      LISTMEMHLP      LISTMODHLP
NEWPSRSHLP      NOWLISTHLP      PACKHLP         PERMITHLP       PERMPFHLP
REPMODHLP       REPPROCHLP      REPULIBHLP      RETAINHLP       REWRITEHLP
SESPARMHLP      SESPROCHLP      SIMDEFHLP       SIMFNUNHLP      SPECTRHLP
TMPFILSHLP      TO900HLP        TO999HLP        TOOLDOCHLP      TWOPAGEHLP
USEINFOHLP      USEMSGHLP       VEGEEHLP        VOLUMEHLP       WHOMAILHLP
XGENMARHLP      XGENMSGHLP      XLITHLP         XREFMODHLP      PLIB DIR
REVERT.     END CATPLIB     HELPLIB -> OUTPUT
```

This example shows CATPLIB output for a library **seshlib** in the catalog of user **mv69**.

2.4 LISTPROC - LIST CONTENTS OF PROCEDURE LIBRARY

LISTPROC performs two main functions to show you what's in an SES procedure library, and to get a printout of any number of procedures from the procedure library. LISTPROC first generates a list of the names of all the procedures specified. Lastly, LISTPROC generates a printout of all procedures specified. Parameters to LISTPROC are :

p or proc :
(optional) list of name(s) of Procedure(s) to be extracted. If you don't code this parameter, LISTPROC extracts all procedures from the procedure library specified by the l parameter. Ranges of procedure names may also be coded.

l or b :
(optional) name of procedure Library to be processed. If you don't code the b parameter, LISTPROC uses the value of profile variable **plib** as the procedure library name, and if there's no such variable defined, the LISTPROC uses the default name **plib**.

o or output :
(optional) name of file to receive the Output from LISTPROC. If you don't code the o parameter, LISTPROC places the output on some unique file name that it generates. The output doesn't default to file **output** because, if you're running LISTPROC interactively, you probably don't want the rather voluminous output on the screen.

un :
(optional) User Name in whose catalog the procedure library specified by l is to be found. If you don't code the **un** parameter, LISTPROC uses the value of the profile variable **plibown**, and if there isn't such variable defined, LISTPROC uses the user name of the current user.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **batch** mode, but it can be run in **local** mode. The **dayfile** parameter is not used by this procedure.

print :

this (optional) parameter indicates how the output of LISTPROC is to be printed. For the print parameter you may code any of the parameters to procedure PRINT. If you don't code the print parameter at all, LISTPROC prints one (1) copy of the output on the ASCII printer.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of LISTPROC Usage

```
ses.listproc print=c=3
 12.34.56. SUBMIT COMPLETE.  JOBNAME IS ACULARS
REVERT.    JOB LISTPROC  SUBMITTED
```

This example shows LISTPROC used to process a library, whatever name is the value of plib. Three copies of all the procedures are being produced. LISTPROC is being run in batch, rather than interactively.

```
ses.listproc b=sesplib o=procs print=(c=2,h='latest/sesplib')
   18.12.00. SUBMIT COMPLETE.  JOBNAME IS ANAPABS
REVERT.    JOB LISTPROC  SUBMITTED
```

This example shows LISTPROC being run in batch to produce two full printouts of the contents and all the procedures in the procedure library sesplib. The title on the listing is 'latest sesplib'.

2.5 WIPEPROC - DELETE PROCEDURE(S) FROM PROCEDURE LIBRARY

WIPEPROC is a procedure for deleting (wiping) procedure(s) from an SES procedure library. Parameters to WIPEPROC are :

p or proc :
list of Procedures to be deleted from a procedure library. Ranges of procedure names may also be coded.

l or b :
(optional) name of procedure Library from which procedure(s) are to be deleted. If you don't code the l parameter, WIPEPROC uses the value of profile variable plib as the name of the library. If there's no such profile variable defined, WIPEPROC uses the name plib as the name of the procedure library.

nl or nb :
(optional) name of New procedure Library to be created at the end of the WIPEPROC run. If you don't code the nl parameter, WIPEPROC uses the value of profile variable newplib as the name of the new procedure library. If there's no such profile variable defined, WIPEPROC places the new procedure library back over the old procedure library specified by the l parameter.

un :
(optional) User Name in whose catalog the procedure library specified by l/nl is to be found, if l/nl is not in the catalog of the current user. If you don't code the un parameter, WIPEPROC uses the value of profile variable plibown as the user name from whose catalog the procedure library is to be obtained, and if there's no such variable defined, WIPEPROC uses the current user's catalog.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the lock parameter determines the name of the interlock file. Interlocking is the <u>default</u> action when the base being updated is in another user's catalog. If you don't code either of the lock or nolock <u>keys</u>, interlocking is controlled by the lokmode profile variable. Refer to the introductory sections of this

chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :

These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

3.0 PRINTING FILES

3.0 PRINTING FILES

This section covers the facilities for printing files. Most of the discussion revolves around the ASCII printer, and the SES procedures to use the ASCII printer are as described below :

PRINT prints file(s) on the ASCII printer (the "normal" CDC printer is an option). If necessary, "carriage control" is added to the files before printing (it is possible to override this automatic determination but it is very seldom necessary to do so). Choices of character sets (full ASCII or the 64 character ASCII subset) are available.

PRINTID places "large print" heading banners at the start of a printout. Lines of title, date and time formats, and number of banners are selectable. PRINTID is used by PRINT to (optionally) place large print banners at the start of the file(s) to be printed.

COPYSAF COPY Shifted ASCII File. Copies an ASCII file to prepare it for printing at a later stage. The file is "shifted" if necessary. Character set conversion is also performed as required.

BANNER place "large print" banners of your bin number on the front of a printout. BANNER is used at sites where output is placed in "bins", identified by a bin number.

FICHE set up list file for microfiche processing.

3.0 PRINTING FILES
3.1 PRINT - PRINT FILE(S)

3.1 PRINT - PRINT FILE(S)

PRINT is a powerful facility for printing one or more file(s). It looks after the mundane matters of character set conversions, special disposition (route) codes and, perhaps most importantly, automatically determining whether each file to be printed has already been or needs to be "formatted" for a printer.

f or i :
specifies the name of the File, or a list of files to be printed. If the file(s) are not already local when PRINT is called, PRINT tries to GET or ATTACH the file(s) for you.

copies or c or n :
(optional) specifies the Number of Copies of the printout you require. If you don't code the c parameter, PRINT produces 1 (one) copy.

h or id :
(optional) specifies the Heading banner to be printed at the start of your printout. If you don't code the h parameter, the first two pages of the printout will have a "large print" banner of the file name, date and time.

hn or idn :
(optional) specifies the Number of Heading banners to be placed at the start of the printout. If you don't code the hn parameter, PRINT produces 2 (two) heading banners.

shn or sidn :
(optional) this parameter is used when printing more than one file at a time. When a list of files is printed, PRINT outputs a heading banner for the whole printout, and places a sub heading before each new file is started. The shn parameter specifies the Number of SubHeading banners to be printed before each new file. If you don't code the shn parameter, PRINT produces 2 (two) subheading banners.

printer or pr :
this (optional) parameter specifies on which printer the printout is to appear. If you don't code this parameter or if you specify **ascii**, the printout is routed to the

full ASCII printer. If you specify **ascii64**, the printout is routed to 64 character ASCII subset printer. If you specify **cdc**, the printout is routed to a "normal" CDC printer. Any other value for this parameter is illegal.

bin :

If you are at a site allowing bin references, printout needs some sort of a banner containing a Bin Number, to tell operations which bin to place the output in. Printouts without a bin number frequently end up in a bin labelled NOS/VE BIN NUMBER. At the time of writing, all bins have identifiers of the form digit digit letter, which means you must code the **bin** parameter in the form of a string, for example, **bin='22a'**. The string delimiters are required if you don't want SES complaining about invalid numbers. If you don't code a **bin** parameter, PRINT uses the value of profile variable **userbin**, and if there's no such profile variable defined, generates a listing banner of NO-ID, which almost certainly guarantees that your output goes to the NOS/VE BIN NUMBER bin.

setid :

setid is another site dependent special for sites allowing bin references. If your printout is destined for the "normal" (printer=cdc) display code printer, not the ASCII printer, and if your printout is large (more than about 100 pages).

compres :

(optional) keyword specifies multiple copies are printed together on one listing with only one NOS banner. When **compres** is used the number of heading banners is defaulted to zero unless overridden by specification of values for **hn** or **shn**.

cs612 or cs64 or cs812 :

these (optional) keys can be used to specify the character set of the file(s) to be printed. If you don't code any of these keys or if you code the **cs612** key, the file(s) is (are) assumed to be in the NOS 6/12 ASCII character set. If you code the **cs64** key, the file(s) is (are) assumed to be in the 64 character ASCII subset or display code character set. If you code the **cs812** key, the file(s) is (are) assumed to be in the "8 out of 12" ASCII character set.

3.0 PRINTING FILES
3.1 PRINT - PRINT FILE(S)

shift or noshift or cmps :
these (optional) keys control whether the printout is SHIFTed before disposal to the printer. If you don't code any of these keys, PRINT examines each file to determine how to prepare it for printing (i.e. which of these keys should have been selected). **Note** : that this automatic determination works for the vast majority of files and that this parameter is provided for use only in those rare cases where the automatic method fails. Note also, that the option you select with this parameter applies to all files to be printed, and therefore in the automatic case a mix of "shifted" and "unshifted" files can be printed with one procedure call. If you code the **shift** key, PRINT shifts (i.e. adds "carriage control" characters at the left of each line) each file in order to "format" the file(s) for printing. If you code the **noshift** key, PRINT assumes each file is already "formatted" for printing and therefore does no shifting. The **cmps** key is a special form of **noshift**, where any line that contains only a 1 in column one is deleted, and any line starting with an asterisk (*) in column one is also deleted. This is intended for printing the output of programs such as COMPASS and CATALOG which try to print on "prime pages". This idea is out of date these days, and the **cmps** key helps save paper.

date or etad :
this (optional) key specifies the form in which the DATE is to be output on the banner. If you code the **date** key, PRINT uses the form 01 AUG, 78, which is the default. If you code the **etad** key, you get the standard system date format.

time or ampm :
this (optional) key specifies the form in which you get the TIME on the banner. If you code the **ampm** key, PRINT uses the form 4:23 PM, which is the default. If you code the **time** key, you get the standard system time format.

un :
(optional) allows you to specify the user number for a printer not at the central site.

fc :
(optional) allows you to specify a forms code.

3.0 PRINTING FILES
3.1 PRINT - PRINT FILE(S)

Examples of PRINT Usage

```
ses.print (myfile, hisfile, herfile)
REVERT.    END PRINT    MYFILE..HERFILE
```

This example prints three files : myfile, hisfile and herfile on the ASCII printer. The example shows the message that appears at the end of the print process.

```
ses.print formout, c=3, noshift
REVERT.    END PRINT    FORMOUT
```

This example prints three copies of the file formout without shifting the file (the noshift key).

3.0 PRINTING FILES
3.1 PRINT - PRINT FILE(S)

PROFILE Variables for PRINT

If you wish, you may establish default options for PRINT by setting variables in your PROFILE. The profile variables that print uses are defined here.

copies default number of copies to print.

hn default number of heading banners to print at the start of a printout.

shn default number of subheading banners to print at the start of every file in a multi file printout.

printer default printer to be used. Refer to the **printer** parameter in the procedure description above.

incset default character set of file(s) to be printed. Refer to the **cs612** parameter described above.

shift sets your default for shifting files or not. Refer to the **shift key** description above.

userbin used only at sites allowing bin references. **userbin** sets up your default bin number.

printer_un default user number of a remote printer to be used. Refer to the **un** parameter description above.

printer_fc default forms code to be used. Refer to the **fc** parameter description above.

In addition to the profile variables described above, there are profile variables that PRINTID uses. See the description of these at the end of the PRINTID procedure description.

Note : because the **userbin** profile variable is used by PRINT, it is also used indirectly by any SES procedure that uses PRINT.

userbin is also used by those SES procedures that can run in batch. If you run procedures in batch, **userbin** is used to place bin number banners on the batch job output.

3.2 PRINTID - PRINT "LARGE PRINT" HEADING BANNERS

PRINTID is mainly used as a building block for other procedures. It produces a "large print" heading banner, that occupies a full page, using a program called JOBID which is described in an appendix to this document. Parameters to PRINTID are :

h or id :
this parameter may be any arbitrary character string which specifies the text to be output. The string used can be anything that's legal for the JOBID program.

copies or c or n or hn or idn :
(optional) Number of COPIES of the banner that you want output. If you don't code the copies parameter, PRINTID produces two (2) copies.

o or output :
(optional) name of file to which the banner is to be Output. If you don't code the o parameter, PRINTID uses file OUTPUT.

date or etad :
this (optional) key specifies the form in which the DATE is to be output on the banner. If you code the date key, PRINTID uses the form 01 AUG, 78, which is the default. If you code the etad key, you get the standard system date format.

time or ampm :
this (optional) key specifies the form in which you get the TIME on the banner. If you code the ampm key, PRINTID uses the form 4:23 PM, which is the default. If you code the time key, you get the standard system time format.

cs612 or cs64 or cs812 :
these (optional) keys specify the character set in which PRINTID output is to appear. If you don't code any of these keys or if you code the cs64 key, output is in the 64 character ASCII subset (can also be thought of as display code) character set. If you code the cs612 key, output is in the NOS 6/12 ASCII character set (since PRINTID output does not include any of the so-called

extended characters, cs612 is effectively equivalent to cs64 for this procedure). If you code the cs812 key, output is in the "8 out of 12" ASCII character set. **Note**: that when PRINTID is called by the PRINT procedure, that this parameter is set according to the destination printer; the PRINT procedure has a parameter with these keys and it refers to the character set of the file(s) to be printed.

Note that the **date** and **time** parameters only have effect when the **h** parameter is specified as a name. If you give the **h** parameter as something other than a name, it is up to you to include the date and time in the heading if you want it.

Example of PRINTID Usage

```
ses.printid id='final/debug/document', o=banner
REVERT.    END PRINTID    BANNER
```

This example shows a title banner being formatted onto a file called banner.

PROFILE Variables for PRINTID

You can establish defaults for various PRINTID parameters by coding them in your PROFILE. The profile variables are described here. Also refer to the PRINT profile variables.

myid is a character string specifying the default format for heading banners. For complete understanding of the following description refer to the documentation for the JOBID program (which produces the banner's) and to the SES Procedure Writer's Guide.
The default heading banner format would be defined via this profile variable as follows :

```
\ myid ='''/''++VALS(h)++'''//)''++mydate++'''//)''++mytime'
```

Other heading banner formats can be constructed in an analogous manner. To eliminate heading banners altogether (as the default case) you should code the following line in your profile :

```
\ myid = NO
```

Note that in order to produce a heading banner, the length of the string assigned to the myid variable must be at least 4 (four).

mydate is a character string containing your version of the date. Refer to the **date key** in the procedure description above.

mytime is a character string containing your version of the time. Refer to the **time key** in the procedure description above.

outcset can be used to specify the default character set for PRINTID output. Refer to the **cs64 key** in the procedure description above. Note that this variable has no effect when PRINTID is being called by the PRINT procedure.

3.3 COPYSAF - COPY SHIFTED ASCII FILE TO PREPARE FOR PRINTER

COPYSAF prepares a file for printing on the NOS ASCII printer. The file in question is converted to the ASCII8 character set if required, and shifted (carriage control added) if required. Parameters to COPYSAF are :

i or f :
name of Input File to be prepared for the printer.

o :
(optional) name of file to receive the Output from COPYSAF. If you don't code the o parameter, the output appears on the file specified by the i parameter.

incset or ic :
(optional) designator for the INput file's Character SET. The table below describes the allowed designators.

outcset or oc :
(optional) designator for the OUTput file's Character SET. The table below describes the allowed designators.

shift or noshift or cmps :
these (optional) keys control whether the printout is SHIFTed before conversion. If you don't code any of these keys, COPYSAF examines the file to determine how to prepare it for printing (that is, which of these keys should have been selected). Note : that this automatic determination works for the vast majority of files and that this parameter is provided for use only in those rare cases where the automatic method fails. If you code the shift key, COPYSAF shifts (that is, adds "carriage control" characters at the left of each line) the file in order to "format" the file for printing. If you code the noshift key, COPYSAF assumes the file is already "formatted" for printing and therefore does no shifting. The cmps key is a special form of noshift, where any line that contains only a 1 in column one is deleted, and any line starting with an asterisk (*) in column one is also deleted. This is intended for printing the output of programs such as COMPASS and CATALOG which try to print on "prime pages". This idea is out of date these days, and the cmps key helps save trees.

3.0 PRINTING FILES
3.3 COPYSAF - COPY SHIFTED ASCII FILE TO PREPARE FOR PRINTER

The **incset** parameter has a default of **cs612** (NOS 6-12 character set) and the **outcset** parameter defaults to **cs812** (ASCII character set suitable for the NOS ASCII printer), as described in the table below, but these defaults can be overridden by defining in your profile defaults via variables with names the same as the parameters. The following table defines the allowed designators for the **incset** and **outcset** parameters :

Designator	Meaning
cs612	NOS 6/12 ASCII character set
cs64	64 character ASCII subset (display code) character set
cs812	"8 out of 12" ASCII character set

Examples of COPYSAF Usage

```
ses.copysaf  listing, topress, noshift
REVERT.    END COPYSAF  LISTING -> TOPRESS


ses.copysaf  outfile, cs64
REVERT.    END COPYSAF  OUTFILE
```

The first example of COPYSAF converts the file **listing** from the 6-12 character set to the 8-12 character set, placing the result on file **topress**. The **noshift** key ensures that the file is not shifted during the conversion. The second example is converting a display code file to the 8-12 character set ready for printing, and COPYSAF determines whether to add carriage control to the file.

3.4 BANNER - WRITE BIN NUMBER ON LARGE PRINT BANNER PAGE

This procedure is a special form of the PRINTID procedure for use by sites allowing bin references to identify printed output.

BANNER is intended for writing the user's bin number on a large print banner page (actually it produces two pages of banner) on printouts or in the dayfile of a batch job. BANNER produces two pages with your bin number arranged in a geometrical pattern that makes it easy for the process control persons to see. BANNER outputs the page in different patterns depending on the number of characters in your bin number. Parameters to the BANNER procedure are :

bin :
is your (optional) bin number. This must be specified in the form of a string, for example, bin='018d'. If you don't code the bin parameter, BANNER uses the value of profile variable **userbin**, and if there's no such profile variable defined, generates a bin number of NO-ID.

o or output :
is the (optional) name of the file on which the banner is to be written. If you don't code the o parameter, the banner appears on file **output** by default, making this a convenient default for batch jobs.

Examples of BANNER Usage

```
ses.banner
REVERT.    END BANNER


ses.banner  bin='13b', o=tempout
REVERT.    END BANNER  TEMPOUT
```

The first example shows the simplest use of BANNER with the banner going to file **output**, and the bin number either obtained from the **userbin** variable in the profile, or the default NO-ID generated. The second example shows how to code the bin number parameter on the procedure line.

PROFILE Variables for BANNER

The only PROFILE variable used by BANNER is :

userbin defines your output "bin" identifier as a character string.

Note : that the **userbin** profile variable is also used by PRINT, and thus indirectly by any SES procedure that uses PRINT. **userbin** is also used by those SES procedures that can run in batch. If you run procedures in batch, **userbin** is used to place bin number banners on the batch job output.

3.5 FICHE - SET UP LIST FILE FOR MICROFICHE PROCESSING

FICHE accepts an input file consisting of CYBIL, 180 Assembler, META Assembler or COMPASS code and writes it to a tape in a format suitable for microfiche processing at Comsource Headquarters. For any module name encountered in the listing file, a banner page will be generated, for visual aid on the microfiche, and it will appear in the index of the microfiche. The tape which is made from running FICHE should be sent with a 'Microfilm Work Request' (form no. AA5167) to Terry Larson - HQC00H. The 'Program ID' used on the 'Microfilm Work Request' is ARH1833. A 'Master Charge No.' will also be required to submit this 'Microfilm Work Request'. Please contact Kathy Beatty (482-2338) for information needed to acquire a 'Master Charge No.' Parameters to the FICHE procedure are :

list or l :
name of the **list** file to be used as input.

id :
fiche **id** title which will appear at the top of the microfiche. The **id** supplied should be a string limited to no more that 38 characters.

tapevsn or vsn :
number of the tape to be used for the output of FICHE.

Example of FICHE Usage

```
ses.fiche l=ulist id='ses utilities' vsn=ses016
```

This example shows the input file **ulist** being used to generate a fiche from the output contained on the tape **vsn ses016**. The title at the top of the fiche when it comes back from processing will be **ses utilities.**

4.0 DOCUMENT FORMATTING SYSTEM

4.0 DOCUMENT FORMATTING SYSTEM

The documentation formatting system is a powerful set of programs that can produce documents from unformatted source text, containing directives to control margin alignment, itemizing, tables and so on. There are facilities to compare two versions of a document to generate revision bars to indicate changes, and facilities to change heading formats and so on.

The major components of the documentation system are two processors called TXTCODE and TXTFORM. TXTCODE is in fact a high level preprocessor for TXTFORM. Type SES.TOOLDOC to reference the manuals that describe TXTCODE and TXTFORM.

After the main descriptions of FORMAT and FORMREV, there are a number of "data flow diagrams" showing the processes that are run by coding different parameter combinations on a FORMAT or a FORMREV procedure usage.

FORMAT runs the complete document formatting system in all its manifestations and options, while the rest of the procedures described after FORMAT are the individual components of the system.

FORMREV runs the document formatting system to generate a revision package, complete with change pages and change directive summary.

SPELL produces a list of suspected misspelled words from a text file.

GRADLVL computes the reading difficulty of a document.

TWOPAGE places two formatted document pages on one printer page.

DIAGRAM aids the construction of block diagrams such as those found later in this section.

4.0 DOCUMENT FORMATTING SYSTEM

MEMO produces a standard format memo.

TXTCODE runs the high level preprocessor for TXTFORM.

TXTFORM produces a formatted document.

TXTHEAD a TXTFORM post processor which changes heading banners on document pages.

GENREVB works in conjunction with TXTFORM to format a document with revision bars down the right side of the page to show changes from a previous document.

GENREVP builds a revision package by comparing the output of two formatted documents.

4.0 DOCUMENT FORMATTING SYSTEM
4.1 FORMAT - RUN DOCUMENT FORMATTING SYSTEM

4.1 FORMAT - RUN DOCUMENT FORMATTING SYSTEM

FORMAT produces a formatted document either from TXTCODE source (the default), or from TXTFORM source. FORMAT normally runs in batch rather than interactively, but you can run FORMAT at the terminal if you wish, by coding the **local** key when you run it. It is usually better to run the document formatters in batch since they consume some time. Parameters to FORMAT are :

i or f :
name of Input File containing source data to be formatted by TXTFORM or TXTCODE.

b or old :
(optional) name of file containing a Base or OLD version of the document source. If you code the b parameter, FORMAT compares the files specified by b and i such that revision bars are printed on the formatted output document.

l or listing :
(optional) name of file to receive the output of the formatter in a form suitable for printing on the ASCII printer. If you don't code the l parameter, and you're running FORMAT LOCAL at your terminal, the output appears on a file called **listing**.

d or display :
(optional) name of file to receive the output of FORMAT in a form suitable for DISPLAYing at a terminal, that is, the formatted document is truncated for a terminal screen. If you don't code the d parameter, FORMAT doesn't produce any display file (unless you code the **k** parameter described below).

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **batchn** mode, but it can be run in **local** mode. The **dayfile** parameter is used by this procedure.

print :

(optional) controls the action of the PRINT procedure, used to print the output of FORMAT. If you run FORMAT as a batch job (the default case) and you don't code the print parameter, FORMAT prints one (1) copy of the document on the ASCII printer. If you run FORMAT in local mode or "while you wait", FORMAT does not print any copies of the document. For a complete description of the values that may be coded for the print parameter, see the description of procedure PRINT.

s or source :

this (optional) key specifies that the SOURCE of the document is to be printed in addition to the formatted output. If you don't code the s key, the source isn't printed.

k or keepout :

(optional) name of file in which to KEEP the OUTput of FORMAT. You can use this facility where you may want to run many copies of a document at a later date. If there's no file of name specified by the k parameter in your catalog, FORMAT places the output in a DIRECT access file of the specified name. If such a file already exists, FORMAT overwrites it with the output of the run.

Note : that if you code the k parameter, FORMAT produces a display file (see above) and saves that, rather than saving the full size listing file which is usually large.

code or form or txtcode or txtform :

these (optional) keys determine which formatter is to be used by FORMAT to process the document source. If you don't code any of these keys, FORMAT runs the TXTCODE process, which is the same as if you coded the code or txtcode keys. If you code the form or txtform key, FORMAT runs the TXTFORM process. Note: you can change the default selection by defining in your profile a variable called FORMAT and setting its value to 'TXTFORM' or 'FORM'.

head or txthead :

this (optional) key invokes TXTHEAD, a special postprocessor for the document formatting system. TXTHEAD is used to change the headings that appear at the top of the document pages, where the standard headings are

inappropriate. Documentation for TXTHEAD can be found in an appendix to this User Handbook. If you code the head key, FORMAT runs TXTHEAD.

twopage :

this (optional) parameter, if coded, directs FORMAT to run the TWOPAGE procedure (described in this chapter) which places two pages on one printer page. Since the maximum width of a printer page is 136 columns, if you happen to have wide platen documents, the output tends to look rather strange, so for effective use of twopage, your documents should be less than 68 columns wide. twopage may be coded just as a key, or it may have a list of values, namely the pw parameter and the seq key, as in the description of the TWOPAGE procedure later in this chapter.

Note : that the twopage option only affects the printed copy of the output. The actual data in the files specified by the d or l parameters is not affected.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

backup :

this (optional) key specifies that the source is to be processed by the old processor (TEXTCODE/TEXTFORM). If you don't code the backup key, FORMAT uses the new processor (TEXTPRO).

Examples of FORMAT Usage

```
ses.format myfile
 17.42.55. SUBMIT COMPLETE.  JOBNAME IS ADCQBUG
REVERT.    JOB FORMAT  SUBMITTED
```

This is the very simplest example of using FORMAT. myfile contains document source written in TXTCODE form. The document is processed by FORMAT, and one (1) copy is printed on the ASCII printer. FORMAT runs as a batch job, freeing up your terminal for further processing.

```
ses.format newfile, oldfile, print=(c=2,h=no), s, jobtl=4000(8)
 13.32.55. SUBMIT COMPLETE.  JOBNAME IS ADUQBOG
REVERT.    JOB FORMAT  SUBMITTED
```

This example is more complex. The files newfile and oldfile contain respectively the latest and previous versions of a document. Both versions are coded in TXTCODE form. FORMAT processes both lots of source, uses the revision bar generator to generate change bars on the final document. The source of newfile is printed because the s key is coded on the call line. It's assumed that the documents are large and require lots of computer time to process them, so jobtl=4000(8) is coded to give FORMAT twice the normal time. Finally, the print parameter specifies that two (2) copies are to be printed with no heading banners.

```
ses.format i=newfile, b=oldfile, txtform, print=c=4
 11.35.43. SUBMIT COMPLETE.  JOBNAME IS ADRICAD
REVERT.    JOB FORMAT  SUBMITTED
```

This example shows FORMAT used to format the document newfile, with oldfile as the base version for generating revision bars. The txtform key indicates that the source of the two documents is in TXTFORM format, and so the TXTCODE stage of FORMAT is bypassed. Finally, four (4) copies of the output are printed, because of the print=c=4 parameter. This last parameter indicates how the print parameter is coded when it has only one subparameter in its list, as opposed to the example above, where the print parameter has two subparameters in its list, so that they must be enclosed in parentheses.

4.0 DOCUMENT FORMATTING SYSTEM
4.1 FORMAT - RUN DOCUMENT FORMATTING SYSTEM

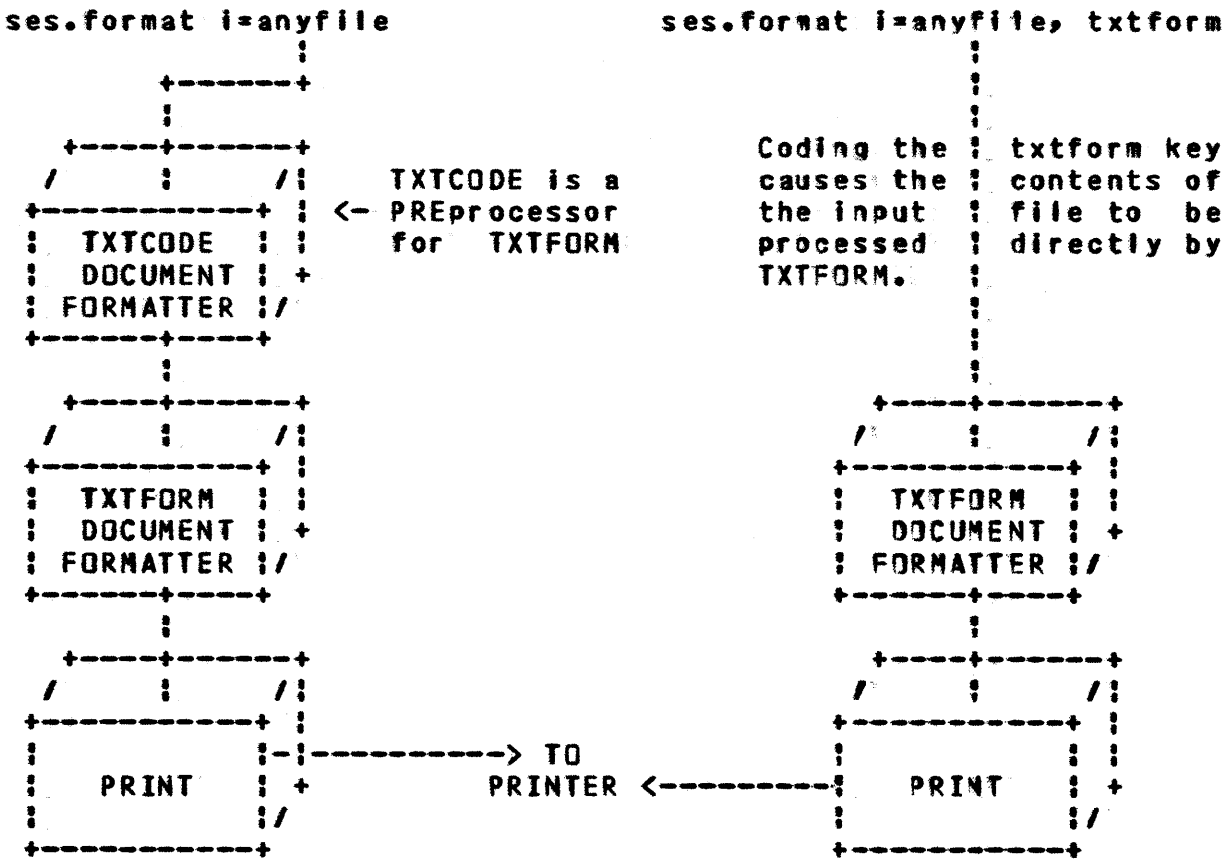
```
ses.format i=grabage local print d=looksee
*    RUNNING TXTCODE ON GRABAGE
*    RUNNING TXTFORM
*    FORMATTED DOCUMENT ON FILE LISTING
*    PRINTING LISTING
*    CREATING DISPLAYABLE DOCUMENT LOOKSEE
REVERT.    END FORMAT  GRABAGE -> LISTING  LOOKSEE
```

This example shows how to use FORMAT at the terminal, or "while you wait". The file grabage is assumed to contain the source of a document in TXTCODE form. FORMAT runs while you wait because the local key is coded. FORMAT places the formatted document on a file called listing and prints one (1) copy of it on the ASCII printer (because the print key was specified). The file called looksee contains a copy of the formatted document suitable for displaying at a terminal. The example shows the informative messages output by FORMAT to reassure you that there's something going on.

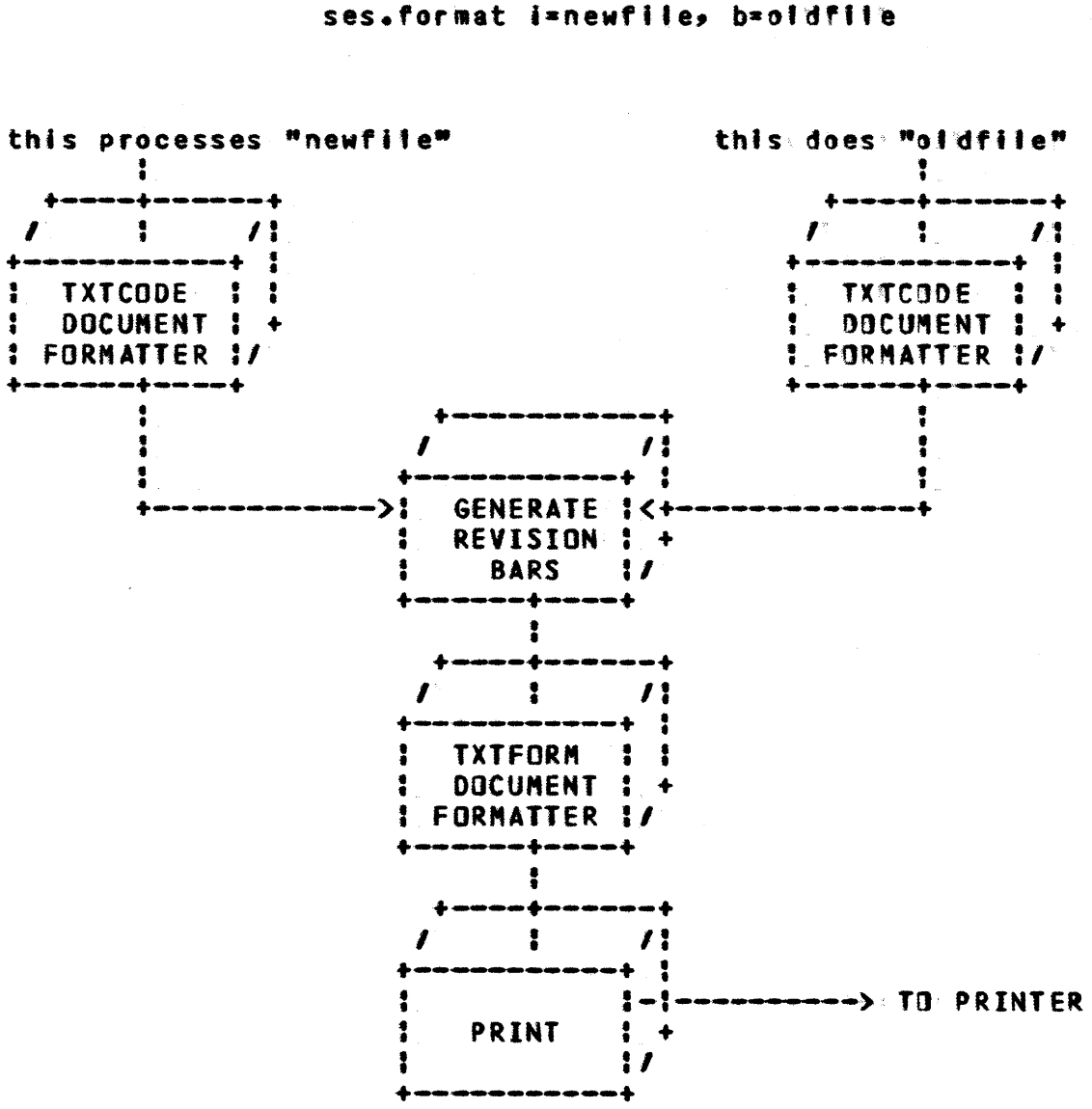
```
ses.format draft batch twopage=seq
 12.10.30. SUBMIT COMPLETE.  JOBNAME IS AMTBSOB
REVERT.    JOB FORMAT  SUBMITTED
```

This example shows the twopage option used to double up on the printed output.

Data_Flow_of_FORMAT_Procedures



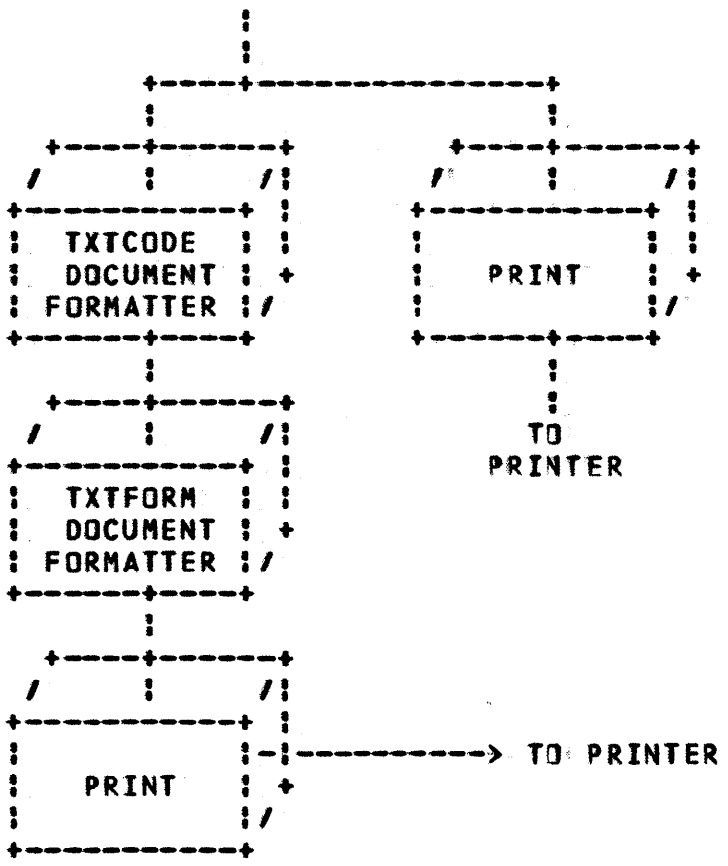
The diagram shows two of the ways that FORMAT can run, depending on the options you code. The left hand diagram shows the normal and simplest case of FORMAT. The right hand diagram shows that coding the txtform or form key eliminates the TXTCODE process.



This diagram shows the action taken when you run FORMAT with a current document and a "base" version of the document. The revision bar generator compares the two files after TXTCODE has processed them, and generates the commands such that TXTFORM knows where to place revision bars in the final document.

4.0 DOCUMENT FORMATTING SYSTEM
4.1 FORMAT - RUN DOCUMENT FORMATTING SYSTEM

ses.format myfile source



This diagram shows what happens when you run FORMAT and code the source key. The source of the document is printed directly, then the formatting process takes place to produce a formatted document.

4.2 FORMREV - FORMAT A REVISION PACKAGE FROM DOCUMENT SOURCE

FORMREV is intended to format documents and run the GENREVP package described later, to produce a revision package for a document. FORMREV uses the FORMAT and GENREVP procedures in a manner suitable for most of the standard day to day document applications. If you have any complicated situations you should use FORMAT and GENREVP "manually" to cater for the situation you want. Parameters to FORMREV are :

i or f or new :
name of NEW Input File containing the new document source to be processed by FORMREV.

b or old :
(optional) name of file containing OLD document source to be processed by FORMREV. If you don't code this b parameter, you should supply the file specified by the r parameter.

r or rlist :
(optional) name of file containing the output of a previous FORMAT run. If you don't code the r parameter, you should supply the file specified by the b parameter. Of course, both b and r may be coded.

l or listing :
(optional) name of file to receive the listing of the selected pages after the comparison. If you don't code the l parameter, FORMREV uses file listing.

s or summary :
(optional) name of file to receive the summary of the changes, and a "how to update the document" description. If you don't code the s parameter, FORMREV uses file summary.

o or output :
(optional) name of file to receive the OUTPUT of FORMREV. The output file contains error messages and statistics. If you don't code the o parameter, FORMREV attempts to use PROFILE variable output, and if such a variable doesn't exist, file output is used.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **batchn** mode, but it can be run in **local** mode. The **dayfile** parameter is used by this procedure.

print :
(optional) parameter list (as for the PRINT procedure) which controls the PRINTing of the l, s and o files. If you run this procedure in **local** mode, no printing is performed. If you run this as a batch job, the default action is to print one (1) copy of those files.

format or format1 :
(optional) list of parameters to control the FORMAT run that produces the new version of the text. The parameters are as for the FORMAT procedure described previously.

format2 :
(optional) list of parameters that control the FORMAT run that produces the old version of the text, in the case that the r parameter wasn't coded. The parameters are as for the FORMAT procedure described previously.

genrevp :
(optional) list of parameters that control the actions of the GENREVP procedure described below, when GENREVP is called into play during the run of FORMREV to produce the final output listings. The parameters are as for GENREVP described below.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of FORMREV Usage

```
ses.formrev i=newdoc, b=olddoc
 09.45.34. SUBMIT COMPLETE.  JOBNAME IS AEXICOG
REVERT.    JOB FORMREV  SUBMITTED
```

This example shows FORMREV used to generate a revision package by comparing **newdoc** (the current version) with **olddoc** (the previous version). FORMREV produces a printout consisting of those pages which differ between the two documents, and a summary indicating which pages are changed. The changed pages have revision bars against those lines that differ.

```
ses.formrev newdoc, r=oldlist, format=txtform, jobtl=5000(8)
 14.45.56. SUBMIT COMPLETE.  JOBNAME IS JACIBHI
REVERT.    JOB FORMREV  SUBMITTED
```

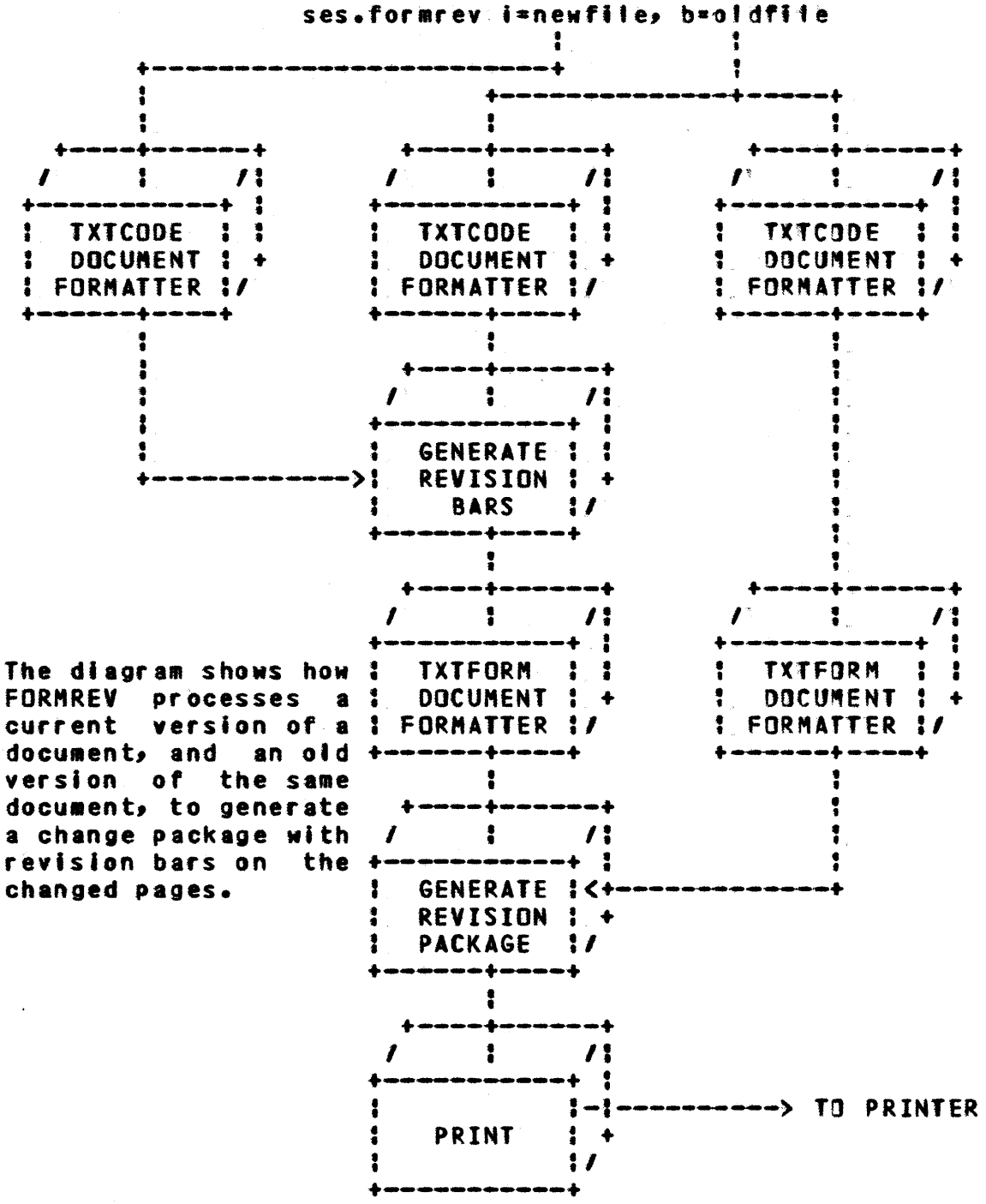
This example generates a revision package by comparing the results of formatting the current document **newdoc** with the old listing file **oldlist**. The formatting process is to be done with the TXTFORM formatter, bypassing the TXTCODE process, as indicated by the **format=txtform** parameter. The **job time limit** is set to 5000 seconds (octal) by the **jobtl=5000(8)** parameter.

```
ses.formrev i=current, b=dated, genrevp=(how=comfile, folio)
 16.32.48. SUBMIT COMPLETE.  JOBNAME IS AFTXBUD
REVERT.    JOB FORMREV  SUBMITTED
```

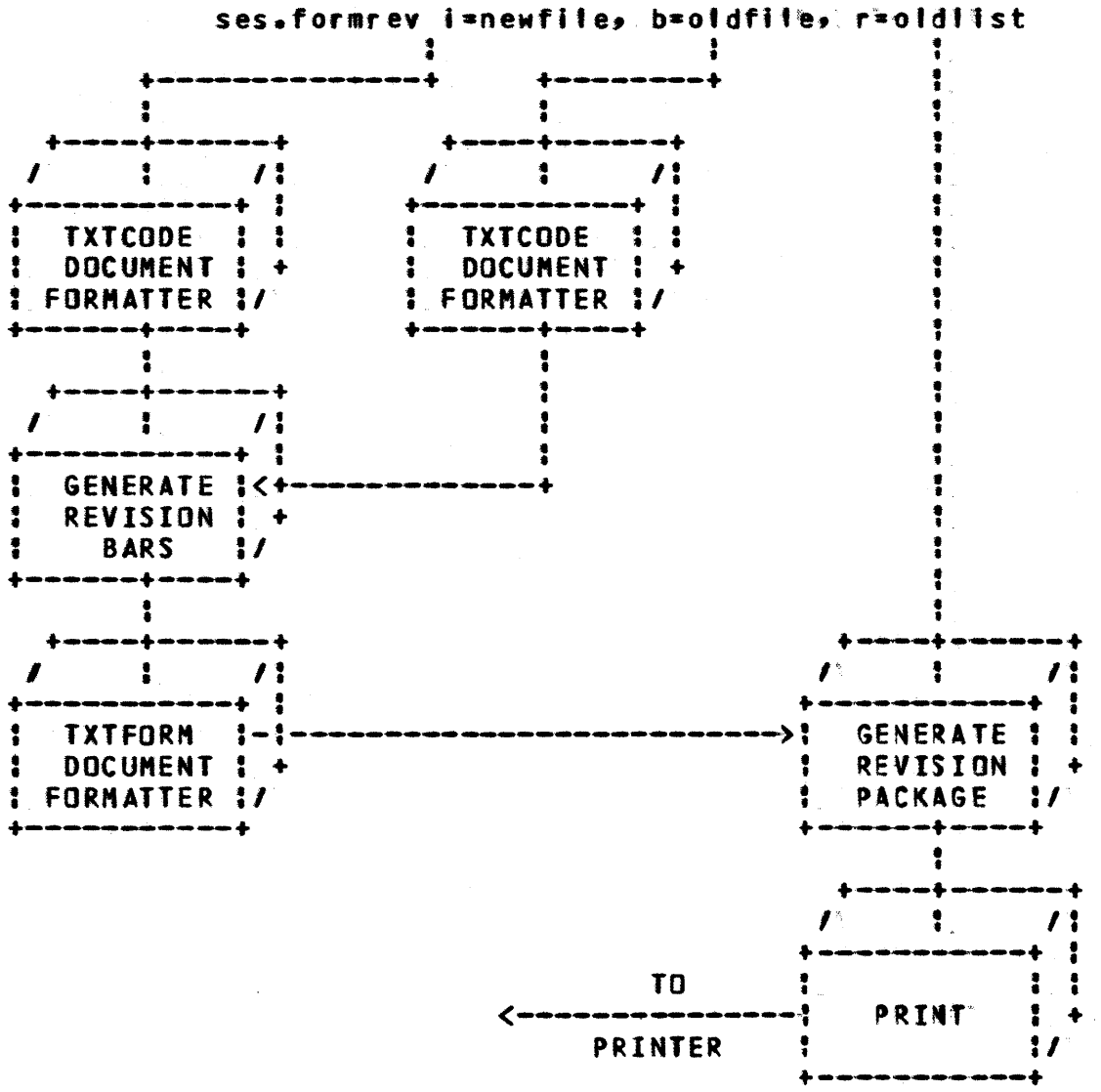
This example shows FORMREV used to generate a change package for the documents **current** and **dated**. The portions of the listing pages that are to be ignored are not the standard ones normally used by the GENREVP procedure, so the **genrevp** parameter is coded to indicate that the directives are on file **comfile**, and that the **folio** line (indicated by the **folio** key) is also to be ignored for the purpose of comparison.

4.0 DOCUMENT FORMATTING SYSTEM
4.2 FORMREV - FORMAT A REVISION PACKAGE FROM DOCUMENT SOURCE

Data_flow_of_FORMREV_Procedures

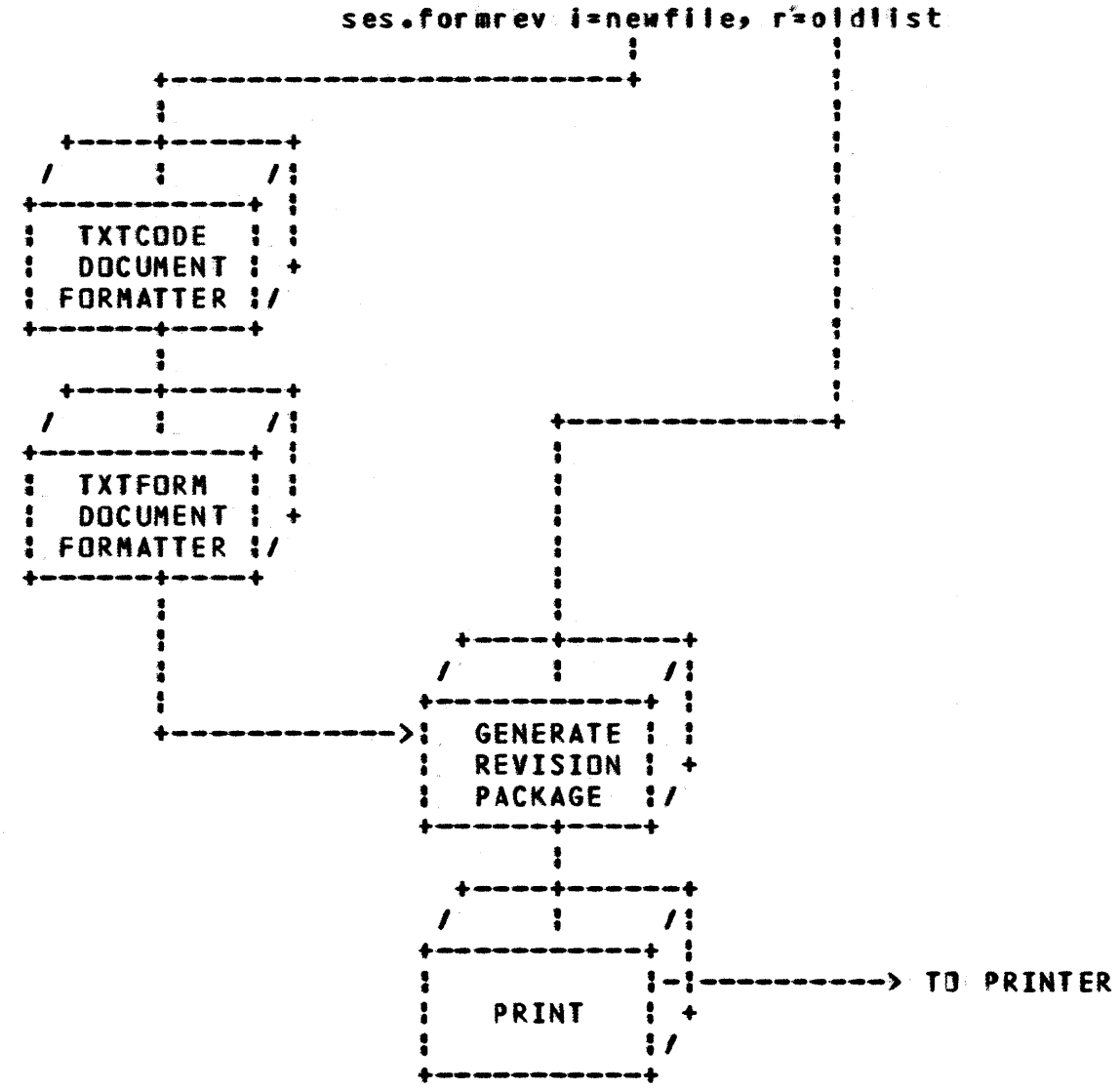


The diagram shows how FORMREV processes a current version of a document, and an old version of the same document, to generate a change package with revision bars on the changed pages.



This diagram shows FORMREV used to generate a revision package where all three forms of the document are used : the current version of the document, **newfile**, the old version of the document source, **oldfile**, and the old document listing, **oldlist**. In this case, FORMREV uses the old listing as the base against which to compare the processed version of the new document. Because the old document source was used as well, the change pages produced by FORMREV have revision bars on them.

4.0 DOCUMENT FORMATTING SYSTEM
4.2 FORMREV - FORMAT A REVISION PACKAGE FROM DOCUMENT SOURCE



This diagram shows the simplest use of FORMREV. The current version of the document is compared with the old listing of the document in order to generate a change package. In this case there are no revision bars generated on the final output since there was no old document source given to FORMREV to be processed.

4.3 SPELL - CHECK FILE FOR SPELLING MISTAKES

SPELL checks for words in a document that are spelled incorrectly. Parameters to SPELL are :

i or f :
name of Input File to be checked for spelling mistakes.

o :
(optional) name of the file to receive the Output from SPELL. If you don't code the o parameter, SPELL uses file OUTPUT.

dict or d :
(optional) name of a file containing words (one per line, left justified, all letters in upper case) that are to be used in conjunction with standard "dictionary" used by SPELL. If you don't code this parameter, SPELL uses only the standard dictionary (containing approximately 50000 words) when checking for misspelled words.

dictun or dun :
(optional) Username in whose catalog the additional dictionary specified by **dict** is found. The default is the current user's catalog.

lo :
(optional) List options available for output format.
brief, the default, displays three columns containing each "mispelled" word, the input file line number at which the word was first encountered, and the number of subsequent references of that word in the text.
full is a proof reader's format. The entire text line containing the "misspelled" word is printed out and the word in question is "flagged" on every occurrence.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

msg or nomsg :
These (optional) **keys** control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of SPELL Usage

```
ses.spell newdoc
REVERT.    END SPELL  NEWDOC
```

This example shows the simplest use of SPELL. A list of the words in file NEWDOC that are suspected to be misspelled along with a count of the number of times each occurs is written to file OUTPUT. Only the standard dictionary was used since the dict parameter was omitted.

```
ses.spell i=somedoc, dict=jargon, o=baddies
REVERT.    END SPELL  SOMEDOC
```

This example shows SPELL applied to a file called SOMEDOC with additional words from dictionary file JARGON. The output of SPELL appears on file BADDIES.

4.4 GRADLVL - DETERMINES READING LEVEL OF A DOCUMENT

GRADLVL reads the input file and computes the reading difficulty of the document. Difficulty is expressed in terms of the U.S. grade level that would find the document understandable. Two reading level measurments are computed, the Flesch and the Kincaid. In general, the Flesch measure will report a slightly higher grade level than the Kincaid. For technical material, the Kincaid value is probably more accurate. The 'hical' option will provide a list of document words of three or more syllables on the output file. If the reading level of your document is too high, try substituting simpler words. Parameters to GRADLVL are :

i or f :
name of input file containing text to compute the reading difficulty of.

o :
(optional) name of file to receive the output of GRADLVL. If you don't code the o parameter, GRADLVL uses file OUTPUT.

incset :
(optional) designator for the INput file's Character SET. The table below describes the allowed designators.

outcset :
(optional) designator for the OUTput file's Character SET. The table below describes the allowed designators.

hical :
(optional) key provides a list of document words of three or more syllables on the output file.

Both the **incset** and **outcset** parameters default to cs612 (see below). The following table defines the allowed designators for the **incset** and **outcset** parameters :

Designator	Meaning
cs612	NOS 6/12 ASCII character set
cs64	64 character ASCII subset (display code) character set
cs812	"8 out of 12" ASCII character set

4.5 TWOPAGE - PRINT TWO DOCUMENT PAGES SIDE BY SIDE

TWOPAGE takes the output of any of the text processors and places two output pages on one printer page. This reduces both paper usage and print time, which is invaluable while documents are in their development stages and being turned around fast.

The input for TWOPAGE must be in 8 of 12 ASCII format and not in NOS 6/12 ASCII. Its output is also in 8 of 12 ASCII. Parameters to TWOPAGE are :

i or f :
name of Input File to be processed by TWOPAGE.

o :
(optional) name of Output file from TWOPAGE. If you don't code the o parameter, the output appears on a file of the same name as the input file specified by the i parameter.

pw or width :
(optional) Page Width of the output pages. If you don't code the pw parameter, TWOPAGE uses a default page width of 137 (136 columns plus one column of carriage control). Remember to cater for the carriage control column when specifying the page width.

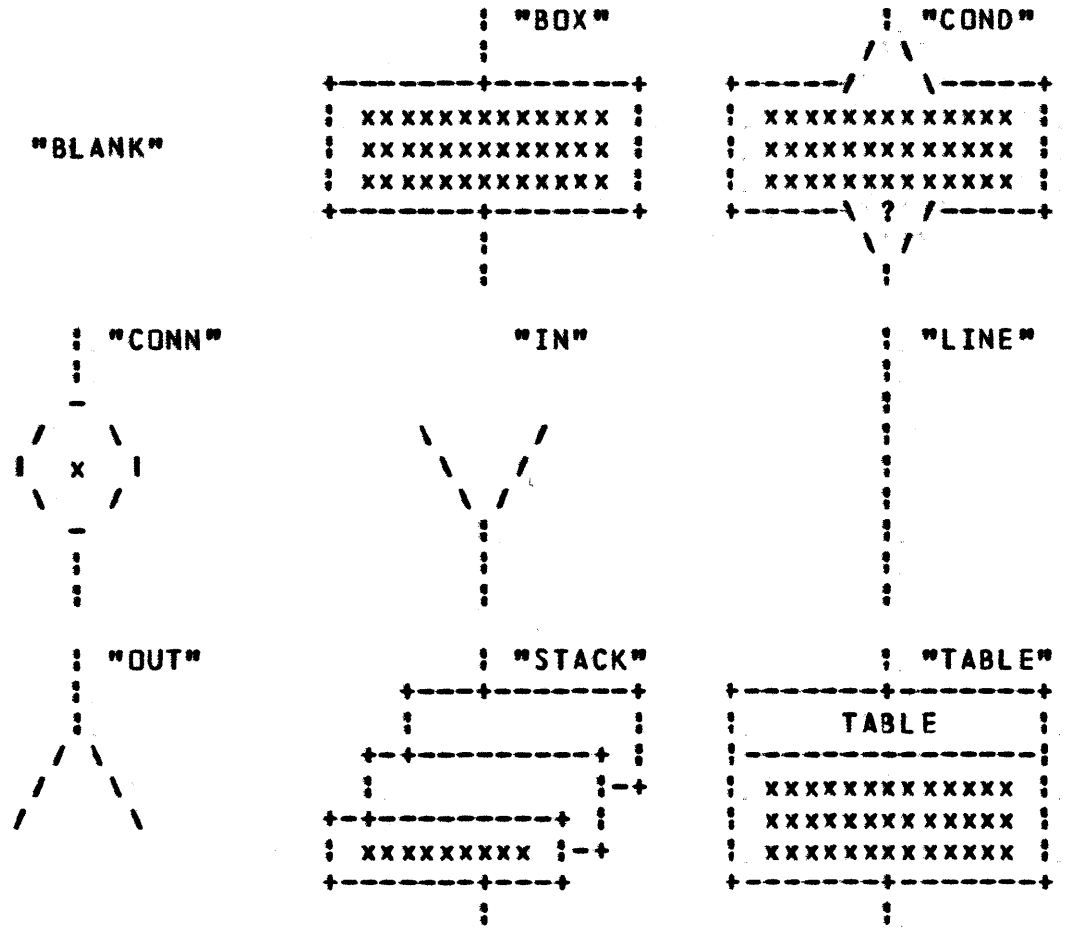
seq :
this (optional) key indicates that SEQuence numbers are to be placed on the final printout. The sequence numbers appear down the center of the printer page between the two document pages. Default action in the absence of the seq key is no sequence numbers.

Examples of TWOPAGE Usage

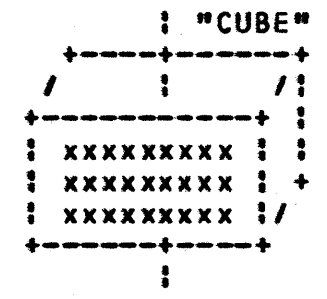
```
ses.twopage i=textout, o=listing, seq
REVERT.    END TWOPAGE  TEXTOUT -> LISTING


ses.twopage formatd
REVERT.    END TWOPAGE  FORMATD
```

4.6 DIAGRAM - DIAGRAM DRAWING AID

DIAGRAM is a documentation aid for drawing diagrams. The "data flow diagrams" in this section of the SES User's Handbook were drawn using DIAGRAM. Basically, a page is considered as a three column, five row grid, into which you can place various shapes. You still have to do some manual editing to join lines and add text, but the bulk of the pain of drawing the shapes is done for you. Shapes available via DIAGRAM are those shown below, with their "names" given for reference.



4.0 DOCUMENT FORMATTING SYSTEM
4.6 DIAGRAM - DIAGRAM DRAWING AID



To use DIAGRAM you call up the SES DIAGRAM procedure with the following parameters :

o :
name of the file to receive the output from DIAGRAM. This parameter is required.

The next three parameters are introduced by the keywords c1 and c2 and c3. They represent a list of shapes that are to be placed in column one, column two and column three of the page grid, respectively. For example, the grid of shapes given above was generated by the following SES call :

```
ses.diagram o=anyfile, c1=(blank,conn,out,cube), ...
    c2=(box,in,stack), c3=(cond,line,table)
```

where anyfile is the output file used for the example. The shape "names" were added by editing the output file. It is usual to include diagrams in your file in "asis" mode.

4.7 MEMO - GENERATE STANDARD MEMO HEADER

MEMO places a standard TXTCODE format banner at the front of a file containing the text of a memo. Date, 'to', 'from' and subject information can be entered as parameters to MEMO. Internal to the resultant file, the TXTCODE table directives are set up such that further recipients' names may be added easily. Parameters to MEMO are as follows :

i or f :
name of Input File containing the text of your memo, to which the standard header is to be added.

o :
(optional) name of the file to receive the Output of MEMO. If you don't code the o parameter, MEMO places the output on the file specified by the i parameter.

to :
(optional) name and location of the person to whom the memo is to go. It is coded as a list; the first item in the list is the recipient's name and the second item is the location.

from :
(optional) name and location of the person sending the memo. It is coded similar to the to parameter.

when :
(optional) date to be placed on the memo. If you don't code the when parameter, MEMO uses today's date as a default.

s or subject :
(optional) string which is the subject line to be placed in the memo.

Some of the parameters above can get default information from your profile, as follows :

faccode is the FACility CODE variable. The value of faccode are

used both for the to and from location if they aren't specified as parameters.

myname is the profile variable containing your name, which is also used by the mail facilities. If you don't code the first element of the from parameter list, MEMO uses the myname profile variable.

By this time you should be suitably confused by this explanation so an example is in order :

```
ses.memo i=anyfile, to=('Ethel Snerge', london),...\.
    from=('Stanislaus Bonokowonokowitch', paris),..\..
    subject='hunting the snark'
REVERT.    END MEMO  ANYFILE
```

The result of this would be a standard memo header at the start of **anyfile** as shown on the next page :

4.0 DOCUMENT FORMATTING SYSTEM
4.7 MEMO - GENERATE STANDARD MEMO HEADER

```
\para
\setu~
\seth^
\setb@
\blank010
\skip4
\asis
\bold
                MM    MM  EEEEEEE  MM    MM   00000
       C        M M M M  E        M M M M  M  0     0        C
       D        M  M  M  EEEEE    M  M  M  M  0     0        D
       C        M     M  E        M     M  M  0     0        C
                M     M  EEEEEEE  M     M   00000
\nobold
\table;1,45
\skip4
 ;^^^DATE^:^MAR 14, 1979
\skip1
\table;1,45
 ;^^^^^TO^:^ETHEL SNERGE;LOCATION^:^LONDON
\table;11,56
\skip1
\table;1,45
 ;^^^FROM^:^STANISLAUS BONOKOWONOKOWITCH;LOCATION^:^PARIS
\table;11,56
\skip2
\table;1,45
 ;SUBJECT^:^~HUNTING THE SNARK~
\block
\skip4
```

4.8 TXTCODE - RUN TXTCODE DOCUMENT PREPROCESSOR

TXTCODE is a high level preprocessor for TXTFORM. This procedure runs TXTCODE in a stand alone fashion. It may be used as a building block for larger procedures to form a complete document formatting system. Parameters to TXTCODE are :

i or f :
name of Input File containing source text to be processed by TXTCODE.

o :
(optional) name of file to receive the Output from TXTCODE. If you don't code the o parameter, the output appears on the file specified by the i parameter.

e :
(optional) name of file to receive the Error output from TXTCODE. If you don't code the e parameter, TXTCODE doesn't produce any error file.

Example of TXTCODE Usage

```
ses.txtcode codein codeout errlist
REVERT.    END TXTCODE  CODEIN > CODEOUT, ERRLIST
```

This example shows TXTCODE used to process the data contained in file codein, and produce the TXTFORM source on file codeout. Any errors go to file errlist.

4.9 TXTFORM - RUN TXTFORM DOCUMENT PROCESSOR

TXTFORM is the main component of the document processing system. This procedure runs TXTFORM stand alone. It can be used as a building block for larger procedures to form a complete document formatting system. Parameters to TXTFORM are :

i or f :
name of Input File containing source text to be processed by TXTFORM.

o :
(optional) name of file to receive the Output from TXTFORM. If you don't code the o parameter, the output appears on the file specified by the i parameter.

Example of TXTFORM Usage

```
ses.txtform i=forminp, o=formout
REVERT.    END TXTFORM  FORMINP -> FORMOUT
```

4.10 TXTHEAD - PAGE HEADING PROCESSOR

TXTHEAD is a postprocessor for TXTFORM. TXTHEAD is used to change the format of the heading banners on document pages, where the standard heading format is inappropriate. There is documentation on TXTHEAD in and appendix to this User Handbook.

This TXTHEAD procedure runs TXTHEAD in a stand alone fashion. It may be used as a building block for larger procedures or jobs to form a complete document formatting system. Parameters to TXTHEAD are :

i or f :
name of Input File containing TXTFORM output which is to be processed by TXTHEAD.

o :
(optional) name of file to receive the Output from TXTHEAD. If you don't code the o parameter, the output appears on the file specified by the i parameter.

Example of TXTHEAD Usage

```
ses.txthead i=formout, o=headout
REVERT.    END TXTHEAD  FORMOUT -> HEADOUT
```

This example shows TXTHEAD being used to process the output of TXTFORM contained on the file **formout**, and produce the output file **headout**.

4.11 GENREVB - GENERATE REVISION BARS FOR DOCUMENTS

GENREVB is a procedure that works in conjunction with TXTFORM, to format a document with bars down the right hand side of the page to show changes from a previous document. GENREVB functions by comparing the source text of two documents and generating a set of special directives to TXTFORM. Parameters to GENREVB are :

i or f or new :
name of File containing the NEW version of the document in TXTFORM source format.

b or old :
name of file containing the OLD (or Base) version of the document in TXTFORM source format.

o :
(optional) name of file to receive the Output from GENREVB. If you don't code the o parameter, the output appears on the file specified by the i parameter.

newcset :
(optional) designator for the NEW file's Character SET. The table below describes the allowed designators.

oldcset :
(optional) designator for the OLD file's Character SET. The table below describes the allowed designators.

outcset :
(optional) designator for the OUTput file's Character SET. The table below describes the allowed designators.

ls or ignorls :
these (optional) keys specify whether or not to IGNORe Leading Spaces on lines being compared. The default action is to recognize leading spaces (the ls option). If you code the ignorls key, GENREVB ignores leading spaces on text lines.

The default character set designator for all three cset parameters is **cs612**. The following table defines the allowed designators for the cset parameters :

Designator	Meaning
cs612	NOS 6/12 ASCII character set
cs64	64 character ASCII subset (display code) character set
cs812	"8 out of 12" ASCII character set

Example of GENREVB Usage

```
ses.genrevb i=newtext, b=oldtext, o=changes
REVERT.    END GENREVB  NEWTEXT : OLDTEXT -> CHANGES
```

This example shows GENREVB used to compare the two document source files **newtext** and **oldtext**, and produce the change directives in the text on file **changes**

4.12 GENREVP - GENERATE A REVISION PACKAGE FOR A DOCUMENT

GENREVP builds a revision package by comparing the output of two formatted documents. Parameters to GENREVP are :

new or newdoc :
name of NEW DOCument to be processed by GENREVP.

old or olddoc :
name of OLD DOCument to be processed by GENREVP.

how or howfile :
is (optional) and specifies HOW the processing is to be performed, that is, which lines are to be ignored for the purposes of the comparison (date, time and such). The how parameter is either the name of a file containing the directives, or it's a list of character strings on the control card which specify the directives. If you don't code the how parameter, GENREVP attempts to use the file name associated with PROFILE variable myhowf. If there isn't such a variable in your profile, GENREVP uses a standard set of directives suitable for the standard output of FORMAT.

l or listing :
(optional) name of file to receive the listing of the selected pages after the comparison. If you don't code the l parameter, GENREVP uses file listing.

s or summary :
(optional) name of file to receive the summary of the changes, and a "how to update the document" description. If you don't code the s parameter, GENREVP uses file summary.

o or output :
(optional) name of file to receive the OUTPUT of GENREVP. The output file contains error messages and statistics. If you don't code the o parameter, GENREVP attempts to use PROFILE variable output, and if such a variable doesn't exist, file output is used.

folio :
this (optional) key indicates that the FOLIO line of the documents is not to be compared for the purposes of generating the revision package.

GENREVP Directives

Directives to GENREVP (specified via the how parameter) are of the following form (one per line) :

L1..L2 C1..C2 comments

L1..L2 specifies a range of lines (if you only want to indicate a single line, the ..L2 can be omitted; and if you want to indicate all lines, use an *); and C1..C2 specifies a range of columns (if you only want to indicate a single column, the ..C2 can be omitted; and if you want to indicate all columns, use an *). Any comments present on a directive line are ignored. The directives are used to tell GENREVP what parts of what lines are to be ignored when comparing pages of formatted documents for differences. The standard set of directives is :

2	60..79	page number
3..4	60..79	date
*	80..136	formatting codes (junk)
60	*	folio line

The last directive is generated only if the folio parameter is given.

Examples of GENREVP Usage

```
ses.genrevp thisdoc, thatdoc, folio
REVERT.    END GENREVP  THISDOC,THATDOC --> LISTING,SUMMARY
```

This example of GENREVP compares the two document listings **thisdoc** and **thatdoc** to generate a listing of the changed pages on the file **listing**, and a summary of the changes on the file **summary**. The **folio** key indicates that the folio line is also to be ignored.

4.0 DOCUMENT FORMATTING SYSTEM
4.12 GENREVP - GENERATE A REVISION PACKAGE FOR A DOCUMENT

```
ses.genrevp grab, hold, how=('3..4 60..79', '* 80..999') folio
REVERT.    END GENREVP  GRAB,HOLD --> LISTING,SUMMARY
```

This example shows the how parameter of GENREVP used to indicate that character positions 60 through 79 on lines 3 through 4 are to be ignored for the purposes of comparison, and that character positions 80 through 999 on all (*) lines are to be ignored. In addition, the folio key indicates that the folio line is also to be ignored.

5.0 SOURCE TEXT MAINTENANCE

5.0 SOURCE TEXT MAINTENANCE

SES provides a comprehensive set of procedures to manage source text libraries. The source text maintenance package of SES is based on a special version of the NOS utility MODIFY called **MADIFY**. MADIFY differs from the standard version in that it caters to the NOS 6/12 character set; and it supports nested and conditional calls to common decks (see the description of the GENCOMP procedure for details on the CALL directives available to the MADIFY user).

It's assumed that if you're using these source text maintenance procedures, that you're moderately familiar with the concepts of MODIFY, including things such as decks, common decks, correction idents and so on. Before giving a short summary of the source text maintenance procedures, we'll introduce some of the terminology that SES uses.

BASE is a library of source text. All the source text maintenance procedures use a default name of BASE. You can establish a default base in your profile by coding

```
\ base = 'whatever_your_base_is'
```

MODULE is a single record of source text on a base. A module may be a COMMON module (COM or C or OPLC), or a module may be a REGULAR module (REG or R or OPL)

GROUP is a file containing text in the form of MODIFY "source" records. That is, each logical record in the file has its name as the first line of the record, and if the record is a common deck, the second line of the record is the word COMMON.

5.0 SOURCE TEXT MAINTENANCE

Summary Of Source Text Maintenance Procedures

GETMOD(S) GET MODuleS out of a base library onto a group file. The modules appear on the group file in MODIFY "source" format as one module per logical record.

REPMOD(S) REPlace or add MODuleS on a base library. You also use REPMOD(S) in a special way to create a brand new base library from scratch.

COLLECT collect a group of modules and accumulate them on a group file.

GENCOMP GENerate a COMPile file for a module or a list of modules.

GENCOR(S) GENerate a CORrection Set for a module.

TEMPCOR Make TEMPorary CORrections to a base without actually creating a new base.

MODIFY MODIFY a base library with a correction set file creating a new base library.

CATBASE CATalogs a BASE library to produce a short list of the names and types of modules contained in a base library.

LISTMOD LIST MODules. LISTMOD generates a cross reference listing of all the modules on a base (using the XREFMOD procedure described below), and optionally a source listing of all the modules or all the common modules on the base.

SORTMOD SORT a MODify library. Modules appear on the sorted base in alphabetical order by deck name (the deck type is not used in the sort process).

WIPEMOD WIPE (delete) MODules from a base.

5.0 SOURCE TEXT MAINTENANCE

XREFMOD Produces a cross (X) REFerence of MODules in a source library. For each module is listed the modules it calls (directly or indirectly) and the modules that call it (directly or indirectly). The LISTMOD procedure, described above, can be used to run XREFMOD as a batch job.

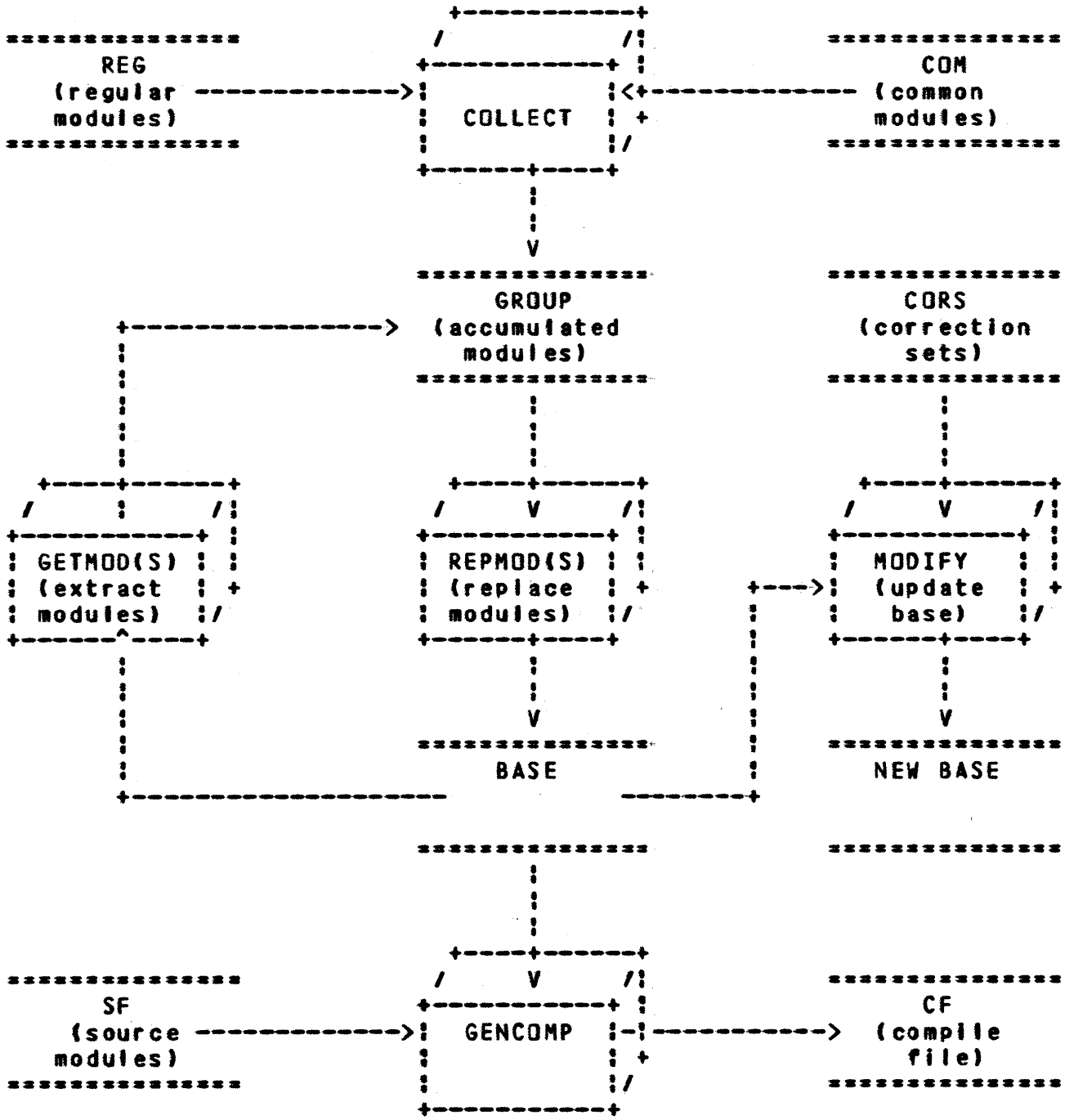
GETCOMN ACQUIRES the CYBCCMN or CYBICMN common deck program library, in the MADIFY or SCU form, and makes it local to your control point.

SYSTEMX a MADIFY based CYBIL system cross reference facility.

SCOOP Compares MADIFY or UPDATE source files.

5.0 SOURCE TEXT MAINTENANCE

Data_flow_of_the_source_text_maintenance_procedures



5.0 SOURCE TEXT MAINTENANCE

Parameter Naming Convention for Source Text Maintenance

This section introduces the parameter naming convention common to most of the procedures in the source text maintenance system.

b or l	name of Base Library on which the operations are to be performed.
c or com or oplc	list of name(s) of COMmon module(s).
r or reg or opl	list of name(s) of REGular module(s).
m	stands for Module(s), and is used in those applications where the module names do not have to be explicitly specified as common or regular.
g or group	name of file upon which a GROUP of modules are accumulated when adding or replacing modules on a base, or when modules are extracted from a base. A group file contains one module per logical record, and may be edited using procedure MULTED, and also may be processed by procedures PACK and UNPACK.
cors or c	name of CORrection Set(s) to be applied against a base for maintaining correction histories.
nb or nl	name of a New Base or New Library, used in those procedures that perform operations that permanently alter a base, requiring that a new base be created. An explanation of the method used to update a base is contained in a subsequent section.

5.0 SOURCE TEXT MAINTENANCE

PROFILE Variables for Source Text Maintenance

The variables described below may be declared in your PROFILE in order to establish default information for the parameters of the source text maintenance procedures.

base name of base library where modules are to be found or replaced.

baseown user name in whose catalog the base is to be found.

newbase name of new base library for those procedures which update a base library, thereby creating a new one. The default name for newbase is the same name as the old base.

cors used by all procedures as name of file containing CORrection Sets to be applied to a base. The default name for this file is **cors**.

group used by all procedures to designate a group file, or a file containing modules in source format, one module per logical record. The default name for group is **group**.

cfseq used by GENCOMP to indicate whether to place sequence numbers on the compile file. If you wish to use this, declare it as **seq**.

cfwidth used by GENCOMP to specify the width of the compile file. The default value for this is 110.

sestmpb used by all of the procedures that update bases to specify the temporary base file during the update process. The default name for this file is **sestmpb**.

lokmode used by all of the procedures that update bases, to determine the default interlock action. lokmode may be set to one of 'LOCK' or 'NOLOCK'. See also the sections below on updating bases and interlocking.

5.0 SOURCE TEXT MAINTENANCE

intrlok used by all of the procedures that update bases or libraries to specify the interlock file to be used during the update process. The default name for this file is **intrlok**.

5.0 SOURCE TEXT MAINTENANCE

Base Updating Process

Some of the source text maintenance procedures physically alter a base, requiring that a new base be created. The process is common to all such procedures, and is as follows :

If you code both the b and nb parameters, the new base is created simply by performing the required operations on the old base, and writing the newly created base onto the file specified by the nb parameter.

If you code only the b parameter, the update takes place in two stages. The updating operation is performed and the new base written to a file called sestmpb (SES TeMPorary Base). sestmpb is then physically copied back over the file specified by the b parameter, and finally sestmpb is purged.

Note that if the file specified by the nb (or b if nb is omitted) does not already exist, a DIRECT access READ mode file is DEFINEd for the base (this, of course, can only be done if the owner of the base is the current user).

Note that if anything goes wrong during an updating run, the original base is always left intact, and only the new base or sestmpb is potentially incorrect. This means that you can recover by purging the new base or sestmpb and starting the run again. If something goes wrong before the base appears on sestmpb, the original base is intact, and sestmpb may be purged, whereas if something goes wrong after the new base is on sestmpb, that is, during the "rewrite" phase of the update, it is sestmpb that is safe, and the old base that is wrong. In this case, you may use the REWRITE procedure to do the rewrite again, or alternatively, you may just purge the old base, then change the name of sestmpb to be the base name.

5.0 SOURCE TEXT MAINTENANCE

Interlock Process for Updating a Base or Library

When updating a base or library you can interlock the update so that only one user at a time can update the base or library. Those procedures that update a base or library are set up so that if the base or library to be updated is in another user's catalog, the base or library is interlocked by default. You can override default actions by defining a **lokmode** variable in your profile, or by coding a **lock** or **nolock** key on the procedure, as shown in the diagram on the next page.

Interlocking of a base or library is done via an interlock file. Such a file **must** be a **DIRECT** access file in the same user's catalog as the base or library being updated. Naturally the interlock file must be a PUBLIC, WRITE MODE file if other users are likely to be using it. The default name used by the SES source code maintenance and library management procedures for the lock file is **intrlok**. You can have an interlock file of any name by defining the **intrlok** profile variable, or by coding a file name as a value for the **lock** parameter on the appropriate procedures.

Coded On Procedure	Action Taken
lock	library or base is locked regardless of contents of lokmode profile variable or other conditions.
nolock	library or base is not locked regardless of contents of lokmode profile variable or other conditions.

Nothing	Coded in Profile	Action Taken
	\ lokmode='LOCK'	library or base is locked unless overridden by nolock parameter on procedure.
	\ lokmode='NOLOCK'	library or base is not locked unless overridden by lock parameter on procedure.

Nothing	Owner Of Base Or Library	Action Taken
	Current User	library or base is not locked unless the lokmode profile variable is set to 'LOCK' or the lock parameter is coded on the procedure.
	Another User	library or base is locked unless the lokmode profile variable is set to 'NOLOCK' or the nolock parameter is coded on the procedure

5.0 SOURCE TEXT MAINTENANCE
5.1 GETMOD OR GETMODS - EXTRACT MODULE GROUP FROM BASE

5.1 GETMOD OR GETMODS - EXTRACT MODULE GROUP FROM BASE

GETMOD and GETMODS are synonyms for a procedure designed to extract a designated group of modules (or all modules) from a MADIFY library, and place the extracted modules on a MADIFY "source" format file, that is, each extracted module occupies one logical record on the "source" file. Parameters to GETMOD(S) are :

m or all :
Coding the **all** key indicates that ALL modules are to be extracted from the specified base. The m parameter refers to a module or list of modules to be extracted. Ranges of modules may also be specified.

g or group :
(optional) name of file to receive the modules extracted from the base. If you don't code the g parameter, GETMOD(S) uses the value of profile variable **group** as the group file name, and if there's no such variable defined, GETMOD(S) uses the name **group** as the name of the group file. See the note on GROUP files at the beginning of this section.

b or l :
(optional) name of Base Library from which modules are to be extracted. If you don't code the b parameter, GETMOD(S) uses the value of profile variable **base** as the name of the base library, and if there's no such variable defined, GETMOD(S) uses the name **base**.

un :
(optional) User Name in whose catalog the base specified by b is to be found, if b is not in the catalog of the current user. If you don't code the un parameter, GETMOD(S) uses the value of profile variable **baseown** as the user name from whose catalog the base is to be obtained, and if there's no such variable defined, GETMOD(S) uses the current user's catalog.

cors or c :
(optional) is a file containing a CORrection Set to be applied against the base b before the module m is

extracted. If you code this parameter, SES procedure TEMPCOR is used to apply a TEMPorary CORrection set. Note that TEMPCOR doesn't actually alter the base, the correction set is only applied for the duration of the GETMOD(S) run.

w or width :
(optional) maximum Width of modules retrieved.

status or sts :
these (optional) keys can be used for those cases where GETMOD(S) is being used as a building block of more sophisticated procedures or jobs. The **status** keys cause GETMOD(S) to set the NOS job control register EFG to the value zero if GETMOD(S) successfully completed, and to non zero if anything went wrong during the run of GETMOD(S). If **status** is not specified, the procedure will EXIT if an error occurs.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of GETMOD(S) Usage

```
ses.getmods (outlind, limitsd..oprtypk, ststypk, symentd..symk)
REVERT.    END GETMODS  GROUP <- BASE
```

This example shows GETMODS used to extract a bunch of modules from a base. Modules **outlind**, all modules **limitsd** through **oprtypk**, module **ststypk**, and all modules **symentd** through **symk** are extracted. Every parameter to GETMODS is set to its default value. The modules appear on the file **group**.

5.0 SOURCE TEXT MAINTENANCE
5.1 GETMOD OR GETMODS - EXTRACT MODULE GROUP FROM BASE

```
ses.getmods all b=cidbase un=dt73 g=junk
REVERT.    END GETMODS  JUNK <- CIDBASE
```

This example extracts ALL modules from base cidbase in the catalog of user dt73 onto a group file junk

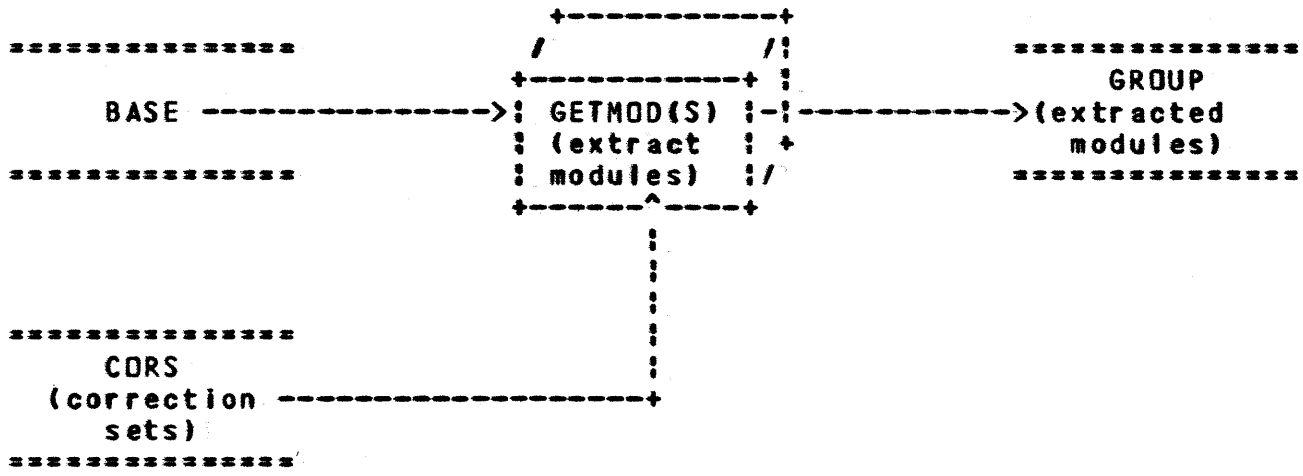
Note : A possible method of using this procedure is to extract a group of modules using GETMODS, to edit those modules using MULTED, and finally to replace the modules in the base library using REPMOD. For example :

```
ses.getmods all b=cidbase; multed; repmods b=cidbase
```

The example gets all modules from base cidbase, uses procedure MULTED to edit the group file (MULTED's default input file name is group), then replaces all the modules on cidbase again.

5.0 SOURCE TEXT MAINTENANCE
5.1 GETMOD OR GETMODS - EXTRACT MODULE GROUP FROM BASE

Data_flow_of_GETMOD(S)_usage



GETMOD(S) is a fairly simple process : a list of modules is extracted from the base library, and appear on file group in MADIFY source form, that is, one module per logical record. An optional correction set (the cors parameter) may be applied temporarily to the base before the modules are extracted. Procedure TEMPCOR applies the temporary correction sets.

5.2 REPMOD OR REPMODS - ADD OR REPLACE MODULES

REPMOD and REPMODS are synonyms for a procedure for adding or replacing module(s) on a base. Parameters to REPMOD(S) are :

m :

(optional) list of Modules to be added to or replaced on the base. If you don't code the m parameter, REPMOD(S) assumes that the module(s) to be added or replaced are on the group file specified by the g parameter. Note that each file specified by the m parameter is treated as a MADIFY source file (or "group" file) with exactly one module on it. See the note on GROUP files at the beginning of this section.

g or group :

(optional) name of the group file to be used to accumulate the modules. If you don't code the g parameter, REPMOD(S) uses the value of profile variable group as the group file name, and if there's no such variable defined, REPMOD(S) uses the name group for the group file. See the note on GROUP files at the beginning of this section.

b or l :

(optional) name of Base Library to be updated. If you don't code the b parameter, REPMOD(S) uses the value of profile variable base as the name of the base library, and if there's no such variable defined, REPMOD(S) uses the name base.

nb or nl :

(optional) name of the New Base to receive the updated library. If you don't code the nb parameter, REPMOD(S) uses the value of profile variable newbase as the name of the new base library, and if there's no such variable defined, REPMOD(S) uses the name of the base library specified by the b parameter.

un :

(optional) User Name in whose catalog the base specified by b/nb is to be found, if b/nb is not in the catalog of the current user. If you don't code the un parameter, REPMOD(S) uses the value of profile variable baseown as the user name from whose catalog the base is to be

obtained, and if there's no such variable defined, REPMOD(S) uses the current user's catalog.

w or width or inwidth :
(optional) maximum Width of the input modules.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of REPMOD(S) Usage

```
ses.repmod (hertz, maxwell, marconi), b=radio, nomsg
REVERT.    END REPMOD  GROUP -> RADIO
```

This example shows three modules being collected onto a group file group (the default), replaced on the base called radio. Since the nomsg key is coded, no informative messages are output during the run of REPMOD.

```
ses.repmods (grab, hold) b=zilch g=temp w=80 lock msg
*    GRAB -> TEMP
*    HOLD -> TEMP
*    LOCKING ZILCH VIA INTRLOK
*    ZILCH LOCKED
*    REPLACING/ADDING MODULES ON ZILCH
*    NEW BASE ON SESTMPB
*    NEW BASE NOW ON ZILCH
*    SESTMPB PURGED
*    BASE ZILCH UNLOCKED
REVERT.    END REPMODS  TEMP -> ZILCH
```

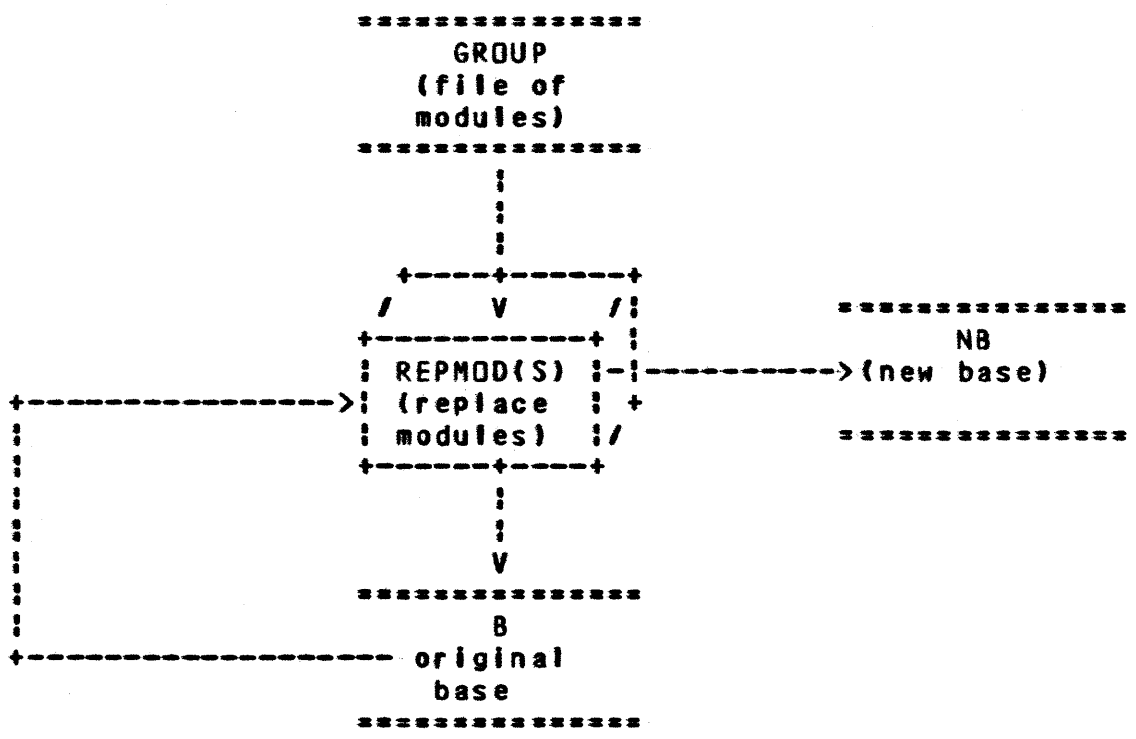
This example shows two modules being replaced on the base zilch through a group file temp. The width of the input lines is limited to 80 (ASCII) characters (because of the w parameter). Because the lock key was coded, the base is locked during the run using file intrlok (the default). Informative messages are output during the REPMODS run because the msg key was coded.

Creating a New Base with REPMOD(S)

If you are creating a brand new base from scratch, you use REPMOD(S) slightly differently from the normal case, in that you code a b (base) parameter and an nb (new base) parameter, but the b parameter refers to a non existent base, as in this example :

```
ses.repmods (york,hunt) b=nonsuch nb=newstuf
*     YORK -> GROUP
*     HUNT -> GROUP
*     CREATING NEW BASE  NEWSTUF
*     NEW BASE ON NEWSTUF
REVERT.     END REPMODS  GROUP -> NEWSTUF
```

Data_flow_of_REPMOD(S)_process.



REPMOD(S) takes modules from a group file (which, as shown in the diagram, could well have come from the old base in the first place), and replaces (or adds) them to produce either a completely new base, or rewrites the result over the old base.

5.3 COLLECT - COLLECT MODULE(S) TO BUILD A GROUP FILE

COLLECT is useful in conjunction with source text maintenance procedures such as REPMOD(S). COLLECT builds a group file by COLLECTing modules of a specified type, adding in the module name and COMMON lines, and copying the modules to a group file. Also, COLLECT can accumulate members of different types onto the group file, and so may be used in conjunction with the library management procedures described in another section. Parameters to COLLECT are :

m or mem or mems or member or members :
(optional) list of Members to be accumulated onto the group file. When used in conjunction with other Source Code Maintenance procedures, COLLECT assumes that files specified with the m parameter already have the deck name as the first line of the file (and COMMON as the second line for a common deck). **Note** : if the file specified by the m parameter is a multi record file, COLLECT does gather all records in the file. This capability is not available with the c and r parameters as it is not truly applicable in those cases.

c or com or opic :
(optional) list of COMmon modules to be accumulated onto the group file. COLLECT assumes that such modules are straight text files, and so COLLECT places the module name and COMMON lines at the start of the module.

r or reg or opl :
(optional) list of REGular modules to be accumulated onto the group file. COLLECT assumes that such modules are straight text files, and so COLLECT places the module name line at the start of the module.

g or group :
(optional) name of group file onto which the member(s) or module(s) are to be accumulated. If you don't code the g parameter, COLLECT uses the value of profile variable group, and if there's no such variable defined in your profile, COLLECT uses the filename group.

msg or nomsg :
these (optional) keys control the generation of

informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

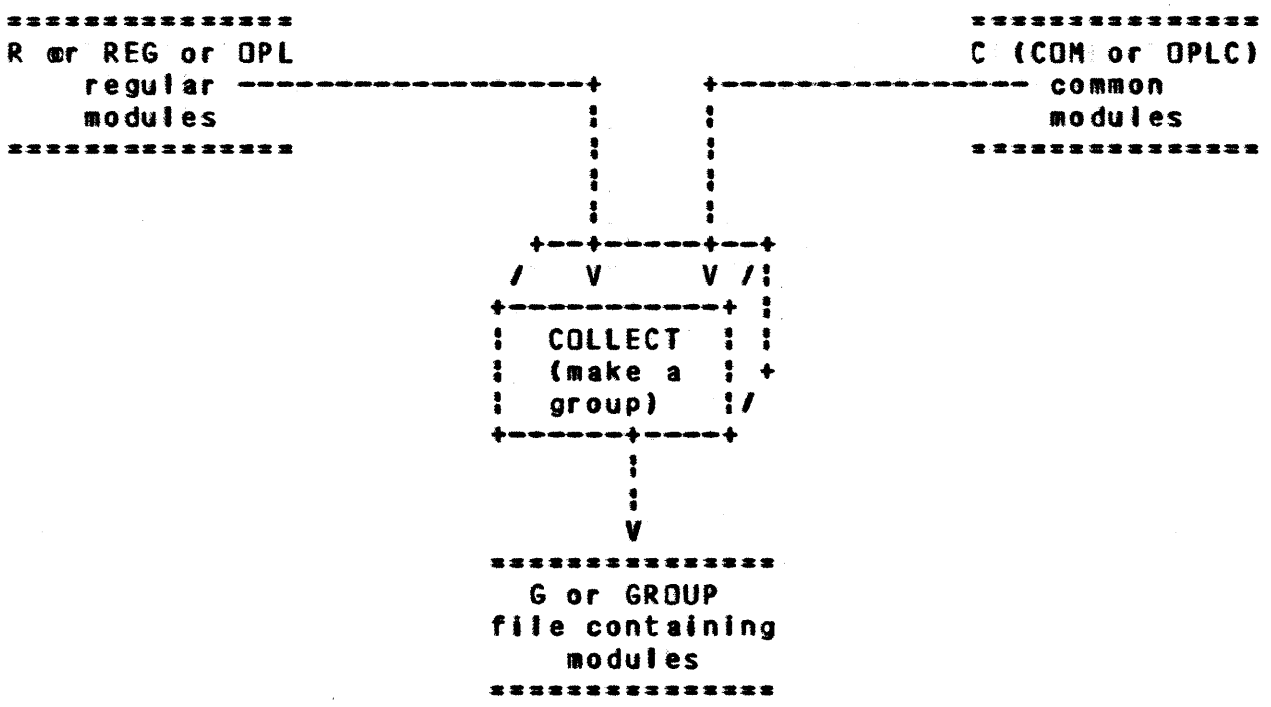
Examples of COLLECT Usage

```
ses.collect c=(cdc, icl, ibm), r=(sdc, ifs, csc), g=acronym
*     CDC/COM -> ACRONYM
*     ICL/COM -> ACRONYM
*     IBM/COM -> ACRONYM
*     SDC/REG -> ACRONYM
*     IFS/REG -> ACRONYM
*     CSC/REG -> ACRONYM
REVERT.     END COLLECT  ACRONYM
```

This example shows six straight text files being collected onto a group file acronym. The first three are specified as common modules (the c parameter), so COLLECT adds the module name and COMMON lines at the front of the file before adding it to the group file. The last three are specified as regular modules (the r parameter), so COLLECT adds the module name line at the start of each one before adding it to the group file.

5.0 SOURCE TEXT MAINTENANCE
5.3 COLLECT - COLLECT MODULE(S) TO BUILD A GROUP FILE

Data_flow_of_COLLECT_processing



As shown in the diagram, COLLECT takes files designated as either regular or common modules, and gathers them onto a group file. COLLECT can also just gather up records which already have header information in them (the m parameter).

5.4 GENCOMP - GENERATE COMPILE FILE FOR MODULE(S)

GENCOMP generates a compile file for a module or a group of modules. Only regular modules can be GENCOMPed. The GENCOMP process resolves calls to common modules (including nested calls) according to the following "compile file" directives :

***CALL** module
This directive unconditionally calls the specified common module.

***CALLC** module
This directive only calls the specified common module if it has not already been called within the current regular module. The information about which common modules have been called is cleared at the end of processing each regular module.

***IFCALL** name module
This directive only calls the specified common module if the specified name has been defined (see the define parameter described below).

***NIFCALL** name module
This directive only calls the specified common module if the specified name has not been defined (see the define parameter described below).

GENCOMP permits the specification of a list of alternate bases from which to resolve common deck calls (or in which to find modules to be GENCOMPed). Also, a source file may be specified which contains source for modules (or common modules) to be used in the GENCOMP process. In addition, correction sets may be applied prior to the compile file being generated.

When generating the compile file, MODIFY searches for decks in the following order :

- If the SF parameter is coded, the specified file is searched.
- If the AB parameter is coded, the specified files are searched from right to left.
- If the CYBCCMN or CYBICMN parameter is coded, the designated common deck base is searched.
- Finally, the base specified by the B parameter is searched.

Parameters to GENCOMP are :

5.0 SOURCE TEXT MAINTENANCE
5.4 GENCOMP - GENERATE COMPILE FILE FOR MODULE(S)

m or all :

This (optional) parameter specifies a list of modules (including ranges of modules) for which the compile file is to be generated. If the all key is coded, a compile file for ALL of the (regular) modules in the specified base(s) are produced. If you don't code the m parameter at all, you must code the sf parameter (see below).

cf :

(optional) name of the Compile File to be generated. If you don't code the cf parameter, GENCOMP writes the compile file to a file called compile.

b or l :

(optional) name of Base Library from which to generate the compile file. If you don't code the b parameter, GENCOMP uses the value of profile variable base as the name of the base library, and if there's no such variable defined, GENCOMP uses the name base.

un :

(optional) User Name in whose catalog the base specified by b may be found. If you don't code the un parameter, GENCOMP uses the value of profile variable baseown as the name of the base library, and if there's no such variable defined, GENCOMP assumes that the base is in the current user's catalog.

ab or al :

(optional) list of Alternate Bases from which to satisfy calls to common modules. The ab parameter may be coded as a multi valued list of sublists. Each element of the value list is either a single name, in which case it refers to the name of an alternate base already assigned to the job, or in the current user's catalog, or an element is a sublist, in which case, the last sub element in the sublist is a user name in whose catalog the bases referred to by the other sub elements of the sublist may be found. The example at the end of this description should (with luck) make this clear.

sf :

(optional) name of a Source File containing source text for one or more modules or common modules that are to be used in place of or in addition to the modules on the

specified base(s). If the sf parameter is given, the file it specifies is the object of a *CREATE directive to MADIFY. Note : that if you don't code the m parameter (described above), GENCOMP assumes that the file specified by the sf parameter contains one (regular) module and that, unlike the normal case for a MADIFY source module, its first line is not the module name but is the first line of data in the module (in this case a default module name of SESOPT is used). If you don't code the sf parameter, you must code the m parameter.

cors or c :
(optional) name of file containing CORrection Sets to be temporarily applied against the base b prior to generating the compile file. If you code the cors parameter, GENCOMP uses procedure TEMPCOR to make the temporary corrections before generating the compile file.

seq or noseq :
(optional) The default for GENCOMP is NO SEQence numbers on the compile file, and a compile file width of 100 (ASCII) characters. You can code noseq=value to change the width, or you can code seq to get SEQuence numbers at a width of 100 characters, or you can code seq=value to get SEQuence numbers at compile file width of value characters. You control the sequence numbers and width by defining the profile variables cfseq and cfwidth.

define or def :
this (optional) parameter specifies a name or list of names to be *DEFINEd for reference by the *IFCALL and *NIFCALL "compile file" directives described above.

status or sts :
these (optional) keys are used for those cases where GENCOMP is being used as a building block of more sophisticated procedures or jobs. The status key causes GENCOMP to set the NOS job control register EFG to the value zero if GENCOMP successfully completed, and non zero if anything went wrong during the run of GENCOMP. If status is not specified, the procedure will EXIT if an error occurs.

cybccmn or cybicmn :
(optional) key specifies an of SES supplied Alternate Base

for use in satisfying calls to common modules (see the **ab** parameter described above). If you code the **cybccmn** key, the base containing CoMmoN modules for use by CYBIL CC programs is selected. If you code the **cybicmn** key, the base containing CoMmon modules for use by CYBIL CI programs is selected. By default, no base is selected for use in generating the compile file.

nest or **nonest** :
these (optional) keys control whether nested calls to common modules are processed or ignored. If you omit this parameter or code the **nest** key, GENCOMP processes nested calls to common decks. If you code the **nonest** key GENCOMP treats nested calls as ordinary lines of text.

msg or **nomsg** :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

nodrop :
if this (optional) key is coded, it specifies that an "unknown" common deck (one that couldn't be found during the compile file generation) is not considered to be an error, otherwise it is. Note : other errors are also ignored if **nodrop** is given, so use this feature at your own risk.

Examples of GENCOMP Usage

```
ses.gencomp nexlinr
*    GENERATING COMPILE FILE NEXLINR
REVERT.    END GENCOMP  COMPILE <- OURBASE
```

This example shows the simplest use of GENCOMP, generating a compile file for a module nexlinr. Since the m parameter of GENCOMP is the first positional parameter, we can omit the m keyword. There is no base specified in the example since there is a directive like :

```
\ base = 'OURBASE'
```

in our profile. When GENCOMP has finished, the generated compile file appears on the file compile. The next example is more complicated, it shows the use of multiple modules and alternate bases used to generate the compile file.

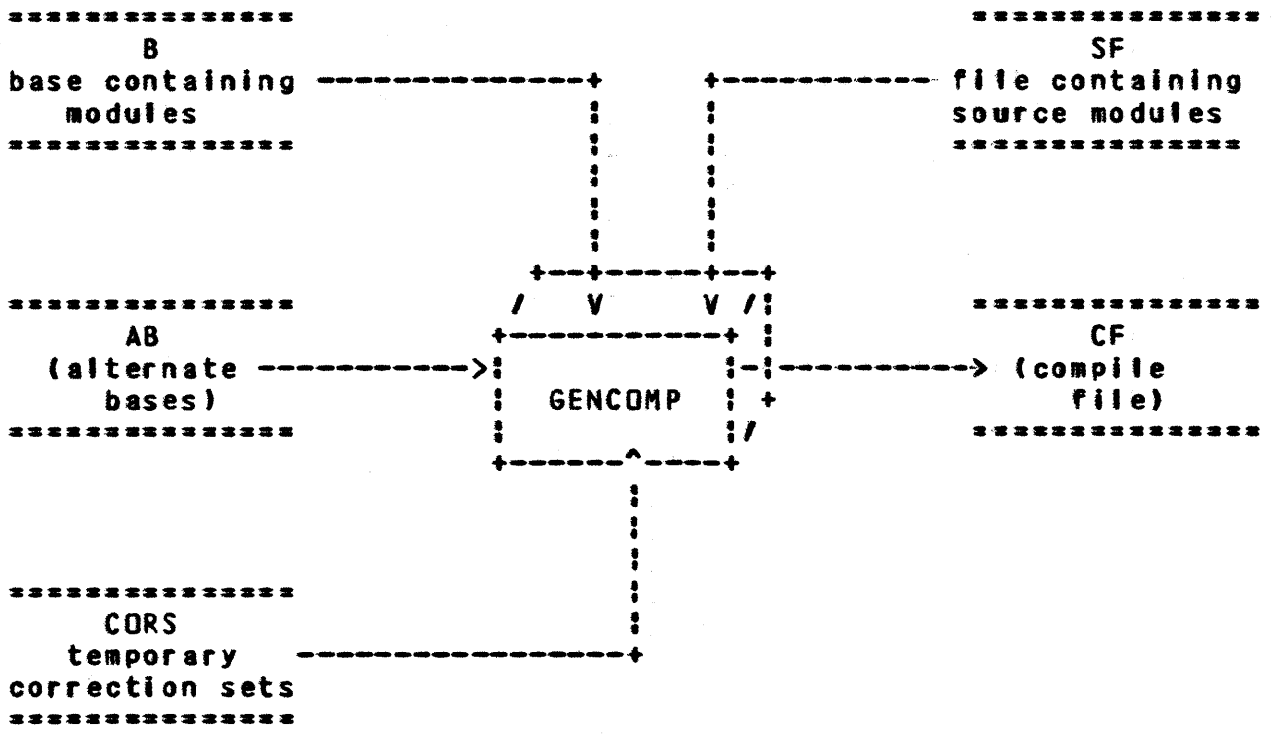
```
ses.gencomp (asterix, obelix..getafix), b=gaul, ab=(hasp,....
..? (grasp, cics, power, lr69), milos)
*    GENERATING COMPILE FILE  COMPILE
REVERT.    END GENCOMP  COMPILE <- GAUL
```

This example generates a compile file for modules asterix, and all modules obelix through getafix in the base called gaul. Alternate bases hasp and milos are to be obtained from the current user's catalog, and alternate bases grasp, cics and power are to be obtained from the catalog of user lr69. The compile file appears on file compile. The example also illustrates the use of continuation lines, showing the multi valued ab parameter split across lines.

```
ses.getmods (special, general), b=theory; multed; ..
..? gencomp (special, general), cf=albert, sf=group, ..
..? b=theory, define=lorentz, seq=79
*    GENERATING COMPILE FILE  ALBERT
REVERT.    END GENCOMP  ALBERT <- THEORY
```

In this example modules special and general are extracted from base theory onto file group and edited. Then a compile file is generated for the edited version of the modules (via the sf parameter) with the lorentz "option" defined. The resulting file albert will have sequencing information to the right of column 79.

Data_flow_of_GENCOMP_usage



GENCOMP generates a compile file for modules which may come from off the base (the m parameter), or from a source file (the sf parameter), or from both places. The ab parameter specifies a list of alternate bases to be used to get common decks from. A temporary correction set may be applied (the cors parameter) via the TEMPCOR procedure before generating the compile file.

5.5 GENCOR OR GENCORS - GENERATE CORRECTION SETS

GENCOR and GENCORS are synonyms for a procedure that GENerates a CORrection Set for a module. Parameters to GENCOR(S) are :

m :
name of the Module within the base library b for which the correction set is to be generated. Also, m is the default name of the file (either local or in your catalog) which contains the new version of the module (see the sf parameter described below).

b or l :
(optional) name of Base containing the module specified by m for which the correction set is to be generated. If you don't code the b parameter, GENCOR(S) uses the value of profile variable base as the name of the base library, and if there's no such variable defined, GENCOR(S) uses the name base.

un :
(optional) User Name where the base library specified by b is to be found. If you don't code the un parameter, GENCOR(S) uses the value of profile variable baseown as the user name in whose catalog b is to be found, and if there's no such variable defined, GENCOR(S) assumes that the base library is in the current user's catalog.

cors or c :
(optional) is a file containing a CORrection Set to be applied against the base b before the module m is extracted. If you code this parameter, SES procedure TEMPCOR is used to apply a TEMPorary CORrection set. Note that TEMPCOR doesn't actually alter the base, the correction set is only applied for the duration of the GENCORS run.

ncors or nc :
(optional) name of file to receive the New CORrection Set. If you don't code the nc parameter, GENCOR(S) uses the name ncors for the new correction set file.

id :

(optional) IDent name for the GENCORS *IDENT directive in the generated correction set. id can be any character string that forms a valid MODIFY correction identifier. If you don't code the id parameter, GENCORS generates a unique correction set identifier.

sf :

this (optional) parameter specifies the name of the file containing the new version of the module. If you don't code the sf parameter, GENCOR(S) uses the name specified by the m parameter as the Source File name.

ls or ignorls :

these (optional) keys specify whether or not to IGNORe Leading Spaces on lines being compared. The default action is to recognize leading spaces (the ls option). If you code the ignorls key, GENCOR(S) ignores leading spaces on text lines.

fl :

(optional) parameter to increase the field length. If you don't code the fl parameter, GENCOR(S) defaults to a field length of 100K.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

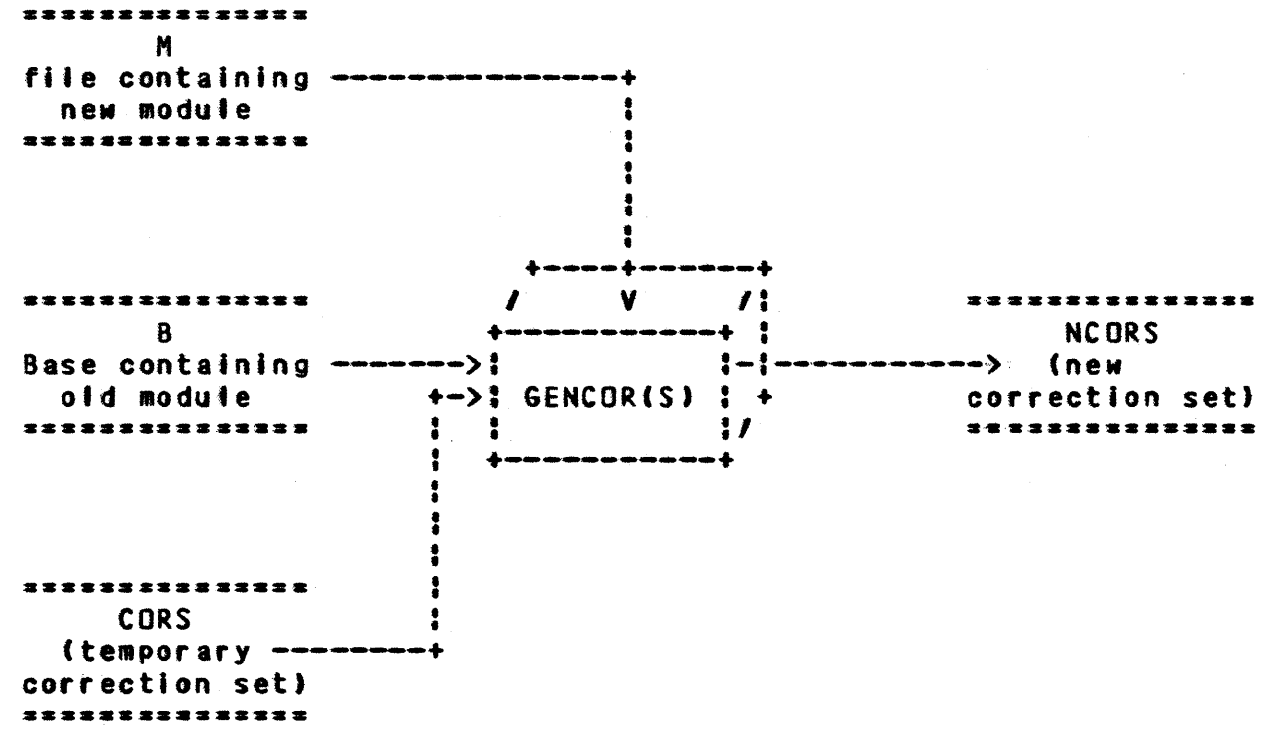
Example of GENCOR(S) Usage

```
ses.gencors m=nice, b=neat, un=by73, nc=corset
*    GENERATING CORRECTION SET FOR NICE ON CORSET
REVERT.    END GENCORS  NICE -> CORSET
```

This example shows a correction set being generated for module nice. The original module is in base neat in the catalog of user by73.

5.0 SOURCE TEXT MAINTENANCE
5.5 GENCOR OR GENCORS - GENERATE CORRECTION SETS

Data_flow_of_GENCOR(S)



The picture shows the flow of GENCOR(S). The file containing the new module M is in the user's catalog. The old version of the module is in the base B. An optional temporary correction set (CORS) may be applied if desired, in which case GENCOR(S) uses the TEMPCOR procedure to apply the correction set before generating the new correction set, which appears on the file specified by the ncors parameter.

5.6 TEMPCOR - MAKE TEMPORARY CORRECTIONS TO BASE

TEMPCOR makes temporary corrections to a base library, without permanently updating the library, providing checkout of the corrections applied before permanent modifications are made. Parameters to TEMPCOR are :

b or l :
name of Base Library against which the corrections are to be applied. If you don't code the b parameter, TEMPCOR uses the value of profile variable base as the name of the base library, and if there's no such variable defined, TEMPCOR uses the name base.

un :
(optional) User Name in whose catalog the base library specified by b may be found. If you don't code the un parameter, TEMPCOR uses the value of profile variable baseown as the user name where b may be found, and if there's no such variable defined, TEMPCOR uses the current user's catalog.

cors or c :
name of file containing the CORrectionS to be applied against b.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Example of TEMPCOR Usage

```
ses.tempcor b=hisbase, un=nr06, c=corsets
*    CREATING TEMPORARY BASE
*    TEMPORARY BASE NOW BEING USED
REVERT.    END TEMPCOR  CORSETS -> HISBASE
```

This example shows the correction set **corsets** being applied to the base **hisbase**. The base **hisbase** is left local to the job, such that further references to **hisbase** actually use a temporary copy, and not the original.

5.7 MODIFY - UPDATE BASE WITH CORRECTION SET(S)

MODIFY is intended to take a correction set file and update a base library with the correction sets contained therein. Parameters to MODIFY are :

cors or c or i :
name of file containing the CORrection Set(s) to be applied against the base specified by the b parameter. If you don't code the cors parameter, MODIFY uses the name cors.

b or i :
(optional) name of Base to be updated from the correction set file. If you don't code the b parameter, MODIFY uses the value of profile variable base as the name of the base, and if there's no such variable defined, MODIFY uses the default name base.

nb or ni :
(optional) name of New Base to be created when the update has been completed. If you don't code the nb parameter, MODIFY uses the value of profile variable newbase as the name of the new base, and if there's no such variable defined, MODIFY writes the new base over the old base specified by the b parameter.

lo :
this (optional) parameter specifies the Options to be used when producing the Listing. Each designator selects an option to a maximum of seven options. The designators must not be separated. Designators are:

a	list active lines in deck
c	list directives other than INSERT, DELETE, RESTORE, MODNAME, I, or D
d	list deck status
e	list errors
i	list inactive lines in deck
m	list modifications performed
s	include statistics on listing
t	list text input
w	list compile file directives

un :

(optional) User Name in whose catalog the base specified by **b/nb** is to be found, if **b/nb** is not in the catalog of the current user. If you don't code the **un** parameter, MODIFY uses the value of profile variable **baseown** as the user name from whose catalog the base is to be obtained, and if there's no such variable defined, MODIFY uses the current user's catalog.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Example of MODIFY Usage

```
ses.modify b=dusbase c=corslib
*    APPLYING CORSLIB TO DUSBASE
*    NEW BASE ON SESTMPB
*    NEW BASE NOW ON DUSBASE
*    SESTMPB PURGED
REVERT.    END MODIFY  CORSLIB -> DUSBASE
```

This example shows a correction set file called **corslib** applied against a base library called **dusbase**.

5.8 CATBASE - PRODUCE LIST OF MODULES IN A BASE LIBRARY

CATBASE is used to display a list of the modules in a base library. CATBASE outputs the modules list in a condensed format, four modules per line, showing only the module names and their type. Parameters to CATBASE are :

b or l :
(optional) name of Base Library which is to have its module names displayed. If you don't code the b parameter, CATBASE uses the value of profile variable base as the name of the base, and if there's no such variable defined, CATBASE uses a file name of base.

un :
(optional) User Name in whose catalog the base specified by the b parameter is to be found. If you don't code the un parameter, CATBASE uses the value of profile variable baseown as the user name, and if there's no such variable defined, CATBASE uses the current user's catalog.

o :
(optional) name of file to receive the Output from CATBASE. If you don't code the o parameter, CATBASE places the module list on file output.

short or long :
these (optional) keys determine the format of the output from CATBASE. If you omit this parameter or code the short key, CATBASE produces its output in the form shown in the example below. If you code the long key, CATBASE produces its output in the same format as the NOS utility CATALOG.

Example of CATBASE Usage

```
ses.catbase b=dusbase un=vb55
 DOUSERM..OPL       DUINTRO..OPLC      DUPRINT..OPLC      DUFORMT..OPLC
 DUSCODM..OPLC      DULIBRM..OPLC      DUFILEM..OPLC      DUCOMPL..OPLC
 DUTEXTP..OPLC      DUMAILB..OPLC      DUMULTI..OPLC      DUCONVS..OPLC
 DUGOODY..OPLC      DUPROFL..OPLC      DUMODES..OPLC      DUMESGS..OPLC
 DUSYNTX..OPLC      DUAQUIR..OPLC      DUXTRAC..OPLC      DUDOMSG..OPLC
 DUUPDAT..OPLC      OPL.....OPLD
REVERT.     END CATBASE  DUSBASE
```

This example shows CATBASE output for base library **dusbase** in the catalog of user **vb55**. Regular modules are identified by the type code **OPL**, and common modules are identified by the type code **OPLC**.

5.9 LISTMOD - LIST CONTENTS OF BASE

LISTMOD performs two main functions : to show you what's in a base, and to get a printout of the modules in the base if necessary. LISTMOD first generates a cross reference of the base, showing which modules call which modules, and which modules are called by which modules (LISTMOD uses procedure XREFMOD to generate the cross reference - see the description of XREFMOD if you're interested). Lastly, LISTMOD generates a printout of all modules or common modules in the base--you can suppress this if you only want the cross reference without the whole base's contents. Parameters to LISTMOD are :

b or i :
(optional) name of Base to be processed. If you don't code the b parameter, LISTMOD uses the value of profile variable **base**, and if there's no such variable defined, LISTMOD uses the default name **base**.

o :
(optional) name of file to receive the Output from LISTMOD. If you don't code the o parameter, a scratch file is used for the output and is returned once it has been printed. If you code the o parameter and LISTMOD is run in **local** mode, the output is placed on the specified file and is **not** printed unless the **print** parameter is given.

un :
(optional) User Name in whose catalog the base specified by **b** is to be found. If you don't code the **un** parameter, LISTMOD uses the value of profile variable **baseown**, and if there's no such variable defined, LISTMOD uses the user name of the current user.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **batch** mode, but it can be run in **local** mode. The **dayfile** parameter is not used by this procedure.

print :
(optional) indicates how the output of LISTMOD is to be

printed. For the **print** parameter you may code any of the parameters to procedure PRINT. If you don't code the **print** parameter at all, LISTMOD prints one (1) copy of the output on the ASCII printer.

short or **common** :
these (optional) keys control which modules are listed following the cross reference of the base. If you omit this parameter, LISTMOD lists the source for all modules in the base. If you code the **common** key, LISTMOD lists only the common modules from the base. If you code the **short** key, no modules are listed (i.e., only the cross reference is produced).

Examples of LISTMOD Usage

```
ses.listmod  print=copies=3, short
 11.17.34. SUBMIT COMPLETE.  JOBNAME IS AVOPBUG
REVERT.     JOB LISTMOD  SUBMITTED
```

This example shows LISTMOD processing the base whose name is contained in profile variable **base**. Three copies of the **short** list (cross reference only) are produced. LISTMOD is run in **batch**, rather than interactively. Running LISTMOD in batch is the preferred mode, since it uses rather a large amount of resources.

```
ses.listmod  b=cidbase, o=cidprin,....
..? print=(copies=2, h='latest/version of/cidbase')
 10.19.57. SUBMIT COMPLETE.  JOBNAME IS ACULBXG
REVERT.     JOB LISTMOD  SUBMITTED
```

This example shows LISTMOD run in **batch** to produce two printouts of the cross reference of base **cidbase**, plus the text of all the modules in **cidbase**.

5.0 SOURCE TEXT MAINTENANCE
5.10 SORTMOD - SORT BASE INTO ALPHABETICAL ORDER

5.10 SORTMOD - SORT BASE INTO ALPHABETICAL ORDER

SORTMOD rearranges a base such that all the modules appear in the base in alphabetical order. This is a very useful facility, for when a base becomes large and has a large number of modules in it, it's easier to find modules in the base if they're in some regular order. Parameters to SORTMOD are :

b or l :
(optional) name of Base to be sorted. If you don't code the b parameter, SORTMOD uses the value of profile variable **base**, and if there's no such profile variable defined, SORTMOD uses the name base.

nb or nl :
(optional) name of New Base to be created at the end of the SORTMOD run. If you don't code the nb parameter, SORTMOD the value of profile variable **newbase**. If there's no such profile variable defined, SORTMOD writes the new base on the file specified by the b parameter.

un :
(optional) User Name in whose catalog the base specified by **b/nb** is to be found, if **b/nb** is not in the catalog of the current user. If you don't code the **un** parameter, SORTMOD uses the value of profile variable **baseown** as the user name from whose catalog the base is to be obtained, and if there's no such variable defined, SORTMOD uses the current user's catalog.

o :
(optional) name of file to receive the Output of the programs used to run SORTMOD. If you don't code the o parameter, SORTMOD sets **o=0**, or no output. This is usually the desirable default, since the sort process generates large volumes of output.

edtsort :
(optional) pair of filenames representing the input and output files for the EDT text editor. You use edtsort to EDIT the SORT output to impose your own ordering on the modules in the base.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

5.11 WIPEMOD - DELETE MODULE(S) FROM BASE

WIPEMOD is intended as a means of deleting module(s) from a base. Parameters to WIPEMOD are :

b or i :
(optional) name of Base from which module(s) are to be deleted. If you don't code the b parameter, WIPEMOD uses the value of profile variable base as the name of the base. If there's no such profile variable defined, WIPEMOD uses the name base as the name of the base.

nb or ni :
(optional) name of New Base to be created at the end of the WIPEMOD run. If you don't code the nb parameter, WIPEMOD uses the value of profile variable **newbase** as the name of the new base. If there's no such profile variable defined, WIPEMOD places the new base back over the old base specified by the b parameter.

un :
(optional) User Name in whose catalog the base specified by b/nb is to be found, if b/nb is not in the catalog of the current user. If you don't code the un parameter, WIPEMOD uses the value of profile variable **baseown** as the user name from whose catalog the base is to be obtained, and if there's no such variable defined, WIPEMOD uses the current user's catalog.

c or com or opic :
(optional) list of COMmon module(s) to be deleted.

r or reg or opl :
(optional) list of REGular or OPL module(s) to be deleted.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the lock parameter determines the name of the interlock file. Interlocking is the <u>default</u> action when the base being updated is in another user's

catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :

These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Example of WIPEMOD Usage

```
ses.wipemod mybase c=(cdecka, cdeckb) r=(main..street, comd)
*    DELETING MODULES FROM MYBASE
*    NEW BASE ON SESTMPB
*    NEW BASE NOW ON MYBASE
*    SESTMPB PURGED
REVERT.    END WIPEMOD  MYBASE
```

This example of WIPEMOD deletes common modules **cdecka** and **cdeckb** from the base **mybase**. It also deletes regular modules **main** through **street** inclusive, and regular module **comd**. The example illustrates the sort of informative messages output by WIPEMOD.

5.12 XREFMOD - CROSS REFERENCE OF A BASE

XREFMOD produces a cross reference listing of a base. (XREFMOD is primarily intended as a "building block" for other procedures. In particular, see the description of the LISTMOD procedure which runs XREFMOD as a batch job.)

Optionally, a single module (deck) name can be specified for which are produced (instead of the normal output listing) a list of the regular modules that reference (call) the module either directly or indirectly (see the description of the m parameter for more information on this feature).

In the normal (i.e., not the single module option) mode of output, XREFMOD produces for each MODule (deck) the following information :

1. the name of the deck
2. the date on which the deck was created
3. the type of deck (see deck categories described below)
4. the number of lines in the deck
5. the position of the deck in the base
6. a list of all the decks that reference (call) this deck
7. a list of all the decks that are referenced (called) by this deck

Items 2 through 5 are replaced by a "flag" if the deck is "external" to the base being cross referenced. An external deck is one which is referenced (called) by one or more decks in the base but which is not itself contained in the base. A separate list of all the external decks is produced following the main deck list.

In items 6 and 7, a deck name may be preceded by 'I' indicating an Indirect reference, and/or by '*' indicating an external deck.

All deck lists are arranged in alphabetical order. The page width and page length are selectable via parameters.

5.0 SOURCE TEXT MAINTENANCE
5.12 XREFMOD - CROSS REFERENCE OF A BASE

Modules are categorized according to the following criteria :

Category_Type	Criteria*
COMMON	common deck
MODULE	first program line is a MODULE line
IDENT	first program line is an IDENT line
PROGRAM	first program line is a PROGRAM line
SESPROC	first two characters of the first program line are '\ ', that is, the SES processor master control character followed by a space.
TXTFORM	first character of the first line is a reverse slant (\)
KCL	first character of the first line is an asterisk (*) and not immediately followed by "CALL", "CALLC", "IFCALL", or "NIFCALL".
CCL	first line is a .PROC line
UNKNOWN	(you guessed it) none of the above

* Up to five lines are examined if the first separator character

of the previous line(s) is a double quote (") or a question mark (?).

Parameters to XREFMOD are :

b or l :
(optional) name of Base Library for which the cross reference is to be generated. If you don't code the b parameter, XREFMOD uses the value of profile variable **base**, and if there's no such variable defined, XREFMOD uses the default name of base.

un :
(optional) User Name in whose catalog the base specified by the b parameter is to be found. If you don't code the un parameter, XREFMOD uses the value of profile variable **baseown** as the user name, and if there's no such variable defined, XREFMOD assumes that the base is in the current user's catalog.

o :
(optional) name of file to receive the Output from the cross reference run. If you don't code the o parameter, XREFMOD sends the output to file OUTPUT.

m or mod :
(optional) name of a module (possibly external) for which a list of the regular modules that reference (call) it either directly or indirectly are produced. Each module name is written on a separate line with a space in column 1, the module name in columns 2 through 8 (left justified, space filled, with letters in upper case) and a period (.) in column 9. The module names are written in alphabetical order.

pw or pwidth :
this (optional) parameter specifies the Page Width for the cross reference listing. If you code this parameter, you must give a value of at least 70. If you omit the pw parameter, XREFMOD uses a page width of 132.

pl or plength :
this (optional) parameter specifies the Page Length for

the cross reference listing. If you code this parameter, you must give a value of at least 20. If you omit the pl parameter, XREFMOD uses a page length of 60.

5.13 GETCOMN - ACQUIRE CYBIL COMMON DECK LIBRARY

GETCOMN acquires the latest MADIFY or SCU version of the CYBIL CC or CI common deck library as a local file named CYBCCMN or CYBICMN - parameters to GETCOMN are :

mad or scu :
this (optional) key indicates the format of the CYBIL CC or CI common deck library desired. If you code the mad key, the MADIFY formatted version of the common deck library is made local. If you code the scu key or do not code either of these keys, the SCU formatted version of the common deck library is made local.

cc or ci :
(optional) key indicates the version of the CYBIL common deck library is desired. If you code the ci key, the MADIFY or SCU formatted CYBIL CI common deck library is made local. If no key is coded, the MADIFY or SCU formatted CYBIL CC common deck library is made local.

You may also define PROFILE variables to override the default file assignments for the various forms of the CYBIL common deck library. For the CYBIL CC version the PROFILE variables should be :

```
ccmn      CYBCCMN in MADIFY format
ccmnun    owner of 'ccmn'
ccscu     CYBCCMN in SCU format
ccscun    owner of 'ccscu'
```

For the CYBIL CI version, the PROFILE variables should be :

```
icmn      CYBICMN in MADIFY format
icmnun    owner of 'icmn'
icscu     CYBICMN in SCU format
icscun    owner of 'icscu'
```

Examples of GETCOMN Usage

```
ses.getcomn
REVERT.   SCU VERSION OF CYBCCMN MADE LOCAL
```

5.0 SOURCE TEXT MAINTENANCE
5.13 GETCOMN - ACQUIRE CYBIL COMMON DECK LIBRARY

```
ses.getcomn mad ci
REVERT.   MADIFY VERSION OF CYBICMN MADE LOCAL
```

The first example shows GETCOMN acquiring the SCU version of CYBCCMN (both parameters take their defaults). The second example shows how to acquire the MADIFY version of CYBICMN.

Note : please ensure that you **RETURN** the common deck library when you're finished with it.

5.14 SYSTEMX - GLOBALLY CROSS REFERENCES A CYBIL SYSTEM

SYSTEMX (a MADIFY based CYBIL system cross reference facility) initially builds a data base containing a global cross reference of all CYBIL modules which reside on a specified set of MADIFY PL's. Then SYSTEMX can:

1) query the data base for information pertaining to a CYBIL identifier or MADIFY deck name

   or

2) output the entire data base to nine-track, high-density (GE) tape(s) in a format suitable for microfiche processing at Comsource Headquarters.

The tape(s) made from running SYSTEMX with the **fiche** parameter specified should be sent with a 'Microfilm Work Request' (form no. AA5167) to Terry Larson - HQCOOH. The 'Program ID' used on the 'Microfilm Work Request' is ARH144. A 'Master Charge No.' is also required to submit the 'Microfilm Work Request'. Please contact Kathy Beatty (482-2338) for information needed to acquire a 'Master Charge No.'.

If building the data base, the default jobmode is **batchn** and either the **build** or **restart** parameter must be given. If writing the data base to microfiche formatted tape, the default jobmode is **batchn** and the **fiche, sysname** and **vsn** parameters must be given when beginning the job, or just the **fiche** parameter when restarting the job. If neither the **build, restart** or **fiche** parameter is given, then SYSTEMX assumes the query function where the default jobmode is **local** and either the **identifier** or **deck** parameter must be given.

The Build Function

The parameters in effect are :

product,pd
build
bases,ba
packnam,pn
modules,md
restart
ph1,ph2,ph3,ph4
batch job parameters

The permanent files left in your catalog after phase4 completion are :

SYSXB1..SYSXBn where 1 ≤ N ≤ 99 equals the total number of MADIFY PL's input during the build

and

SYSXRST	SYSXPH1	SYSXDKS	SYSXDR1
SYSXDR2	SYSXSRT	SYSXMST	SYSXTMP
SYSXTXT			

Of these permanent files, only :

SYSXRST	SYSXTXT	SYSXMST	SYSXTMP
SYSXDR2	SYSXSRT	SYSXDR1	

are required as input to the query or fiche functions.

The remaining permanent files :

SYSXB1..SYSXBn	SYSXPH1	SYSXDKS

which were created during phase1 of the build should be retained only if a subset of your system was cross referenced via the product parameter and you will create another sub-system cross reference later. Then retention of these files saves having to run phase1 of the build over again.

Note that all of these files can be dumped to tape via the SES.DUMPPF procedure and then restored via the SES.LOADPF procedure. If these files are dumped from one catalog and restored into another, then text line two of file SYSXRST must be changed to reflect the new user_number.

Should the build require more than one processing session to complete (i.e., restart(s)), then some permanent files left by SYSTEMX in your catalog will begin with 'ZQ' and others with 'SYSX'. Please note that prior to beginning a SYSTEMX build, all of the above mentioned filenames beginning with 'SYSX' must be purged from your catalog.

Between restarts of the build function, users are strongly encouraged to backup on tape what has been produced up to that point. NOS could go down for the night or crash before SYSTEMX shuts itself down, in which case all has been lost if previous output wasn't backed up to tape. To determine which files to dump, simply list file SYSXRST and match file names from that list to those in your catalog. Dump those files which match and also file SYSXRST.

Phase1 Description :

5.0 SOURCE TEXT MAINTENANCE
5.14 SYSTEMX - GLOBALLY CROSS REFERENCES A CYBIL SYSTEM

Phase1 creates a new set of MADIFY based PL's which are almost identical to the original set of PL's input. The difference is that comments now bracket the text of each deck and contain information about the deck's origin.

SYSTEMX must complete phase1 and shut itself down pending a restart during one processing session. If your system is large, then eight or more hours of wall clock time should be available for processing when you begin this phase.

Phase2 Description :

Phase2 creates and sorts the initial set of files required by phase3. SYSTEMX must complete phase2 during one processing session, though the wall clock time required is very short.

Phase3 Description :

Phase3 performs the bulk of the build process. Each CYBIL module in your system is built, CYBFORMed, compiled and fed into the global cross referencing logic. SYSTEMX defaults to perform this phase in 100 module increments after which it voluntarily shuts itself down pending a subsequent restart.

The 100 module default can be changed at any restart occurring before phase3 completion to any number from 1 to 99999, depending on the amount of processing time available. Please note that the last figure selected remains in effect during subsequent phase3 restarts if a new figure isn't selected at that time via the **modules** parameter.

SYSTEMX must complete phase3 or voluntarily terminate after processing a preset number of modules. On a moderately loaded CYBER 176, SYSTEMX can process about 30 modules per hour of wall clock time. Of course, this can vary dramatically depending on the load.

Phase4 Description :

Phase4 performs the sorts and merges necessary to draw the global cross reference together. SYSTEMX must complete phase4 during one processing session. If your system is large, than eight or more hours of wall clock time should be available for processing when you begin this phase.

The_Fiche_Function

5.0 SOURCE TEXT MAINTENANCE
5.14 SYSTEMX - GLOBALLY CROSS REFERENCES A CYBIL SYSTEM

The parameters in effect are :

fiche
entries
sysname
sysversion
syslevel
vsn
batch job parameters

Output from the fiche function goes to nine-track, high-density tape or tapes containing one to many files. The important thing you as the user need do is insure that the reels are externally numbered sequentially (i.e., one, two, three, etc.) as they come off the drive. This sequencing function must continue through fiche restart(s) (i.e., do not start sequencing back at one if fiche restart(s) are necessary). Also, mention to the operators to 'blank label a tape' if necessary.

If output of the data base to tape requires more than one processing session to complete (i.e., fiche restart(s)), then SYSTEMX will leave a permanent file in your catalog called SYSXFCH. If this file exists in your catalog, then you are not finished yet. SYSTEMX defaults to process the data base in 15,000 entry increments after which it voluntarily shuts itself down pending a subsequent restart.

The 15,000 entry default can be changed at the beginning of the fiche job or at any subsequent fiche restart via the **entries** parameter to any number from 1 to 99999, depending on the amount of processing time available. Please note that the last number selected remains in effect during subsequent fiche restarts if a new number isn't selected at that time via the entries parameter.

SYSTEMX must complete the fiche job or voluntarily terminate after processing a preset number of entries. On a moderately loaded CYBER 176, SYSTEMX can process about 2000 entries per hour of wall clock time.

The microfiche themselves will have two sections. The Source Deck section will consist of a sorted set of deck reports as described under the query function. The Global Crossref section will consist of a sorted set of identifier reports as described under the query function.

Between fiche restarts users are strongly encouraged to copy file SYSXFCH to another indirect access file in their catalog. If NOS goes down for the night or crashes before SYSTEMX shuts itself down, then SYSTEMX can properly restart only if file SYSXFCH can be restored to it's condition prior to the fiche restart which failed.

If a fiche restart fails, simply restore file SYSXFCH to it's condition prior to that restart and attempt another fiche restart. Those tapes associated with the restart which failed should be re-used beginning with the lowest numbered tape. If the fiche function fails during the initial processing session, then purge file SYSXFCH and start over using the absolute lowest numbered tape.

The_Query_Function

The parameters in effect are :

identifier,id
output,o
qual#
deck
batch job parameters

Any user can query a SYSTEMX data base. Only a copy of file SYSXRST must reside in that user's catalog. Line two of SYSXRST must contain the user_number of the catalog where the entire data base actually resides and the user must be validated to read those files.

Identifier Report :

For all CYBIL identifiers, except module identifiers, a complete system-wide cross reference report is returned. For a module identifier, an unabridged compile file and standard CYBIL cross reference report is returned.

Deck Report :

Different CYBIL identifiers, each bearing the same name, can be a common occurrence within a CYBIL system. SYSTEMX qualifies each identifier and each reference to that identifier by deck_name, base_name, user_name, and deck_line_number.

The deck_line_number corresponds to the line number as the phase3 build received the deck. The comments placed around each source deck during phase1 build, and potentially the CYBFORMing process, have altered the one-to-one relationship to line numbers on the original source decks so SYSTEMX makes available a listing of each source deck as phase3 received it to restore that one-to-one relationship to the user.

Parameters to SYSTEMX are :

5.0 SOURCE TEXT MAINTENANCE
5.14 SYSTEMX - GLOBALLY CROSS REFERENCES A CYBIL SYSTEM

identifier or id :
any valid CYBIL identifier is legal (id=clm$input_stack_manager).

product or pd :
(optional) file name of a local or permanent file which contains a list of module deck names and/or module deck prefixes (if your system has such a convention) which specify to SYSTEMX a subset of your system you wish to obtain a global cross reference of. If the file name specified is both local and permanent, the local one is used. The module deck names or prefixes can be delimited by spaces or EOL only and can occur on multiple text lines in this file. Please note that the file name 'FULL' has a special meaning and is not examined, but rather a complete system-wide cross reference is produced. This parameter can be used in one of two ways. It can be specified during the initial call to SYSTEMX to begin the build, in which case the **build** parameter was used, or in conjunction with the **restart** parameter. If it is not specified at the initial build time, the default is to produce a complete system-wide cross reference. When used with the **restart** parameter, permanent files (SYSXB1..SYSXBn, SYSXPH1, and SYSXDKS) must already exist. These files are created during phase1 build. All other 'SYSX' files must be purged (i.e., SYSXRST, SYSXDR1, SYSXDR2, SYSXSRT, SYSXMST, SYSXTMP, and SYSXTXT). (pd=prodfile)

unused :
TBD

output or o :
(optional) local file name for **output** from a SYSTEMX query. If the default (SYSXOUT) is used, it is rewound before **output**. For any other file name, the **output** is appended to the file. (o=outfile)

qual# :
(optional) positive integer SYSTEMX uses to uniquely **qualify** an **identifier** which is defined more than once in your system. When querying SYSTEMX about an **identifier** the first time, skip the **qual#** parameter and if the **identifier** is multiply defined, SYSTEMX will return enough information so you can retry the query using the **qual#** parameter. (qual#=3)

5.0 SOURCE TEXT MAINTENANCE
5.14 SYSTEMX - GLOBALLY CROSS REFERENCES A CYBIL SYSTEM

deck :
any valid MADIFY deck name is legal. (deck=cimism)

build :
key parameter tells SYSTEMX to begin the build process from scratch. (build)

bases or ba :
(optional) parameter default is to use the basename 'BASE' with the username of the current user. This parameter may be coded as a multi-valued list of sublists. Each element of the value list is either a single name, in which case it is the name of a base in the current user's catalog, or the element is a sublist where the last subelement in the sublist is a username in whose catalog the bases referred to by the preceding subelements in that sublist may be found. Please note that a search order to MADIFY is implied when using this parameter. That order is from left to right. (ba=((ses11bd,ses), (pscompl,build), (oslpi,intl), (pascmn,pcgsrc8,pfesrc,ip3)))

packnam or pn :
(optional) default for this parameter is to use the current user's permanent file device for the permanent files SYSTEMX creates. For large systems, enough mass storage may not be available on the user's permanent file device, in which case two auxiliary devices can be specified (only two). This parameter is only used at initial build time or with a restart where the product parameter was also specified. This parameter may be coded as a multi-valued list of sublists. Each element of the value list is a sublist where the last subelement in the sublist is the device type of the preceding packnames in that sublist. (pn=((nve1,nve2,dl1)) or pn=((nve1,dj1),(nve2,dl1)))

modules or md :
(optional) number of modules to process during phase3 build before voluntarily shutting down pending a subsequent restart. The range can be from 1 to 99999. The default is 100 modules or the last modules specification input by the user. (md=40)

restart :
this key tells SYSTEMX to restart the build process. If

the product parameter is also specified, the build process is restarted from the beginning of phase2; otherwise, the build process is continued from wherever it left off. When restart is specified, the bases and packnam parameters are disregarded. (restart)

unused :
TBD

unused :
TBD

ph1, ph2, ph3, ph4 :
(optional) key telling SYSTEMX when to shut down pending a subsequent restart. ph1 corresponds to phase1 and so on. The default is ph4 (which means do the whole thing). Users are strongly encouraged to always use this parameter and step the build up one phase at a time. If a permanent file 'DAYFILE' doesn't appear in your catalog giving you the last 100 lines of the job's dayfile in the case of an abort, you can assume that the last build step attempted completed properly.

fiche :
this key tells SYSTEMX to begin writing the data base to microfiche formatted tape. If file SYSXFCH exists in your catalog, this key tells SYSTEMX to restart this process where it left off, and the sysname, sysversion, syslevel and vsn parameters are disregarded. (fiche)

entries :
(optional) number of entries to process during production of the fiche tape(s) before volutarily shutting down pending a subsequent fiche restart. The range can be from 1 to 99999. The default is 15,000 entries or the last entries specification input by the user. One entry is defined as a single 'deck report' or identifier report' as described under the query function. (entries=20,000)

sysname :
string parameter specifies the name of your system and can be any eight characters enclosed within single quotes. (sysname='NOS/VE')

sysversion :
(optional) string parameter specifies the version of your system and can be any four characters enclosed within single quotes. (sysversion='1.12')

syslevel :
(optional) string parameter specifies the level of your system and can be any five characters enclosed within single quotes. (syslevel='5')

vsn :
string parameter enclosed within single quotes specifies any legal NOS vsn to be used by SYSTEMX when it issues a LABEL request for the tape required for fiche processing.

batch job parameters :
these parameters are described in the section entitled SES PROCEDURES RUN AS BATCH JOBS. The default for the SYSTEMX build and fiche functions is to run in batchn mode, though they can be run in batch, defer or local modes as well. The default for the SYSTEMX query function is to run in local mode, though it can be run in batch, batchn, or defer modes as well. The dayfile parameter is used by SYSTEMX and if the job aborts, the last 100 lines of the dayfile are dumped to your catalog. Please note that use of the jobtl parameter is strongly encouraged (jobtl=77777(8)) to set the time limit to unlimited.

Examples of SYSTEMX Build Usage

```
ses.systemx build ba=((nosvepl,oslpl,dsplpl,cybicmn,int2)) ..
..? pn=((nve1,nve2,dl1)) ph1 jobtl=77777(8)
```

This example shows the build process being initiated for the Advanced System.

```
ses.systemx restart ph2 jobtl=77777(8)
```

This example shows a restart of the same job to step through phase2 of the build.

```
ses.systemx restart ph3 md=40 jobtl=7777(8)
```

This example shows a restart of the same job to step through the first 40 modules of processing in phase3 of the build.

```
ses.systemx restart ph3 md=250 jobtl=7777(8)
```

This example shows a restart of the same job to step through the next 250 modules of processing in phase3 of the build.

Note : When the first text line in file 'SYSXRST' says 'PH3' you know that phase3 is complete and the final phase can be started.

```
ses.systemx restart ph4 jobtl=7777(8)
```

This example shows a restart of the same job to step through the final phase (phase4) of the build process.

Examples of SYSTEMX Fiche Usage

```
ses.systemx fiche entries=20000 sysname='nos/ve' ..
..? sysversion='1.12' syslevel='5' vsn='tp001' jobtl=7777(8)
```

This example shows the fiche process being initiated for the Advanced System.

```
ses.systemx fiche entries=40000 jobtl=7777(8)
```

This example shows a fiche restart of the same job to output the next 40,000 entries in the data base to tape.

Examples of SYSTEMX Query Usage

```
ses.systemx id=tree_node_contents qual#=2 o=treefle
```

This example shows a query of a multiply defined identifier tree_node_contents where the output is directed to file treefle.

```
ses.systemx deck=listcmn qual#=2
```

This example shows a query of a multiply defined deck listcmn where output is defaulted to SYSXOUT.

5.15 SCOOP - COMPARES MADIFY OR UPDATE SOURCE FILES

SCOOP compares an 'old' file and a 'new' file, lists the changes that transform 'old' to 'new', and generates source code utility corrections. Any MADIFY or UPDATE source files may be compared. Parameters to SCOOP are :

o or old :
(optional) name of the old file which you are comparing to the new file. The lines in this file will appear on the left side of the listing. If you don't code this parameter, the procedure expects a default file name of LEFT.

n or new :
(optional) name of the new file. If you don't code this parameter, SCOOP expects a default file name of RIGHT.

l or list :
(optional) output listing file generated by SCOOP. If you don't specify a value for this parameter, it defaults to a file name of LISTING.

s or short or nm or nomatch :
(optional) key selecting short lists changes only (additions and deletions). Otherwise, lines that are common to both files are also listed. errcnt restricts the number of changes listed.

e or err or errs or errcnt :
(optional) number of changes to be listed. Coding a numeric value for this parameter restricts the number of changes to list. The default value is 999999.

so :
(optional) key selects SHORT, L=OUTPUT.

pch :
(optional) filename for punch output. This is the file containing MADIFY or UPDATE INSERT and DELETE directives needed to change the old file to the new file. This file will not be generated unless you specify a value for this

parameter.

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT

SES supplies a comprehensive set of procedures to manage multi record files, using the NOS LIBEDIT utility as a basis. It's assumed that you're fairly familiar with the ideas of multi record file and LIBEDIT. The terminology that SES procedure descriptions of library management use are as follows :

LIBRARY A collection of records in a file. All SES library management procedures use an l parameter to refer to a Library.

MEMBER A single record in a library file, or a file containing a member which is added to or replaced on a library. All SES library management procedures use an m parameter to refer to MEMBER(S).

GROUP a collection of member(s) that occupy one record per member on the file.

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT

Summary Of Library Management Procedures

GETMEM(S) GET MEMberS out of a library onto a group file. The members appear one per logical record.

REPMEM(S) REPlace or add MEMberS on a library. You also use REPMEM(S) in a special way to create a brand new library from scratch.

REPULIB REPlace or add members on a User Library.

COLLECT collect a group of members onto a group file.

CATLIB CATalog a LIBrary to display the names and the types of members in the library, in short form output.

LISTMEM LIST MEMbers. LISTMEM generates a catalog of all the members on a library, and optionally prints every TEXT member from the library.

SORTMEM SORT MEMbers in a library. The library is arranged so that all members of the same type appear together. Within each member type, the members are sorted into alphabetical order.

SRTULIB SoRT members in a User LIBrary.

WIPEMEM WIPE (delete) MEMbers from a library.

WIPULIB WIPe (delete) members from a User LIBrary.

LIBEDIT use the LIBEDIT utility directly.

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT

Parameter Naming Convention for Library Management

This section introduces the parameter naming convention that's common to most of the procedures in the library management system.

l or b	name of Library on which the operations are to be performed.
m	stands for Member(s).
g or group	name of file upon which a GROUP of members are accumulated when adding or replacing members on a library, or when members are extracted from a library. A group file contains one member per logical record.
nl or nb	name of a New Library or New Base, used in those procedures that perform operations that permanently alter a library, requiring that a new library be created. An explanation of the method used to update a library is contained in a subsequent section.

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT

PROFILE Variables for Library Management

The variables described below may be declared in your PROFILE in order to establish default information for the parameters of the library management procedures.

lib name of library where members are to be found or replaced.

libown user name in whose catalog the library is to be found.

newlib name of new library for those procedures which update a library, thereby creating a new one. The default name for newlib is the same name as the old library.

group used by all procedures to designate a group file, or a file containing one module per logical record. The default name for group is **group**.

sestmpl used by all of the procedures that update libraries to specify the temporary library file during the update process. The default name for this file is **sestmpl**.

lokmode determines the interlocking action when updating a library. The **lokmode** variable may be set to 'LOCK' (lock by default unless the **nolock** parameter is coded on the procedure) or 'NOLOCK' (no lock by default unless the **lock** parameter is coded on the procedure).

intrlok used by all of the procedures that update libraries or bases to specify the interlock file to be used during the update process. The default name for the interlock file is **intrlok**.

nx used by all of the procedures that update User LIBraries to specify whether a cross reference of entry points is to be included in the library. If nx is set to any non-null value, the entry point cross reference is not included by default, otherwise it is.

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT

Library Updating Process

Some of the library management procedures physically alter a library, requiring that a new library be created. The process is common to all such procedures, and is as follows :

If you code both the l and nl parameters, new library is created simply by performing the required operations on the old library, and writing the newly created library onto the file specified by the nl parameter.

If you code only the l parameter, update takes place in two stages. The updating operation is performed and the new library written to a file called sestmpl (SES TeMPorary Library). sestmpl is then physically copied back over the file specified by the l parameter, and finally sestmpl is purged.

Note that if the file specified by the nl (or l if nb is omitted) does not already exist, a DIRECT access READ mode file is DEFINEd for the library (this, of course, can only be done if the owner of the library is the current user).

Note that if anything goes wrong during an updating run, the original library is always left intact, and only the new library or sestmpl is potentially incorrect. This means that you can recover by purging the new library or sestmpl and starting the run again. If something goes wrong before the library appears on sestmpl, original library is intact, and sestmpl may be purged, whereas if something goes wrong after the new library is on sestmpl, that is, during the "rewrite" phase of the update, it is sestmpl that is safe, and the old library that is wrong. In this case, you may use the REWRITE procedure to do the rewrite again, or alternatively, you may just purge the old library and then change the name of sestmpl to be the library name.

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT

Interlock Process for Updating a Library

When updating a library you can interlock the update so that only one user at a time can update the library. Those procedures that update a library are set up so that if the library to be updated is in another user's catalog, the library is interlocked by default. You can override default actions by defining a **lokmode** variable in your profile, or by coding a **lock** or **nolock** key on the procedure, as shown in the diagram on the next page.

Interlocking of a library is done via an interlock file. Such a file **must** be a **DIRECT** access file in the same user's catalog as the library being updated. Naturally the interlock file must be a PUBLIC, WRITE MODE file if other users are likely to be using it. The default name used by the SES source code maintenance and library management procedures for the lock file is **interlok**. You can have an interlock file of any name by defining the **intrlok** profile variable, or by coding a file name as a value for the **lock** parameter on the appropriate procedures.

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT

Coded On Procedure	Action Taken
lock	library or base is locked regardless of contents of lokmode profile variable or other conditions.
nolock	library or base is not locked regardless of contents of lokmode profile variable or other conditions.

Nothing	Coded in Profile	Action Taken
	\ lokmode='LOCK'	library or base is locked unless overridden by nolock parameter on procedure.
	\ lokmode='NOLOCK'	library or base is not locked unless overridden by lock parameter on procedure.

Nothing	Owner Of Base Or Library	Action Taken
	Current User	library or base is not locked unless the lokmode profile variable is set to 'LOCK' or the lock parameter is coded on the procedure.
	Another User	library or base is locked unless the lokmode profile variable is set to 'NOLOCK' or the nolock parameter is coded on the procedure

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT
6.1 GETMEM OR GETMEMS - EXTRACT MEMBER(S) FROM LIBRARY

6.1 GETMEM OR GETMEMS - EXTRACT MEMBER(S) FROM LIBRARY

GETMEM(S) is intended for extracting any number of members from a library onto a group file, with one member per logical record. Parameters to GETMEM(S) are :

m or all :
if you code the **all** keyword, GETMEM(S) extracts **ALL** members from the library specified by the l parameter. Otherwise, you code the m parameter as a list of name(s) of Member(s) to be extracted. The list of member(s) must all be of the same type (as specified by parameter 6). Ranges of member names may also be coded, as shown in the examples at the end of this description.

g or group :
(optional) name of **group** file to receive the member(s) extracted from the library. If you don't code the g parameter, GETMEM(S) uses the value of profile variable **group**, and if there's no such variable defined, uses the name **group** for the group file.

l or b :
(optional) name of Library from which the member(s) are to extracted. If you don't code the l parameter, GETMEM(S) either uses the value of profile variable **lib**, or if that's undefined, GETMEM(S) uses the name **lib** for the name of the library.

un :
(optional) User Name in whose catalog the library specified by l is to be found, if it isn't in the catalog of the current user. If you don't code the un parameter, GETMEM(S) uses the value of profile variable **libown**, and if there's no such variable defined, the GETMEM(S) uses the user name of the current user.

status or sts :
these (optional) keys can be used for those cases where GETMEM(S) is being used as a building block of more sophisticated procedures or jobs. The **status** key causes the GETMEM(S) procedure to set the NOS job control register EFG to the value zero if GETMEM(S) successfully completed, and non zero if anything went wrong during the

run of GETMEM(S). If status is not specified, the procedure will EXIT if an error occurs.

text or opl or oplc or opld or rel or abs or ovl or pp or ppu or ulib or cap or proc :
these (optional) keys specifies the type of the members to be removed from the library. If you don't code any of these keys, GETMEM(S) assumes the members are of type TEXT. A description of the meanings of these rather funny looking keys can be found in the NOS Reference Manual under the description of LIBEDIT.

Examples of GETMEM(S) Usage

```
ses.getmem format l=proclib un=ir77 text
REVERT.    END GETMEM  GROUP <- PROCLIB
```

The example uses GETMEM to get one text member called format from a library called proclib in the catalog of user ir77.

```
ses.getmem (txtcode,txtform,txthead) binary l=prglib un=mv73 ovl
REVERT.    END GETMEM  BINARY <- PRGLIB
```

This example extracts a group of ovl type records from a library called prglib in the catalog of user mv73 and places them on a group file called binary.

```
ses.getmem (comput..piscan, ciilev..ciiord) l=dbuglib rel
REVERT.    END GETMEM  GROUP <- DBUGLIB
```

This example illustrates how ranges may be coded for a parameter. The example extracts all rel members comput through piscan inclusive, and all members ciilev through ciiord inclusive, from a library called dbuglib, and places the extracted members on the file called group.

6.2 REPMEM OR REPMEMS - ADD OR REPLACE MEMBER(S) ON LIBRARY

REPMEM and REPMEMS are synonyms for a procedure to add new member(s) to, or replace member(s) on a library. **Note** : to add or replace members on a User LIBrary, for use by the CYBER loader, use the **REPULIB** procedure described below. Parameters to REPMEM(S) are :

m or mem or mems or member or members :
(optional) list of Member(s) to be added to or replaced on the library. If you don't code the m parameter, REPMEM(S) assumes that all the member(s) to be added or replaced are already on a file specified by the g parameter.

g or group :
(optional) name of a **group** file on which member(s) are accumulated as one member per logical record. If you don't code the g parameter, REPMEM(S) uses the value of profile variable **group**. If there's no such profile variable defined, REPMEM(S) uses filename **group** as the group file.

l or b :
(optional) name of Library on which the member(s) are to be added or replaced. If you don't code the l parameter, REPMEM(S) uses the value of profile variable **lib** as the name of the library. If there's no such profile variable defined, REPMEM(S) uses a library name of **lib**.

nl or nb :
(optional) name of New Library to be created when REPMEM(S) has completed its run. If you don't code the nl parameter, REPMEM(S) uses the value of profile variable **newlib**, and if there's no such profile variable defined, REPMEM(S) writes the new library over the file specified by the l parameter.

un :
(optional) User Name in whose catalog the library specified by l/nl is to be found, if l/nl is not in the catalog of the current user. If you don't code the un parameter, REPMEM(S) uses the value of profile variable **libown** as the user name from whose catalog the library is to be obtained, and if there's no such variable defined,

REPMEM(S) uses the current user's catalog.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

rep or add or addrep or repadd :
these (optional) keys specify whether you're adding and replacing, or just replacing. If you're only replacing member(s), you can code the rep key. If you omit this key altogether, REPMEM(S) assumes that you're adding and replacing member(s) and so issues the appropriate directives to LIBEDIT to perform that function.

dir or build or nodir or nobuild :
these (optional) keys specify whether a directory is to be placed at the end of the library when REPMEM(S) finishes its run. Coding the **dir** or **build** options specifies that you want a directory record to be added to the library. If you code the **nodir** or **nobuild** options, or if you don't code any of these keys, REPMEM(S) doesn't add a directory but if a directory already exists it is updated.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of REPMEM(S) Usage

```
ses.repmem (first, second, third) l=berklib, build
*    FIRST -> GROUP
*    SECOND -> GROUP
*    THIRD -> GROUP
*    REPLACING/ADDING MEMBERS ON BERKLIB
*    LIBRARY NOW ON SESTMPL
*    NEW LIBRARY NOW ON BERKLIB
*    BERKLIB PURGED
REVERT.    END REPMEM  GROUP -> BERKLIB
```

This example illustrates REPMEM used to collect three members onto a group file, and then place the contents of the group file on a library called berklib. The example shows some of the informative messages that appear during a REPMEM run.

Creating a New Library with REPMEM(S)

```
ses.repmem l=dummy, nl=proclib, g=newproc build
*    CREATING NEW LIBRARY PROCLIB
REVERT.    END REPMEM  NEWPROC -> PROCLIB
```

This example illustrates how to build a new library. The l parameter is coded as some "dummy" name that doesn't exist. The new members to be added are on a group file called newproc. The new library is created, and a directory is added because of the build key.

6.3 REPULIB - ADD OR REPLACE MEMBER(S) ON USER LIBRARY

REPULIB is a special version of REPMEM(S) to add or replace member(s) on a User LIBrary. LIBEDIT (the basis for these library management procedures) can't handle user libraries directly, so REPULIB uses the LIBEDIT and LIBGEN utilities to perform the process correctly. Parameters to REPULIB are :

m or mem or mems or member or members :
(optional) list of Member(s) to be added to or replaced on the library. If you don't code the m parameter, REPULIB assumes that all the member(s) to be added or replaced are already on a file specified by the g parameter.

g or group :
(optional) name of a group file on which member(s) are accumulated as one member per logical record. If you don't code the g parameter, REPULIB uses the value of profile variable group. If there's no such profile variable defined, REPULIB uses filename group as the group file.

l or b :
(optional) name of Library on which the member(s) are to be added or replaced. If you don't code the l parameter, REPULIB uses the value of profile variable lib as the name of the library. If there's no such profile variable defined, REPULIB uses a library name of lib.

nl or nb :
(optional) name of New Library to be created when REPULIB has completed its run. If you don't code the nl parameter, REPULIB uses the value of profile variable newlib, and if there's no such profile variable defined, REPULIB writes the new library over the file specified by the l parameter.

un :
(optional) User Name in whose catalog the library specified by l/nl is to be found, if l/nl is not in the catalog of the current user. If you don't code the un parameter, REPULIB uses the value of profile variable libown as the user name from whose catalog the library is to be obtained, and if there's no such variable defined,

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT
6.3 REPULIB - ADD OR REPLACE MEMBER(S) ON USER LIBRARY

REPULIB uses the current user's catalog.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

rep or add or addrep or repadd :
this (optional) key specifies whether you're adding and replacing, or just replacing. If you're only replacing member(s) you can code the **rep** key. If you omit this key altogether, REPULIB assumes that you're adding and replacing member(s) and so issues the appropriate directives to LIBEDIT to perform that function.

nx :
this (optional) key indicates that when REPULIB performs the LIBGEN portion of its job, that LIBGEN should not build a cross reference of entry points in the ULIB record. You can set your own default for this parameter by defining the nx variable in your profile (any non-null value causes the cross reference to not be built). Refer to the LIBGEN documentation in the NOS reference manual.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

6.4 COLLECT - COLLECT MEMBER(S) TO BUILD A GROUP FILE

COLLECT is used in conjunction with library management procedures such as REPMEM(S). COLLECT is used to accumulate members onto a group file for subsequent processing by REPMEM(S), LIBEDIT and so on. COLLECT can also be used in conjunction with source text maintenance procedures, to collect text modules, adding headers and COMMON lines if required. Parameters to COLLECT are :

m or mem or mems or member or members :
(optional) list of Members to be accumulated onto the group file. **Note** : if the file specified by the m parameter is a multi record file, COLLECT does gather all records in the file. This capability is not available with the c and r parameters as it is not truly applicable in those cases.

c or com or opic :
(optional) list of COMmon modules to be accumulated onto the group file. COLLECT assumes that such modules are straight text files, and so COLLECT places the module name and COMMON line at the start of the module.

r or reg or opl :
(optional) list of REGular modules to be accumulated onto the group file. COLLECT assumes that such modules are straight text files, and so COLLECT places the module name at the start of the module.

g or group :
(optional) name of group file onto which the member(s) or module(s) are to be accumulated. If you don't code the g parameter, COLLECT uses the value of profile variable group. If there's no such variable defined in your profile, COLLECT uses the filename group.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Example of COLLECT Usage

```
ses.collect (docode,dostat,eatup,elseif,equal) g=coders
*     DOCODE -> CODERS
*     DOSTAT -> CODERS
*     EATUP -> CODERS
*     ELSEIF -> CODERS
*     EQUAL -> CODERS
REVERT.     END COLLECT  CODERS
```

This example of COLLECT illustrates how it is used to accumulate a bunch of records onto a group file.

6.5 CATLIB - PRODUCE LIST OF MEMBERS IN A LIBRARY

CATLIB is used to display a list of the members in a library. CATLIB outputs the members list in a condensed format, four members per line, showing only the member names and their type. Parameters to CATLIB are :

l or b :
(optional) name of Library which is to have its member names displayed. If you don't code the l parameter, CATLIB uses the value of profile variable lib as the name of the library, and if there's no such variable defined, CATLIB uses a file name of lib.

un :
(optional) User Name in whose catalog the library specified by the l parameter is to be found. If you don't code the un parameter, CATLIB uses the value of profile variable libown as the user name, and if there's no such variable defined, CATLIB uses the current user's catalog.

o :
(optional) name of file to receive the Output from CATLIB. If you don't code the o parameter, CATLIB places the member list on file output.

short or long :
these (optional) keys determine the format of the output from CATLIB. If you omit this parameter or code the short key, CATLIB produces its output in the form shown in the example below. If you code the long key, CATLIB produces its output in the same format as the NOS utility CATALOG.

Example of CATLIB Usage

```
ses.catlib b=csiolib un=mv69
 CSIOLIB..ULIB      A59TOIA..REL       A64TOIA..REL       CATLIST..REL
 CNVCASE..REL       CPYA.....REL       C59TOIA..REL       IATOA59..REL
 IATOA64..REL       IATON59..REL       IATON64..REL       IDEBUG...REL
 IDEBUGI..REL       IKMISC...REL       ISCLMSG..REL       ISOLSYS..REL
 ISYSTIM..REL       MCOMP....REL       MCS.....REL        MCSCNV...REL
 MEXPR....REL       MGET.....REL       MGETCSP..REL       MGETKCS..REL
 MGETSYM..REL       MGETTOK..REL       MLOCFIL..REL       MMSG.....REL
 MPDTPVT..REL       MPLIST...REL       MPUT.....REL       MSTREAM..REL
 MTOKCNV..REL       N59TOIA..REL       N64TOIA..REL       PKUNPK...REL
 ZIATOXC..REL       ZXCTOIA..REL       CSIOLIB..OPLD
REVERT.     END CATLIB  CSIOLIB
```

This example shows CATLIB output for a library **csiolib** in the catalog of user **mv69**. The member types are as defined in the NOS reference manual section on LIBEDIT.

6.6 LISTMEM - LIST CONTENTS OF LIBRARY

LISTMEM performs two main functions to show you what's in a library, and to get a printout of the TEXT members in the library if necessary. LISTMEM first generates a normal CATALOG which shows the names of all the members in the library. Lastly, LISTMEM generates a printout of all TEXT members in the library - you can suppress this if you only want the catalog without the whole library's contents. Parameters to LISTMEM are :

l or b :
(optional) name of Library to be processed. If you don't code the **b** parameter, LISTMEM uses the value of profile variable **lib** as the library name, and if there's no such variable defined, the LISTMEM uses the default name **lib**.

o :
(optional) name of file to receive the Output from LISTMEM places the output on some unique file name that it generates. The output doesn't default to file output because, if you're running LISTMEM interactively, you probably don't want the rather voluminous output on the screen.

un :
(optional) User Name in whose catalog the library specified by **l** is to be found. If you don't code the **un** parameter, LISTMEM uses the value of the value of profile variable **libown**, and if there isn't such variable defined, LISTMEM uses the user name of the current user.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **batch** mode, but it can be run in **local** mode. The **dayfile** parameter is not used by this procedure.

print :
this (optional) parameter indicates how the output of LISTMEM is to be printed. For the **print** parameter you may code any of the parameters to procedure PRINT. If you don't code the **print** parameter at all, LISTMEM prints one (1) copy of the output on the ASCII printer.

short :
if you code this (optional) key, LISTMEM only generates the catalog output, and suppresses the listing of all the TEXT members.

Examples of LISTMEM Usage

```
ses.listmem print=c=3 short
 12.34.56. SUBMIT COMPLETE.  JOBNAME IS ACULARS
REVERT.    JOB LISTMEM  SUBMITTED
```

This example shows LISTMEM used to process a library, whatever name is the value of lib. Three copies of the short list (catalog only) are being produced. LISTMEM is being run in batch, rather than interactively.

```
ses.listmem b=proclib o=procs print=(c=2,h='latest/proclib')
   18.12.00. SUBMIT COMPLETE.  JOBNAME IS ANAPABS
REVERT.    JOB LISTMEM  SUBMITTED
```

This example shows LISTMEM being run in batch to produce two full printouts of the catalog and all the TEXT members in the library proclib. The title on the listing is 'latest proclib'.

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT
6.7 SORTMEM - SORT LIBRARY INTO ALPHABETICAL ORDER

6.7 SORTMEM - SORT LIBRARY INTO ALPHABETICAL ORDER

SORTMEM rearranges a library such that all the members of a given type appear in the library together, and within each type, the members are sorted into alphabetical order. **Note** : to sort a User LIBrary, for use by the CYBER loader, use the SRTULIB procedure described below. Parameters to SORTMEM are :

l or b :
(optional) name of Library to be sorted. If you don't code the l parameter, SORTMEM uses the value of profile variable **lib**. If there's no such profile variable defined, SORTMEM uses the name **lib**.

nl or nb :
(optional) name of New Library to be created at the end of the SORTMEM run. If you don't code the nl parameter, SORTMEM uses the value of profile variable **newlib**. If there's no such profile variable defined, SORTMEM writes the new library on the file specified by the l parameter.

un :
(optional) User Name in whose catalog the library specified by l/nl is to be found, if l/nl is not in the catalog of the current user. If you don't code the un parameter, SORTMEM uses the value of profile variable **libown** as the user name from whose catalog the library is to be obtained, and if there's no such variable defined, SORTMEM uses the current user's catalog.

o :
(optional) name of file to receive the Output of the programs used to run SORTMEM. If you don't code the o parameter, SORTMEM sets o=0, or no output. This is usually the desirable default, since the sort process generates large volumes of output.

edtsort :
(optional) pair of filenames representing the input and output files of the EDT text editor. You can use **edtsort** to EDIT the SORT output to impose your own ordering on the members in the library.

lock or nolock :

these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :

These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

nodir or nobuild or dir or build :

these (optional) keys indicate whether a directory is to be placed at the end of the library when SORTMEM has finished its run. If you code the **nodir** or **nobuild** key, SORTMEM doesn't place a directory in the library. If you code the **dir** or **build** key, SORTMEM does place a directory in the library. If you don't code any of these keys, SORTMEM does a **build**, that is, a directory is created for the sorted library.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled - "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of SORTMEM Usage

```
ses.sortmem l=proclib batchn
 17.52.00. SUBMIT COMPLETE.  JOBNAME IS ADXIBEN
REVERT.    JOB SORTMEM  SUBMITTED
```

This example illustrates SORTMEM being run as a batch job by coding the batch key. SORTMEM is being used to sort the file proclib into order.

```
ses.sortmem l=berklib lock build
*    LOCKING BERKLIB VIA INTRLOK
*    BERKLIB LOCKED
*    GENERATING MEMBER LISTS FOR BERKLIB
*    SORTING MEMBER LISTS
*    EXTRACTING MEMBERS FROM BERKLIB
*    SORTING BERKLIB ONTO SESTMPL
*    NEW LIBRARY ON SESTMPL
*    NEW LIBRARY NOW ON BERKLIB
*    SESTMPL PURGED
*    BERKLIB UNLOCKED
REVERT.    END SORTMEM  BERKLIB
```

This example illustrates SORTMEM run while you wait, at the terminal. The example shows the informative messages output by SORTMEM during its run.

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT
6.8 SRTULIB - SORT USER LIBRARY INTO ALPHABETICAL ORDER

6.8 SRTULIB - SORT USER LIBRARY INTO ALPHABETICAL ORDER

SRTULIB is a special version of SORTMEM for sorting a User LIBrary into alphabetical order. LIBEDIT (the basis for these library management procedures) can't handle user libraries directly, so SRTULIB uses the LIBEDIT and LIBGEN utilities to perform the process correctly. Parameters to SRTULIB are :

l or b :
(optional) name of Library to be sorted. If you don't code the l parameter, SRTULIB uses the value of profile variable **llb**. If there's no such profile variable defined, SRTULIB uses the name **llb**.

nl or nb :
(optional) name of New Library to be created at the end of the SRTULIB run. If you don't code the **nl** parameter, SRTULIB uses the value of profile variable **newlib**. If there's no such profile variable defined, SRTULIB writes the new library on the file specified by the l parameter.

un :
(optional) User Name in whose catalog the library specified by l/nl is to be found, if l/nl is not in the catalog of the current user. If you don't code the un parameter, SRTULIB uses the value of profile variable **llbown** as the user name from whose catalog the library is to be obtained, and if there's no such variable defined, SRTULIB uses the current user's catalog.

o :
(optional) name of file to receive the Output of the programs used to run SRTULIB. If you don't code the o parameter, SRTULIB sets **o=0**, or no output. This is usually the desirable default, since the sort process generates large volumes of output.

edtsort :
(optional) pair of filenames representing the input and output files of the EDT text editor. You can use **edtsort** to EDiT the SORT output to impose your own ordering on the members in the library.

lock or nolock :

these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :

These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

nx :

this (optional) key indicates that when SRTULIB performs the LIBGEN portion of its job, that LIBGEN should not build a cross reference of entry points in the ULIB record. You can set your own default for this parameter by defining the **nx** variable in your profile (any non-null value causes the cross reference to not be built). Refer to the LIBGEN documentation in the NOS reference manual.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT
6.9 WIPEMEM - DELETE MEMBER(S) FROM LIBRARY

6.9 WIPEMEM - DELETE MEMBER(S) FROM LIBRARY

WIPEMEM is intended as a means of deleting member(s) from a library. **Note** : to delete member(s) from a User LIBrary, for use by the CYBER loader, use the **WIPULIB** procedure described below. Parameters to WIPEMEM are :

l or b :
(optional) name of Library from which member(s) are to be deleted. If you don't code the l parameter, WIPEMEM uses the value of profile variable **lib** as the name of the library. If there's no such profile variable defined, WIPEMEM uses the name **lib** as the name of the library.

nl or nb :
(optional) name of New Library to be created at the end of the WIPEMEM run. If you don't code the **nl** parameter, WIPEMEM uses the value of profile variable **newlib** as the name of the new library. If there's no such profile variable defined, WIPEMEM places the new library back over the old library specified by the l parameter.

un :
(optional) User Name in whose catalog the library specified by **l/nl** is to be found, if **l/nl** is not in the catalog of the current user. If you don't code the **un** parameter, WIPEMEM uses the value of profile variable **libown** as the user name from whose catalog the library is to be obtained, and if there's no such variable defined, WIPEMEM uses the current user's catalog.

text :
(optional) list of TEXT member(s) to be deleted.

opic or com or c :
(optional) list of OPLC Common member(s) or COMmon modules to be deleted.

opl or reg or r :
(optional) list of OPL or REGular member(s) to be deleted.

rel :

(optional) list of RELocatable binary member(s) to be deleted.

ovl :
(optional) list of OVerLay type member(s) to be deleted.

abs :
(optional) list of ABSolute type member(s) to be deleted.

ppu :
(optional) list of PPU type member(s) to be deleted.

pp :
(optional) list of PP type member(s) to be deleted.

cap :
(optional) list of CAP type member(s) to be deleted.

proc :
(optional) list of CCL (not SES) PROCedure member(s) to be deleted.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the <u>default</u> action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** <u>keys</u>, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file <u>must</u> be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :

These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Example of WIPEMEM Usage

```
ses.wipemem l=thislib, text=(heavy,user's), rel=last..access
*    DELETING MEMBERS FROM THISLIB
*    NEW LIBRARY ON SESTMPL
*    NEW LIBRARY NOW ON THISLIB
*    SESTMPL PURGED
REVERT.    END WIPEMEM  THISLIB
```

This example of WIPEMEM illustrates the way in which multiple lists of record types may be coded, and also shows a range of record types coded for the **rel** type.

6.10 WIPULIB - DELETE MEMBER(S) FROM USER LIBRARY

WIPULIB is a special version of WIPEMEM for deleting (wiping) member(s) from a User LIBrary. LIBEDIT (the basis for these library management procedures) can't handle user libraries directly, so WIPULIB uses the LIBEDIT and LIBGEN utilities to perform the process correctly. Parameters to WIPULIB are :

l or b :
(optional) name of Library from which member(s) are to be deleted. If you don't code the l parameter, WIPULIB uses the value of profile variable lib as the name of the library. If there's no such profile variable defined, WIPULIB uses the name lib as the name of the library.

nl or nb :
(optional) name of New Library to be created at the end of the WIPULIB run. If you don't code the nl parameter, WIPULIB uses the value of profile variable newlib as the name of the new library. If there's no such profile variable defined, WIPULIB places the new library back over the old library specified by the l parameter.

un :
(optional) User Name in whose catalog the library specified by l/nl is to be found, if l/nl is not in the catalog of the current user. If you don't code the un parameter, WIPULIB uses the value of profile variable libown as the user name from whose catalog the library is to be obtained, and if there's no such variable defined, WIPULIB uses the current user's catalog.

mem or m or rel :
list of member(s) to be deleted.

lock or nolock :
these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the lock parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the lock or nolock keys, interlocking is controlled by the lokmode profile variable. Refer to the introductory sections of this

chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

batch job parameters :

These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

nx :

this (optional) key indicates that when WIPULIB performs the LIBGEN portion of its job, that LIBGEN should not build a cross reference of entry points in the ULIB record. You can set your own default for this parameter by defining the **nx** variable in your profile (any non-null value causes the cross reference to not be built). Refer to the LIBGEN documentation in the NOS reference manual.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT
6.11 LIBEDIT - RUN LIBEDIT UTILITY

6.11 LIBEDIT - RUN LIBEDIT UTILITY

LIBEDIT is intended as a means of calling NOS LIBEDIT utility directly. Although SES library management procedures use the LIBEDIT utility themselves, not all the functions of LIBEDIT are available, since SES procedures are oriented towards the most common functions. The LIBEDIT procedure gives you direct control over the LIBEDIT utility, but still provides all SES convenience of ATTACHing your files for you and so on. Parameters to LIBEDIT are :

g or group or lgo :
(optional) name of file containing record(s) to be inserted or replaced on the library specified by **l**. If you don't code the g parameter, LIBEDIT uses the value of profile variable **group**, and if there's no such variable defined, uses the default filename group.

l or b :
(optional) name of the old Library to be manipulated by LIBEDIT. If you don't code the l parameter, LIBEDIT uses the value of profile variable **llb** as the name of the library, and if there's no such variable defined, LIBEDIT uses the default name of **llb**.

nl or nb :
(optional) name of the New Library after the LIBEDIT run. If you don't code the **nl** parameter, LIBEDIT uses the value of profile variable **newlib**, and if there's no such variable defined, the LIBEDIT writes the new library back on top of the library specified by **l**.

un :
(optional) User Name in whose catalog the library specified by **l/nl** is to be found, if **l/nl** is not in the catalog of the current user. If you don't code the **un** parameter, LIBEDIT uses the value of profile variable **llbown** as the user name from whose catalog the library is to be obtained, and if there's no such variable defined, LIBEDIT uses the current user's catalog.

i :
(optional) list of character strings which represent LIBEDIT utility directives. Using the i parameter, it is possible to perform many LIBEDIT operations using a single

control statement. If you don't code the i parameter, LIBEDIT assumes the directives are coming from file INPUT. If you code the i parameter as a single name, LIBEDIT assumes that it's the name of a file containing LIBEDIT directives.

o :

this (optional) parameter specifies the name of the file to receive the listing output from LIBEDIT. If you don't code this parameter, LIBEDIT places its listing output on file OUTPUT.

lock or nolock :

these (optional) parameters determine whether the base update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the base being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the base being updated. If the interlock file cannot be found, the procedure aborts.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Example of LIBEDIT Usage

```
ses.libedit l=proglib i='*i,ovl/catalog,catlist,sesmsg'
.....
   .....  here appears the output from LIBEDIT utility
*    NEW LIBRARY ON SESTMPL
*    NEW LIBRARY NOW ON PROGLIB
*    SESTMPL PURGED
REVERT.    END LIBEDIT  PROGLIB
```

The example illustrates how the LIBEDIT utility input directive may be coded as a string parameter on the procedure call line. The text preceded by periods indicates where, in the actual use of LIBEDIT, output from the LIBEDIT utility itself appears.

7.0 GETTING INFORMATION

7.0 GETTING INFORMATION

This chapter describes a collection of SES procedures to find out what's going on and what's available. There are two main areas of interest :

1. Information about your local environment on NOS - what files you have, your validation limits, the time, parts of your dayfile, and others.
2. Information about SES - documents, procedures and status.

CATLIST display comprehensive information about files in a user's catalog.

DAYFILE displays a selected part of your dayfile.

CATALOG gives a condensed list of the records in a library.

FILES display information about local files.

PERMIT gives permission information for files in a user's catalog.

LIMITS displays your LIMITS information in a condensed format.

TIME "talks" the current time as a dayfile message.

DISPLAY displays various information about your job environment.

DAYWEEK tells you what day of the week a date fell on, falls on (that is, today), or will fall on.

EXPLAIN displays online manuals.

CONVOLM converts online manual source to NOS/VE format.

7.0 GETTING INFORMATION

TOOLDOC prints documents describing tools.

INFO displays the latest information about SES.

USSINFO displays the latest information about User Supplied Software.

SESPROC generates a list of names of SES procedures.

SESPARM generates a printout of the parameter lists of SES procedures in any selected procedure library.

7.0 GETTING INFORMATION
7.1 CATLIST - DISPLAY PERMANENT FILE INFORMATION

7.1 CATLIST - DISPLAY PERMANENT FILE INFORMATION

CATLIST gives you a concise but very comprehensive list of information about all files in a catalog. The files are listed in alphabetical order, three files per line, so that you can get 45 files worth of information on one screenload. The world goes by your window more slowly, so to speak. For each file in the specified catalog, CATLIST outputs the information in the following format :

FILENAM type category mode length

where the different things displayed are :

FILENAM is of course the name of file in question.

type is the access type for the file. It is I for an Indirect file, and D for a Direct file.

category is the security of the file. This field is PU for a PUblic file, SP for a Semi Private file, and PR for a PRivate file.

mode is the access mode for the file. This is one of R for Read, W for Write, M for Modify, A for Append, RM for Read Modify, and RA for Read Append and E for Execute.

length is the length of the file in PRUS.

For example, a typical line of outout from CATLIST would be like :

MAILBOX I PU RA ..1 MAXBIN D PU R 342 PROFILE I PR R ..1

This line of CATLIST output shows that MAILBOX is an Indirect access PUblic Read Append mode file of 1 pru, MAXBIN is a Direct access PUblic Read mode file of 342 prus, and PROFILE is an Indirect access PRivate Read mode file of 1 pru.

7.0 GETTING INFORMATION
7.1 CATLIST - DISPLAY PERMANENT FILE INFORMATION

Parameters to CATLIST are :

un :
(optional) User Name of the (alternate) catalog to be CATLISTed. If you don't code the un parameter, the current user's catalog is CATLISTed.

pn :
(optional) Pack Name for files to be CATLISTed. If you don't code the pn parameter, the files on the default pack are CATLISTed.

o :
(optional) name of file to receive the Output from CATLIST. If you don't code the o parameter, CATLIST uses file output.

fpl :
this (optional) parameter is used to tell CATLIST how many file descriptions it should write on each line of the output file. If specified, the Files Per Line parameter must be given a numeric value from 1 to 10. If you don't code the fpl parameter, CATLIST generates 3 files per line.

Note : CATLIST can only handle the first thousand files in a user's catalog.

Example of CATLIST Usage

The example below shows a CATLIST for another user's catalog.

```
ses.catlist un=ls69
    CATLIST OF LS69 FOR LR78   ( 13 FILES, 10769 PRUS )
 CSIOLIB D SP R .167     CSIOPL. D SP R .585     DICTION I SP R .125
 DOCBASE D SP R 1158     DO1PL64 D SP R ..55     DO2R64. D SP R .347
 DO2S64. D SP R .783     INTRLOK D PU R ...0     ISOLLIB I SP R ..37
 MAILBOX I SP W ...6     PROCLIB D SP R .409     PROGLIB D SP R 6126
 UTILITY D SP R .971
REVERT.    END CATLIST
```

7.2 DAYFILE - DISPLAY SELECTED PORTIONS OF DAYFILE

DAYFILE displays either the last 14 lines of your dayfile, or to display any number of lines starting at a specified character string. Parameters to DAYFILE are :

n or list :
(optional) number of lines of dayfile to display. If you don't code the n parameter, DAYFILE displays 14 lines.

find :
(optional) character string at which to start displaying the dayfile. If you don't code the **find** parameter, DAYFILE always goes to the line specified by **last_line-n.** If you code a character string for this parameter, DAYFILE goes to the last occurrence of that character string, then displays n lines starting from that string.

o or output :
(optional) name of file to receive the output from DAYFILE. If you don't code the o parameter, DAYFILE uses file **output.**

Note : If you code the **find** parameter, the ? character may not be a part of the character string, as ? is used as the string delimiter for the search.

7.3 CATALOG - SHOW LIST OF RECORDS IN A FILE

CATALOG performs the NOS CATALOG operation on a file. The NOS CATALOG utility outputs a large amount of information about the records, most of which is never used. The SES CATALOG procedure outputs a condensed form of the information from CATALOG, showing only the record names and their types. CATALOG outputs its information in the form :

```
RECNAME TYPE   RECNAME TYPE   RECNAME TYPE   RECNAME TYPE
```

four record names and their types are displayed on one line, so that if you're using a (713) terminal, the names and types of sixty records can be displayed before the information goes off the top of the screen. The world goes by your window more slowly, so to speak. Parameters to CATALOG are :

l or b :
name of Library or Base whose contents are to be CATALOGED. CATALOG is so arranged that if the file to be CATALOGed is local to your running job at the time the CATALOG runs, the file isn't RETURNed. If the file is not local when CATALOG runs, CATALOG GETs or ATTACHes the file for you, and RETURNs it when it has been CATALOGed.

un :
(optional) User Name in whose catalog the file is to be found, if the file is not in the catalog of the current user.

o or output :
(optional) name of file to receive the Output from CATALOG. If you don't code the o parameter, CATALOG uses file output.

short or long :
these (optional) keys determine the format of the output from CATALOG. If you omit this parameter or code the short key, CATALOG produces its output in the form shown in the example below. If you code the long key, CATALOG produces its output in the same format as the NOS utility CATALOG.

See also the descriptions of procedures CATBASE and CATLIB which

provide useful defaulting of the l and un parameters.

Example of CATALOG Usage

This example shows a CATALOG operation performed on a library from another user's catalog. The CATALOG procedure obtains the library, and returns it when it has finished.

```
ses.catalog afile, un=jf73
 CCEFMGR..OPLC      CEDM.....OPL      CEDTEXT..OPL      CIOINT...OPL
 CRACK....OPL       EDBLD....OPL      EFMGR....OPL      ENVINIT..OPL
 EPBLD....OPL       FMTMSG...OPL      MSGDICT..OPL      PINESC...OPL
 POUTESC..OPL       PUSHERR..OPL      REPERR...OPL      SFSKEOL..OPL
 SFWTIL...OPL       SRCHMSG..OPL      STRTUP...OPL      TCS......OPL
 AFILE....OPLD
REVERT.     END CATALOG  AFILE
```

7.4 FILES - DISPLAY LOCAL FILE INFORMATION

FILES gives you a concise list of information about your local files. The output of FILES is a compressed form of the output from the NOS utility ENQUIRE,F. Parameters to FILES are :

o or output :
(optional) specifies the name of the file to receive the local file information. If you don't code this parameter, FILES uses file **output**.

all :
if you code this (optional) key, FILES includes in its output some files that are normally not included because they are always present (e.g., INPUT*, INPUT, and OUTPUT).

Example of FILES Usage

```
ses.files
 LBSRC70 ..817 PM.     UTILSRC .1447 PM*     SCLEDIT ....3 LO.
 GROUP.. ...12 LO.
REVERT.    END FILES
```

The example shows FILES being used to list local file information to the terminal.

7.5 PERMIT - OBTAIN FULL LIST OF PERMISSION INFORMATION

PERMIT provides you with a permission information for all files in your catalog, or for a selected list of files, whereas if you use the NOS CATLIST utility, you can only list permissions one file at a time. Parameters to PERMIT are :

o or output :
(optional) name of file to receive the Output from PERMIT. If you don't code the o parameter, PERMIT uses file output.

f or fn :
(optional) list of File Names for which you wish the permission information to be produced. If you don't code the f parameter, PERMIT gives the information for all files in your catalog.

Example of PERMIT Usage

```
ses.permit  fn=proclib

 CATALOG OF WRB                FM/CLSH176 80/10/28. 08.56.42.

PROCLIB


   1. SES        READ          3 80/10/17. 09.43.18.
   2. SESAUX     READ          1 80/10/28. 08.44.33.
   3. SLE        READ          1 80/10/28. 08.50.29.
   4. EFT        READ          1 80/10/28. 08.54.58.
```

This example illustrates a typical use of PERMIT. The output from the PERMIT procedure is as that of the standard NOS CATLIST utility.

7.6 LIMITS - DISPLAY VALIDATION LIMITS

LIMITS provides you with a condensed version of the display that you'd get by using the NOS LIMITS control statement. The LIMITS control statement uses 35 lines to display the information. By using this LIMITS procedure, you get that information in five lines, thereby conserving space. Parameters to LIMITS are :

o or output :
(optional) name of file to receive the Output from LIMITS. If you don't code the o parameter, LIMITS places the display on file OUTPUT.

Note that this LIMITS procedure formats the LIMITS data such that the word UNLIMITED is represented by an asterisk (*).

Example of LIMITS Usage

This example shows the use of the LIMITS procedure, and the typical output generated by it.

```
ses.limits
UI=612  AB=,  AB=,  AB=,  AB=,  MT=*,  RP=*,  TL=*,  CM=*,  NF=*,
DB=*,  FC=*,  CS=*,  FS=*,  PA=EVEN,  RO=SYSTEM,  PX=HALF,  TT=TTY,
TC=ASCII,  IS=BATCH,  MS=*,  DF=*,  CC=*,  DF=*,  CP=*,  LP=*,
EC=*,  SL=*,  CN=,  PN=,  DS=*,  AW=0000777777770000755?
REVERT.    END LIMITS
```

7.7 TIME - DISPLAY CURRENT TIME OF DAY

TIME is a quick way of finding the time (to the nearest five minutes). The SES TIME procedure "talks" the time as a dayfile message, for example :

```
ses.time
REVERT.    TWENTY FIVE PAST TWELVE
```

7.8 DISPLAY - DISPLAY VARIOUS USEFUL INFORMATION

DISPLAY provides you with an easy way to ask for the display of odd things such as the job control registers, field length, switches, date, time and so on. You can request the display of many things at a time. The display is formatted in a "dense" fashion, to minimize the number of lines of screen used. There are only two parameters to DISPLAY that require values, the rest are just keys to indicate what you want displayed. The two parameters are the expr or e parameter, described below, and the o parameter which determines which file the display goes to. The display goes to file output by default. Keys indicate to DISPLAY what you want to know. You use DISPLAY like this :

```
ses.display thing1, thing2, thing3, etc.
```

where the "things" are any or all of the keys described here.

key	what's displayed
expr or e	displays the value of any valid SES expression. A list of the arithmetic operators available for expressions is given in the description of the MATH procedure below. A full description of the syntax of expressions can be found in the SES Procedure Writer's Guide.
date	displays the current date in the form : AUG 1, 1978
time	"talks" the time in the form : TWENTY FIVE TO ELEVEN. The time given in this form is always rounded to the nearest five minutes.
clock	displays the clock time in the form : 4:23 PM
fl	displays your current Field Length
r1	displays the contents of job control register R1
r2	displays the contents of job control register R2
r3	displays the contents of job control register R3
r1g	displays the contents of job control register R1G
ef	displays the contents of the Error Flag
efg	displays the contents of the Global Error Flag
jcr	displays the contents of all four Job Control Registers, R1, R2, R3 and EF.
sw	displays the six sense SWitches 1 to 6 in left to right order.

7.0 GETTING INFORMATION
7.8 DISPLAY - DISPLAY VARIOUS USEFUL INFORMATION

all displays ALL the above information

Example of DISPLAY Usage

```
ses.display all
 FL = 45000(8),     R1 = 4500, 10624(8),     R2 = 0
 R3 = 6789,     15205(8),     R1G = 0,     EF = 0,     EFG = 0,
 SW = (F,T,F,T,T,F),     JAN 25, 1980,     TWENTY PAST FOUR
REVERT.     END DISPLAY
```

7.9 DAYWEEK - DISPLAY DAY OF THE WEEK

DAYWEEK is used to determine what day of the week any date (including today) fell on, is, or will fall on. Parameters to DAYWEEK are :

day :
(optional) specifies the number of the day in the month. If you don't code this parameter, the current day of the month is used.

month :
(optional) specifies the month either as a number (1..12) or as a string (of which the first 3 characters are used). If you don't code this parameter, the current month is used.

year :
(optional) specifies the year. Note that all the digits of the year must be given. If you don't code this parameter, the current year is used.

o or output :
(optional) specifies the name of the file to receive the output from DAYWEEK. If you don't code this parameter, DAYWEEK writes its output to file output.

7.10 EXPLAIN - DISPLAY ONLINE MANUALS

EXPLAIN is used to access the information available in SES and NOS online manuals. If access is requested to a manual that does not exist at your site, the following message will be issued:

COULD NOT FIND A MANUAL NAMED <manual_name>

The parameters to EXPLAIN are:

m :

(optional) name of the online manual to be accessed. If you don't code the manual parameter, EXPLAIN uses the default online manual menu named SESINFO, which allows you to access the available online manuals by menu selection. The values that can currently be used on the manual parameter are:

SESINFO	- Online Manual Menu
TOOLS, SES	- SES Users Handbook Online Manual
CYBIL	- CYBIL Online Reference Manual
MISC	- Miscellaneous Routines Interface Reference Manual
GRAPH, CADSG	- NOS GRAPH Graphics Online Users Handbook
DEFINE, DD, CADSDD	- NOS DEFINE Data Dictionary Tools Online Users Handbook
CID	- Cyber Interactive Debug Online Manual
COMMAND	- NOS COMMAND Online Reference Manual
CONTEXT	- Online Manual Reference Manual
FTN5, FORTRAN	- FORTRAN Version 5 Online Reference Manual
SORT5, SORT	- SORT/MERGE Version 5 Online Reference Manual

topic or t :

(optional) string to be used in an index topic search of the named online manual. The online manual will be entered at the screen pointed to by the index topic given in this parameter, rather than at the main menu of the online manual. This parameter must be specified in the form of a string, for example, t='this index topic'.

Examples of EXPLAIN Usage

```
ses.explain
```

```
REVERT.  END EXPLAIN

ses.explain cybil
REVERT.  END EXPLAIN CYBIL

ses.explain m=ftn5
REVERT.  END EXPLAIN FTN5
```

7.11 CONVOLM - CONVERT ONLINE MANUAL SOURCE TO NOS/VE FORMAT

CONVOLM is used to convert NOS online manual source files to NOS/VE format. The converted online manual source file must be moved to NOS/VE by the user. The parameters to CONVOLM are :

input or i or f :
name of the NOS online manual source input file to be converted to NOS/VE format.

output or o :
(optional) name of the output file which will receive the converted online manual source file. If the output parameter is not coded, the output appears on the file specified by the input parameter.

listing or l :
(optional) name of the listing file which will contain the error and statistics messages created during the conversion. If the listing parameter is not coded, the error/statistics output appears on a file called LISTING.

Examples of CONVOLM Usage

```
ses.convolm i=nosmans
REVERT.  END CONVOLM NOSMANS -> NOSMANS,LISTING

ses.convolm nosmans vemans
REVERT.  END CONVOLM NOSMANS -> VEMANS,LISTING

ses.convolm i=nosmans o=vemans l=stats
REVERT.  END CONVOLM NOSMANS -> VEMANS,STATS
```

7.12 TOOLDOC - PRINT TOOL DOCUMENT

TOOLDOC is the means by which on-line and/or off-line copies of TOOL DOCuments are made available. Parameters to TOOLDOC are :

docs or d :
This (optional) parameter is used to indicate which document(s) you wish to obtain. Document names are taken from the "document number" column of the document index display (see the 'doclist' parameter). The absence of this parameter will default the proc to display the entire document index.

doclist or dl :
(optional) used to indicate that the document index is to be displayed - or retained on a file named by the 'listing' parameter. The 'doclist' parameter is ignored if the 'docs' parameter is specified. Specific portions of the document index are displayed by using the following keys with this parameter:

CYBIL	- CYBIL Compiler related documents.
CADS	- CADS related documents.
SCU	- SCU related documents.
C170	- SES Development and Support tools.
C180	- C180 Operating System development documents.
M68000	- Motorola 68000 related documents.
OTHER	- Documents that don't fit into the categories above.
SITE	- Documents available only at the local user's site.
ALL	- All of the above categories. This is the default if none of the above keys are indicated.

listing or l :
(optional) name of file to receive the document index display (when using TOOLDOC without specifying any document(s)), or the formatted document(s) themselves. If you code the listing parameter to direct the document(s) or the document index to a local file, that file is not printed.

bin :
(optional) BIN number used at sites requiring bin numbers to indicate the bin number the output is to go to. If you don't code the bin parameter, TOOLDOC uses the value of profile variable userbin, and if there's no such variable defined, generates a bin number of NO-ID.

Note that the standard SES batch job parameters are available. The default is to run the job in batch mode.

Example of TOOLDOC Usage

```
ses.tooldoc arh1737
10.36.10.  SUBMIT COMPLETE.  JOBNAME IS AAEQAXA
REVERT.    JOB  TOOLDOC  SUBMITTED
```

This example would cause the ERS for TXTFORM to be printed on the system line printer.

```
ses.tooldoc i=doclist local
*    INDEX DISPLAY -> DOCLIST
REVERT.    TOOLDOC DISPLAY COMPLETED
```

This example copies the document index to the file DOCLIST.

7.13 INFO - ACCESS SES INFORMATION

INFO is a means whereby users may get up to the minute bulletins on changes in SES. Parameter(s) to INFO are :

o or output :
(optional) name of file to receive the Output from INFO. If you don't code the o parameter, INFO outputs the INFO file to OUTPUT.

7.14 USSINFO - ACCESS USS INFORMATION

USSINFO is a means whereby users may get up to the minute bulletins on changes in User Supplied Software. Parameter(s) to USSINFO are :

o or output :
(optional) name of file to receive the output from USSINFO. If you don't code the o parameter, USSINFO outputs the USSINFO file to OUTPUT.

7.15 SESPROC - LIST SES PROCEDURE NAMES

SESPROC displays a list of procedure names already used by SES. This is of use if you are a procedure writer and wish to create new procedures for use in SES. You can then use SESPROC to ensure that there isn't any name conflict. Parameters to SESPROC are :

un :
(optional) list of user names whose catalogs are to be searched for procedure libraries. SES uses the SES Library NAMe as the name of the procedure library to look for. If you don't code the un parameter, SESPROC displays the names of procedures in the SESPLIB of the current user, and the names of procedures on SES library.

o or output :
(optional) name of file to receive the Output from SESPROC. If you don't code the o parameter, SESPROC uses file output.

Example of SESPROC Usage

```
ses.sesproc  un=ed73
 UN = ED73
          CETGEN    COLLDOC   COLLSUB   EDTDOC    EDTMOD    EDTSUB
 ICGEN    MOVDOC    MOVSUB    PERT      PRTDOC    PRTSUB    RELDOC
 RELSUB   REPDOC    REPSUB    SENDKC    WIPEDOC   WIPESUB
REVERT.     END SESPROC
```

This example shows SESPROC used to determine the names of SES procedures in the SESPLIB of user ed73.

7.16 SESPARM - PRINT PARAMETER REQUIREMENTS FOR SES PROCS

SESPARM prints out the parameter definitions for all the procedures on a library. It runs **batch** or **local** with **batch** as the default. Parameters to SESPARM are :

l :
name of Library whose parameters are to be printed.

un :
(optional) User Name where the library is to be found. If you don't code the **un** parameter, SESPARM assumes that the library is in the catalog of the current user.

o or output or listing :
(optional) name of file to receive the Output of SESPARM. If you don't code the **o** parameter, the output appears on file LISTING.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **batchn** mode, but it can be run in **local** mode. The **dayfile** parameter is not used by this procedure.

print :
(optional) parameter to control the PRINTing of the output. When the job runs in batch, one copy is printd automatically. If you run the procedure in **local** mode, no copies are printed, unless you specifically code some **print** parameter. For example, you could code **print=c=3** to obtain three copies of the parameter listing. The format of the **print** parameter is that of the parameters for the PRINT procedure.

Note that the parameter list of each procedure **must** terminate with a PARMEND directive, or, for those records in a library which don't have a PARMEND directive by virtue of the fact that they're INCLUDE files, the parameter list must be terminated by a

" \ PARMEND "

Example of SESPARM Usage

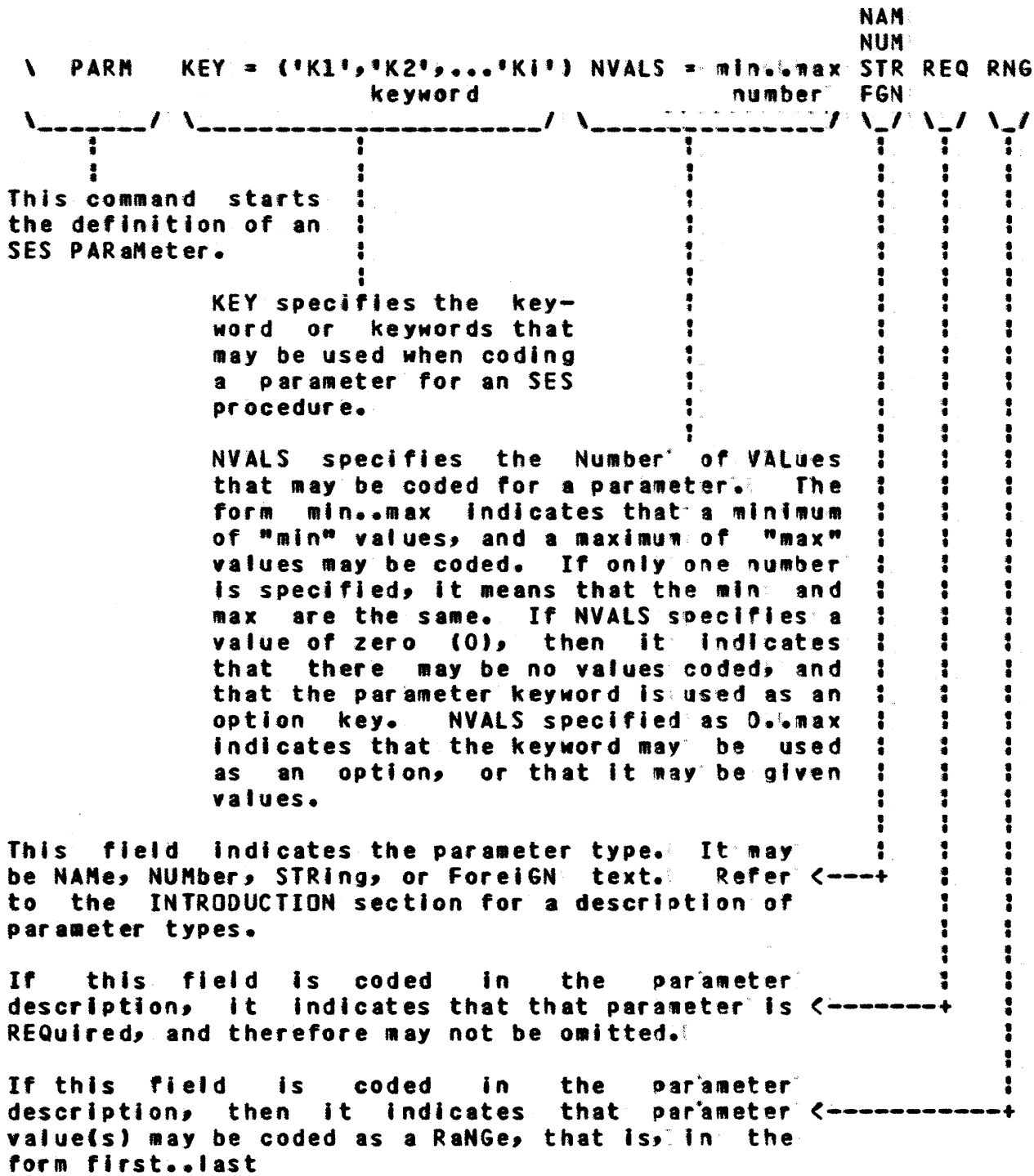
```
ses.sesparm l=proclib un=hk77 local noprint
 DEBUG
 \  PARM    KEY = 'f'              NVALS = 1..MAXVALS   NAM
 \  PARM    KEY = ('args', 'a')    NVALS = 1            STR
 \  PARM    KEY = 'p'              NVALS = 1..MAXVALS   NAM
 \  PARM    KEY = 'i'              NVALS = 1            NAM
 \  PARM    KEY = 'e'              NVALS = 1..MAXVALS   STR
 \  PARM    KEY = 'xid'            NVALS = 1..MAXVALS   STR
 \  PARMEND

==========
 DBUGFTN
 \  PARM  KEY = ('i', 'f')  NVALS = 1  NAM
 \  PARM  KEY =  'i'        NVALS = 1  STR
 \  PARM  KEY =  'b'        NVALS = 1  STR
 \  PARM  KEY =  'lo'       NVALS = 1  NAM
 \  PARMEND
REVERT.    END  SESPARM  PROCLIB/HK77 -> OUTPUT
```

Although this is a rather contrived example of a procedure library with only two procedures in it, it illustrates the output that SESPARM generates. SESPARM is run in local mode, or while you wait, to produce the parameter definitions for procedures in library proclib in the catalog of user hk77. The noprint key inhibits printing of the generated output.

Explanation of the output of SESPARM

The diagram on the next page shows the alternative forms of parameter definitions of SES procedures. Alternative forms are arranged in the vertical columns. The blocks of text below provide a brief description of each field of a parameter definition.

7.0 GETTING INFORMATION
7.16 SESPARM - PRINT PARAMETER REQUIREMENTS FOR SES PROCS



8.0 FILESPACE MANAGEMENT

8.0 FILESPACE MANAGEMENT

SES provides some useful utilities to aid you in handling your files. The procedures described here are :

RETAIN GETs or ATTACHes all your files to stop them being archived.

DUMPPF
LOADPF interfaces to a dump and load utility to provide easy to use dumping and loading of files.

REWRITE provides a "safe" way to rewrite a file after it has been modified.

8.0 FILESPACE MANAGEMENT
8.1 RETAIN - ACCESS ALL FILES IN USER'S CATALOG

8.1 RETAIN - ACCESS ALL FILES IN USER'S CATALOG

RETAIN GETs or ATTACHes all your files to ensure that they aren't archived. RETAIN also provides the facility to CATALOG either designated files, or all DIRECT access files in your catalog. Parameters to RETAIN are :

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". RETAIN runs in **batchn** mode by default. RETAIN doesn't use the **dayfile** parameter.

c or noprint :
(optional) coding **c** alone as a key causes all DIRECT access files in the catalog to be CATALOGed. Coding a list of files for **c** causes only those specified files to be CATALOGed. Coding the **noprint** key suppresses all output from RETAIN.

s or short :
coding the (optional) **short** key directs RETAIN to generate a **short** SES style CATLIST output instead of the NOS long form CATLIST. RETAIN generates the NOS style long form CATLIST by default.

8.2 DUMPPF / LOADPF - DUMP / LOAD PERMANENT FILES

DUMPPF and LOADPF provide an easy way of dumping files to magnetic tape and retrieving them when required. DUMPPF and LOADPF use the same parameters, so they are described here together with variations explained as required.

Note that DUMPPF and LOADPF interface to the permanent file utility RECLAIM. The parameters described below are a minimal subset of RECLAIM's capabilities that are needed to dump and load permanent files. RECLAIM creates and uses a database in the user's catalog; therefore, a direct access file by the name of RECLDB should not be purged.

f or file or files :
(optional) list of files to be processed. If you don't give any file names, all files (or files determined by other selection criteria) are dumped or loaded. LOADPF will not overwrite an existing file by the same permanent file name. Therefore, if overwriting is desired, those files should be purged beforehand.

tape or vsn :
the VSN (volume serial number) of the tape that is to be used for dumping or loading. If the tape VSN is already present in file RECLDB, it may not be necessary to specify this parameter.

seven or mt or nine or nt :
(optional) indicates whether to use seven or nine track tape. The default is to use nine track. This parameter may be specified in the profile by the variable **tapetrk**.

dn or density :
(optional) tape density to be used. Densities may be LO, HI and HY (default) for seven track tapes and HD, PE (default) and GE for nine track. This parameter may be specified in the profile by the variable **tapeden**.

fmt or format :
(optional) tape format to use. The default is I which is NOS Internal. This parameter may be specified in the profile by the variable **tapefmt**.

tapelist :
coding this (optional) key will cause a listing to be produced that shows what files are on the tape. It also gives other information such as file type, length, date last modified, etc.

ty or type :
(optional) TYPE of files to process. If you don't code the type parameter, all files will be processed (both direct and indirect). Other options are TY=I (to process only Indirect files) or TY=D (to process only Direct files). The DUMPPF/LOADPF procedure doesn't check the validity of any options you code here.

lo :
this (optional) key directs the LOADPF procedure to load files as local files only. This parameter may not be used when dumping files.

na or noabort :
this (optional) key directs DUMPPF/LOADPF to NOt ABORT if any errors are encountered during processing. The normal default is to abort.

xp :
(optional) eXtra Parameters which are options to be passed to RECLAIM. You code this in the form of a character string.

batch job parameters :
these parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS" - of the SES USER's Handbook. DUMPPF/LOADPF run in batch mode by default.

Examples of DUMPPF / LOADPF Usage

```
ses.dumppf vsn=backup
 07.48.35. SUBMIT COMPLETE.  JOBNAME IS AGRIBUS
REVERT.    JOB DUMPPF  SUBMITTED
```

This is the simplest example of DUMPPF usage, your entire catalog is dumped to nine track tape backup.

```
ses.dumppf (wilson,heath,nixon,agnew), tape=abc123, d=hy
 19.17.15. SUBMIT COMPLETE.  JOBNAME IS ABXTCUB
REVERT.    JOB DUMPPF  SUBMITTED
```

In this example, the four files in the parenthesised list are dumped to tape ABC123 where the tape Density (the d parameter) is set to HY (800 bpi) instead of the default PE (1600 bpi).

```
ses.loadpf tape=backup
 16.42.55. SUBMIT COMPLETE.  JOBNAME IS ADRQBIT
REVERT.    JOB LOADPF  SUBMITTED
```

This example loads all files from tape BACKUP. Note that existing permanent files are skipped during the load of files from tape BACKUP.

8.3 REWRITE - REWRITE FILE

REWRITE is mainly used as a building block for the source text maintenance and library management procedures, where rewriting of a base or library is required. REWRITE is a "safe" way of performing the overwrite process. If for some reason REWRITE fails to perform the rewrite, it tries five (5) times before it gives up. Parameters to REWRITE are :

i or rewriti :
name of Input file to REWRITE.

o or rewrito :
name of Output file after the REWRITE process is complete. This is the new file created by the rewrite.

un or rewritu :
(optional) User Name in whose catalog the file specified by o is to be found, if it isn't in the catalog of the current user. If you don't code the un parameter, REWRITE uses the user name of the current user.

status or sts :
these (optional) keys can be used for those cases where REWRITE is being used as a building block of more sophisticated procedures or jobs. The status key causes the REWRITE procedure to set the NOS job control register EFG to the value zero if REWRITE is successfully completed, and non zero if anything went wrong during the run of REWRITE. If status is not specified, the procedure will EXIT if an error occurs.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Note : If the new file (given by the o parameter) is an existing INDIRECT access file, it is REPLACEd. If the new file is an existing DIRECT access file, REWRITE performs an overwrite process by ATTACHing the file in WRITE mode. If the new file doesn't exist, REWRITE DEFINEs the new file as a DIRECT access READ mode file

(this, of course, can only be done if the un parameter is omitted or is given as the user name of the current user).

9.0 COMPILING, LINKING, AND DEBUGGING

9.0 COMPILING, LINKING, AND DEBUGGING

This section describes the SES procedures available for compiling (assembling) programs written in various languages, and for linking relocatables into executable form. The procedures currently available in this category are :

CYBIL	run the CYBIL compiler.
CYBIL	run the CYBIL compiler.
ISWL	run the ISWL CC compiler.
SYMPL	run the SYMPL compiler.
FTN	run the ForTraN extended compiler.
FTN5	run the ForTraN-5 compiler.
COBOL5	run the COBOL5 compiler.
COMPASS	run the CPU/PPU ASSembler for the CYBER 170.
ASM180	run the 180 ASseMbler for the CYBER 170.
CPAS180	run the CPu ASsembler for the CYBER 180.
PPAS180	run the PPu ASsembler for the CYBER 180.
LINK170	link relocatable binaries to form an absolute program.
XREFMAP	generate a cross-reference from a load map.
GETCCDB	gets the CYBIL CC Interactive Debugger ready for use.

9.0 COMPILING, LINKING, AND DEBUGGING

GETLIB(S) get CYBIL run time library local to the job.

CYBENV run the CYBIL environment.

9.1 CYBIL - RUN CYBIL COMPILER

CYBIL runs the compiler for the CDC CYBER IMPLEMENTATION LANGUAGE. The CC compiler is available to generate code to be run on a CYBER 170 machine. The CI compiler is available to generate code to be run on a CYBER 180 machine. The CM compiler is available to generate code to be run on a Motorola 68000 microprocessor. The CP compiler is available to generate UCSD pcode. The CS compiler is available to generate code to be run on a CYBER 200 machine.

The procedure is arranged so that it does not abort. If there are any compile-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the compiler.

Parameters to the CYBIL procedure are :

i or f :
(optional) name of Input File to be compiled. If you don't code the i parameter, CYBIL assumes that the input is on a file called **compile**.

l :
(optional) name of file to receive the Listing from the compiler. If you don't code the l parameter, the compiler output appears on a file called **listing**. This listing file is not rewound by the CYBIL procedure.

b :
(optional) name of file to receive the Binary object code generated by the compiler. If you don't code the b parameter, the binaries appear on a file called **bgo**. This **bgo** file is not rewound by the CYBIL procedure. If you give a value of null to the b parameter, no binary is generated.

cc or ci or cm or cp or cs :
(optional) key determines the compiler used. By default, the CYBIL CC compiler is used. If you code the **ci** key, the CYBIL CI compiler is used. If you code the **cm** key, the CYBIL CM compiler is used. If you code the **cp** key, the CYBIL CP compiler is used. If you code the **cs** key, the CYBIL CS compiler is used. Note : you can set your own default compiler selection in your profile by setting

a variable called cybil to this key.

Note : this parameter is used to select a version of the CC or CI or CM or CP or CS compiler other than the standard one released via the SES. This is done by coding as a value for this parameter the user name of the owner of the compiler you're after. For example, entering cc=lp3 accesses the version of the CC compiler residing in the LP3 catalog. It is necessary for the owner of such a compiler to use the file name CYBILC for the CC version or CYBILI for the CI version or CYBILM for the CM version or CYBILP for the CP version or CYBILS for the CS version in order to allow the compiler to be accessed via this procedure. For the CS version, the file CSTOSS must also be present in the owner's catalog.

chk or rc :

this (optional) parameter specifies the run-time checks the compiler is to generate. If you code no for this parameter, no run-time checking code is generated. If you code any combination of n (check for de-reference of NIL pointer), r (check value ranges--range checking may add as much as 3 1/2 words to the generated object code for each range check performed) and s (check subscript ranges), the corresponding checking code is generated and omitted checking options cause the corresponding checks to be suppressed. If you omit the chk parameter, CYBIL generates all flavors of run-time checks.

lo :

this (optional) parameter specifies the Options to be used when producing the Listing. Any combination of the following designators may be coded for this parameter :

a specifies that a map of the source input block structure and stack is to be produced

f specifies that a full listing is to be produced (selects list options S, A, and R)

o specifies that the listing should include the generated object code in an assembler-like format

s specifies that the listing of the source input file is to be produced

w specifies that the listing of warning diagnostic messages is to be suppressed

r specifies that a symbolic cross reference listing is to be produced

ra specifies that a symbolic cross reference listing of all program entities is to be produced whether referenced or not

x works in conjunction with the LISTEXT pragmat such that LISTings can be EXTernally controlled via the compiler call statement.

If no is specified, no listing is produced. If the lo parameter is omitted, a listing of the source input file only is produced.

debug or d or nodebug :

this (optional) parameter is used to select the debug options to be in effect for the compilation. The available debug options are :

sd If this option is selected, the compiler includes Symbol Table information in the object code file to be used by the CYBIL interactive debugger. If this option is not selected it isn't possible to symbolically debug the module(s) being compiled.

fd This option produces the same tables as d=sd plus stylizes the generated code to provide a step facility when using the symbolic debugger (it is only meaningful, currently, when used with the CC compiler).

ds If this option is selected the Debugging Statements appearing between ?? nocompile ?? and ?? compile ?? pragmats is compiled, otherwise such statements are skipped by the compiler.

If you omit this parameter or you code the nodebug key, none of the sd, fd or ds options are selected. If you code the debug or d keys but don't code any values, debug=sd is used. To select combined options you can code debug=(sd,ds) or debug=(fd,ds).

pad :

coding a numeric value for this (optional) pad parameter generates the specified number of no-ops between machine instructions. This feature is used for the P3 diagnostics.

opt :

this (optional) parameter is used to select optimization levels in the generated code (not supported for all compiler implementations). Legitimate values are :

0 Provides for keeping constant values in registers.

1 Provides for keeping local variables in registers.

2 Eliminate redundant memory references and common subexpressions.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Example of CYBIL Usage

```
ses.cybil lo='or', chk='sn'
REVERT.    END CYBIL  COMPILE -> LISTING, LGO
```

This example shows the CYBIL CC compiler being used to process the program on file COMPILE. A listing is produced consisting of the source input, generated object code, and a symbolic cross reference, on file LISTING. The object code is written to file LGO and will include run-time checking code for Subscript ranges and NIL pointer de-reference.

9.2 ISWL - RUN ISWL CC COMPILER

ISWL runs the ISWL CC compiler.

The procedure is arranged so that if there are any compile-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the compiler.

Parameters to the ISWL procedure are :

i or f :
(optional) name of Input File to be compiled. If you don't code the i parameter, ISWL assumes that the input is on a file called **compile**.

l :
(optional) name of file to receive the Listing from the compiler. If you don't code the l parameter, the compiler output appears on a file called **listing**. This **listing** file is not rewound by the ISWL procedure.

b :
(optional) name of file to receive the Binary object code generated by the compiler. If you don't code the **b** parameter, the binaries appear on a file called **lgo**. This **lgo** file is not rewound by the ISWL procedure.

lo :
if this (optional) parameter is specified as **r**, a symbolic cross Reference map is produced on the file specified by the l parameter. (R is the only List Option available with this ISWL procedure.) If you don't code this parameter, ISWL does not produce a cross reference.

small or medium or large :
these (optional) keys are used to select the size of the ISWL compiler to be used. As you might have already guessed, three sizes are available. The default is **medium** but this can be over-ridden by defining in your profile a variable called **iswlsiz** set to one of these keys. The compiler size determines how large a module can be compiled.

debug or nodebug :
these (optional) keys select whether you want the compiler to generate (in the object code file) information for the ISWL DEBUGger which runs under the SES SUBSYStem. If you code the **debug** key, ISWL generates the debug information. If you code the **nodebug** key or if you don't code either of these keys, ISWL does not generate the debug information.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

9.3 SYMPL - RUN THE SYMPL COMPILER

This procedure runs the SYMPL compiler for the CYBER 170.

The procedure is arranged so that if there are any compile-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the compiler.

Parameters to the SYMPL procedure are :

i or f :
: (optional) name of Input File to be compiled. If you don't code the i parameter, SYMPL assumes that the input is on a file called **compile**.

l :
: (optional) name of file to receive the Listing from the compiler. If you don't code the l parameter, the compiler output appears on a file called **listing**. This **listing** file is not rewound by the SYMPL procedure.

b :
: (optional) name of file to receive the Binary object code generated by the compiler. If you don't code the b parameter, the binaries appear on a file called **lgo**. This **lgo** file is not rewound by the SYMPL procedure.

lo :
: this (optional) parameter is used to specify the List Options for the compilation (see the SYMPL reference manual for legal list options). If you don't code this parameter, SYMPL uses only list option L.

xp :
: this (optional) parameter is used to pass eXtra Parameters to the compiler (see the SYMPL reference manual for a list of allowed SYMPL control statement parameters). If you want to pass extra parameters, the list of them must be specified as a string (within single quotes). The SYMPL procedure does not check the validity of the parameters in the string. If you don't code this parameter, no extra parameters are passed to the compiler.

debug or nodebug :
these (optional) keys select whether you want the compiler to generate information for the CYBER Interactive Debugger (CID). If you code the **debug** key, SYMPL includes the debug information in the object code file. If you code the **nodebug** key or if you don't code either of these keys, SYMPL doesn't generate the debug information.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

9.4 FTN - RUN THE FTN (FORTRAN 4) COMPILER

This procedure runs the FTN (ForTraN extended) compiler for the CYBER 170.

The procedure is arranged so that if there are any compile-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the compiler.

Parameters to the FTN procedure are :

i or f :

(optional) name of Input File to be compiled. If you don't code the i parameter, FTN assumes that the input is on a file called **compile**.

l :

(optional) name of file to receive the Listing from the compiler. If you don't code the l parameter, the compiler output appears on a file called **listing**. This **listing** file is not rewound by the FTN procedure.

b :

(optional) name of file to receive the Binary object code generated by the compiler. If you don't code the b parameter, the binaries appear on a file called **lgo**. This **lgo** file is not rewound by the FTN procedure.

fl :

this (optional) parameter specifies the Field Length to be used while running the compiler. If you don't code this parameter, FTN uses a field length of 70000(8).

xp :

this (optional) parameter is used to pass eXtra Parameters to the compiler (see the FTN reference manual for a list of allowed FTN control statement parameters). If you want to pass extra parameters, the list of them must be specified as a string (within single quotes). The FTN procedure does not check the validity of the parameters in the string. If you don't code this parameter, no extra parameters are passed to the compiler.

debug or **nodebug** :
these (optional) keys select whether you want the compiler to generate information for the CYBER Interactive Debugger (CID). If you code the **debug** key, FTN includes the debug information in the object code file. If you code the **nodebug** key or if you don't code either of these keys, FTN doesn't generate the debug information.

msg or **nomsg** :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

9.5 FTN5 - RUN THE FTN5 (FORTRAN-5) COMPILER

This procedure runs the FTN5 (ForTraN-5, also known as Fortran-77) compiler for the CYBER 170.

The procedure is arranged so that if there are any compile-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the compiler.

Parameters to the FTN5 procedure are :

i or f :
: (optional) name of Input File to be compiled. If you don't code the i parameter, FTN5 assumes that the input is on a file called **compile**.

l :
: (optional) name of file to receive the Listing from the compiler. If you don't code the l parameter, the compiler output appears on a file called **listing**. This listing file is not rewound by the FTN5 procedure.

b :
: (optional) name of file to receive the Binary object code generated by the compiler. If you don't code the b parameter, the binaries appear on a file called **lgo**. This lgo file is not rewound by the FTN5 procedure.

fl :
: this (optional) parameter specifies the Field Length to be used while running the compiler. If you don't code this parameter, FTN5 uses a field length of 70000(8).

xp :
: this (optional) parameter is used to pass eXtra Parameters to the compiler (see the FTN5 reference manual for a list of allowed FTN5 control statement parameters). If you want to pass extra parameters, the list of them must be specified as a string (within single quotes). The FTN5 procedure does not check the validity of the parameters in the string. If you don't code this parameter, no extra parameters are passed to the compiler.

debug or nodebug :
these (optional) keys select whether you want the compiler to generate information for the CYBER Interactive Debugger (CID). If you code the **debug** key, FTN5 includes the debug information in the object code file. If you code the **nodebug** key or if you don't code either of these keys, FTN5 doesn't generate the debug information.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

9.6 COBOL5 - RUN THE COBOL5 COMPILER

This procedure runs the COBOL5 compiler for the CYBER 170.

The procedure is arranged so that if there are any compiler-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the compiler.

Parameters to the COBOL5 procedure are :

i or f :
(optional) name of Input File to be compiled. If you don't code the i parameter, COBOL5 assumes that the input is on a file called **compile**.

l :
(optional) name of file to receive the Listing from the compiler. If you don't code the l parameter, the compiler output appears on a file called **listing**. This **listing** file is not rewound by the COBOL5 procedure.

b :
(optional) name of file to receive the Binary object code generated by the compiler. If you don't code the b parameter, the binaries appear on a file called **lgo**. This **lgo** file is not rewound by the COBOL5 procedure.

fl :
this (optional) parameter specifies the Field Length to be used while running the compiler. If you don't code this parameter, COBOL5 uses a field length of 70000(8).

xp :
this (optional) parameter is used to pass extra parameters to the compiler (see the COBOL5 reference manual for a list of allowed COBOL5 control statement parameters). If you want to pass extra parameters, the list of them must be specified as a string (within single quotes). The COBOL5 procedure does not check the validity of the parameters in the string. If you don't code this parameter, no extra parameters are passed to the compiler.

debug or nodebug :
these (optional) **keys** select whether you want the compiler to generate information for the CYBER Interactive Debugger (CID). If you code the **debug key**, COBOL5 includes the debug information in the object code file. If you code the **nodebug key** or if you don't code either of these keys, COBOL5 doesn't generate the debug information. Note - this is not the COBOL5 debug option which can be selected via the DB parameter. This COBOL5 debug option can be selected by using the xp parameter.

msg or nomsg :
these (optional) **keys** control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

9.7 COMPASS - RUN THE COMPASS ASSEMBLER

COMPASS runs the CPU/PPU assembler (COMPASS) for the CYBER 170.

The procedure is arranged so that if there are any assemble-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the assembler.

Parameters to the COMPASS procedure are :

i or f :
(optional) name of Input File to be assembled. If you don't code the i parameter, COMPASS assumes that the input is on a file called **compile**.

l :
(optional) name of file to receive the Listing from the assembler. If you don't code the l parameter, the assembler output appears on a file called **listing**. This **listing** file is not rewound by the COMPASS procedure. (Note that the assembler's control statement parameter O is set to the same file as specified by the L parameter only when the **nomsg** option (see below) is specified.)

b :
(optional) name of file to receive the Binary object code generated by the assembler. If you don't code the b parameter, the binaries appear on a file called **lgo**. This lgo file is not rewound by the COMPASS procedure.

fl :
this (optional) parameter specifies the Field Length to be used while running the assembler. If you don't code this parameter, COMPASS uses a field length of 70000(8).

xp :
this (optional) parameter is used to pass eXtra Parameters to the assembler (see the COMPASS reference manual for a list of allowed control statement parameters). If you want to pass extra parameters, the list of them must be specified as a string (within single quotes). The COMPASS procedure does not check the validity of the parameters in

the string. If you don't code this parameter, no extra parameters are passed to the assembler.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

9.8 ASM180 - EXECUTE THE 180 ASSEMBLER ON 170

ASM180 executes the 180 assembler on 170. This assembler shares a large percentage of common code with the one which executes in native mode. It produced object modules which may be bound.

Parameters to the ASM180 procedure are :

i or input :
(optional) name of the Input file to be assembled. If you don't code the i parameter, ASM180 assumes that the input is on a file called **input**.

b or binary :
(optional) name of the file to receive the object code generated by the assembler. If you don't code the b parameter the object code will be written to a file called **igo**. This file is not rewound by the ASM180 procedure.

l or listing :
(optional) name of the file to receive the Listing from the assembler. If you don't code the l parameter the assembler output appears on a file called **listing**. This file is not rewound by the ASM180 procedure.

x or xref :
(optional) boolean parameter selecting a cross reference listing. If you don't code this parameter no cross reference listing is produced.

c or checks :
(optional) boolean parameter selecting whether the assembler checks for correct register types. If you don't code this parameter the checks will be performed.

cvt or convert :
(optional) If you don't code this parameter the file specified by the i parameter will be processed directly by the assembler. If you give only the **cvt** parameter name the file specified by the i parameter will be passed through a program to convert the source statements from the form expected by CPAS180 to the form processed by this assembler. If you give the **cvt** parameter name and a file

name as its value that file will receive the converted source statements.

st or status :
(optional) If this parameter is given a value of EFG or R1G, those job control registers will be set to zero for no assembly errors and non-zero if there are assembly errors.

msg or nomsg :
These (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

9.9 CPAS180 - RUN THE CPU ASSEMBLER FOR THE CYBER 180

CPAS180 runs the CYBER 180 CPU assembler. The procedure has two modes of operation :

1. "normal mode" in which object code is generated from the source program
2. "systext mode" in which a "system text" file is produced (containing pre-assembled PROCs, etc.) that is used later as input to "normal mode" assemblies (see the BLDSYST keyword parameter described below)

The procedure is arranged so that if there are any compile-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the assembler.

Parameters to the CPAS180 procedure are :

i or f :
(optional) name of Input File to be assembled. If you don't code the i parameter, CPAS180 assumes that the input is on a file called **compile**.

l :
(optional) name of file to receive the Listing from the assembler. If you don't code the l parameter, the assembler output appears on a file called **listing**. This listing file is not rewound by the CPAS180 procedure.

b :
(optional) name of file to receive the Binary object code generated by the assembler. If you don't code the b parameter, the binaries appear on a file called **lgo**. This lgo file is not rewound by the CPAS180 procedure.

chk :
this (optional) parameter determines whether the assembler checks for correct register types and instruction formats. If you code **no** for this parameter, no checking of types and formats is done and the assembler runs faster. If you code **yes** for this parameter or if you omit the parameter, the assembler performs the checks.

s :

this (optional) parameter is used to specify the name of a System text file. In "normal mode" this parameter is used to provide an alternate to the standard "SYSTEXT" used by the assembler. In the "build system text mode", this parameter specifies the name of the file on which the system text is written (default is SYSTEXT).

fl :

this (optional) parameter specifies the Field Length to be used while running the assembler. If you don't code this parameter, CPAS180 uses a field length of 70000(8).

bldsyst :

if you code this (optional) key, CPAS180 runs in the "BuiLD SYStem Text" mode. If you don't code this key, CPAS180 runs in "normal" mode.

msg or nomsg :

these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

9.10 PPAS180 - RUN THE PPU ASSEMBLER FOR THE CYBER 180

PPAS180 runs the CYBER 180 PPU assembler. This assembler is a modified version of standard CYBER 170 COMPASS so that usage of the two assemblers is very similar.

The procedure is arranged so that if there are any compile-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the assembler.

Parameters to the PPAS180 procedure are :

i or f :
(optional) name of Input File to be assembled. If you don't code the i parameter, PPAS180 assumes that the input is on a file called **compile**.

l :
(optional) name of file to receive the Listing from the assembler. If you don't code the l parameter, the assembler output appears on a file called **listing**. This **listing** file is not rewound by the PPAS180 procedure. (Note that the assembler's control statement parameter O is set to the same file as specified by the L parameter only when the **nomsg** option (see below) is specified.)

b :
(optional) name of file to receive the Binary object code generated by the assembler. If you don't code the b parameter, the binaries appear on a file called **lgo**. This **lgo** file is not rewound by the PPAS180 procedure.

fl :
this (optional) parameter specifies the Field Length to be used while running the assembler. If you don't code this parameter, PPAS180 uses a field length of 70000(8).

xp :
this (optional) parameter is used to pass eXtra Parameters to the assembler (see the COMPASS reference manual for a list of allowed control statement parameters). If you want to pass extra parameters, the list of them must be

specified as a string (within single quotes). The PPAS180 procedure does not check the validity of the parameters in the string. If you don't code this parameter, no extra parameters are passed to the assembler.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

conv or convert :
these (optional) keys specify that the output binary object code file (specified by the b parameter) is to be connected to the NOS/170 object text format that is described in the CYBER LOADER REFERENCE MANUAL. If this keyword is omitted the binary object code file produced is in the NOS/VE object text format.

9.11 LINK170 - LINK RELOCATABLE BINARIES

LINK170 provides an easy way to use the CYBER 170 Loader. Although the CYBER 170 Loader is very powerful and flexible, it suffers from the disadvantage that you can't talk to it interactively, since it reads ahead in the control statement file, and that means that you can only drive it via canned sequences of JCL. This procedure provides a simple to use interface to the loader which allows for most common types of linking. When using CYBIL screen formatting procedures, a 'LIBRARY,SFLIB.' statement should precede the call to LINK170. The procedure is arranged so that if any link errors are detected, the NOS job control register EFG is set to a non-zero value. This allows the procedure to be used as a building block for other procedures. Parameters to the LINK170 procedure are :

f :

(optional) list of Files containing the relocatable binaries that you want LINKed. If you don't code the f parameter, LINK170 assumes that the relocatables are on file lgo. The f parameter may be coded as a multi valued list of sublists. Each element of the value list is either a single name, in which case it refers to the name of an object code file already assigned to the job, or in the current user's catalog, or an element is a sublist, in which case, the last sub element in the sublist is a user name in whose catalog the object code files referred to by the other sub elements of the sublist may be found. The example at the end of this description should (with luck) make this clear. Note that files for which a user name is specified are returned when the link is completed.

b :

(optional) name of file to receive the LINKed program. If you don't the b parameter, LINK170 places the linked program on file lgob.

p :

(optional) list of library files from which external references are to be satisfied. The p parameter may be coded as a multi valued list of sublists. Each element of the value list is either a single name, in which case it refers to the name of an user library already assigned to the job, or in the current user's catalog, or an element is a sublist, in which case, the last sub element in the sublist is a user name in whose catalog the libraries

referred to by the other sub elements of the sublist may be found. The example at the end of this description should (with luck) make this clear. Note that libraries for which a user name is specified are returned when the link is completed.

l :

(optional) name of file to receive the map Listing from LINK170. If you don't code the l parameter, the loader uses the default map file name (normally output).

lo :

(optional) List Options for the LINK170 process. If you don't code the lo parameter, the loader uses its default list options. If you simply code lo without any values supplied, LINK170 sets default values lo=sb. If you code lo=full, LINK170 turns on FULL list options (which are lo=sbex). Otherwise, LINK170 sets whatever list options you say.

ep :

(optional) specifies a list of Entry Points to be included in the header of a multiple entry point "overlay". This list is specified as a string exactly as the list should appear on the NOGO loader directive (a string is used to allow the specification of so-called "special" entry points which have a syntax unacceptable to SES).

xld :

(optional) list of eXtra Loader Directives. This parameter takes the form of a list of character strings which are inserted into the control statement stream, in addition to those normally generated, thus allowing you to supply extra directives over and above the standard ones.

debug :

(optional) controls whether the linked program includes the CYBIL INTERACTIVE DEBUGger (CCDBG) or the CYBER INTERACTIVE DEBUGger (CID). If you code the debug parameter, the program will be able to be run under control of the requested debugger (by preceding a call to the program with the control statement DEBUG(ON)). Legal values are CCDBG and CID. If no value is given, CCDBG is used. NOTE: if DEBUG is specified, the tables used by the debugger are placed on file ZZZZZDT. This file must also

be saved and made local if the absolute binary is to be used at a later time.

iswllib or cybclib :
(optional) **key** provides a convenience when linking programs written in the CYBIL or ISWL language. If you are linking such a program, you are advised to use the appropriate **key** for the language in question, since such usage causes the LINK170 procedure to acquire the necessary library and generate any other necessary loader directives associated with LINKing a program in the designated language.

el or errlevl :
this (optional) parameter controls the Error Level at which the loader aborts without producing any absolutes. Valid values for the el parameter are :

NONE only "terminal" errors cause an abort.
FATAL only fatal errors cause an abort.
ALL all errors (both fatal and nonfatal) cause an abort.
ALL is the **default** for LINK170.

A default value for the el parameter may be set up via the errlevl profile variable if you don't wish to keep overriding the default by coding the parameter.

Examples of LINK170 Usage

```
ses.link170 b=txtform, l=mapform, lo
REVERT.    END LINK170    TXTFORM
```

This example shows LINK170 used to link a program. The relocatable binaries are assumed to be on file lgo, and the LINKed program appears on file txtform. A partial load map is written to file mapform.

```
ses.link170 p=(mylib,yourlib) themap full cybclib debug
REVERT.    END LINK170    LGOB,ZZZZZDT
```

This example illustrates the use of multiple user libraries and in particular the use of the CYBIL run-time library. The relocatable binaries are assumed to be on file lgo, the linked

program (with CYBIL INTERACTIVE DEBUG) is written to file **igob** and a full load map is written to file **themap**. Debug symbol tables are written to file **zzzzzdt**.

```
ses.link170 b=abs, p=(somelib,(utillib,jbf)), .
..? f=((relo, oreo, jee))
REVERT.    END LINK170    ABS
```

This example shows files **relo** and **oreo**, belonging to user **jee**, being linked using library **somelib**, which is either local to the job or belongs to the current user, and library **utillib** which belongs to user **jbf**. The linked program is written to file **abs**. Files **relo**, **oreo**, and **utillib** are returned when the link is completed, but file **somelib** is left local.

9.12 XREFMAP - GENERATE A CROSS_REFERENCE FROM A LOAD MAP

XREFMAP is used to generate a neat cross-reference from a load map. XREFMAP outputs the cross-reference in 3 formats: module names with external references, module name with it's contained entry points and which external modules reference each entry point, and a list of all entry points with containing module name and who references that entry point. Parameters to XREFMAP are :

i or b :
name of load map which is to be cross-referenced. The load map must have been created with, at least, the LO=EX option on the LINK170 call.

un :
(optional) user name of catalog to search for input file. The default is the current user.

l :
(optional) name of the output file to receive the output from XREFMAP. If you don't code the l parameter, a default listing file name of LISTING is used.

print or prt :
this (optional) key controls the automatic printing of the output file.

batch job parameters :
These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in batch mode, but it can be run in local mode.

9.13 GETCCDB - GET CYBIL INTERACTIVE DEBUG

GETCCDB provides an alternate way to acquire CYBIL INTERACTIVE DEBUG (CCDBG). Normally, CCDBG is acquired when necessary by the LINK170 procedure. If the LINK170 procedure is not to be used in the current session, though, the necessary files must be explicitly attached. There are no parameters to the GETCCDB procedure.

Example of GETCCDB Usage

```
ses.getccdb
REVERT.  CYBIL INTERACTIVE DEBUG AVAILABLE
```

9.14 GETLIB OR GETLIBS - ACQUIRE LIBRARY FOR LINKING

GETLIB(S) acquires the CYBCLIB or CYBILIB library ready for linking. Parameters to GETLIB(S) are :

cybclib or **cybilib** :
(optional) key indicates the library required to be made local. Coding the **cybclib** key acquires the CYBIL-CC library. Coding the **cybilib** key acquires the CYBIL-CI library.

Examples of GETLIB(S) Usage

```
ses.getlib  cybclib
REVERT.     END GETLIB  CYBCLIB IS LOCAL
```

```
ses.getlibs  cybilib
REVERT.      END GETLIBS  CYBILIB IS LOCAL
```

9.15 CYBENV - RUN CYBIL ENVIRONMENT

CYBENV provides a CYBIL environment with many options. Facilities include source code creation and modification using FSE, source management, source formatting, compiling, and FSE interfaces in split screen for error correction.

create or **modify** or **compile** or **cybil_to_sct** or **sct** or **add** or **get** or **list_base** or **delmod** or **delcom** :
(required) function name of CYBENV option to execute.

create - Lets you create the deck(s) specified by the m parameter using FSE and then adds the deck(s) to the specified base.

modify - Lets you modify the deck(s) specified by the m parameter using FSE and then replaces the deck(s) to the specified base.

compile - Compiles the deck(s) specified by the m parameter. If there are compilation errors, you have the option of correcting the errors in the source using FSE in a split screen mode. The top part of the screen will be the source and the bottom half will be the listing with the error message(s) under the line(s) with the error(s). You can also read the online CYBIL manual by pressing the ASSIST function key in FSE.

cybil_to_sct or **sct** - Produces structure charts of the specified deck(s). There can be no compilation errors in the deck(s) specified or the structure charts will not be generated. You have an option of using the CADSG procedure to look at the structure charts produced.

get - Gets deck(s) from the base specified and puts the deck(s) on the file specified by the g parameter.

add - Adds deck(s) on the file specified by the g parameter to the base specified.

delmod - Deletes deck(s) from the base specified.

delcom - Deletes common deck(s) from the base specified.

Parameters to the CYBENV procedure are :

m or **all** :

This (required) parameter specifies a list of modules (including ranges of modules) that will be affected by the option specified. If the **all** key is coded, a compile file for ALL of the (regular) modules in the specified base(s) is produced.

b or l :
(optional) name of Base Library from which to generate the compile file. If you don't code the b parameter, CYBENV uses the value of profile variable **base** as the name of the base library, and if there's no such variable defined, CYBENV uses the name **base**.

un :
(optional) User Name in whose catalog the base specified by b may be found. If you don't code the un parameter, CYBENV uses the value of profile variable **baseown** as the name of the base library, and if there's no such variable defined, CYBENV assumes that the base is in the current user's catalog.

Note : If the User Name is different than the current user, and using the **modify** or **create** options, CYBENV will place the deck(s) on a base library that is the value of the profile variable **base**, or the name **base** if no profile variable exists. The specified base library in the different catalog will then be added as an alternate base.

g :
(optional) name of file to receive the modules extracted from the base. If you don't code the g parameter, CYBENV uses the name **group** as the name of the file.

cors or c :
(optional) name of file containing CORrection Sets to be temporarily applied against the base b prior to generating the compile file. If you code the cors parameter, CYBENV uses procedure TEMPCOR to make the temporary corrections before generating the compile file.

ab or al :
(optional) list of Alternate Bases from which to satisfy calls to common modules. The ab parameter may be coded as a multi valued list of sublists. Each element of the

value list is either a single name, in which case it refers to the name of an alternate base already assigned to the job, or in the current user's catalog, or an element in a sublist, in which case, the last sub element in the sublist is a user name in whose catalog the bases referred to by the other sub elements of the sublist may be found. See the example at the end of the description of GENCOMP to help make this clear.

cybccmn or cybicmn :
(optional) key specifies an SES supplied Alternate Base for use in satisfying calls to common modules (see the **ab** parameter described above). If you code the **cybccmn** key, the base containing CoMmoN modules for use by CYBIL CC programs is selected. If you code the **cybicmn** key, the base containing CoMmon modules for use by CYBIL CI programs is selected. By default, **cybccmn** is selected for use in generating the compile file.

cc or ci or cm or cp or cs :
(optional) key determines the compiler used. By default, the CYBIL CC compiler is used. If you code the **ci** key, the CYBIL CI compiler is used. If you code the **cm** key, the CYBIL CM compiler is used. If you code the **cp** key, the CYBIL CP compiler is used. If you code the **cs** key, the CYBIL CS compiler is used. Note : you can set your own default compiler selection in your profile by setting a variable called **cybil** to this key.

lo :
this (optional) parameter specifies the Options to be used when producing the Listing. Any combination of the following designators may be coded for this parameter :

a specifies that a map of the source input block structure and stack is to be produced

f specifies that a full listing is to be produced (selects list options S, A, and R)

o specifies that the listing should include the generated object code in an assembler-like format

s specifies that the listing of the source input file is to be produced

w specifies that the listing of warning diagnostic

messages is to be suppressed

r specifies that a symbolic cross reference listing is to be produced

ra specifies that a symbolic cross reference listing of all program entities is to be produced whether referenced or not

x works in conjunction with the LISTEXT pragmat such that LISTings can be EXTernally controlled via the compiler call statement.

If no is specified, no listing is produced. If the lo parameter is omitted, a listing of the source input file only is produced.

chk or nochk :
this (optional) parameter specifies the run-time checks the compiler is to generate. If you code no for this parameter or nochk, no run-time checking code is generated. If you code any combination of n (check for de-reference of NIL pointer), r (check value ranges--range checking may add as much as 3 1/2 words to the generated object code for each range check performed) and s (check subscript ranges), the corresponding checking code is generated and omitted checking options cause the corresponding checks to be suppressed. If you omit the chk parameter, CYBIL generates all flavors of run-time checks.

debug or d or nodebug :
this (optional) parameter is used to select the debug options to be in effect for the compilation. The available debug options are :

sd If this option is selected, the compiler includes Symbol Table information in the object code file to be used by the CYBIL interactive debugger. If this option is not selected it isn't possible to symbolically debug the module(s) being compiled.

fd This option produces the same tables as d=sd plus stylizes the generated code to provide a step facility when using the symbolic debugger (it is only meaningful, currently, when used with the CC compiler).

ds If this option is selected the Debugging Statements appearing between ?? nocompile ?? and ?? compile ?? pragmats is compiled, otherwise such statements are skipped by the compiler.

If you omit this parameter, all the debug options are selected. If you code the nodebug key, none of the sd, fd or ds options are selected. If you code the debug or d keys but don't code any values, debug=sd is used. To select combined options you can code debug=(sd,ds) or debug=(fd,ds).

opt :

this (optional) parameter is used to select optimization levels in the generated code (not supported for all compiler implementations). Legitimate values are :

0 Provides for keeping constant values in registers.

1 Provides for keeping local variables in registers.

2 Eliminate redundant memory references and common subexpressions.

bin :

(optional) name of file to receive the Binary object code generated by the compiler. If you don't code the bin parameter, the binaries appear on a file called lgo. If you give a value of null to the bin parameter, no binary is generated.

10.0 CYBER 180 VIRTUAL ENVIRONMENT CREATION AND SIMULATION

10.0 CYBER 180 VIRTUAL ENVIRONMENT CREATION AND SIMULATION

"Simulation ain't what it's going to be"
New Scientist

This chapter describes the SES utilities for creating and using the CYBER 180 Virtual Environment on the CYBER 170. Facilities described in this chapter are :

SIM180 run the CYBER 180 Hardware System Simulator.

TO180 convert a NOS/170 text file for use with the simulated NOS/VE I/O available through the SIM180 procedure.

TO170 NOS/170 to NOS/170 INTERFACE file conversion.

FROM180 convert a simulated NOS/VE I/O text file to a NOS/170 text file.

FROM170 converts NOS/170 INTERFACE format files to NOS/170 format.

DUMP180 dumps the contents of a simulated NOS/VE file.

DUMP170 dumps the contents of a NOS/170 INTERFACE format file.

GENCPF link CYBER 180 binaries and generate a Checkpoint File.

VELINK link CYBER 180 binaries and produce Segment Files.

VEGEN generate a CYBER 180 Checkpoint File from Segment Files.

GETLIB(S) get CYBIL run time library local to job.

GETDSI make HCS IOU Deadstart tape generator local to job.

10.0 CYBER 180 VIRTUAL ENVIRONMENT CREATION AND SIMULATION

GETLDB make HCS NOS Environment Interface Loader local to job.

10.0 CYBER 180 VIRTUAL ENVIRONMENT CREATION AND SIMULATION
10.1 SIM180 - RUN THE CYBER 180 HARDWARE SYSTEM SIMULATOR

10.1 SIM180 - RUN THE CYBER 180 HARDWARE SYSTEM SIMULATOR

SIM180 runs the CYBER 180 Hardware System Simulator (often referred to as the HSS). The hardware system that is simulated consists of the following components :

1. CPU
2. Central Memory (16 Megabytes)
3. 10(10) Peripheral Processors (PPs)
4. 12(10) I/O Channels
5. 2 7154 disk controllers sharing 4 844-4x disk storage units.
6. 2 7155-1 disk controllers sharing 2 844-4x and 2 885-1x disk storage units.
7. 2 7155-14 disk controllers sharing 2 844-4x, 2 885-1x and 2 885-42 disk storage units.

SIM180 sets up the total system simulated environment and gives control to the CYBER 180 simulator. At this point, the user may enter any valid simulator command. Simulator commands provide the user with control over the simulated environment which includes the following capabilities :

1. CPU and PP memory loading
2. Interpretive CPU and PP instruction execution
3. CPU and PP breakpoint
4. CPU and PP trace
5. Display and change Central and PP memories
6. Dump Central and PP memories to a specified file
7. Display and change CPU and PP registers
8. Checkpoint / Restart
9. On/off CPU and PP simulation
10. CPU keypoint
11. CPU program performance monitoring

The CYBER 180 simulator is intended primarily for interactive use (local mode) but may be readily used in batch. An example of batch submission of a Simulator run is given below. Parameters to SIM180 are :

restart or rs :
(optional) name of a checkpoint file from which to restart simulation. The checkpoint file may be one previously created by GENCPF (GENerate CheckPoint File), VEGEN (Virtual Environment GENerator), or via the simulator CHECKPOINT command. If you don't code the restart parameter, a complete simulated system is created.

10.0 CYBER 180 VIRTUAL ENVIRONMENT CREATION AND SIMULATION
10.1 SIM180 - RUN THE CYBER 180 HARDWARE SYSTEM SIMULATOR

cf :
(optional) name of a command file which is to be processed immediately upon activation of the simulator. The command file consists of one or more valid simulator command(s). This parameter is required when running in batch.

i180 or i :
(optional) name of file containing input for the special simulator CPU I/O instruction (op code FF(16)). If you're running SIM180 interactively and don't code the i180 parameter, SIM180 takes its input from a file assigned to your terminal. If you run SIM180 in batch and use the CPU I/O instruction for input, you must supply a file containing the input.

o180 or o :
(optional) name of file to receive output from the special simulator CPU I/O instruction (op code FF(16)). If you're running SIM180 interactively and don't code the o180 parameter, SIM180 sends output to a file assigned to your terminal. If you run SIM180 in batch, you must supply a file to receive the output, otherwise the output is lost.

Examples of SIM180 Usage

```
ses.sim180
** I SM 6052: CYBER 180 SIMULATOR VER 6.7 LEV 133 (MIGDS REV S)
```

This example shows SIM180 used interactively, to create a new simulated system environment. The user may enter any simulator command once the simulator banner is displayed.

```
ses.sim180 rs=check1 cf=errata
** I SM 6052: CYBER 180 SIMULATOR VER 6.7 LEV 133 (MIGDS REV S)
```

This example shows SIM180 used to resume simulation from a checkpoint file. The command file errata is processed before requesting additional simulator commands from the user.

```
ses.sim180 checkx i=indata o=outdata
** I SM 6052: CYBER 180 SIMULATOR VER 6.7 LEV 133 (MIGDS REV S)
```

This example shows SIM180 used interactively, to resume simulation from the checkpoint file checkx. The special simulator CPU I/O instruction, when encountered during CPU instruction simulation, is to read input from file indata and is to write output to file outdata.

```
ses.do batchn, file=simcf,.....
..? cs=('ses.sim180 postds, cf=simcf, o=progout',.....
..? 'save,seslog')
  15.54.28. SUBMIT COMPLETE.  JOBNAME IS AAAQCBN
REVERT.    JOB DO  SUBMITTED
```

This example shows SIM180 submitted for batch running via the SES DO procedure. Simulation will resume from the checkpoint file postds, the simulator commands in the file simcf are processed, and any output from the special simulator CPU I/O instruction is directed to file progout. Refer to the description of the SES DO procedure in the "Miscellaneous Useful Goodies" section for an explanation of using DO to submit multiple control statements to batch.

10.2 TO180 - NOS/170 TO SIMULATED NOS/VE FILE CONVERSION

TO180 converts a NOS/170 file containing character data to SIMULATED NOS/VE file format. Only character data can be converted. Parameters of TO180 are:

i or f :
The Input or File parameter specifies the NOS/170 format file to be converted.

o :
The (optional) Output parameter specifies the SIMULATED NOS/VE format file to receive the converted data. If this parameter is omitted then the output and input file names are the same.

cs612 or cs64 :
These (optional) keys indicate whether the NOS/170 format file is upper/lower case ascii (cs612) or whether it is a 64 character set ascii or display code file (cs64). If no keyword is given then cs612 is used.

10.3 T0170 - NOS/170 TO NOS/170 INTERFACE FILE CONVERSION

T0170 converts a NOS/170 file containing character data to the NOS/170 INTERFACE format of simulated NOS/VE I/O. It is similar to T0180 except that the output file content is different. Parameters to T0170 are :

i or f :
Name of the Input File to be converted.

o :
(optional) name of file to receive the Output, that is, the NOS/170 INTERFACE format data. If you don't code the o parameter, the output appears on the file specified by the i parameter.

cs612 or cs64 :
these (optional) keys specify the character set of the input file. T0170 assumes the input file is in cs612 character set.

10.4 FROM180 - SIMULATED NOS/VE TO NOS/170 FILE CONVERSION

FROM180 converts a file in SIMULATED NOS/VE format containing character data to NOS/170 format. Conversion stops whenever an error is found on the SIMULATED NOS/VE format file. Such errors include invalid record headers on the file and non-ASCII data in the records. The NOS/170 format file may be printed with an SES.PRINT command, or input to any other normal NOS/170 program. Parameters of FROM180 are :

i or f :
The Input or File parameter specifies the source data, SIMULATED NOS/VE format file.

o :
The (optional) Output parameter specifies the NOS/170 format file. If this parameter is omitted then Input and Output file names are the same.

cs612 or cs64 :
These keywords indicate the character set of the output, NOS/170 format file. Upper/lower case ASCII is indicated by the keyword **cs612** and the keyword **cs64** indicated a 64 character set ASCII or display code. If this parameter is omitted then **cs612** is used.

10.5 FROM170 - NOS/170 INTERFACE TO NOS/170 FILE CONVERSION

FROM170 converts NOS/170 INTERFACE format file containing character set data to NOS/170 format. Conversion stops when an error is found in the NOS/170 INTERFACE format file. Such errors include invalid record headers in the file, and non ASCII data in the records. The NOS/170 file may be printed via the SES PRINT procedure, or input to any other normal NOS/170 program. Parameters to FROM170 are :

i or f :
name of Input File to be converted.

o :
(optional) name of file to receive the Output of FROM170. If you don't code the **o** parameter, the output appears on the file specified by the **i** parameter.

cs612 or cs64 :
these (optional) keys specify the character set of the output file, and correspond to the same parameters on the FROM180 procedure.

10.6 DUMP180 - SIMULATED NOS/VE FILE DUMP

DUMP180 dumps the contents of a simulated NOS/VE file. It is similar to HEXDMP except that it checks the records and data structure of the file and diagnoses errors which prevent FROM180 from converting the file. Parameters to DUMP180 are :

i or f :
name of the file to be dumped.

o :
(optional) name of file to receive the dump. If you don't code the o parameter, the dump output appears on file output.

cs612 or cs64 :
these (optional) keys specify the character set of the output file, as in the FROM180 procedure.

fw :
(optional) First Word of the file to dump. The default is to start at word 0.

lw :
(optional) Last Word of the file to dump. The default is to dump up to end of information (EOI).

10.7 DUMP170 - NOS/170 INTERFACE FILE DUMP

DUMP170 dumps the contents of a NOS/170 INTERFACE format file. It is similar to DUMP180, except that DUMP170 also checks the record and data structure of the input file and diagnoses errors which prevent FROM170 from converting the file. Parameters to DUMP170 are:

i or f :
name of the Input File to be dumped.

o :
(optional) name of Output file to receive the dump. If you don't code the **o** parameter, the output appears on file **output**.

cs612 or cs64 :
these (optional) keys specify the character set of the output file. DUMP170 assumes the cs612 character set.

fw :
(optional) First Word from which to start dumping. If you don't code the fw parameter, the dump starts from word 0.

lw :
(optional) Last Word to be dumped. If you don't code the lw parameter, the dump goes to end of information (EOI).

10.8 GENCPF - GENERATE A CHECKPOINT FILE (CPF)

GENCPF executes the SES Virtual Environment Linker and the SES Virtual Environment Generator to produce a Checkpoint File (CPF) which can be loaded and executed on the CYBER 180 Hardware System Simulator. The user must provide a file containing CYBER 180 object text, and can optionally provide a Linker Parameter File (LPF), a set of Monitor Segment files, and/or a Virtual Environment Generator DIRective file. The **mf** file contains the map produced by the Virtual Environment Linker and additional information produced by the Virtual Environment Generator.

Note: most of the following parameters aren't needed by the "general public" (examples show general usage).

Parameters to GENCPF are:

ofl : (Filename(s), optional)
Object_File_List - list of up to 10 names of files containing CYBER 180 object text (ver 1.4). This parameter does not have a default.

lfl : (Filename(s), optional)
Library_File_List - list of up to 10 names of Library files containing CYBER 180 object text. This parameter does not have a default.

pep : (String(31), optional)
Primary_Entry_Point - This parameter specifies the entry point at which to start execution. The default is to start execution at the first Transfer symbol encountered.

ns : (String(4), optional)
Name_Seed - This parameter specifies the "name seed" for the Virtual Environment Linker Segment files. The default for this parameter is **segm**. The value of this parameter overrides any Linker Parameter File specification of this field.

mf : (Filename, optional)
Map_File - This parameter specifies the name of the map file. The default for this parameter is **linkmap**.

mo :

this (optional) parameter determines the linker Map Options. mo determines the amount of information output on the linker map. Any value coded for mo overrides the default options chosen by the linker parameter file (LPF). The value of the mo parameter is one of the following characters :

- N no map information; diagnostics are output.
- S Section allocations for every section of every input object module.
- E section allocations plus Entry point names and address assignments.
- M section allocations and entry points plus output segment and common block allocations (this last option is in fact a full linker map).

rewind or norew :

these (optional) keys determine whether to REWIND the linker map file before writing it. The default action is to rewind the map file. The key coded overrides any linker parameter file specifications.

lpf :

(optional) name of Linker_Parameter_File containing Linker parameters that affect the link process. If you don't code the lpf parameter, default values are applied as in the description of the LPF.

cpf : (Filename, optional)

Checkpoint_File - This parameter specifies the name of the file containing the output from the Virtual Environment Generator. The default is cpfile. The value of this parameter overrides any Linker Parameter File specification of this field.

cybilib : (Keyword, optional)

If you code the (optional) key, GENCPF uses CYBILIB to satisfy externals during the Linking process. GENCPF ACQUIRE's CYBILIB from SES and adds it to the Library_File_List. The default is not to use CYBILIB as part of the Link.

dir or ldrdir or d : (Filename, optional)

Directives - This parameter specifies the name of a file

containing directives to the VE Generator. The default is a file containing default directives.

mtrns or mns : (String(4), optional)
Monitor Name_Seed - This parameter specifies the "name seed" of the Virtual Environment Linker Segment files that contain the Monitor code. This parameter is provided so users can have their own Monitor program. The default for this parameter is **mtrx**. The procedure checks the local files, the local PF catalog, and then the SES catalog to get the files.

Examples of GENCPF Usage

The following example shows how to compile a test program written in CYBIL, generate a VE File, and run it on the Simulator.

```
ses.cybil ci i=testprg l=testlst b=testlgo
ses.gencpf ofi=testlgo cpf=testcpf cybillib
ses.sim180 restart=testcpf
? run
? bye
```

The last two lines were Simulator commands.

This example shows how to generate a CPF from several input files of CYBER 180 object text.

```
ses.gencpf ofi=(mylgo1,mylgo2) cpf=mycpf
```

10.9 VELINK - EXECUTE THE VIRTUAL ENVIRONMENT LINKER

VELINK executes the Virtual Environment Linker, which links data from CYBER 180 object files/libraries and produces a map file and a set of SEGMent files. VELINK is set up so that users can specify their own Linker Parameter File (LPF) containing Virtual Environment Linker variables that control the linkage.

Note: VELINK is generally used only in special cases, where GENCPF is inadequate.

Parameters to VELINK are:

ofl : (Filename(s), optional)
Object_File_List - list of up to 10 names of files containing CYBER 180 object text (ver 1.4). This parameter does not have a default.

lfl : (Filename(s), optional)
Library_File_List - list of up to 10 names of Library files containing CYBER 180 object text. This parameter does not have a default.

pep : (String(31), optional)
Primary_Entry_Point - This parameter specifies the entry point at which to start execution. The default is to start execution at the first Transfer symbol encountered.

ns : (String(4), optional)
Name_Seed - This parameter specifies the "name seed" for the Virtual Environment Linker Segment files. The default for this parameter is **segm**. The value of this parameter overrides any Linker Parameter File specification of this field.

mf : (Filename, optional)
Map_File - This parameter specifies the name of the map file. The default for this parameter is **linkmap**. The value of this parameter overrides any Linker Parameter File specification of this field.

mo :

this (optional) parameter determines the linker Map Options. mo determines the amount of information output on the linker map. Any value coded for mo overrides the default options chosen by the linker parameter file (LPF). The value of the mo parameter is one of the following characters :

- N no map information; diagnostics are output.
- S Section allocations for every section of every input object module.
- E section allocations plus Entry point names and address assignments.
- M section allocations and entry points plus output segment and common block allocations (this last option is in fact a full linker map).

rewind or norew :

these (optional) keys determine whether to REWIND the linker map file before writing it. The default action is to rewind the map file. The key coded overrides any linker parameter file specifications.

debug :

code this key if you want the linker to generate a file containing module address and entry point information. The file name is the name seed appended with 'DBG'. If you don't code this key, no file is generated.

lpf :

(optional) name of Linker_Parameter_File containing Linker parameters that affect the link process. If you don't code the lpf parameter, default values are applied as in the description of the LPF.

cybilib : (Keyword, optional)

If you code the (optional) key, VELINK uses CYBILIB to satisfy externals during the linking process. VELINK ACQUIRE's CYBILIB from SES and add it to the Library_File_List. The default is not to use CYBILIB as part of the Link.

Example of VELINK Usage

In the following example the Linker is passed a Linker Parameter File. Object files **york** and **hunt** are linked using **racelib** as an object library.

```
ses.velink ofl=(york,hunt) lfl=mylib lpf=mylpf
REVERT.    END VELINK
```

10.0 CYBER 180 VIRTUAL ENVIRONMENT CREATION AND SIMULATION
10.10 VEGEN - EXECUTE THE SES VIRTUAL ENVIRONMENT GENERATOR

10.10 VEGEN - EXECUTE THE SES VIRTUAL ENVIRONMENT GENERATOR

VEGEN executes the SES Virtual Environment Generator to produce a Checkpoint File (CPF) which can be loaded and executed on the CYBER 180 Hardware System Simulator. The input to the Virtual Environment Generator is the set of Segment files containing the users program, and a default set of segment files that contain a simple monitor program. The monitor program contains code that sets up the exchange packages, stack areas, and also provides a primitive Monitor and Trap Handler routines, and a Termination routine.

Note: VEGEN is generally used only in special cases, where GENCPF is inadequate.

Parameters to VEGEN are:

dir or ldrdir or d : (Filename, optional)
Directives - This parameter specifies the name of a file containing directives to the VE Generator. The default is a file containing default directives.

ns : (String(4), optional)
Name_Seed - This parameter specifies the "name seed" of the Virtual Environment Linker Segment files. The default for this parameter is segm.

mtrns or mns : (String(4), optional)
Monitor Name_Seed - This parameter specifies the "name seed" of the Virtual Environment Linker Segment files that contain the Monitor code. This parameter is provided so users can have their own Monitor program. The default for this parameter is mtrx. VEGEN checks the local files, the current user's catalog, and then the SES catalog to get the files.

cpf : (Filename, optional)
Checkpoint_File - This parameter specifies the name of the file containing the output from the Virtual Environment Generator. The default is cpfile.

mf : (Filename, optional)
Map_File - This parameter specifies the name of the file on which to write the map information. The default is linkmap.

Example of VEGEN Usage

In the following example, the Generator is passed a directive file, a set of segment files containing the test program, a set of monitor files, and produces a CPF and Map file.

```
ses.vegen dir=mydir ns='test' mtrns='mtrx' cpf=mycpf mf=mymap
REVERT.    END VEGEN
```

10.11 GETLIB OR GETLIBS - ACQUIRE LIBRARY FOR LINKING

GETLIB(S) acquires the CYBCLIB or CYBILIB library ready for linking. Parameters to GETLIB(S) are :

cybclib or cybilib :
(optional) key indicates the library required to be made local. Coding the **cybclib** key acquires the CYBIL-CC library. Coding the **cybilib** key acquires the CYBIL-CI library.

Examples of GETLIB(S) Usage

```
ses.getlib  cybclib
REVERT.     END GETLIB  CYBCLIB IS LOCAL

ses.getlibs  cybilib
REVERT.      END GETLIBS  CYBILIB IS LOCAL
```

10.12 GETDSI - ACQUIRE BINARY FOR DEADSTART TAPE GENERATOR

GETDSI acquires the absolute binary to run the HCS IOU deadstart tape generator. There are no parameters.

Example of GETDSI Usage

```
ses.getdsi
REVERT.    END GETDSI  VGENDSI IS LOCAL
```

10.13 GETLDB - ACQUIRE BINARY FOR ENVIR. INT. LOADER

GETLDB acquires the absolute binary to run the HCS Environment Interface loader. There are no parameters.

Example of GETLDB Usage

```
ses.getldb
REVERT.    END GETLDB  VGENLDB IS LOCAL
```

11.0 MOTOROLA 68000 ENVIRONMENT

11.0 MOTOROLA_68000_ENVIRONMENT

This chapter describes the SES utilities for creating and using the Motorola 68000 Environment. Facilities described in this chapter are :

ASM68K	execute the MC68000 assembler.
LINK68K	execute the MC68000 absolute linker.
BIND68K	create a single bound load module.
TRAN68K	generate a file of Motorola S records from the output files of the MC68000 absolute linker.
REFORM68K	reformat a module to H-P object text format.
BLDMI68K	generates an absolute module from the output files of the MC68000 absolute linker.
GENGS68K	translates a global symbols file into H-P format.
PURSYM68K	removes debug symbol tables, generated by CYBIL/CM or the MC68000 assembler, from modules on object files or object libraries.
TRANDILIB	translate DI libraries.

11.0 MOTOROLA 68000 ENVIRONMENT
11.1 ASM68K - EXECUTE THE MOTOROLA 68000 ASSEMBLER ON 170

11.1 ASM68K - EXECUTE THE MOTOROLA 68000 ASSEMBLER ON 170

ASM68K executes the Motorola 68000 assembler on the C170.

This procedure is arranged so that it does not abort. If there are any assembly-time errors detected, the NOS Job Control Register called EFG is set to a non-zero value. When this procedure is being used as a building block, this feature is used to determine what steps should be taken subsequent to running the assembler.

Parameters to the ASM68K procedure are :

i or input :
(optional) name of the Input file to be assembled. If you don't code the i parameter, ASM68K assumes that the input is on a file called **input**.

b or binary :
(optional) name of the file to receive the object code generated by the assembler. If you don't code the b parameter the object code will be written to a file called **lgo**. This file is not rewound by the ASM68K procedure.

l or listing :
(optional) name of the file to receive the Listing from the assembler. If you don't code the l parameter the assembler output appears on a file called **listing**. This file is not rewound by the ASM68K procedure.

x or xref :
(optional) keyword parameter selecting a cross reference listing. If you don't code this parameter no cross reference listing is produced.

debug or d :
if you code this key, the assembler generates debug symbol tables in the object code file. If you don't code this key, the assembler doesn't generate the debug information.

msg or nomsg :
These (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES

11.0 MOTOROLA 68000 ENVIRONMENT
11.1 ASM68K - EXECUTE THE MOTOROLA 68000 ASSEMBLER ON 170

FROM SES PROCEDURES".

col or cols or columns : (left, right)
(optional) left starting column and right ending column of source assembly language statements. The default values are (1,90).

11.2 LINK68K - EXECUTE THE MC68000 ABSOLUTE LINKER

LINK68K executes the MC68000 Absolute Linker, which links data from SES object files/libraries and produces a map file and a set of SEGMent files. LINK68K is set up so that users can specify their own Linker Parameter File (LPF) containing MC68000 Absolute Linker variables that control the linkage.

Parameters to LINK68K are:

ofl : (Filename(s), optional)
Object_File_List - list of up to 10 names of files containing SES object text (ver 1.4). This parameter does not have a default.

lfl : (Filename(s), optional)
Library_File_List - list of up to 10 names of Library files containing SES object text. This parameter does not have a default.

sp : (String(31), optional)
Starting_Procedure - This parameter specifies the entry point at which to start execution. The default is to start execution at the first Transfer symbol encountered.

ns : (String(4), optional)
Name_Seed - This parameter specifies the "name seed" for the MC68000 Absolute Linker Segment files. The default for this parameter is segm. The value of this parameter overrides any Linker Parameter File specification of this field.

mf : (Filename, optional)
Map_File - This parameter specifies the name of the map file. The default for this parameter is linkmap. The value of this parameter overrides any Linker Parameter File specification of this field.

mo :
this (optional) parameter determines the linker Map Options. mo determines the amount of information output on the linker map. Any value coded for mo overrides the default options chosen by the linker parameter file (LPF).

The value of the mo parameter is one of the following characters :

- N no map information; diagnostics are output.
- S section allocations for every section of every input object module.
- E section allocations plus Entry point names and address assignments.
- M section allocations and entry points plus output segment and common block allocations (this last option is in fact a full linker map).

rewind or norew :
these (optional) keys determine whether to REWIND the linker map file before writing it. The default action is to rewind the map file. The key coded overrides any linker parameter file specifications.

lpf :
(optional) name of Linker_Parameter_File containing Linker parameters that affect the link process. If you don't code the lpf parameter, default values are applied as in the description of the LPF.

cybmlib : (Keyword, optional)
If you code this (optional) key, LINK68K uses CYBMLIB to satisfy externals during the linking process. LINK68K ACQUIRE's CYBMLIB from SES and add it to the Library_File_List. The default is not to use CYBMLIB as part of the Link.

debug :
code this key if you want the linker to generate a file containing module address and entry point information. The file name is the name seed appended with 'DBG'. If you don't code this key, no file is generated.

Example of LINK68K Usage

In the following example the Linker is passed a Linker Parameter File. Object files york and hunt are linked using mylib as an object library.

11.0 MOTOROLA 68000 ENVIRONMENT
11.2 LINK68K - EXECUTE THE MC68000 ABSOLUTE LINKER

```
ses.link68k ofl=(york,hunt) ifl=mylib ipf=mylpf
REVERT.    END LINK68K
```

11.3 BIND68K - BIND MC68000 MODULES

The purpose of this command is to create a single bound load module from the specified component object or load modules. The code sections of the old modules are combined into a single code section, data sections with identical attributes are combined into a single section and the binding sections are combined and reorganized for minimum redundancy. Entry point definitions which are referenced from one or more component modules are deleted unless they have been explicitly retained on this command or they have been previously retained.

Parameters to the BIND68K are :

file or f or library or lib :
This parameter may be coded as a multivalued list of sublists. Each element of the value list is a single name, referring to the name of a file already assigned to the job or in the current user's catalog; or the element is a sublist, the last sub-element in the sublist is a user name in whose catalog the files (or libraries) referred to by the other sub-elements may be found. An example at the end should help clear this up.

name or n :
This parameter specifies the module name to be associated with the created module.

module or modules or m :
(optional) This parameter specifies the modules or module subranges from files or libraries that are to be components of the created module. If not specified, all the modules on the specified files and libraries will be used as components.

retain or r :
(optional) This parameter specifies the externally declared names referenced by a component of the created module that are to be retained in the created module. Procedures that have been retained previously need not be respecified in this parameter.

omit or o :
(optional) This parameter specifies the externally declared

names in the component modules whose definitions are to be removed from the output module.

starting_procedure or sp :
(optional) This parameter specifies the externally declared name of the procedure at which execution is to begin. Omission causes last transfer symbol encountered to be used. The **starting procedure** is always retained in the new module.

upon or up :
This parameter specifies the file to contain the created module and the user name in whose catalog the file is to be placed.

base or b or baselib or bl :
(optional) This parameter specifies a file of object code modules that the upon file will be added. SES procedure GOF will be called if the keys b or base or just a file name is specified. GOL will be called if the keys **baselib** or **bl** are specified. GOF and GOL use the upon file as input and the base is rewritten with the created module as the last mdodule.

lock or nolock :
(optional) These parameters determine whether the library update process is interlocked against simultaneous updates. Coding a filename for the lock parameter determines the name of the interlock file. Interlocking is the default action when the upon file being updated is in another user's catalog.

output or out :
(optional) This parameter specifies the file name to which an object map of the newly created module is to be written.

lpf :
(optional) This parameter specifies the file name of the file which contains the above parameter's as commands to BIND68K. This file must be local to your job. The following parameters and their optional names may be specified on this file: (FILE, MODULE, OMIT, RETAIN, NAME, STARTING_PROCEDURE).

Examples of BIND68K Usage

```
ses.bind68k f=(mylib,struct,(sine,cosine,tan,..
      ftnlib)) trig_functions sp=start up=bound
```

The files **mylib** and **struct** are either already local or pulled from your catalog. The files **sine**, **cosine**, and **tan** are in the **ftnlib** catalog or local. The name of the new module will be **trig_functions** and the new module will be on the file **bound**.

```
ses.bind68k ipf=bindipf out=list upon=obout

where bindipf contains:

FILE=(FILE1,FILE2)
MODULE=(MOD1,MOD2,MOD3)
RETAIN=(ENT1,ENT2, ..
ENT3, ..
ENT4, ..
ENT5)
NAME=NEWONE
SP=TEST
END
```

The modules to be bound are obtained from files **FILE1** and **FILE2**. The modules bound are **MOD1**, **MOD2** and **MOD3**. The entry points retained are **ENT1** ... The new module's name is **NEWONE**, and the starting procedure is **TEST**. The listing from BIND68K is on file **LIST** and the bound module is on **OBOUT**.

11.4 TRAN68K - GENERATE S RECORDS

TRAN68K generates a file of Motorola S records from the output files of the MC68000 Absolute Linker.

Parameters to TRAN68K are:

hdr or h :
name of the header file from the Absolute Linker. The file name has the form xxxxHDR from the linker. This file contains information about the other files needed to create the S records.

srec or s :
name of file to receive the S records.

un :
(optional) user number of catalog to search for hdr file and the other files needed. The default is the current user.

Example of TRAN68K Usage

```
ses.tran68k segmhdr srecfil
REVERT.     END TRAN68K
```

This example shows TRAN68K creating **srecfil** from files that are specified by **segmhdr** in the current user's catalog.

11.5 REFORM68K - REFORMAT MODULE TO H-P OBJECT TEXT FORMAT

REFORM68K takes a relocatable module in the CDC object text format and converts it to the H-P format. One CDC file with relocatable object text is reformatted into two files which conform to the H-P format: 1) a file containing relocatable object code (known in the H-P world as a "reloc" type file), and/or 2) a file containing the local symbols for each module (known to the H-P world as an "asmb_sym" type file). One or both of these files can be generated, as needs dictate. The reformatted module(s) can then be downloaded for use on an H-P 64000 workstation. This procedure will work on an input file with more than one module. Each module gets a separate record on each of the output files. Parameters to REFORM68K are :

input or i :
name of file containing the module or modules to be reformatted.

output or o :
(optional) name of file to receive reformatted module or modules. If you don't code this parameter, the value of the profile variable **hpreloc** will be used (if it exists in your profile).

symbols or sym :
(optional) name of file to receive reformatted local symbols for a module or modules. If you don't code this parameter, the value of the profile variable **hplocsym** will be used (if it exists in your profile).

un :
(optional) user number of catalog to search for input file. The default is the current user.

userid or uid :
a string of up to 6 characters that is the user id from an H-P development station.

list or l :
a file to contain the list of warning errors from REFORM68K. The default is to list warning errors to output.

11.0 MOTOROLA 68000 ENVIRONMENT
11.5 REFORM68K - REFORMAT MODULE TO H-P OBJECT TEXT FORMAT

Example of REFORM68K Usage

```
ses.reform68k i=cyrel un=cy170 o=hprel sym=hplsym uid=hpfile
REVERT.  END REFORM68K CYREL -> HPREL,HPLSYM
```

The file **cyrel** is a file of CDC object text in the catalog **cy170.** The object text in H-P format will be on a file named **hprel.** The local symbols will be on a file named **hplsym.** The user id **hpfile** is part of the file **hprel.**

11.6 BLDMI68K - MEMORY IMAGE BUILDER

BLDMI68K generates an absolute module from the output files of the MC68000 Absolute Linker. The file specified by the hdr parameter is acquired by the procedure if it is not local.

Parameters to BLDMI68K are:

name :
this parameter specifies the module name to be associated with the created absolute module.

hdr or h :
name of the header file from the Absolute Linker. The file name has the form xxxxHDR from the linker, where xxx is the name seed given by the linker. This file contains information about the other files needed to create the absolute module.

output or o :
(optional) name of file to receive the new absolute module. If you don't code this parameter, the default file name is MIBMOD.

un :
(optional) user number of catalog to search for hdr file and the other files needed. The default is the current user.

Example of BLDMI68K Usage

```
ses.bldmi68k abs_module segmhdr newmod
REVERT.     END BLDMI68K
```

This example shows BLDMI68K creating an absolute module named abs_module from files that are specified by segmhdr in the current user's catalog. The new module will be on the file newmod.

11.7 GENGS68K - TRANSLATES GLOBAL SYM. FILE INTO H-P FORMAT

GENGS68K accepts as input the 'DBG' file output from LINK68K (the debug keyword must be specified on the call to LINK68K, or no 'DBG' file will be generated). The file is reformatted into an H-P Linker Symbols File (also referred to in H-P manuals as a "link_sym" type file or "File Type 13"). The H-P formatted file can then be downloaded for use on an H-P 64000 work station. Parameters to GENGS68K are :

input or i :
(required) name of the linker symbols file output from LINK68K. The file name has the form 'xxxxDBG', where xxx is the name seed used by LINK68K.

output or o :
(optional) name of a file to receive the H-P formatted Linker Symbols File. If not specified, GENGS68K writes to a local file named HPGSYM.

userid or uid :
(optional) 1-6 character H-P user id. This is the same id entered on the HP64000 in response to the userid prompt. If defaulted, the then current H-P userid will be used when the file is used on the H-P station.

un :
(optional) user number of catalog to search for the input file. If not specified, the default is the current user.

Example of GENGS68K Usage

```
ses.gengs68k i=junedbg uid=hpid
REVERT.    END GENGS68K  JUNEDBG -> HPGSYM
```

In this example, the file named junedbg is a file of linker (i.e., global) symbols; the H-P formatted linker symbols will be on a file named hpgsym and the user id 'hpid' is recorded in that file.

11.0 MOTOROLA 68000 ENVIRONMENT
11.8 PURSYM68K - PURGE SYMBOL TABLES

11.8 PURSYM68K - PURGE SYMBOL TABLES

PURSYM68K removes debug symbol tables, generated by CYBIL/CM or the MC68000 assembler, from modules on object files or object libraries. The input files are acquired by the procedure and the output files are rewritten as permanent files. The same number of files must be specified for both the input and output parameters.

Parameters to PURSYM68K are :

input or i :
name of file or list of files containing debug symbol tables. These files are acquired if they are not local. These files can be object files or object libraries.

output or o :
name of file or list of files that do not contain debug symbol tables when this procedure is finished. All these files are rewritten into the catalog of the user number specified. If the input file is an object file, the output file will be an object file and the same holds true for object libraries.

un :
(optional) user number of catalog to search for input files and where to rewrite the output files. The default is the current user.

Example of PURSYM68K Usage

```
ses.pursym68k (exec monitor) (cexe mntr)
REVERT.     END PURSYM68K
```

This example shows PURSYM68K removing debug symbol tables from modules on the file exec and putting the modules without symbol tables on the file cexe. The same is done with monitor and mntr.

11.9 TRANDILIB - TRANSLATE DI LIBRARIES

TRANDILIB packs object modules on object files or object libraries into byte oriented data mappings so the new file or library can be downloaded to the DI box. The object modules will appear to have been generated by CYBIL on the C180. The number of input files specified must be the same as the number of output files specified.

Parameters to TRANDILIB are :

input or i :
name of file or list of files to be translated. These files are acquired if they are not local.

output or o :
name of file or list of files that have been translated when this procedure is finished. All these files are rewritten into the catalog of the user number specified.

un :
(optional) user number of catalog to search for input files and where to rewrite the output files. The default is the current user.

file or f or lib or l :
(optional) key specifies the format of the output files. If you code lib, then a module directory and an entry point directory are included in the output files. If you don't code either key, then the default is file.

Example of TRANDILIB Usage

```
ses.trandilib (objfil1 objlib1) (objfil2 objfil3)
REVERT.     END TRANDILIB
```

This example shows TRANDILIB packing all the object modules on the object file, objfil1, and the object library, objlib1, into packed object files objfil2 and objfil3. If you had specified lib or l instead of file, objfil2 and objfil3 would have been packed libraries.

12.0 UCSD PCODE ENVIRONMENT

12.0 UCSD_PCODE_ENVIRONMENT

This chapter describes the SES utilities for using the UCSD Pcode Environment. Facilities described in this chapter are :

REFORMP reformat module to UCSD Pcode object text format.

12.0 UCSD PCODE ENVIRONMENT
12.1 REFORMP - REFORMAT MOD TO UCSD PCODE OBJ TEXT FORMAT

12.1 REFORMP - REFORMAT MOD TO UCSD PCODE OBJ TEXT FORMAT

REFORMP takes a relocatable module in the CDC object text format and converts it to the UCSD Pcode format. The re-formatted module can then be downloaded to a UCSD Pcode Version IV.0 P-system. This procedure will work on a file with more than one module. Each module gets a separate record on the output file. Parameters to REFORMP are :

input or i :
name of file containing the module or modules to be reformatted.

output or o :
name of file to receive reformatted module or modules. The default name is PFILE.

un :
(optional) user number of catalog to search for input file. The default is the current user.

Example of REFORMP Usage

```
ses.reformp cyrel pfile un=cy170
```

The file **cyrel** is a file of CDC object text in the catalog **cy170**. The object text in UCSD Pcode format will be on a file named **pfile**.

13.0 SES OBJECT CODE UTILITIES

13.0 SES OBJECT CODE UTILITIES

The Object Code Utilities are a collection of commands which used to create and update libraries, modify certain aspects of individual modules, and list various types of information for each module. The utilities accept as input object modules in loader text format V1.4 generated by cross compilers and assemblers executing on CYBER 170 (ASM68K, CYBIL-CI, CYBIL-CM and PCODE). Here is a brief summary of the Object Code Utility commands. The full descriptions follow.

COM	Change Object Module. Changes characteristics of a module in an object library.
DEOM	DElete Object Module(s) from an object library.
DIOM	DIsplay Object Module(s) information.
GOL	Generate Object Library. Generates or updates an object library.
GOF	Generate Object File. GOF is similar to GOL, except that GOF produces a file which is not a library.
OBJLIST	displays CDC object text in a readable form.

13.0 SES OBJECT CODE UTILITIES

Parameter Naming Convention for Object Code Utilities

This section introduces the parameter naming convention common to most of the procedures in the SES Object Code Utilities.

file or library	name of the object library (or file) on which the object modules reside.
module	name of a module or range of modules
upon	name of new library (or file) for those procedures which update a library (or file).
listing	name of file to receive listing information.

13.0 SES OBJECT CODE UTILITIES

PROFILE Variables for Object Code Utilities

The variables described below may be declared in your PROFILE in order to establish default information for the parameters of the SES Object Code Utilities procedures.

lokmode used by all of the procedures that update libraries, to determine the default interlock action. **lokmode** may be set to one of 'LOCK' or 'NOLOCK'. See also the sections below on updating libraries and interlocking.

intrlok used by all of the procedures that update libraries to specify the interlock file to be used during the update process. The default name for this file is **intrlok**.

13.0 SES OBJECT CODE UTILITIES

Interlock Process for Updating a Library

When updating a library you can interlock the update so that only one user at a time can update the library. Those procedures that update a library are set up so that if the library to be updated is in another user's catalog, the library is interlocked by default. You can override default actions by defining a lokmode variable in your profile, or by coding a **lock** or **nolock** key on the procedure, as shown in the diagram on the next page.

Interlocking of a library is done via an interlock file. Such a file **must** be a **DIRECT** access file **in the same user's catalog as the library being updated**. Naturally the interlock file must be a PUBLIC, WRITE MODE file if other users are likely to be using it. The default name used by the SES Object Code Utilities and library management procedures for the lock file is intrlok. You can have an interlock file of any name by defining the intrlok profile variable, or by coding a file name as a value for the **lock** parameter on the appropriate procedures.

13.0 SES OBJECT CODE UTILITIES

Coded On Procedure	Action Taken
lock	library or base is locked regardless of contents of lokmode profile variable or other conditions.
nolock	library or base is not locked regardless of contents of lokmode profile variable or other conditions.

Nothing	Coded in Profile	Action Taken
	\ lokmode='LOCK'	library or base is locked unless overridden by nolock parameter on procedure.
	\ lokmode='NOLOCK'	library or base is not locked unless overridden by lock parameter on procedure.

Nothing	Owner Of Base Or Library	Action Taken
	Current User	library or base is not locked unless the lokmode profile variable is set to 'LOCK' or the lock parameter is coded on the procedure.
	Another User	library or base is locked unless the lokmode profile variable is set to 'NOLOCK' or the nolock parameter is coded on the procedure

13.0 SES OBJECT CODE UTILITIES
13.1 COM - CHANGE OBJECT MODULE

13.1 COM - CHANGE OBJECT MODULE

COM Changes an Object Module in an object library. Various characteristics of the module may be changed, such as entry point names which may be substituted or removed, gated entry points may be designated or removed, and commentary may be changed. Parameter to COM are :

file or f or library or lib :
name of file (or library) containing the module to be changed, plus an (optional) user name specifying the catalog where the file (or library) resides if it is not in the current user's catalog. The specified file may or may not be a library, but if the keyword used is lib or library, it must be a library. Whether the upon file is a library is determined by the format of this input file. If you code a file name and a user name, they must appear as a list, the first element of which is the name of the file, and the second element the name of the user in whose catalog the file resides, for example : f=(date, palm) to get file date from the catalog of user palm.

module or mo :
name of the module in the library to be changed.

nn :
(optional) name which replaces the name of the specified module.

s :
(optional) entry point pair(s) whose names are to be substituted. The pairs are of the form (<old entry point>,<new entry point>) where <old entry point> is replaced by <new entry point>.

omit or o :
(optional) entry point(s) whose definitions are to be removed from the output module.

gate or g :
(optional) entry point(s) that are to be gated in the output module. Gated entry points can be entered from any ring within the files call bracket.

13.0 SES OBJECT CODE UTILITIES
13.1 COM - CHANGE OBJECT MODULE

ng :
(optional) entry point(s) for which the gated attribute is to be removed.

pro :
(optional) entry point at which execution is to begin (transfer symbol).

comment or co :
(optional) contents of the commentary field in the module header.

upon or up :
name of file to receive the updated file (or library), plus an (optional) user name in whose catalog the file (or library) resides, if the file (or library) is not in the current user's catalog. Whether the upon file is a library is determined by the format of the inout file. If you code a file name and a user name, they must be coded as a list, the first element of which is the name of the file, and the second element the user name in whose catalog the file resides. If you just code the file name and no user name, the upon file is placed in the catalog of the current user. If the user name refers to another user's catalog, the upon file must already exist in that user's catalog.

lock or nolock :
these (optional) parameters determine whether the library update process is interlocked against simultaneous updates; coding a filename for the lock parameter determines the name of the interlock file. Interlocking is the default action when the library being updated is in another user's catalog. If you don't code either of the lock or nolock keys, interlocking is controlled by the lokmode profile variable. Refer to the introductory sections of this chapter for information on the interactions of the lokmode profile variable and the lock and nolock parameters. If you don't code a filename for the lock parameter, the contents of profile variable intrlok is used as the interlock filename; if there's no such profile variable, the name intrlok is used as the lock filename. The interlock file must be in the same catalog as the library being updated. If the interlock file cannot be found, the procedure aborts.

13.0 SES OBJECT CODE UTILITIES
13.1 COM - CHANGE OBJECT MODULE

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of COM Usage

```
ses.com library=lib1 module=modx nn=mody upon=lib2
REVERT.    MODULE MODX CHANGED ON FILE LIB2
```

This command creates a new library, lib2 which is identical to library lib1 except that the name of modx is changed to mody.

```
ses.com file1 modx s=((ep1,newep1),(ep2,newep2)) up=(file2,wrb)
*    LOCKING FILE2 VIA INTRLOK
*    FILE2 LOCKED
*    FILE2 UNLOCKED
REVERT.    MODULE MODX CHANGED ON FILE FILE2
```

This command creates a new file file2 (may be library) identical to file1 except that entry point name newep1 replaces ep1 and newep2 replaces ep2. The destination file, file2 is given with the user name wrb, so the automatic interlocking process takes place during the update of file2.

```
ses.com lib1 mo=modulex o=ent1 g=ent2 ng=ent3 pro=ent4 ..
..? co='cpu migds review' up=lib2
REVERT.    MODULE MODULEX CHANGED ON FILE LIB2
```

This command creates a new library lib2 identical to lib1 except that entry points in module modulex are modified as follows : ent1 is removed; ent2 is set to gated; ent3 is set to not gated; the transfer symbol is changed to ent4; and the commentary field is changed as shown.

13.2 DEOM - DELETE OBJECT MODULE(S)

DEOM DEletes Object Module(s) or ranges of object modules from an object library. Parameters to DEOM are :

file or f or library or lib :
name of the file (or library) from which module(s) or module subrange(s) are to be deleted, plus an (optional) user name in whose catalog the file or library resides if it isn't in the current user's catalog. The specified file may or may not be a library, but if the keyword used is **lib** or **library**, it must be a library. Whether the upon file is a library is determined by the format of this input file. If you code a file name and a user name, they must appear as a list, the first element of which is the name of the file, and the second element the name of the user in whose catalog the file resides, for example : **f=(coconut, palm)** to get file **coconut** from the catalog of user **palm**.

module or mo :
module(s) or module subrange(s) to be deleted from the file. If a specified module is not on the file, a fatal error is issued.

upon or up :
name of the file (or library) on which the remaining modules are written, plus an (optional) user name in whose catalog the updated file (or library) is to be placed, if it is not in the current user's catalog. Whether the upon file is a library is determined by the format of the input file. If you code a file name and a user name, they must be coded as a list, the first element of which is the name of the file, and the second element the user name in whose catalog the file resides. If you just code the file name and no user name, the upon file is placed in the catalog of the current user. If the user name refers to another user's catalog, the upon file must already exist in that user's catalog.

lock or nolock :
these (optional) parameters determine whether the library update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the library being updated is in another user's

catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the library being updated. If the interlock file cannot be found, the procedure aborts.

listing or list :
(optional) name of listing file on which the names of the modules on the new library are listed in the order in which they occur as well as the file from which they came. If you don't code the **list** parameter, no "listing" output is produced.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of DEOM Usage

```
ses.deom library=lib1 module=modx upon=lib2
REVERT.    MODULES DELETED ON FILE LIB2
```

This command creates a library lib2 identical to lib1 except that module modx is deleted.

```
ses.deom file=(file1, user1) mo=(moda.,modb,modc) up=file2
REVERT.    MODULES DELETED ON FILE FILE2
```

This command creates a file file2 (is library if file1 is library) identical to file1 except that modules between moda and modb inclusive and modc are deleted. The file (or library) file1 is being obtained from the catalog of user1.

13.3 DIOM - DISPLAY OBJECT MODULE INFORMATION

DIOM DIsplays various pieces of information about Object Module(s) in an object library. The format of the list file produced is described in the object code utilities ERS (ARH2922). Parameters to DIOM are :

file or f or library or lib :
name of file (or library) whose contents are to be displayed, plus an (optional) user name in whose catalog the file (or library) is to be found, if it is not in the current user's catalog. The specified file may or may not be a library, but if the keyword used is lib or library, it must be a library. If a file (or library) name and a user name is coded, they must appear as a list, for example : lib=(offthe, wall) to get library offthe from the catalog of user wall.

module or mo :
(optional) module(s) or module subrange(s) about which information is to be displayed. If a subrange is specified, all modules in the subrange are displayed. Omission causes all modules in the file or library to be displayed.

listing or list :
(optional) name of file on which the display information is to be written. Omission causes the information to be written to the job output file.

on :
(optional) level of information to be displayed. Only the options selected are in effect. Valid specifications are:

D time and date module was created
E entry point definitions of the module
H module header information
X external references made by the module
A all information printed by D, E, H and X (default)

Examples of DIOM Usage

```
ses.diom library=lib1 module=modx list=listx on=a
REVERT.    MODULES FROM FILE LIB1 DISPLAYED
```

This command lists all information about module modx of library lib1 on file listx.

```
ses.diom file=(filex,losery) mo=(moda.-modb,modc) on=(d,h,x)
REVERT.    MODULES FROM FILE FILEX DISPLAYED
```

This command lists date and time created, module header information, and external references about modules moda to modb inclusive and modc of filex (may be library) on file output (default). The file filex is acquired from the catalog of user losery.

13.4 GOL - GENERATE OBJECT LIBRARY

GOL Generates an Object Library, or updates an existing object library. Also, several files can be combined into one. GOL requires that there be an input file specified either by the file parameter, or base parameter.

The add, replace, and combine parameters, specify exactly which modules from the file(s) specified by the file parameter to include on the new library, while the after and before parameters control the position of the new modules. Parameters to GOL are :

file or f or library or lib :
(optional) list of filename-username pairs describing files (or libraries) from which object modules specified by the following three parameters are to be obtained for the new library. The specified file(s) may or may not be library(s), but if the keyword used is lib or library, they must be library(s). Each element in the list of file (or library) names can either be a single file (or library) name, or the element can itself be a list of two elements, the first is the file (or library) name, and the second is the user name in whose catalog the file (or library) resides. If only a single element, file (or library) name is given, the file (or library) is acquired from the current user's catalog.

The next three parameters (combine, add, and replace) specify module(s) or module subrange(s) to be included on the new library from the files specified by the file or library parameter. None, one, two or all three parameters may be specified. If none are specified, all the modules on the files specified by the file or library parameter is included on the new library. Only the first occurrence of duplicate modules are included in the new library. As explained below, the particular parameter(s) used give you control over whether the given modules should exist on the base library(s).

combine or co :
(optional) module(s) to be included in the new library. The specified module(s) may or may not exist on the base library(s). If they exist, they are replaced.

add :
(optional) module(s) to be ADDed on the new library. If a specified module duplicates a module already on a library

specified by the base parameter, a fatal error is issued.

replace or rep :
(optional) module(s) to be REPLACEd on the new library. If a specified module does not already exist in a library specified by the base parameter, a fatal error is issued.

after or af or before or be :
(optional) module on a base library after or before which to position the new modules. Default position is after the last module on the last base library.

base or b or baselib or bl :
(optional) list of filename-username pairs that determine a list of base file(s) or library(s) to be included in the new object file. All modules from the base files (or libraries) become part of the new object file except duplicate modules (the first occurrence of the module takes precedence). The specified file(s) may or may not be library(s), but if the baselib or bl keyword is used to code the parameter, they must be library(s). Each element in the list of files (or libraries) can itself be a list of two elements, the first being the name of the file (or library), and the second being the name of the user in whose catalog the file (or library) resides. If only one element is specified, the file (or library) is acquired from the current user's catalog.

upon or up :
name of file to receive the generated object library, plus an (optional) user name in whose catalog the new object library is to be placed, if it is not in the current user's catalog. If you code a file name and a user name, they must be coded as a list of two elements, the first of which is the file name, and the second the user name. If you just code the file name, the upon file is placed in the current user's catalog. If the user name refers to another user's catalog, the upon file must already exist in that user's catalog.

lock or nolock :
these (optional) parameters determine whether the library update process is interlocked against simultaneous updates; coding a filename for the lock parameter determines the name of the interlock file. Interlocking is the default

action when the library being updated is in another user's catalog. If you don't code either of the lock or nolock keys, interlocking is controlled by the lokmode profile variable. Refer to the introductory sections of this chapter for information on the interactions of the lokmode profile variable and the lock and nolock parameters. If you don't code a filename for the lock parameter, the contents of profile variable intrlok is used as the interlock filename; if there's no such profile variable, the name intrlok is used as the lock filename. The interlock file must be in the same catalog as the library being updated. If the interlock file cannot be found, the procedure aborts.

listing or list :
(optional) name of listing file on which the names of the modules on the new library are listed in the order in which they occur as well as the file from which they came. If this parameter is not specified, no "listing" output is produced.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

ipf :
(optional) parameter specifies that the other parameters may be specified on a local file. The parameters which may be specified are: FILE, COMBINE, ADD, REPLACE, AFTER/BEFORE, and BASE. The parameters are specified on the file the same way they are specified on the command line.

Examples of GOL Usage

```
ses.gol file=((coconut,palm)) list=rocker upon=newlib
REVERT.    OBJECT LIBRARY GENERATED
```

This command creates a library newlib which contains all the

modules from file coconut (may be library), obtained from the catalog of user palm. Display information appears on file rocker.

```
ses.gol lib=inplib add=moreof rep=sameas bl=baselib up=newlib
REVERT.    OBJECT LIBRARY GENERATED
```

This command creates a library newlib identical to library baselib except that module sameas from library inplib replaces sameas and module moreof from inplib is added at the end. Display information appears on file output (default).

```
ses.gol base=(file1,file2,file3) upon=newlib
REVERT.    OBJECT LIBRARY GENERATED
```

This command creates a library newlib which is the combination of libraries file1, file2 and file3, all obtained from the current user's catalog. Only the first occurrence of duplicate modules appears on the new library. Display information appears on file output.

```
ses.gol (filea,fileb) co=(tea..dinner,nitecap) after=lunch ..
..? base=baselib up=nova
REVERT.    OBJECT LIBRARY GENERATED
```

This command creates a library nova using library baselib as a base. Modules tea thru dinner inclusive and nitecap from files filea and fileb (may be libraries), both obtained from the current user's catalog, are added after module lunch. Display information appears on file output.

```
ses.gol ipf=gollpf list=listf upon=new
.DATA,GOLLPF.
FILE=(FILE1, FILE2)
COMBINE=(MOD1..MOD3) ..
  MOD6
BASE=MYBASE
(EOR)
```

This command creates a library new by combining files FILE1,FILE2 and the modules MOD1..MOD3 and MOD6. The base file is MYBASE. Note that the parameters are obtained from GOLLPF.

13.0 SES OBJECT CODE UTILITIES
13.5 GOF - GENERATE OBJECT FILE

13.5 GOF - GENERATE OBJECT FILE

GOF Generates an Object File, or updates an existing object file. Also, several files can be combined into one. GOF requires that there be an input file specified either by the file parameter, or base parameter.

The add, replace, and combine parameters, specify exactly which modules from the file(s) specified by the file parameter to include on the new object file, while the after and before parameters control the position of the new modules. Parameters to GOF are :

file or f or library or lib :
(optional) list of filename-username pairs describing files (or libraries) from which object modules specified by the following three parameters are to be obtained for the new object file. The specified file(s) may or may not be library(s), but if the keyword used is lib or library, they must be library(s). Each element in the list of file (or library) names can either be a single file (or library) name, or the element can itself be a list of two elements, the first is the file (or library) name, and the second is the user name in whose catalog the file (or library) resides. If only a single element, file (or library) name is given, the file (or library) is acquired from the current user's catalog.

The next three parameters (combine, add, and replace) specify module(s) or module subrange(s) to be included on the new object file from the files specified by the file or library parameter. None, one, two or all three parameters may be specified. If none are specified, all the modules on the files specified by the file or library parameter is included on the new object file. Only the first occurrence of duplicate modules are included in the new object file. As explained below, the particular parameter(s) used give you control over whether the given modules should exist on the base library(s).

combine or co :
(optional) module(s) to be included in the new object file. The specified module(s) may or may not exist on the base library(s). If they exist, they are replaced.

add :
(optional) module(s) to be ADDed on the new object file. If a specified module duplicates a module already on a

library specified by the base parameter, a fatal error is issued.

replace or rep :
(optional) module(s) to be REPLACEd on the new object file. If a specified module does not already exist in a library specified by the base parameter, a fatal error is issued.

after or af or before or be :
(optional) module on a base library after or before which to position the new modules. Default position is after the last module on the last base library.

base or b or baselib or bl :
(optional) list of filename-username pairs that determine a list of base file(s) or library(s) to be included in the new object file. All modules from the base files (or libraries) become part of the new object file except duplicate modules (the first occurrence of the module takes precedence). The specified file(s) may or may not be library(s), but if the baselib or bl keyword is used to code the parameter, they must be library(s). Each element in the list of files (or libraries) can itself be a list of two elements, the first being the name of the file (or library), and the second being the name of the user in whose catalog the file (or library) resides. If only one element is specified, the file (or library) is acquired from the current user's catalog.

upon or up :
name of file to receive the generated object file, plus an (optional) user name in whose catalog the new object file is to be placed, if it is not in the current user's catalog. If you code a file name and a user name, they must be coded as a list of two elements, the first of which is the file name, and the second the user name. If you just code the file name, the upon file is placed in the current user's catalog. If the user name refers to another user's catalog, the upon file must already exist in that user's catalog.

lock or nolock :
these (optional) parameters determine whether the library update process is interlocked against simultaneous updates;

coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is the default action when the library being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the library being updated. If the interlock file cannot be found, the procedure aborts.

listing or list :
(optional) name of listing file on which the names of the names of the modules on the new library are listed in the order in which they occur as well as the file from which they came. If this parameter is not specified, no "listing" output is produced.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

lpf :
(optional) parameter specifies that the other parameters may be specified on a local file. The parameters which may be specified are: FILE, COMBINE, ADD, REPLACE, AFTER/BEFORE, and BASE. The parameters are specified on the file the same way they are specified on the command line.

Examples of GOF Usage

```
ses.gof file=((filex,usx),(filey,usy)) list=listx up=newfile
REVERT.    OBJECT FILE GENERATED
```

This command creates an object file newfile which contains all

the modules from the object files filex (obtained from the catalog of usx) and filey (obtained from the catalog of usy). File filex and filey can in fact be library(s). Display information appears on file listx.

```
ses.gof lib=inplib add=mod1 rep=mod2 b=baselib up=(nfile,nusr)
*    LOCKING NEWFILE VIA INTRLOK
*    NEWFILE LOCKED
*    NEWFILE UNLOCKED
REVERT.    OBJECT FILE GENERATED
```

This command creates an object file nfile identical to library baselib except that module mod2 from library inplib replaces mod2 and module mod1 from inplib is added at the end. Note that file nfile is specified as being in the catalog of nusr, so the object library update is automatically interlocked. Display information appears on file output (default).

```
ses.gof base=(file1,file2,file3) upon=latest
REVERT.    OBJECT FILE GENERATED
```

This command creates an object file latest which is the combination of libraries file1, file2 and file3. Only the first occurrence of duplicate modules appears on the new library. Display information appears on file output.

```
ses.gof (filea,fileb) co=(moda..modb,modc) after=modx ..
..? base=((baselib,olduser)) up=newobjf
REVERT.    OBJECT FILE GENERATED
```

This command creates an object file newobjf using library baselib (obtained from the catalog of olduser) as a base. Modules moda thru modb inclusive, and modc, from files filea and fileb (may be libraries) are added after module modx. Display information appears on file output.

```
ses.gof ipf=gollpf list=listf upon=new
.DATA,GOLLPF.
FILE=(FILE1,FILE2)
COMBINE=(MOD1 ..  MOD3) ..
  MOD6
BASE=MYBASE
(EOR)
```

This command creates a file NEW which is a combination of FILE1 and FILE2 and the modules MOD1 .. MOD3 and MOD6. The base file is

MYBASE. Note that the parameters are obtained from **GOLLPF.**

13.0 SES OBJECT CODE UTILITIES
13.6 OBJLIST - LIST CDC OBJECT TEXT

13.6 OBJLIST - LIST CDC OBJECT TEXT

The action taken by OBJLIST is determined by the DEBUG parameter. OBJLIST will either print out CDC object text in a readable form or produce a symbol table file for use with the CYBER 180 simulator. Parameters to OBJLIST are :

i or f :
(optional) name of Input File which is to have its contents listed out. If you don't code the i parameter, OBJLIST assumes the object text is on file lgo.

o :
(optional) name of file to receive the Output from OBJLIST. If you don't code the o parameter, the output appears on file output.

debug :
(optional) key determines whether OBJLIST prints CDC object text in a readable form or produces a symbol table file. If you code the debug key, OBJLIST produces a symbol table file. If you fail to code the debug key, OBJLIST produces a readable listing of CDC object text.

Examples of OBJLIST Usage

```
ses.objlist  o=listing
REVERT.    END OBJLIST  LGO -> LISTING

ses.objlist  i=binary, o=prinfyl
REVERT.    END OBJLIST  BINARY -> PRINFYL
```

14.0 SOURCE CODE UTILITY ON NOS 170

14.0 SOURCE_CODE_UTILITY_ON_NOS_170

SES provides a series of procedures for access to the NOS 170 prototype implementation of the Source Code Utility. The procedures currently available in this category are:

SCU Acquire the executable binary for SCU.

SOLO Run the SCU editor in stand alone mode.

SCUCOMP Generate an SCU modification for a deck.

EDSCU Edit a deck from an SCU library.

14.0 SOURCE CODE UTILITY ON NOS 170
14.1 SCU - ACQUIRE EXECUTABLE BINARY FOR SCU

14.1 SCU - ACQUIRE EXECUTABLE BINARY FOR SCU

SCU acquires the executable binary for the Source Code Utility and copies it to a local file named SCU. In response to calling this procedure the user may be sent a short message warning him that a change has been introduced and that there is additional information available through the STATUS facility. There are no parameters for the SCU procedure.

The library CYBCCMN or CYBICMN which contain declarations for interfaces supported by the SES group may be accessed in the form of SCU source libraries through the use of the GETCOMN procedure.

14.0 SOURCE CODE UTILITY ON NOS 170
14.2 SOLO - STAND ALONE SCU EDITOR

14.2 SOLO - STAND ALONE SCU EDITOR

SOLO is a stand alone version of the SCU editor, that can be used to edit any text file. A delay will be encountered when entering and exiting this editor because a temporary library is created on entry from which the text file is extracted on exit. Parameters to SOLO are :

base or ba :
(optional) name of text file to be edited. If you don't code the base parameter, SOLO uses a local file named **base**.

result or r :
(optional) name of file to receive the result of the editing session (SOLO doesn't, by default, write the edited file back over the text input file specified by the **base** parameter). If you don't code the **result** parameter, SOLO writes the result on a local file named **result**.

input or i :
(optional) name of input file from which SOLO reads commands. If you don't code the **input** parameter, commands are read from **input**.

list or l :
(optional) name of file on which the editor displays are written. If you don't code the **list** parameter, SOLO writes its displays to file **output**.

14.3 SCUCOMP - GENERATE SCU CORRECTION SET

SCUCOMP compares an "old" version of an SCU deck (obtained from an SCU source library) with a "new" version of the same deck (obtained from a "source" file). The "source" file must have neither line identifiers nor deck directives in it. The output of SCUCOMP is a file of SCU EDITor commands that transform the "old" version of the deck into the "new" version.

You can select a character to be used as the value of the UNTIL parameter on SCU EDITor INSERT and REPLACE commands. The default character is the hyphen. If any lines of insertion or replacement text end in the "text delimiter" character, they are followed on the edit commands file with a line that indicates the problem, and the NOS job control register EFG is set to a non-zero value; otherwise EFG is set to zero. Parameters to SCUCOMP are :

name or na :
NAME of an SCU deck against which the correction set are generated.

source or s :
name of a source file containing a modified version of the deck.

ec :
name of a file to which SCU editor commands are to be written.

base or ba :
(optional) name of an SCU source library containing the "unmodified" version of the SCU deck with which the source file are compared. If you don't code this parameter, SCUCOMP attempts to access a file named base.

un :
(optional) User Name in whose catalog the source library specified by base is to be found.

td :
(optional) single character value which defines the Terminating Delimiter to be used on text insertion commands generated by SCUCOMP. The value of td may be given as any

single character, or as a positive integer in the range 1..127. If you don't code the td parameter, SCUCOMP uses the hyphen - as the default character.

Is or ignoris :
These (optional) keys specify whether or not to IGNORe Leading Spaces on lines being compared. The default action is to recognise leading spaces (the is option). If you code the **ignoris** key SCUCOMP ignores leading spaces on text lines.

Example of SCUCOMP Usage

```
ses.scucomp 'scm$editor_help_command' formed ./
ec=modform ba=sl0619
```

This example of SCUCOMP compares formatted source statements on file FORMED with the SCU deck SCM$EDITOR_HELP_COMMAND from the source library on file SL0619 to build an SCU modification for the deck on file MODFORM. In this case lines differing by leading spaces were recognised as changed.

14.4 EDSCU - EDIT A DECK FROM AN SCU LIBRARY

EDSCU provides the user with a convenient way to make modifications to decks from an SCU library without having to directly use the SCU editor. The user may use FSE, XEDIT, EDIT, or EDT depending on their preference. Parameters to EDSCU are :

na or name :
(optional) name of deck to be edited. If you don't code this parameter, EDSCU will prompt you for it unless you coded the end key and are in SCU.GROUP mode (see the group key below).

ba or base :
(optional) name of the base where the deck to be edited can be found. The base parameter should only be coded when you aren't in SCU.GROUP mode. If you aren't in SCU.GROUP mode and haven't coded this parameter, EDSCU uses the value of the profile variable scubase, and if there's no such variable defined in your profile then EDSCU will prompt you for the base name.

nb or newbase :
(optional) name of the new base where base is to be rewritten after the deck changes are complete. Default is to write back over the base name specified by the base parameter. EDSCU will not prompt you for this parameter so to change it you must code it on the call to EDSCU. This parameter can be set or changed at any time. Last value coded is used when the end key is coded.

mod or modification :
(optional) modification name to use when your editing changes are supplied to the SCU editor. If you aren't in SCU.GROUP mode and haven't coded this parameter, EDSCU will prompt you for it. You must code this parameter on the call to EDSCU to change the value while in SCU.GROUP mode. This parameter can be set or changed at any time. Last value coded is used on subsequent calls to this proc until the end key is coded.

e or editor :
(optional) name of the editor you wish to use. If you don't code this parameter, EDSCU uses the value of the

profile variable editor, and if there's no such variable defined in your profile EDSCU will use FSE as the editor. EDSCU will not prompt you for this value. This parameter can be set or changed at any time. The last value coded is used on subsequent calls to this proc until the end key is coded.

g or group :

(optional) key initiating SCU.GROUP mode. The use of this parameter is recommended if your changes will be associated with more than one modification name or your changes will occur in more than one deck. This parameter should be coded only when you aren't already in SCU.GROUP mode. EDSCU assumes you are in SCU.GROUP mode if either files ZZZZZGF or SESSCCG are local. EDSCU will not prompt you for this value.

end :

(optional) key terminating SCU.GROUP mode. EDSCU will not prompt you for this value.

An Example of EDSCU Usage

```
ses.edscu g
ENTER DECK NAME
?  ddump
ENTER BASE NAME
?  testba
ENTER MODIFICATION NAME
?  xyzmod
* BEGINNING EDSCU ON DDUMP OF TESTBA

* BEGINNING GROUP FOR TESTBA
* XEDIT AVAILABLE
 XEDIT 3.1.00
??  i
?  C HELLO THERE ONE AND ALL
??  e
ZQPQDXN IS A LOCAL FILE
* GENERATING CHANGES FOR DDUMP
* INSERTING CHANGES TO DDUMP
 ** E SC 8004: Continue specified for unknown modification
* MODIFICATION XYZMOD ADDED TO DDUMP
* DECK DDUMP EDIT COMPLETE
```

14.0 SOURCE CODE UTILITY ON NOS 170
14.4 EDSCU - EDIT A DECK FROM AN SCU LIBRARY

```
* ENTER *SES.EDSCU* TO EDIT NEXT DECK
* ENTER *SES.EDSCU END* IF FINISHED
$REVERT.
ses.edscu end
* ENDING *EDSCU* EDIT SESSION TESTBA --> TESTBA
$REVERT.
```

Please note that the error message (8004) generated in this example can be expected if the modification name used is a brand new one.

15.0 SOURCE TEXT PREPROCESSORS

15.0 SOURCE TEXT PREPROCESSORS

The utilities described in this section comprise a general set of reformatting facilities for source text. They are described briefly here, and more detailed descriptions follow.

CYBFORM	a source code formatter for CYBIL.
ISWLFRM	a source code formatter for ISWL.
PSEUDO	text preprocessor that can generate repetitive sequences of text.
F5FORM	formats Fortran 5 source text.
PASTOCYB	runs the PASCAL to CYBIL source code translator.

15.0 SOURCE TEXT PREPROCESSORS
15.1 CYBFORM - CYBIL SOURCE TEXT REFORMATTER

15.1 CYBFORM - CYBIL SOURCE TEXT REFORMATTER

CYBFORM is a source code reformatter for CYBIL, such that program readability and consistency of presentation are enhanced. The operation of this procedure differs from that described in the CYBFORM ERS in that this procedure rewinds both the input and output files before and after the reformatting operation. Note that CYBFORM can handle multi record files. Parameters to CYBFORM are :

i or f :
name of Input File containing the CYBIL source text to be reformatted.

o :
(optional) name of file to receive the Output from CYBFORM. If you don't code the o parameter, CYBFORM places the output on the file specified by the i parameter.

15.0 SOURCE TEXT PREPROCESSORS
15.2 ISWLFRM - ISWL SOURCE TEXT REFORMATTER

15.2 ISWLFRM - ISWL SOURCE TEXT REFORMATTER

ISWLFRM reformats ISWL source code, such that program readability and consistency of presentation are enhanced. Parameters to ISWLFRM are :

i or f :
name of Input File containing the ISWL source text to be reformatted.

o :
(optional) name of file to receive the Output from ISWLFRM. If you don't code the o parameter, ISWLFRM places the output on the file specified by the i parameter.

Summary of Formatting Directives

Directives are given to the ISWL formatter by embedding certain comment toggle options in the input ISWL source code :

"$P+"	Select comment formatting.
"$P-"	Turn off comment formatting.
"$PR"	Discard existing line numbers, if any, and generate new line numbers.
"$PN"	Discard existing line numbers, if any, and do not generate new line numbers.
"$PS"	Preserve existing line numbers.
"$PB"	Preserve existing blank lines.
"$PU"	Capitalize ISWL keywords.
"$M<n>"	ISWL Margin chop. Formatter wraps line around at N-1 character position.

Default selections : "$M72" "$PS" "$P-"

Note : ISWLFRM assumes a syntactically correct ISWL program.

Syntax errors may cause undesired results on the output file.

15.0 SOURCE TEXT PREPROCESSORS
15.3 PSEUDO - RUN PSEUDO PREPROCESSOR

15.3 PSEUDO - RUN PSEUDO PREPROCESSOR

PSEUDO is a text preprocessor which enables parameterized replication of selected portions of a text file. The parameters to this procedure are:

i or f :
name of the Input File containing the text to be processed by PSEUDO.

o :
(optional) name of file to receive the Output from PSEUDO. If you don't code the o parameter, PSEUDO directs the output to the file specified by the i parameter.

m :
(optional) name of file to receive the secondary output (Messages) from PSEUDO. If you don't code the m parameter, PSEUDO places the Messages output (if any) on a file called MSGS.

op :
(optional) name of the file to receive the statistics (and any error messages) from the run. If you don't code the op parameter, PSEUDO writes this information to file OUTPUT.

15.4 F5FORM - FORTRAN 5 SOURCE TEXT REFORMATTER

F5FORM reformats FORTRAN-5 source text to reflect the nesting levels of IF-THEN-ELSE blocks and DO loops. Parameters are :

i or **f** :
name of Input File containing the FTN5 source text to be reformatted by F5FORM.

o :
(optional) name of file to receive the Output from F5FORM. If you don't code the o parameter, F5FORM places the resultant formatted text on the file specified by the i parameter.

Examples of F5FORM Usage

```
ses.f5form  i=messy, o=lovely
REVERT.     END F5FORM  MESSY -> LOVELY

ses.f5form  modern
REVERT.     END F5FORM  MODERN
```

15.5 PASTOCYB - PASCAL TO CYBIL TRANSLATOR

PASTOCYB runs the PASCAL to CYBIL source code translator. This translator is based on the University of Minnesota PASCAL compiler. It performs syntax checking on the source input and will translate incomplete decks, although complete source input files produce the best results. Note that this translator may not work for all of the many existing dialects of PASCAL.

The procedure is arranged so that it will not abort. If source code conversion errors are encountered, the NOS Job Control Register called EFG is set to a non-zero value upon exiting the procedure.

Parameters to the PASTOCYB procedure are :

i or f :
(required) name of PASCAL source input file.

o :
(optional) name of the output file to receive the CYBIL source code produced by the translator. If you don't code the o parameter, a file named CYBFILE is used.

l :
(optional) name of the file to receive the listing produced from the translation process. If you don't code the l parameter, a file named LISTING will be used.

ml :
(optional) this parameter specifies the maximum line length of PASCAL source to be translated. The default for this parameter is 120 characters.

Example of PASTOCYB Usage

```
ses.pastocyb i=psource o=csource
```

This example shows PASTOCYB translating the PASCAL source file psource into a CYBIL source file named csource. The listing produced defaults to the file named LISTING.

16.0 SES COMMUNICATIONS

16.0 SES COMMUNICATIONS

User to User Communications

SES provides a communication system such that users can send files between sites, and can also MAIL messages to each other. To receive mail, you need a MAILBOX in your catalog. A MAILBOX is created by using the SES NEWMAIL procedure. When you create a PROFILE using the BuiLD PROFILE (BLDPROF) procedure mentioned in the introduction, a MAILBOX is also created.

SEND	send files between sites with NOS development machines.
MAIL	send mail to other users.
GETMAIL	displays the mail from your mailbox at your terminal, or alternatively, the mail can be placed in a file for subsequent printing or whatever.
NEWMAIL	creates a new mailbox. You also use NEWMAIL when you want to "clear out" the contents of the mailbox.
ANYMAIL	tells you how many items of mail are in your mailbox.
WHOMAIL	tells you who sent the items of mail that are in your mailbox.
SAVMAIL	saves the contents of your mailbox.
CHKMAIL	a procedure SEGMENT that you can INCLUDE into your PROFILE, so that if there is any mail in your mailbox, you get a message every time you use SES.
FEEDBACK	SES user feedback mechanism.

16.0 SES COMMUNICATIONS

The MAIL procedure puts a line at the start of every message that shows you the user name (and possibly the real name) of the person who sent you the message. It's possible to have your name (in characters) included in this by defining a **myname** PROFILE variable as a string, as follows :

```
\ myname = 'Ethel Snerge'
```

Now every time you send a mailbox message to somebody via the MAIL procedure, the header of the message in the destination mailbox reads :

```
FROM USER ES91 ( Ethel Snerge ) ON NOV 19, 1977 AT 9:33 AM
```

MAIL also provides the facility to equate peoples' real names with their user names, so that you can send mail to, say, **fred**, instead of fjc0795 or some other cryptic machine oriented name. You can do this by putting statements like this in your profile :

```
\  IF PROCNAM = 'MAIL' THEN
\    fred = 'FJC0795'
\    bill = 'WAH0001'
\    andy = 'AJL6655'
\  IFEND
```

This series of SES directives in your profile defines the names given, but only when the procedure being used is MAIL. Now, you can send mail like this :

```
ses.mail letters to (fred, bill, andy)
```

Note that when you are using equated names of this type you must use the **to** keyword on the MAIL procedure instead of the **un** keyword.

16.0 SES COMMUNICATIONS

Remote Mainframe Communications

SES provides a set of procedures to submit jobs and manipulate permanent files on remote linked mainframes. Here is a summary of the remote access procedures :

SUBMIT SUBMITs a file to be processed as a batch job on the specified mainframe.

GETPF GETs a Permanent File (direct or indirect) from a remote mainframe to a permanent file (direct) on the local mainframe.

SENDPF SENDs a copy of a Permanent File (direct or indirect) to a direct permanent file on a remote mainframe.

CHANGPF CHANGes parameters of a Permanent File on a remote mainframe.

PURGPF PURGEs (removes) a Permanent File from a remote mainframe.

PERMPF PERMits another user access to a Permanent File.

CATPF displays CAIlist information about Permanent Files existing on the remote mainframe.

DISPOSE PRINTs a file from the local mainframe on a printer at the remote mainframe.

16.0 SES COMMUNICATIONS

Each mainframe has been assigned a site identifier. These site identifiers (located in Arden Hills) are :

M01 - CY176 - S/N 101
M42 - CY760 - S/N 420
M10 - CY73 - S/N 101
M02 - CY835 - S/N 002 - S3
M04 - CY835 - S/N 104 - S2
M05 - CY174 - S/N 805
M07 - CY875 - S/N 907

Note : As additional systems are added to the network, they are assigned a site identifier.

In addition to the site-id, each user must have a valid user name, password, charge number and project number for the target machine. These may be coded into your PROFILE. The PROFILE variables are created by concatenating the site-id with 'fam' for family name, 'use' for user name, 'pas' for password, 'cha' for charge, and 'pro' for project. Users may have PROFILE variables for each remote mainframe they are validated to run jobs on. In interactive mode, the remote link procs requests the family name, user name, password, charge, and project variables if they are not in your PROFILE.

Note : that in batch mode, the variable must exist in your PROFILE, or the job aborts.

EXAMPLE : The variables needed for a S/N 420 user to run jobs on the Arden Hills 101 would be :

```
PROFILE
\ M01USE = 'your 101 user name'
\ M01PAS = 'your 101 password'
\ M01CHA = 'your 101 charge number'
\ M01PRO = 'your 101 project number'
\ M01FAM = 'your 101 family name'
```

If the value for a remote site charge parameter is set to a null string (for example M42CHA = ''), the remote link procedures omit the CHARGE statement in the job control statement stream that is sent to the remote mainframe for execution.

If a remote site family name is set to a null string (for example M01FAM = ''), the family parameter isn't specified on the USER statement.

16.0 SES COMMUNICATIONS
16.1 SEND - TRANSMIT FILES BETWEEN NOS SITES

16.1 SEND - TRANSMIT FILES BETWEEN NOS SITES

SEND is a facility to transmit files between NOS sites.

When you SEND a file to a remote user, a copy of the file is placed in the special catalog C8E at the remote site. This copy of the file is a DIRECT access file which is defined with a UNIQUE name. The receiver is given WRITE permission on the file, and a message is mailed to the receiver to inform him of the real name of the file you sent, and the unique name assigned to the file in the C8E catalog. The receiver should copy the file to his own catalog, and then PURGE the file from the C8E catalog. Note that if the receiver doesn't have a mailbox, he won't be informed that the file is there. Parameters to SEND are :

file or f or i :
name of the file to be sent to the remote user.

un or to :
User Name of the user TO whom the file is to be sent.

at or in :
Identifier of the site (7 characters or less) AT or IN which the receiver resides.

16.0 SES COMMUNICATIONS
16.1 SEND - TRANSMIT FILES BETWEEN NOS SITES

Example of SEND Usage

```
ses.send binfile to hm00612 at svl452
REVERT.    END SEND  BINFILE -> HM00612 AT SVL


ses.send regards to rdp at arh101
REVERT.    END SEND  REGARDS -> RDP AT ARH
```

The two examples show SEND used with all its parameters. One sender is assumed to be in Arden Hills (ARH101) sending to someone in Sunnyvale (SVL452), and the other example shows the reverse situation.

Example of MAIL messages received from SEND

When a remote user sends you files, you receive mail messages something like the following :

```
ses.getmail
*    MAILBOX CONTENTS
*

FROM USER C8E ( CYBER 18 RECEIVER ) ON JAN 14, 1980 AT 12:33 PM

FILE ASORT=ZQ06BS2/UN=C8E FROM REF (RODNEY FOTHERINGAY) IS AVAILABLE

FROM USER C8E ( CYBER 18 RECEIVER ) ON JAN 15, 1980 AT 12:45 PM

FILE GOF=ZQZWBXG/UN=C8E FROM ERP (EARNEST POSTLETHWAITE) IS AVAILABLE

REVERT.    END GETMAIL
```

Now you ATTACH the files from the C8E catalog, copy them to your own catalog, and finally purge them from the C8E catalog (and please remember the last step).

16.2 MAIL - SEND MAIL TO OTHER USERS

MAIL is the procedure to send mail to other users' mailbox(s). Parameters to MAIL are:

i or f :
(optional) name of Input File containing the mail to be transmitted to the mailboxe(s). If you don't code the i parameter, MAIL takes the message from file INPUT, so that if you're at the terminal, it prompts for the message.

un or to :
list of User Names to whom the mail is to be sent TO.

at or in :
(optional) identifier of the site (machine identifier) AT which the addressees reside. All addressees in the list (if more than one) must be at the same site. If you don't code the at parameter, mail is sent to users on the machine you're running on.

MAIL Addressees who do not have a MAILBOX

It may happen that one or more people to whom you're sending mail don't have a MAILBOX. In this case, if the source of mail is a terminal file (such as INPUT), MAIL saves the message you so laboriously typed in on a unique named file, and issues a message to that effect. If the mail message was on a permanent file, of course it is left local anyway.

16.0 SES COMMUNICATIONS
16.2 MAIL - SEND MAIL TO OTHER USERS

Example of MAIL Usage

```
ses.mail  un=j303
?  Alex :
?       I've made the modifications to SESPROC and
?  replaced it on SESLNAM.
?
REVERT.    END MAIL  INPUT -> J303



ses.mail  missifs  to (sherman, bob, jim)  at m01
REVERT.    END MAIL  MISSIFS -> SHERMAN..JIM
```

16.3 GETMAIL - DISPLAY MAIL FROM MAILBOX

GETMAIL displays the contents of your mailbox on a file, output (the terminal if you're an interactive user) by default. Parameters to GETMAIL are :

o or output :
(optional) name of file to receive the Output from GETMAIL. If you don't code the o parameter, GETMAIL displays the mail on file OUTPUT.

seq :
if you code this (optional) key, GETMAIL writes the contents of your mailbox with each line SEQuenced (the sequence number will be on the left end of the lines). This feature is useful, for example, for extracting information from your mailbox via the SELECT procedure (desribed in another section of this document).

at or from :
(optional) identifier of the site (machine identifier) from which mail is to be read. If you don't code this parameter, GETMAIL reads mail from the mailbox on the mainframe you're running on.

user or un or u :
(optional) user name whose mailbox is to be read. This parameter allows you to read mail in mailboxes under other user names if you are authorized to read them. If you don't code the user parameter, your user name is assumed.

Example of GETMAIL Usage

```
ses.getmail
*      MAILBOX CONTENTS
*
FROM USER HN03 (Valery Vitriolix) ON NOV 4, 77 AT 4:33 PM
   Sam :
      New version of program now available on the
      project program library.  .....Val
REVERT.    END GETMAIL
```

16.4 NEWMAIL - CREATE A NEW MAILBOX / CLEAR EXISTING MAILBOX

NEWMAIL creates a new mailbox if one doesn't already exist in your catalog. If a mailbox already exists, NEWMAIL deletes any items previously examined via GETMAIL. There are no parameters to NEWMAIL, you simply code :

```
ses.newmail
```

Also see the introduction to the Handbook for the description of the BuiLD PROFile (BLDPROF) procedure, which in addition to creating a profile for you, creates your mailbox as part of setting up your initial SES environment.

16.5 ANYMAIL - COUNT NUMBER OF ITEMS IN MAILBOX

ANYMAIL informs you of how many items there are in your mailbox. Parameters to ANYMAIL are :

output or o :
(optional) name of the file to which ANYMAIL writes it's output. If you don't code the output parameter, ANYMAIL writes it's output on file OUTPUT.

at or on :
(optional) identifier of the site (machine identifier) at which mail is to be counted. If you don't code the at parameter, ANYMAIL counts items in your mailbox on the mainframe you're running on.

user or un or un :
(optional) user name whose mailbox items are to be counted. This parameter allows you to check for mail in other mailboxes you are authorized to read. If you don't code the user parameter, your user name is assumed.

Examples of ANYMAIL Usage

```
ses.anymail
      4  LETTER(S)  IN MAILBOX
REVERT.    END  ANYMAIL
ses.getmail storage
REVERT.    END GETMAIL
ses.newmail
REVERT.    END NEWMAIL
ses.anymail
 NO MAIL TODAY
REVERT.    END ANYMAIL
```

The example shows ANYMAIL stating that there are four letters in your mailbox. After doing a GETMAIL to place the mailbox contents on file storage, and a NEWMAIL to clear out your mailbox, the second call to ANYMAIL informs you that there is no mail in the mailbox.

16.6 WHOMAIL - DISPLAY LIST OF USERS WHO HAVE SENT MAIL

WHOMAIL displays the list of users WHO have sent MAIL to your mailbox since the last time you did a NEWMAIL. The user name of each user is displayed along with the date and time that the message was sent. There are no parameters to WHOMAIL.

Example of WHOMAIL Usage

```
ses.whomail
FROM USER AEN (Alfred E. Neumann) ON NOV 2, 79 AT 5:10 PM
FROM USER FDR ON DEC 11, 79 AT 4:32 AM
FROM USER JEK ON JAN 2, 80 AT 3:10 PM
FROM USER JXH (Jack Horner) ON FEB 5, 80 AT 8:15 AM
REVERT.    END WHOMAIL
```

This example shows the results when there are 4 messages in your mailbox at the time WHOMAIL is used. If there are no messages, the only output from WHOMAIL is the ending message (REVERT. END WHOMAIL).

16.7 SAVMAIL - SAVE MAILBOX

SAVMAIL saves your mailbox as a text record on a multirecord, direct access, permanent file called OLDMAIL. If there's no file called OLDMAIL in your catalog, SAVMAIL creates one. Parameters to SAVMAIL are:

title or t :
(optional) seven (7) character name of the text record on OLDMAIL to contain the current contents of your mailbox. If you don't code the **title** parameter, SAVMAIL creates a name beginning with the letter 'M' and followed by the current date in the form mmddyy.

Examples of SAVMAIL Usage

```
ses.savmail title=psr421
REVERT.    END SAVMAIL  MAILBOX -> OLDMAIL
```

This example shows the case where it was desired that the message in the mailbox be saved under an identifiable name like a PSR number.

```
ses.savmail
REVERT.    END SAVMAIL  MAILBOX -> OLDMAIL
```

If this example were run on April 22, 1979, your mailbox would be saved as record M042279 on OLDMAIL.

Note : see the Chapter on LIBRARY MANAGEMENT, especially the GETMEM(S) procedure, for some words on how to handle multi record files.

16.0 SES COMMUNICATIONS
16.8 CHKMAIL - CHECK YOUR MAILBOX

16.8 CHKMAIL - CHECK YOUR MAILBOX

CHKMAIL is not actually an SES procedure that you can use via a **ses.chkmail** statement, rather it is an INCLUDE file. The way to use it is to INCLUDE it into your PROFILE, with an SES directive like this in your PROFILE:

```
\ INCLUDE 'CHKMAIL', L=UNIQUE(NAME), LPFN=SESLNAM, UN=SESUNAM
```

This SES directive line arranges that every time you use any SES procedure (with the exception of the MAIL procedures), the CHKMAIL segment checks your MAILBOX for messages, and if there are any, it issues a message to tell you to look at your mail.

16.0 SES COMMUNICATIONS
16.9 FEEDBACK - SES USER FEEDBACK MECHANISM

16.9 FEEDBACK - SES USER FEEDBACK MECHANISM

FEEDBACK allows you to record a comment or suggestion on the usability of SES procedures, or anything you want. The comments will be collected and reviewed for implementation by the SES project. (The SES project will collect the comments directly on S/N 101 in Arden Hills. The SES tools installer at other sites have been requested to do the same and forward them to the SES project. You should confirm that this will be done prior to using FEEDBACK.)

The comment may be entered interactively when FEEDBACK is executed, or given as a previously created file. The time, date and your user name are recorded with each comment. Parameters are :

i or f or file :
(optional) name of input file containing the comment to be transmitted. If you don't code this parameter, FEEDBACK takes the message from file INPUT, so that if you're at the terminal, it prompts for the message.

Examples of FEEDBACK Usage

```
ses.feedback file=comment
REVERT.   END FEEDBACK
```

This example shows use of FEEDBACK with the previously created file comment.

```
ses.feedback
? The GIBRISH procedure HELP documentation doesn't include
? BATCH params.
? (CR)
REVERT.   END FEEDBACK
```

This example shows use of FEEDBACK with the comment being entered interactively.

16.0 SES COMMUNICATIONS
16.10 SUBMIT - SUBMIT A JOB TO BE PROCESSED AT A REMOTE SITE

16.10 SUBMIT - SUBMIT A JOB TO BE PROCESSED AT A REMOTE SITE

SUBMIT provides a method for running a job on a remote mainframe. A user provided job file is processed as a batch job on the specified mainframe. A user job for 170 mode execution is responsible for getting its output sent to any site other than the one the job runs on. This can be done by coding an SES.DISPOSE or an SES.SENDPF as part of the job. If neither of these are coded, the output is printed at the remote site where the job is run.

A user job for 180 mode execution (applicable to the S2 site only) automatically has its output returned and printed at the local site where the user originated the job.

f or fn :
name of file containing the job to be processed.

at or on :
site-id of the mainframe that the job is to run on.

mode :
(optional) value (170 or 180) to indicate execution mode for the job at the remote site; default is mode=170; the mode parameter is applicable only to jobs to be run on the S2 remote site.

You can set a default for the **mode** parameter by defining the **submode** variable in your profile. For instance, \SUBMODE=180 in the PROFILE submits all jobs in 180 mode unless mode=170 is coded when the SUBMIT proc is run.

16.0 SES COMMUNICATIONS
16.11 GETPF - GET A PERMANENT FILE FROM A REMOTE SITE

16.11 GETPF - GET A PERMANENT FILE FROM A REMOTE SITE

GETPF ACQUIRE's a file from a remote mainframe and makes a copy of it in your catalog on the local mainframe. The copy created on the local mainframe is a direct access file. If the file is not under your user number on the remote mainframe, you must have permission to access the file.

The procedure is arranged so that it does not abort. If there are any errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the procedure.

Parameters to GETPF are :

rfn :
name of file to be acquired on the remote mainframe.

at or on :
site-id of the mainframe from which the file is to be acquired.

un :
(optional) user number of the catalog the file is under on the remote mainframe. If this parameter is not coded, GETPF uses the user number value for this site from your PROFILE.

hfn :
(optional) name to be used for the copy of the file created in your catalog on the local mainframe. If this parameter is not coded, GETPF uses the value for rfn. If hfn already exists in your catalog, GETPF terminates with an error status. If hfn doesn't exist, GETPF creates hfn as a direct access file.

passwrd or pw :
(optional) password for the file on the remote mainframe.

16.12 SENDPF - SEND A PERMANENT FILE TO A REMOTE SITE

SENDPF copies a file from a local mainframe to a remote mainframe. If the file already exists, it is rewritten.

The procedure is arranged so that it does not abort. If there are any errors detected, the NOS Job Control Register called EFG is set to a non-zero value. This feature is used when this procedure is being used as a building block, to determine what steps should be taken subsequent to running the procedure.

Parameters to SENDPF are :

hfn :
name of file on the local mainframe that is to be copied to the remote mainframe.

at or on :
site-id of the remote mainframe.

un :
(optional) user name that the local file is cataloged under. If this parameter is not coded, your user name is used.

rfn :
(optional) name of file to be created on the remote mainframe. If this parameter is not coded, SENDPF uses the value of hfn. If file rfn already exists on the remote mainframe, SENDPF writes the new file over it. SENDPF creates rfn as a direct access file.

passwrd or pw :
(optional) password associated with the file on the remote mainframe.

16.13 CHANGPF - CHANGE PARAMETERS OF A PERMANENT FILE

CHANGPF modifies parameters of a permanent file on a remote mainframe without re-defining it. Parameters for CHANGPF are :

ofn :
name of the file on the remote mainframe to be changed.

at or on :
site-id of the remote mainframe.

nfn :
(optional) new name of the file on the remote mainframe.

ct :
(optional) new file category. Must be P (private), S (semiprivate) or PU (public). If the parameter is not coded, the file category is not changed.

m :
(optional) new access mode for the file. Must be W(write), M(modify), A(append), R(read), RM(readmd), RA(readap), E(execute), or N(null-removes permission granted by permission commands). If the parameter is not coded, the access mode isn't changed.

pw :
(optional) new password (1 to 7 characters) for the file. If this parameter is not coded, the password isn't changed.

status or nostat :
coding the (optional) status key causes a message to be placed in your mailbox when CHANGPF completes. When the nostat key is coded, you aren't notified when CHANGPF completes. The default is no message upon completion.

16.0 SES COMMUNICATIONS
16.14 PURGPF - REMOVE (PURGE) A PERMANENT FILE

16.14 PURGPF - REMOVE (PURGE) A PERMANENT FILE

PURGPF removes a permanent file from a remote mainframe. If the file is not in your own catalog, you must have write permission in order to purge it. Parameters for PURGPF are :

ofn :
name of file to be purged.

at or on :
the site-id of the remote mainframe.

un :
(optional) user name on the remote mainframe that **ofn** is cataloged under.

status or nostat :
coding the (optional) status key causes a message to be placed in your mailbox when PURGPF completes. When the nostat key is coded you aren't notified when PURGPF completes. The default is no message upon completion.

16.15 PERMPF - PERMIT ACCESS TO A PERMANENT FILE

PERMPF explicitly permits another user to access a file in your catalog on a remote mainframe. Parameters to PERMPF are :

ofn :
name of file for which access is being permitted.

at or on :
site-id of the mainframe the file is on.

un :
user name to be permitted access to the file.

m :
(optional) permitted mode of access. If this parameter is not coded, R (read) permission is used.

status or nostat :
coding the (optional) status key causes a message to be placed in your mailbox when PERMPF completes. When the nostat key is coded, you aren't notified of PERMPF's completion. the default is no message upon completion.

16.16 CATPF - DISPLAYS INFORMATION ABOUT A FILE

CATPF displays CATlist information about Permanent Files on a remote mainframe. Parameters to CATPF are :

ofn :
(optional) file name that catalog information is desired for.

at or on :
(required) site-id of remote mainframe to get catlist information from.

un :
(optional) user number of an alternative catalog on the remote mainframe.

lo :
(optional) list options for NOS CATLIST command. Default is full information (F).

listing or l :
(optional) name of file on the local mainframe to receive listing. If this parameter is not coded, the listing is printed at the local mainframe central site.

status or nostat :
coding the (optional) status key causes a message to be placed in your mailbox when CATPF completes. When the nostat key is coded, you aren't notified of CATPF's completion. Default is no message upon completion.

16.17 DISPOSE - PRINT A FILE AT A REMOTE SITE

DISPOSE prints a file from the local mainframe on a printer at the remote mainframe. Parameters to DISPOSE are :

file or f :
name of file to be printed.

at or on :
site-id of remote mainframe where file is printed.

print or p :
(optional) string containing parameters to control the action of the SES PRINT procedure which is used to print the file at the remote site. If you don't code the print parameter, DISPOSE prints one copy of the file at the remote site. For a complete description of the values that may be coded for the print parameter, see the description of the procedure PRINT in the SES USERS HANDBOOK.

Alphabetical Summary of Remote Link Procedures

Non-bracketed parameters are required.

```
CATPF     [ofn], at, [un], [lo],     [listing], [status]

CHANGPF   ofn,   at, [nfn], [ct],    [m],       [pw],     [status]

DISPOSE   file,  at, [print]

GETPF     rfn,   at, [un], [hfn],    [pw]

PERMPF    ofn,   at,  un,  [m],      [status]

PURGPF    ofn,   at, [un], [status]

SENDPF    hfn,   at, [un], [rfn], [pw]

SUBMIT    f,     at, [mode]
```

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES

This section describes a list of facilities to perform general massaging of text files, character set conversion and so on. The facilities to be described are :

EDT	run the EDT text editor on a file.
LOWTOUP	convert lower case letters to upper case.
UPTOLOW	convert upper case letters to lower case.
COPYACR	copy ASCII coded record.
ASORT	sort ASCII files.
UNIQUE	removes adjacent duplicate lines from a text file.
MERGE	merge up to five files on a columnar basis.
DEMERGE	split a file into one to five files on a columnar basis.
COMPARE	compare two text files and produce a listing of the differences between them.
COUNT	counts characters, words, or lines in a text file.
SELECT	extracts (displays) selected lines and/or ranges of lines from a text file.
MULTED	MULTI record file EDIT facility.

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES

PACK	"Packs" a multi record or multi file file into a single text record, with control characters such that it can be "unpacked" again.
UNPACK	"Unpacks", that is, reverses the action of PACK.
CONV	CONVert the character set of a text file.
JABFORM	convert source for the TEXTJAB document formatter to source for the TXTFORM document formatter.
WANG2CYBER	send text from a WANG document to a CYBER file.
CYBER2WANG	send text from a CYBER file to a WANG document.
SNOBOL	8-bit SNOBOL interpreter.
FIND	search a file for specified text patterns.
CHANGE	change matched text lines in a file.
XLIT	transliterate characters in a file.

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES
17.1 EDT - RUN THE EDT TEXT EDITOR

17.1 EDT - RUN THE EDT TEXT EDITOR

EDT runs the EDT text editor for you. EDT is an enhanced version of the standard NOS text editor. A description of EDT can be found in an appendix to this document. This EDT procedure is mainly intended to work in conjunction with procedures such as GETMOD(S), GETMEM(S), PACK, UNPACK and so on, so that you can, for instance, string a bunch of procedures together, like :

```
ses.getmod m=nurdle; edt; repmod
```

to get a module off a base, edit it and replace it all in one procedure line. Parameters to EDT are :

i or f :
(optional) name of Input File containing text to be EDiTed. If you don't code the i parameter, EDT uses the value of profile variable **group** as the name of the input file, and if there's no such variable defined, EDT uses a default file name of **group**.

ec or input :
(optional) name of file containing Edit Commands to EDT. If you don't code the **ec** parameter, EDT uses file **input**.

eo or output :
(optional) name of file to receive Edit Output. If you don't code the **eo** parameter, EDT uses file **output**.

t or tabs :
(optional) TABS parameter for EDT. If you don't code the t parameter, there are no tabs set. You may set up a default tabs name in your profile by defining the **tabs** variable.

save or rep :
if you code this (optional) key, EDT REPlaces the file specified by the i parameter in your catalog when you've finished editing. If you don't code the rep key, the file is left local.

17.2 LOWTOUP - CONVERT LOWER CASE TO UPPER CASE

LOWTOUP converts all LOWer case letters in a text file to UPper case letters. Parameters for LOWTOUP are :

i or f :
name of Input File containing text to be processed by LOWTOUP.

o :
(optional) name of file to receive the Output from LOWTOUP. If you don't code the o parameter, the output appears on the file specified by the i parameter.

Example of LOWTOUP Usage

```
ses.lowtoup i=lowcase, o=upcase
REVERT.    END LOWTOUP  LOWCASE -> UPCASE
```

This example shows LOWTOUP converting all lower case letters in a file called lowcase into upper case letters, and produce the output on a file called upcase.

17.3 UPTOLOW - CONVERT UPPER CASE TO LOWER CASE

UPTOLOW converts all UPper case letters in a text file to LOWer case. Parameters for UPTOLOW are :

i or f :
name of Input File containing text to be processed by UPTOLOW.

o :
(optional) name of file to receive the Output from UPTOLOW. If you don't code the o parameter, the output appears on the file specified by the i parameter.

Example of UPTOLOW Usage

```
ses.uptolow i=upper, o=lower
REVERT.    END UPTOLOW  UPPER -> LOWER
```

This example shows UPTOLOW converting all upper case letters in a file called upper to lower case letters, and producing the output on a file called lower.

17.4 COPYACR - COPY ASCII CODED RECORD(S)

COPYACR stands for COPY ASCII Coded Record(s), and as its name implies, it is the ASCII equivalent to the standard NOS COPYCR utility. This COPYACR procedure does a bit more than the standard NOS COPYCR : it acquires the input file for you, and rewrites the output file back over the input file if you specify it that way. COPYACR can also perform some character set conversions for you (see below). Parameters to COPYACR are :

i or f :
name of Input File containing text to be processed by COPYACR.

o :
(optional) name of file to receive the Output from COPYACR. If you don't code the o parameter, the output appears on the file specified by the i parameter.

cols or c :
(optional) specification of the COLumnS between which the file is to be copied. The **c** parameter must be coded as a range, or in the form **low..high**, where **low** is the left column at which to start copying, and **high** is the right column beyond which the file is not to be copied. If you don't code the **c** parameter, COPYACR uses the range **1..80** for the copy.

incset or ic :
(optional) designator for the INput file's Character SET. The table below describes the allowed designators.

outcset or oc :
(optional) designator for the OUTput file's Character SET. The table below describes the allowed designators.

Both the **incset** and **outcset** parameters default to cs612 (see below) but this can be over-ridden by defining in your profile defaults via variables with names the same as the parameters. The following table defines the allowed designators for the **incset** and **outcset** parameters :

Designator	Meaning
cs612	NOS 6/12 ASCII character set
cs64	64 character ASCII subset (display code) character set
cs812	"8 out of 12" ASCII character set

Example of COPYACR Usage

```
ses.copyacr grab hold 11..17
REVERT.    END COPYACR  GRAB -> HOLD
```

This example shows COPYACR copying file grab between columns 11 thru 17 inclusive, and placing the result of the copy on file hold. You might wonder what's the use of doing that? Well the NOS CATALOG utility outputs the record names starting in column 11, so you could use this example of COPYACR to isolate all the record names.

17.5 ASORT - SORT ASCII FILES

ASORT is a utility to sort ASCII files according to sets of sort directives. Parameters to ASORT are :

i or f :
name of Input File to be processed by ASORT.

o :
(optional) name of file to receive the Output from ASORT. If you don't code the o parameter, ASORT places the output file on the file specified by the i parameter.

keys :
list of sort KEYS to direct the sort process. Each element in the list of keys consists of three items. The first item is the start column of the field, the second item is the (optional) length of the field (default 1 character), and the third item is the (optional) sort order, A for Ascending, and D for Descending. The default is Ascending order. There is an example of ASORT at the end of this description.

retain :
this (optional) key is the SORT5 RETAIN option. If you don't code the retain key, SORT5 outputs identically keyed records in arbitrary order. If you do code the retain key, such records are output in their original order.

Note : the collating sequence for ASORT is the ASCII collating sequence. Upper and lower case letters are considered identical for the sort process, so that alphabetic items aren't separated by an entire case of the alphabet.

Example of ASORT Usage

```
ses.asort i=shuffld, o=orderly, keys=((1,7),(10,7,d),(21,4))
REVERT.    END ASORT  SHUFFLD -> ORDERLY
```

This example shows ASORT sorting the file shuffld onto the file orderly. The keys used are column 1 for a length of seven characters, in ascending order (since that's the default), column 10 for a length of seven characters in descending order, and column 21 for a length of four characters, in ascending order (again defaulted).

```
ses.asort i=higgldy, o=linedup, keys=((14, 3, d))
REVERT.    END ASORT  HIGGLDY -> LINEDUP
```

This example of ASORT highlights one of the minor inconveniences of parameters that are lists of lists, namely, that even when you've only got one key to sort on, you still need double parentheses. The following discussion explains the keys parameter.

The keys parameter is a list of sort keys; each element of that list is itself a list of up to three elements. So at the top level you get this situation :

keys=(key_1, key_2, key_3,...., key_n)

each key_i is itself a list of one to three elements :

(start_column, key_length, sort_direction)

start_column is the column position of the start of the sort key.

key_length is the number of characters in the key, inclusive of start_column - the default key_length is 1.

sort_direction is a single letter indicator : A for Ascending order (the default), D for Descending order.

So let's look at some real live examples of sort keys :

keys=((4, 2, d), (11, 7, d))

is a list for two sort keys, the first starts in column 4, is 2 characters long, and is sorting in descending order; the second starts in column 11, is 7 characters long, and is also sorting in descending order. Here's the same list, but this time the sort order is ascending, so you can leave out the direction indicators :

keys=((4, 2), (11, 7))

Now let's look at some more default cases; suppose the column 4 key is only <u>one</u> character long - now you can write the keys like this :

keys=((4, 1), (11, 7))

but the key_length of one is the default, so it can be written :

keys=((4), (11, 7))

harking way back to chapter 1 and the section called "lists of values for parameters", recall that a single element list doesn't need the parentheses :

keys(4, (11, 7))

Now let's shorten it still further by reducing the seven character field to one :

keys=(4, 11)

Getting down to the minimum case of a single key of length one in ascending order, you get :

keys=4, keys=((4, 1)) and **keys=((4, 1, a))** are all identical, whereas :

keys=(7, 12) means **keys=((7, 1), (12, 1))**

17.6 UNIQUE - REMOVE ADJACENT DUPLICATE LINES FROM A FILE

UNIQUE removes adjacent duplicate lines from a text file. Such a situation usually arises when a file has been sorted, (as for example with the ASORT procedure previously described) and duplicate lines in the file are all together. UNIQUE provides the capability of reducing the file to one instance of each unique line. For example, UNIQUE acts as follows :

Input File	**Output File**
Just the place for a Snark	Just the place for a Snark
the Bellman cried	the Bellman cried
the Bellman cried	as he landed his crew with care
as he landed his crew with care	supporting each one
supporting each one	on the top of the tide
on the top of the tide	with a finger entwined in his hair
on the top of the tide	
on the top of the tide	
with a finger entwined in his hair	

With that example to show how it works, the parameters to UNIQUE are :

i or f :
: name of the Input File containing the text which is to have the adjacent duplicate lines removed.

o :
: is the (optional) name of the Output file from the UNIQUE process. If you don't code the o parameter, the results appear on the file specified by the i parameter.

Examples of UNIQUE Usage.

```
ses.unique i=sorted, o=removed
REVERT.    END UNIQUE  SORTED -> REMOVED


ses.unique duplics
REVERT.    END UNIQUE  DUPLICS
```

17.7 MERGE - MERGE UP TO FIVE FILES INTO ONE FILE

MERGE combines up to five files on a columnar basis. Parameters to MERGE are :

o :
name of file to receive the Output of MERGE.

m or merge :
a list of up to five column widths specifying how the input files are to be MERGEd into the output file (see the example below).

i or f :
(optional) list of up to five Input Files to be MERGEd onto the file specified by the o parameter. If you don't code values for the i parameter, MERGE uses filenames tape1 thru tape5 for its input files.

Example of MERGE Usage

```
ses.merge o=diagram, m=(21,21,21), i=(column1, column2, column3)
REVERT.    END MERGE  COLUMN1,COLUMN2,COLUMN3 -> DIAGRAM
```

In this example, MERGE is constructing a file called diagram, by merging together the three files column1, column2, and column3. The contents of file column1 occupy the first 21 columns in the diagram file, the contents of file column2 occupy columns 22 thru 42 in the diagram file, and the contents of file column3 occupy columns 43 thru 63 in the diagram file.

17.8 DEMERGE - SPLIT FILE APART BY COLUMNS

DEMERGE splits a file into one to five files on a columnar basis. Parameters to DEMERGE are :

i or f :
name of Input File which is to be DEMERGEd.

d or demerge :
a list of up to five column widths specifying how the input file is to be DEMERGEd into the output files (see the example below).

o :
(optional) list of up to five files to receive the Output of DEMERGE. If you don't code values for the o parameter, DEMERGE uses **tape1** thru **tape5**.

Example of DEMERGE Usage

```
ses.demerge i=compile, d=(81,23), o=(source, numbers)
REVERT.    END DEMERGE  COMPILE -> SOURCE, NUMBERS
```

This example shows DEMERGE splitting up a file called **compile** into two components. In this example (taken from real life), the first 81 columns of **compile** are the source statements from a master audit of a deck in an UPDATE PL, and the last 23 columns are the statement numbers. The two separate sets of data appear in the example on files **source** and **numbers**.

17.9 COMPARE - COMPARE TEXT FILES

The COMPARE procedure compares two text files and produces a "listing" of the differences between them in terms of inserted, deleted, and replaced lines. Parameters to COMPARE are :

new :
this (optional) parameter specifies the name of the file containing the "new" version of the text. If you don't code this parameter, COMPARE uses a file called new.

old :
this (optional) parameter specifies the name of the file containing the "old" version of the text. If you don't code this parameter, COMPARE uses a file called old.

o :
this (optional) parameter specifies the name of the file to receive the output from COMPARE. If you don't code this parameter, COMPARE writes the results of the comparison to file output. Note that if the two files are equivalent, COMPARE doesn't write anything to the o file.

newcset :
(optional) designator for the NEW file's Character SET. The table below describes the allowed designators.

oldcset :
(optional) designator for the OLD file's Character SET. The table below describes the allowed designators.

outcset :
(optional) designator for the OUTput file's Character SET. The table below describes the allowed designators.

ls or ignorls :
these (optional) keys specify whether or not to IGNORe Leading Spaces on lines being compared. The default action is to recognise leading spaces (the ls option). If you code the ignorls key, COMPARE ignores leading spaces on text lines.

The default character set designator for all three **cset** parameters is **cs612**. The following table defines the allowed designators for the **cset** parameters :

Designator	Meaning
cs612	NOS 6/12 ASCII character set
cs64	64 character ASCII subset (display code) character set
cs812	"8 out of 12" ASCII character set

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES
17.10 COUNT - COUNT THINGS IN A FILE

17.10 COUNT - COUNT THINGS IN A FILE

COUNT counts one of the following three things in a file :

characters number of ASCII characters in a file.

lines number of lines.

words number of words in a file, where a "word" is defined as a contiguous set of characters delimited by space(s), tab(s) or newline(s).

Parameters to the COUNT procedure are :

c or chars or l or lines or w or words :
these keys define which thing is to be counted (Characters, Lines or Words).

i or f or in :
name of Input File IN which the things are to be counted.

o or to :
is the (optional) name of the Output file IO which the results of the counting process are to go. If you don't code the to parameter, the results go to file output.

Examples of COUNT Usage

```
ses.count lines in source
7396
REVERT.     END COUNT  SOURCE


ses.count words in guide to results
REVERT.     END COUNT  GUIDE -> RESULTS
```

The first example shows COUNT counting the number of lines in a file called source, with the result going to file output by default. The second example counts the number of words in a file called guide, with the answer written to a file called results. Note that the choice of parameter keywords means we can specify the COUNT process in a more "English" form.

17.11 SELECT - COPY SELECTED LINE RANGES FROM A FILE

SELECT copies selected lines or ranges of lines from one file to another. The default output file used by SELECT is in fact file **output**, so that the utility provides an easy way to look at selected parts of a file without needing to edit the file. Parameters to the SELECT procedure are :

line or lines :
is a list of line numbers or line number ranges to be selected from the input file.

i or f or of :
is the name of the Input File from which the lines or ranges of lines are to selected.

o or to :
is the (optional) name of the Output file TO which the selected lines are to be copied. If you don't code the **to** parameter, the selected lines go to file **output** by default. You can code the output file name to be the same as the input file name in which case the input file gets overwritten by the selected lines.

Examples of SELECT Usage

```
ses.select line 50 of somfile
Beware the frumious Bandersnatch my son.
REVERT.     END SELECT  SOMFILE


ses.select lines 10..19 of nurdle to twisted
REVERT.     END SELECT  NURDLE -> TWISTED


ses.select lines (5,7,9..17,19,23..47) of primes to sifted
REVERT.     END SELECT  PRIMES -> SIFTED
```

The three examples shown above should give a good idea of the way that SELECT is used, showing a single line, a range of lines, and finally a list of lines and line ranges. Invalid line numbers and

ranges are diagnosed. Line ranges in which the high end of the range is larger than the number of lines in the input file cause the selection to go to the end of the input file.

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES
17.12 MULTED - EDIT MULTI RECORD FILE

17.12 MULTED - EDIT MULTI RECORD FILE

MULTED provides the ability to edit a multi record file. This is especially useful for instance, where major and global changes must be made to many decks of a source text program library. It is easy to generate a SOURCE file, MULTED that file, then replace the decks on the PL. MULTED uses the PACK and UNPACK procedures to change the multi record file to a single record file ready for editing, and to reverse the process when editing is complete. MULTED invokes the EDT text editor, an enhanced version of the standard NOS text editor. A description of EDT can be found in an appendix to this document. Parameters to MULTED are:

i or f :
name of File containing MULTED input.

o :
(optional) name of file to receive the Output from MULTED. If you don't code the o parameter, the output appears on the file specified by the i parameter.

ec or input :
(optional) name of file containing Edit Commands to be used by EDT. If you don't code the ec parameter, MULTED assumes that the edit commands are coming from file INPUT.

eo or output :
(optional) name of file to receive Edit Output. If you don't code the eo parameter, MULTED assumes that the output is going to file OUTPUT.

t or tabs :
(optional) TABS parameter for EDT. If you don't code the t parameter, MULTED doesn't set any tabs. You may set up a default tabs name in your profile by defining the tabs variable.

cc :
(optional) Control Character to be used for the PACK-UNPACK process. If you don't code the cc parameter, MULTED uses a control character of slash (/).

save or rep :
if you code this (optional) key, MULTED REPlaces the file when editing is complete. IF you don't code either of these keys, MULTED leaves the file local. This parameter is only applicable on indirect access files.

Example of using MULTED.

```
ses.multed rigor, mortice, cc='??'
```

This example PACKs file **rigor**, invokes EDT to edit the PACKed file, and finally UNPACKs the file onto file **mortice**. The PACK process turns all physical EOR's and EOF's into the character strings ?EOR and ?EOF respectively, and the UNPACK process reverses the operation.

17.13 PACK - PACK MULTI RECORD FILE

PACK packs a multi record file into a single record in such a way that the record structure is maintained, and may be reconstituted by a subsequent UNPACK. It is a useful facility when you want to edit a multi record text file for instance. Parameters to PACK are:

i or f :
(optional) name of Input File to be PACKed. If you don't code the i parameter, PACK uses the name associated with profile variable **group**. If there isn't such a profile variable defined, PACK uses a filename of **group**.

o :
(optional) name of file to receive the Output from PACK. If you omit this parameter, the output appears on the file specified by the i parameter.

cc :
(optional) Control Character to be used for the PACK process. If you don't code the cc parameter, PACK uses a control character of slash (/).

incset or ic :
designator for the INput file's Character SET. See the table below for a description of the allowed designators.

outcset or oc :
designator for the OUTput file's Character SET. See the table below for a description of the allowed designators.

The following table defines the designators allowed for the **incset** and **outcset** parameters:

Designator	Meaning
CS612	6/12 character set (default)
CS64	display code character set
CS812	8 out of 12 ascii character set

Example of PACK Usage

```
ses.pack i=multrec, o=packed, cc='!'
REVERT.    END PACK  MULTREC -> PACKED
```

This example shows PACK being used to "pack" a multi record file called **multrec** and produce the output of the process on a file called **packed**. The control character ! is used, so that every physical EOR in the file is replaced by the character string !EOR in the output.

17.14 UNPACK - UNPACK TEXT FILE TO MULTI RECORD FILE

UNPACK provides the converse operation to that provided by PACK, or it unpacks a single record text file into a multi record file. Parameters to UNPACK are:

i or f :
(optional) name of Input File containing data to be processed by UNPACK. If you don't code the i parameter, UNPACK uses the name associated with profile variable group. If there isn't such a profile variable defined, UNPACK uses a file name of group.

o :
(optional) name of file to receive the Output from UNPACK. If you don't code the o parameter, the output appears on the file specified by the i parameter.

cc :
(optional) Control Character to be used for the UNPACK process. If you don't code the cc parameter, UNPACK uses a control character of slash (/).

incset or ic :
designator for the INput file's Character SET. See the table below for a description of the allowed designators.

outcset or oc :
designator for the OUTput file's Character SET. See the table below for a description of the allowed designators.

The following table defines the designators allowed for the **incset** and **outcset** parameters:

Designator	Meaning
CS612	6/12 character set (default)
CS64	display code character set
CS812	8 out of 12 ascii character set

Example of UNPACK Usage

```
ses.unpack i=packed, o=unpackd, cc='!'
REVERT.    END UNPACK  PACKED -> UNPACKD
```

This example shows UNPACK being used to reverse the "pack" process described in the previous section. The file **packed** is 'unpacked' onto the file **unpackd**. The control character of ! is used so that every line that has a !EOR starting in column one is replaced with a physical end of record in the output file.

17.15 CONV - CONVERT CHARACTER SET

CONV converts text files from one character set to another. If the text file contains EOFs, CONV will stop at the first EOF encountered. NOTE - Due to its flexibility, the conversion process is fairly time consuming. Parameters to CONV are :

i or f :
name of the Input File containing text to be converted.

o :
(optional) name of the file to receive the Output from CONV. If you don't code the o parameter, the output appears on the file specified by the i parameter.

incset or ic :
designator for the INput file's Character SET. See the table below for a description of the allowed designators.

outcset or oc :
designator for the OUTput file's Character SET. See the table below for a description of the allowed designators.

The following table defines the designators allowed for the **incset** and **outcset** parameters:

Designator	Meaning
N63	NOS 63 6-bit codes
N64	NOS 64 6-bit codes
N612	NOS 6/12 ASCII 6-bit and 12-bit codes
N612U	same as N612 except that all letters are upper case
N612L	same as N612 except that all letters are lower case
NBE63	NOS/BE 63 6-bit codes
NBE64	NOS/BE 64 6-bit codes

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES
17.15 CONV - CONVERT CHARACTER SET

NBE63A	NOS/BE ASCII 63 character subset 7-bit codes (5 per word)
NBE64A	NOS/BE ASCII 64 character subset 7-bit codes (5 per word)
ASCII4	full ASCII 7-bit codes (4 per word)
ASCII5	full ASCII 7-bit codes (5 per word)

A table of these characters sets can be found in an appendix to this document.

17.16 JABFORM - CONVERT TEXTJAB SOURCE TO TXTFORM SOURCE

JABFORM converts source for the TEXTJAB document formatter to source for the TXTFORM document formatter. JABFORM only works on TEXTJAB source format directives that are delimited. Parameters to JABFORM are :

i or f :
: name of the Input File containing the TEXTJAB document source.

o :
: name of the file on which the TXTFORM source is written Out. Note that unlike many other SES procedures JABFORM doesn't overwrite the Input file with the Output file.

e :
: (optional) name of the file to receive error messages from JABFORM. If you don't code the e parameter, JABFORM uses file ERRLIST.

A complete description of the conversions performed by JABFORM can be found in an appendix to this document.

17.17 WANG2CYBER - SEND TEXT FROM WANG DOC. TO CYBER FILE

WANG2CYBER transmits a document from a WANG to a text file on a CYBER. To perform the transfer, a WANG telecommunications environment is necessary. Parameters for WANG2CYBER are :

o :

(required) name of the text file on CYBER. If this file already exists, WANG2CYBER will append the WANG document to the end of this file. If this file doesn't exist, WANG2CYBER will create a direct access file in the catalog of the current user.

un :

(optional) user name in whose catalog the text file specified by the o parameter is to be found. If you don't code the un parameter, WANG2CYBER uses the user name of the current user.

17.18 CYBER2WANG - SEND TEXT FROM CYBER FILE TO WANG DOC.

CYBER2WANG transmits a text file from a CYBER to a document on a WANG. To perform the transfer, a WANG telecommunications environment is necessary. Parameters for CYBER2WANG are :

i or f :
(required) name of the text file on CYBER.

un :
(optional) user name in whose catalog the text file specified by the i or f parameter is to be found. If you don't code the un parameter, CYBER2WANG uses the user name of the current user.

txtcode or txtform :
(optional) **key** indicating to CYBER2WANG that the text file specified by the i or f parameter is **txtcode** or **txtform** source. If you don't code the **txtcode** or **txtform** **key** CYBER2WANG assumes that the text is neither **txtcode** or **txtform** source and transmits the text asis.

(WARNING: The TXTCODE or TXTFORM source must be capable of being formatted by using the BACKUP parameter on SES.FORMAT.)

If you transmit **txtcode** or **txtform** source, a WANG conversion glossary is available to further massage your document once WANG has it. Please call Betty Hawes on Controlnet No. 235-3019 (or 612-482-3019) and arrange to obtain the WANG Format Glossary 00001. Once you have obtained the glossary, attach it to your document and execute glossary entry P; then execute glossary entry L.

17.19 SNOBOL - 8-BIT SNOBOL INTERPRETER

SNOBOL is an 8-bit version of the SNOBOL4 interpreter. SNOBOL afficionados will realize that SNOBOL requires both program and data on the same input file; using the SES SNOBOL procedure you can (if you want) have the program and data on separate files, in which case SES gathers them onto the same file before running SNOBOL. The SNOBOL interpreter requires its input in 8-12 ASCII and produces its output in the same character set. Parameters are :

i or **f** :
name of Input File containing the SNOBOL source program, and, if you wish, data for the program. You can also specify a data file with the d parameter.

d or **data** :
is the (optional) name of a Data file for the SNOBOL program. If you do code this parameter, the data file is appended to the input file specified by the i parameter.

o :
(optional) name of file to receive the Output of the SNOBOL run. If you don't code the o parameter, the output appears on the file specified by the i parameter.

17.20 FIND - FIND PATTERNS IN A FILE

FIND searches a file line by line, looking for specific **text patterns** or **regular expressions**. There is a discussion on regular expressions later. The pattern to be located in the file may be specified on the SES control statement, or, if there are lower case characters which can't be represented, an argument file (default input) may be supplied. Parameters to FIND are :

p or pattern or with or using :
(optional) pattern specification. If you don't code this parameter at all, FIND takes its argument pattern from file input, and prompts you accordingly. The **p or pattern** keywords or no keyword indicate that the pattern is right there on the SES control statement. The **with or using** keywords indicate that the pattern is in an argument file which is specified as the value of the parameter.

i or in or f :
is the name of the Input File IN which the regular expressions are to be located.

o or onto :
(optional) name of the file ONTO which the Output of FIND is to appear. If you don't code the o parameter, the output appears on file **output** by default.

Examples of FIND Usage

```
ses.find '[0-9]$' in=myfile onto=output
REVERT.    END FIND  MYFILE -> OUTPUT


ses.find using=patfile in=oldfile onto=foundit
REVERT.    END FIND  OLDFILE -> FOUNDIT


ses.find '#.*#' in=compile
REVERT.    END FIND  COMPILE
```

The first example shows FIND locating all lines in **myfile** that have a digit at the end of the line. The result appears on file **output**. The second example shows FIND used with a pattern file **patfile**. The third example is locating all lines that contain a '#' sign followed by a run of any characters followed by another '#' sign. Such lines are reasonable candidates for being SYMPL comment lines.

Text Patterns and Regular Expressions

This discussion introduces the notation for regular expressions used by FIND and the CHANGE procedure in the next section. The notation is terse but versatile, and has been applied in a number of pattern matching utilities.

A **text pattern** can be a simple thing, like the letter **a** or a more elaborate construct built up from simple things, like the string **format**. To build arbitrary text patterns you need only know a few rules.

Any **literal character**, such as **a**, is a text pattern that matches that same character in the text being scanned. A sequence of literal characters like **7926** or **grabhold** is a pattern that matches any occurrence of that sequence of characters in a line of

the input.

A pattern is said to **match** part of a text line if the text line contains an occurrence of the pattern. For instance the pattern **aa** matches the line **aabc** once at position 1, the line **aabcaabc** in two places, and the line **aaaa** in three (overlapping) places. Matching is done on a line by line basis : no pattern can match across a line boundary. Text patterns may be **concatenated** : a text pattern followed by another text pattern forms a new pattern that matches anything matched by the first, followed immediately by anything matched by the second. A sequence of literal characters is an example of concatenated patterns.

find has other capabilities - the ability to search for patterns that match **classes** of characters, that match patterns only at particular positions on a line, or that match text of indefinite length.

To be able to express these more general patterns, we have to preempt some characters to represent other types of text patterns, or to delimit them. For example, we use the character **.** as a text pattern that matches **any** single character except a newline. The pattern **x.y** matches **x+y**, **xay**, **x?y** and similar strings.

The **.** and other reserved characters are often called **metacharacters**. We try to choose characters which won't appear with high frequency in normal text, but still there are occasions when we want to look for a literal occurrence of a metacharacter. Thus the special meaning of a metacharacter may be turned off by preceding it with the **escape** character **@**. Thus **@.** matches a literal period, and **@@** matches a literal at sign.

The metacharacter **[** signals that the characters following, up to the next **]** form a **character class**, that is, a text pattern that matches any single character from the bracketed list. For example, **[aA]** matches **a** or **A**, **[a-z]** matches any lower case letter. There is a **negated character class**, such that **[~a-z]** matches any character except a lower case letter, and so forth. Note that a negated character class never matches a newline. The escape convention can also be used inside character classes if the character class is to contain **~** or **-** or **@** or **]**.

Two other metacharacters don't match literal characters but rather match positions on the input line. **^** matches the **beginning** of a line : **^abc** is a pattern that matches **abc** **only** if it appears as the first three characters of an input line. Similarly, **$** matches the newline at the end of a line : **abc$** matches **abc** only if it is the last thing on a line before the newline. Of course these can work together : **^abc$** matches a line that contains **only** **abc**, and **^$** matches only empty lines (lines

containing only a newline).

Any of the text patterns above that match a single character (everything but ^ and $) can be followed by the character * to make a text pattern which matches zero or more successive occurrences of the single character pattern. The resulting pattern is called a **closure**. For example, a* matches zero or more a's; aa* matches one or more a's; [a-z]* matches any string of zero or more lower case letters.

Since a closure matches zero or more instances of the pattern, which do we pick if there's a choice? It turns out to be most convenient to match the longest possible string even when a null string match would be equally valid. Thus [a-zA-Z]* matches an entire word (which may be a null string), [a-zA-Z][a-zA-Z]* matches an entire word (one or more letters but not a null string), and .* matches a whole line (which may be a null string). Any ambiguity in deciding which part of a line matches a pattern is resolved by choosing the match beginning with the leftmost character, then choosing the longest possible match at that point. So [a-z][a-z0-9]* matches the leftmost FORTRAN identifier on a line, (.*) matches anything between parentheses, and ..* matches an entire line of one or more characters (but not a line containing only a newline).

Finally, no pattern matches across a line boundary. This is often most natural and useful, and it prevents an unwise .* from eating up an whole file.

Summary of Regular Expressions

The previous discussion introduced a lot of new notation for matching text patterns and regular expressions. Here is a summary of what went before.

c	literal character (for example s)
.	any character except newline
^	beginning of line
$	end of line (null string before the newline)
[...]	character class (any one of these characters). for example, [ijklmn] matches the first character of a Fortran integer identifier.
[~...]	negated character class (all but these characters)
*	closure (zero or more occurrence of previous pattern)
@c	escaped character (for example, @^, @$, @[, @*)

Any special meaning of characters in a text pattern is lost when escaped, inside a [...], or for :

^	not at beginning
$	not at end
*	at beginning

A character class consists of zero or more of the following elements, surrounded by [and] :

c	literal character, including [
a-c	range of characters (digits, lower or upper case letters)
~	negated character class if at beginning
@c	escaped character (@~ @- @@ @])

Special meaning of characters in a character class is lost when

escaped or for :

~ not at beginning (for example [a~9jug])

- at beginning or end, for example [-+:z] or [+=abc-]

Note : the following three **substitution patterns** apply to the CHANGE utility described in the next section. A **substitution pattern** consists of zero or more of the following elements :

c literal character

& ditto, that is, whatever was matched

c escaped character (@&)

An **escape sequence** consists of the character @ followed by a single character :

@n newline

@t tab

@c c (including @)

17.21 CHANGE - CHANGE LINES THAT MATCH SPECIFIED PATTERNS

CHANGE changes patterns in a file. The from and to patterns may be specified on the control statement, or they may be given as the first two lines of an argument file. Both from and to must be specified, or a pattern file may be given via the with or using parameter. If none of these parameters are supplied, CHANGE takes its pattern argument from file input. Parameters to CHANGE are :

i or f :
name of Input File which is to have its patterns changed.

from :
(optional) character string which represents the from string to be matched.

to :
(optional) character string which represents the to substitution string.

with or using :
(optional) name of a file containing the from and to patterns on two separate lines. This file is used if there are lower case patterns in the from or to parameters which can't be entered via the control statement.

onto or o :
(optional) name of file to receive the Output from CHANGE. If you don't code the o parameter, the output appears on file output by default.

Examples of CHANGE Usage

```
ses.change myfile from '^[0-5]*' to '=MARK=&' onto outfile
REVERT.    END CHANGE  MYFILE -> OUTFILE


ses.change oldfile using patfile
REVERT.    END CHANGE  OLDFILE
```

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES
17.21 CHANGE - CHANGE LINES THAT MATCH SPECIFIED PATTERNS

The first example shows CHANGE locating all lines that have an arbitrary number of digits between 0 and 5 at the beginning of the line, and changing the pattern so found into the string '=MARK=' followed by the string of digits that was found. The second example shows CHANGE operating on a file, where the pattern arguments are supplied via the argument file patfile.

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES
17.22 XLIT - TRANSLITERATE CHARACTERS

17.22 XLIT - TRANSLITERATE CHARACTERS

XLIT transliterates characters in a file. The from and to patterns may be specified on the control statement, or they may be given as the first two lines of an argument file. Both from and to must be specified, or a pattern file may be given via the with or using parameter. If none of these parameters are supplied XLIT takes its pattern argument from file input. Parameters to XLIT are :

i or f :
name of Input File which is to have its characters transliterated.

from :
(optional) character string which represents the from string to be transliterated.

to :
(optional) character string which represents the to transliteration string.

with or using :
(optional) name of a file containing the from and to patterns on two separate lines. This file is used if there are lower case characters in the from or to parameters which can't be entered via the control statement.

onto or o :
(optional) name of file to receive the Output from XLIT. If you don't code the o parameter, the output appears on file output by default.

Examples of XLIT Usage

```
ses.xlit myfile from 'ABCDE' to 'VWXYZ' onto outfile
REVERT.    END XLIT  MYFILE -> OUTFILE
```

```
ses.xlit oldfile using patfile
REVERT.    END XLIT  OLDFILE
```

Summary of XLII Capabilities

This is a short summary of the facilities provided by XLIT. The examples below just show the from and to patterns. The simplest form of XLIT is to simply convert one character to another, like :

```
from x   to y
```

and have all occurrences of x in the input file be replaced by y in the output file. Multiple translations are also handy :

```
from xy   to yx
```

changes all x's into y's and all y's into x's. There is also a nice shorthand notation

```
from a-z   to A-Z
```

to translate all lower case letters to upper case, or

```
from a-zA-Z   to A-Za-z
```

to do case reversal. Even good typists prefer

```
A-Z  a-z
```

to

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ  abcdefghijklmnopqrstuvwxyz
```

There are times when we would like to translate a whole class of characters into just one character, and then to collapse runs of that translation into just one instance. For example, translating spaces, tabs and newlines into newlines and then collapsing multiple newlines leaves each of the words in a document on a separate line, ready for further processing. Or we might want to convert all alphabetic symbols in a program into a's and all numbers into n's. We specify this collapsing operation by giving a second argument that is shorter than the first :

```
from a-zA-Z   to a
from 0-9   to n
```

The implication is that the last character in the second argument (the to string) is to be replicated as often as necessary to make a string as long as the first argument, and that this replicated character should never appear twice in succession in the output.

xlit also provides an escape convention for writing tabs and newlines so that they are visible and cause a minimum of grief for programs that must inspect the arguments. We use the at-sign @ as an escape character : whatever character follows the escape character is in some way special. In particular, we define @t to be a tab and @n to be a newline, so we can write :

```
@t@n @n
```

to change all occurrences of "whitespace" (spaces, tabs and newlines) to just one newline and leave one word per line.

18.0 MISCELLANEOUS USEFUL GOODIES

18.0 MISCELLANEOUS_USEFUL_GOODIES

This section covers a number of useful SES procedures that fall into no other category than miscellaneous. The facilities in this section are described briefly here.

IAF sets up IAF terminal defaults for specific terminal types.

AUTOMOD automatically modifies deck names on a MADIFY library from the standard 170 length of 7 characters or less to NVE allowable length of 31 characters or less.

DO a "control statement generator" that can optionally generate a batch job "on the fly".

MATH a quick and dirty desk calculator.

BELL rings the bell as an attention getter.

BYE logs out your current interactive session.

HEXDMP produces a hexadecimal dump of a (binary) file.

CONCAT concatenates a list of files into a single file, with the resulting file "packed" into a single "logical record".

SCATTER performs the reverse action of COLLECT, that is it "scatters" a multi record file to a number of individual files.

18.0 MISCELLANEOUS USEFUL GOODIES
18.1 IAF - SET UP INITIAL IAF TERMINAL PARAMETERS

18.1 IAF - SET UP INITIAL IAF TERMINAL PARAMETERS

The IAF procedure defines your terminal to the NOS IAF facility. In addition, it will notify you of MAILBOX messages, and updates to SES.INFO.

There are defaults for all the different terminal types, and normally, the user only specifies his/her terminal type either on the proc call or in his/her PROFILE. All the strange parameters, including the terminal type, may be set up in your profile. As an example, if you're logged in to a CDC 713 terminal, you can set up the terminal controls by simply typing :

```
ses.iaf term=cdc713
```

The table on the next page describes the parameters for IAF, the defaults, and the profile variables that you can set to change the defaults chosen by the IAF procedure.

Note : while the IAF procedure is running, do not hit carriage return (CR) to find out what's going on, even if it does seem a little slow. At the time of writing, due to the way the NOS TRMDEF command works, hitting a carriage return interferes with the TRMDEF, which will issue a line of garbled data to your terminal, and the TRMDEF command gets ignored. In other words, the IAF procedure won't do anything.

18.0 MISCELLANEOUS USEFUL GOODIES
18.1 IAF - SET UP INITIAL IAF TERMINAL PARAMETERS

Parameter Name	Default Value	Profile Variable	Parameter Usage
term, t	No Default	term	Terminal type. For a list of the terminal types currently recognized by the IAF procedure, see the list at the end of this table.
intchr, ic	see below	intchr	Character to INTERRUPT a process.
termchr, tc	see below	termchr	Character to TERMINATE a process.
canchr, cc	see below	canchr	Character to CANCEL a line input at the terminal.
conchr, ct	see below	conchr	Character to use as a CONTROL character when giving commands (such as page wait on or off) to the TIP.
bschr, bs	see below	bschr	Character to use to delete previous input character.
width, pw	see below	width	Determines maximum number of characters a terminal can print on one output line.
length, len	see below	length	Determines maximum number of lines that can be printed as one page.
pgwait, pg	no	pgwait	Determines whether page waiting is to be performed for the terminal.
half/full	full	duplex	Determines whether your terminal is to run in HALF or FULL duplex. These are just keys.

18.0 MISCELLANEOUS USEFUL GOODIES
18.1 IAF - SET UP INITIAL IAF TERMINAL PARAMETERS

ci	0	ci	Carriage return idle count for IST3 or TI745.
li	0	li	Line feed idle count for IST3, TI745 or CDC751.
hardcopy	no	hardcopy	If you use a hardcopy device on your IST3, the defaults for ci and li are changed to 50 each.

The defaults given above, namely "IAF default", are those chosen by IAF (part of NOS), as opposed to IAF (the SES procedure). We have copped out by not stating what they are here, since they change from time to time. For a list of the defaults, consult the IAF reference manual, or see your local NOS wizard.

The term (for terminal) profile variable is set up when you use BLDPROF (BuiLD PROFile), and the IAF procedure uses this profile variable. If you use the IAF procedure and no terminal type is specified, either via the term parameter or the term profile variable, IAF prompts for the terminal type. You can edit your profile to insert all the other defaults if you wish.

Terminal types that the IAF procedure knows about at present are :

CDC713, CDC722, CDC722A, CDC751, CDC752, DX132A, DC300, DATAPAC, NCR620, TI745, TTY43, DECW3, TEK4014, IST3, CDC721, Z19.

18.0 MISCELLANEOUS USEFUL GOODIES
18.1 IAF - SET UP INITIAL IAF TERMINAL PARAMETERS

The table below describes the control characters and other parameters that the IAF procedure establishes for you. **Note** : that a page width of zero (0) allows the maximum page width.

TERMINAL TYPE	CONTROL CHAR *	CANCEL CHAR *	INTERRUPT CHAR *	TERMINATE CHAR *	BS CHAR *	PAGE WIDTH *	PAGE LEN *	PAGE WAIT *	ECHOPLEX MODE *
CDC713	ESCAPE	LINE CLEAR	STX	ETX	BS	0	16	N	FULL
CDC722	ESCAPE	EOL	CTRL/P	TAB	BS	0	24	N	FULL
CDC722A	ESCAPE	EOL	CTRL/P	TAB	EM	0	24	N	FULL
CDC751	ESCAPE	GS	CTRL/P	CTRL/T	BS	0	24	N	FULL
CDC752 Z19	ESCAPE	CTRL/X	CTRL/P	CTRL/T	BS	0	24	N	FULL
DX132A	%	CTRL/X	CTRL/P	CTRL/T	BS	132	30	N	FULL
DC300	ESCAPE	VT	FF	HT	CTRL/H	118	N/A	N/A	FULL
DATAPAC	ESCAPE	CTRL/X	CTRL/P	CTRL/T	CTRL/H	0	N/A	N/A	HALF
NCR260 TI745 TTY43 DECW3	ESCAPE	CTRL/X	CTRL/P	CTRL/T	CTRL/H	0	N/A	N/A	FULL
TEK4014	ESCAPE	CTRL/X	CTRL/P	CTRL/T	CTRL/H	0	64	N	FULL
CDC721	ESCAPE	CTRL/X	CTRL/P	CTRL/T	CTRL/H	0	30	N	FULL
IS'13	ESCAPE	CTRL/X	CTRL/P	CTRL/T	CTRL/H	0	34	N	FULL

The default values can be changed by coding a parameter that corresponds with one of the columns marked with *. If half duplex is desired, simply code the half key.

Note : when specifying an octal code to change a default value, remember that the radix, enclosed in parentheses, must follow the number, for example :

conchr=33(8)

18.0 MISCELLANEOUS USEFUL GOODIES
18.1 IAF - SET UP INITIAL IAF TERMINAL PARAMETERS

Note : character values can be changed by entering the desired character enclosed in string quotes (apostrophes), instead of octal numbers.

18.2 AUTOMOD - AUTOMATIC MODIFY TO SCU NAME CHANGE GENERATOR

AUTOMOD is a MODIFY library to SCU library tool that eases the conversion process by automatically determining the 31 character deck names when a standard way of determination exists. The output of this process is a file of commands that in turn can be used as input to the SCU conversion command, CONVERT_MODIFY_TO_SCU. This reduces the number of deck names that must be changed manually (one at a time). Note that AUTOMOD can potentially be unable to handle extra large libraries. It is suggested that merely creating two libraries, each with half the decks, should solve the problem. Parameters to AUTOMOD are :

b or l :
: name of Base Library containing the MODIFY decks to be searched for the long name.

o :
: (optional) name of file to receive the Output from AUTOMOD. If you don't code the o parameter, AUTOMOD places the output on the file NEWNAME.

un :
: (optional) User Name in whose catalog the base library specified by b exists.

The program uses a simple algorithm to do the translation. It searches the deck for a procedure name with the same first two characters as the deck name. The SIS specifies the legal types of decks and the coding of the short names. 'D' decks do not consistently contain an identifiable choice for the long name, so no attempt is made to find one. 'X' decks contain the XREF declaration for a procedure or function and this name is used as the long name. 'H' decks contain a commented explanation of the 'X' deck and the procedure name is simply searched for and used with 'H' substituted as the third character to distinguish them from the long names for the 'X' decks. 'E' decks have not been consistently used for the same purpose, but most of them end up with the likely long name anyway. 'M' decks should end up with the long module name off of the same line as the keyword MODULE. 'A' decks should work the same off of the word ASSEMBLY.

18.3 DO - CONTROL STATEMENT GENERATOR

DO accepts a list of character strings, or a file of commands, and outputs them to the NOS control statement stream. The stream of commands so generated may be submitted to the batch system if desired. For example :

```
ses.do ('rewind,*','get,myfile','dispose,myfile=pr','ses.catlist')
```

All sorts of NOS control statements may be given as parameter values to the DO procedure. Note that a perfectly valid DO parameter value is a call on SES processor itself, so that if you were feeling sufficiently playful, you could code lines like :

```
ses.do 'ses.do ''ses.catlist'''
```

Parameters to DO are :

cs :
(optional) list of Control Statements to be output to the control statement file.

i :
(optional) name of file containing control statments to be executed. If you don't code the i parameter, DO prompts you to enter the statments from file input. Coding the i parameter causes DO to submit a batch job containing the stream of commands from the file.

l :
(optional) name of Library in which the text record given by the i parameter may be found. The library must have a directory. You use this feature to place canned streams of JCL (such as build jobs) in a LIBEDIT style library. See the chapter on LIBRARY FILE MANAGEMENT USING LIBEDIT for hints on how to manage LIBEDIT libraries.

un :
(optional) User Name in whose catalog the file given by the i parameter, or the library given by the l parameter may be found, if it is not in the current user's catalog.

f or file or files :
(optional) list of **LOCAL** FILES that are to be copied onto

the end of the job file if DO is being used to create a batch job (one of the **batch**, **batchn** or **defer** keys are coded). If you code any values for the **files** parameter, DO generates the job control to COPYBF each of the local files onto the end of the job file before SUBMITTING it and also generates the control statements in the jobfile itself that are necessary to COPYBF those files from INPUT when the job runs. If any of the filenames coded for the **files** parameter are not assigned to the job when you code the **ses.do** statement, the DO procedure aborts with an error message.

batch job parameters :

These parameters are described in the section entitled "SES PROCEDURES RUN AS BATCH JOBS". The default for this procedure is to run in **local** mode, but it can be run in **batch** mode. The **dayfile** parameter is not used by this procedure.

Example of Using DO to submit a batch job

```
ses.do ('ses.gencomp mincIud', 'ses.cybil', .&
'ses.print listing', 'save,igo'), batch;  bye
 10.23.56. SUBMIT COMPLETE.  JOBNAME IS ABLEBJD
REVERT.    JOB DO  SUBMITTED
```

The four control statements (mostly consisting of SES procedures of course!) are built into a batch job and submitted. The bye procedure then logs you off.

18.4 MATH - INTERACTIVE KEYBOARD CALCULATOR

MATH is an SES procedure that interacts with the user to provide a "quick and dirty" interactive calculator. Only integer arithmetic of 48 bits precision is provided.

MATH works on the concept of the current variable. When MATH is started, it starts with a variable called VALUE, whose value is 0. Every time an expression is assigned to a variable, that variable becomes the current variable. An assignment statement with no variable name on the left of the operator assigns the expression to the current variable. Type END to terminate this procedure. Valid entries are :

= expression	assigns the value of "expression" to the current variable.
+ expression	adds the value of "expression" to the current variable.
- expression	subtracts the value of "expression" from the current variable.
* expression	multiplies the current variable by the value of "expression" and assigns the result to the current variable.
/ expression	divides the current variable by the value of "expression" and assigns the result to the current variable.
// expression	performs the calculation current variable MODULO "expression" and assigns the result to the current variable.
** expression	raises the current variable to the power "expression" and assigns the result to the current variable.
name = expression	assigns the value of "expression" to the variable specified by "name", which then becomes the current variable.

18.0 MISCELLANEOUS USEFUL GOODIES
18.5 BELL - ATTENTION GETTER OR WAKE UP

18.5 BELL - ATTENTION GETTER OR WAKE UP

BELL is a useful facility for those user's who fall asleep at the terminal. This BELL procedure "rings the bell" a specified number of times. Parameters to BELL are :

n :

is the (optional) number of times to "ring the bell". If you don't code the n parameter, BELL rings twice.

18.0 MISCELLANEOUS USEFUL GOODIES
18.6 BYE - LOG OUT

18.6 BYE - LOG OUT

BYE is an SES procedure that sends a special control code to the user's terminal, to log it out. This provides the useful facility that you can set a stream of SES procedures going, with the last one being BYE, and you can then walk away from the terminal and let the system get on with the job. For example, the following is a fairly typical sequence :

```
ses.gencomp themod1; compass; print listing; 'save,lgo'; bye
```

The example shown GENerates a COMPile file for module themod1, compiles it with the COMPASS assembler, PRINTS the listing output of the assembler, saves the binary output of the assembler on file lgo, and finally logs the terminal off via the BYE procedure. The programmer is able to type that line of SES procedure calls, then go and have tea while it's all happening.

18.7 HEXDMP - GENERATE HEXADECIMAL DUMP

HEXDMP produces a printable dump of a binary file. In the dump, CYBER words (addressed in octal) are shown in hexadecimal with 8/12 ASCII interpretation. Parameters to HEXDMP are :

i :
name of the local file from which to produce the dump.

o :
name to be used for the local file containing the printable dump.

p or t :
(optional) type of dump to produce :
P printable - 8 words per line for line printer
T terminal - 4 words per line for terminal output (default)

a or h :
these (optional) keys determine if ASCII interpretation of the dump is to be included in the output. If you code the a key, ASCII is included - this is the default action. If you code the h key (Hex only) key, ASCII is not included in the dump.

Example of HEXDMP Usage

```
ses.hexdmp file1 tlist t
REVERT.   FILE DUMP COMPLETE
```

This command creates a file 'TLIST' suitable for displaying on a terminal showing the contents of file 'FILE1' in hexadecimal.

18.0 MISCELLANEOUS USEFUL GOODIES
18.8 CONCAT - CONCATENATE FILES

18.8 CONCAT - CONCATENATE FILES

CONCAT concatenates files onto a single "group" file. It is similar to COLLECT except that CONCAT works with files instead of records, and the resultant group file is PACKed upon completion of the process. Parameters to CONCAT are :

i or f :
is a list of names of files to be concatenated onto the group file.

g or group :
(optional) name of group file to receive the final concatenated set of files. If you don't code the group parameter, CONCAT uses the value of profile variable group, and if there's no such variable defined, uses the name group.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

Examples of CONCAT Usage

```
ses.concat (myfile,hisfile,anyfile)
*    MYFILE -> GROUP
*    HISFILE -> GROUP
*    ANYFILE -> GROUP
REVERT.    END CONCAT  MYFILE..ANYFILE -> GROUP


ses.concat (grab,hold,grip,clutch) g=titefin, nomsg
REVERT.    END CONCAT  GRAB..CLUTCH -> TITEFIN
```

18.0 MISCELLANEOUS USEFUL GOODIES
18.9 SCATTER - SCATTER A MULTI RECORD FILE TO OTHER FILES

18.9 SCATTER - SCATTER A MULTI RECORD FILE TO OTHER FILES

SCATTER copies the records of a multi record file to a number of other files. Parameters to SCATTER are :

m or f :
: list of names of Files or Member files to which the records of the single file are to be copied. Names may appear in the list more than once, if you want to divert sundry records to the same file.

g ot group :
: (optional) name of GROUP file containing the records to be copied. If you don't code the g parameter, SCATTER uses the value of profile variable group, and if there's no such variable defined, uses the file name group.

nr :
: if you code this (optional) key, SCATTER does not rewind the group file before starting the copy sequence. If you don't code the nr key, SCATTER rewinds the group file before doing any copying.

Examples of SCATTER Usage

```
ses.catalog newprox
GETPXDB..TEXT     TOOLDOC..TEXT     GETMAIL..TEXT     COMPARE..TEXT
GENCORS..TEXT     GENREVB..TEXT     CAPS999..TEXT
REVERT.    END CATALOG  NEWPROX
ses.scatter (cerbic,varnel,taviot,cerbic,varnel,cerbic,footle),....
..? g=newprox
REVERT.    END SCATTER  CERBIC..FOOTLE <- NEWPROX
ses.catalog cerbic
GETPXDB..TEXT     COMPARE..TEXT     GENREVB..TEXT
REVERT.    END CATALOG  CERBIC
ses.catalog varnel
TOOLDOC..TEXT     GENCORS..TEXT
REVERT.    END CATALOG VARNEL
```

The example shows how SCATTER is used to separate the records from the file newprox onto the different files given in the list on

the SCATTER call. The CATALOG procedures are used here to show the "before and after" effect of SCATTER.

19.0 HANDLING UPDATE PROGRAM LIBRARIES

19.0 HANDLING UPDATE PROGRAM LIBRARIES

For those who use UPDATE as the source text maintenance regime for software development, the set of SES procedures described below supply a fairly easy to use interface to UPDATE. A short summary of these SES procedures appears first, with the more detailed descriptions following. The SES procedures to drive UPDATE are as follows :

GENUPCF GENerate UPdate Compile File. Generates a compile file for specified decks with modsets optionally applied. Either a Quick mode or Full mode UPDATE can be performed.

GETDECK GET DECK(s) for editing. This is intended as a preliminary to a GENMOD(S) procedure usage in order to generate modsets.

GENMOD(S) GENerate MODification Sets for decks on a PL. GENMODS uses the SCOOP utility to compare two files and generate UPDATE modification directives.

GENUPSF GENerate UPdate Source File. A source file may be generated for specific decks or common decks, with modsets optionally applied.

UPDATE applies a list of modsets to an old program library and creates a new program library.

19.0 HANDLING UPDATE PROGRAM LIBRARIES

PROFILE Variable for UPDATE Facilities

The update procedure interfaces use only three profile variables, but these are of great utility when you're working on the same PL for long intervals. The profile variables are :

oldpl name of the program library to be used when generating compile files, edit files and so on. It is the default for the **pl** or **oldpl** parameter on the procedures.

newpl name of a new program library when running the UPDATE procedure. It is the default for the **npl** or **newpl** parameter on the UPDATE procedure.

plowner PL OWNER is the user name of the catalog in which the program library resides. It is the default for the **un** parameter on all the procedures.

updtmcc UPDaIe Master Control Character. If you're working with such things as 'slash PL's', this profile variable is useful for overriding the standard master control character which is the asterisk (*) character.

19.0 HANDLING UPDATE PROGRAM LIBRARIES

Working with the UPDATE Procedures

In order to get the best use out of the UPDATE interface facilities, it is anticipated that users will follow the set of working procedures outlined below.

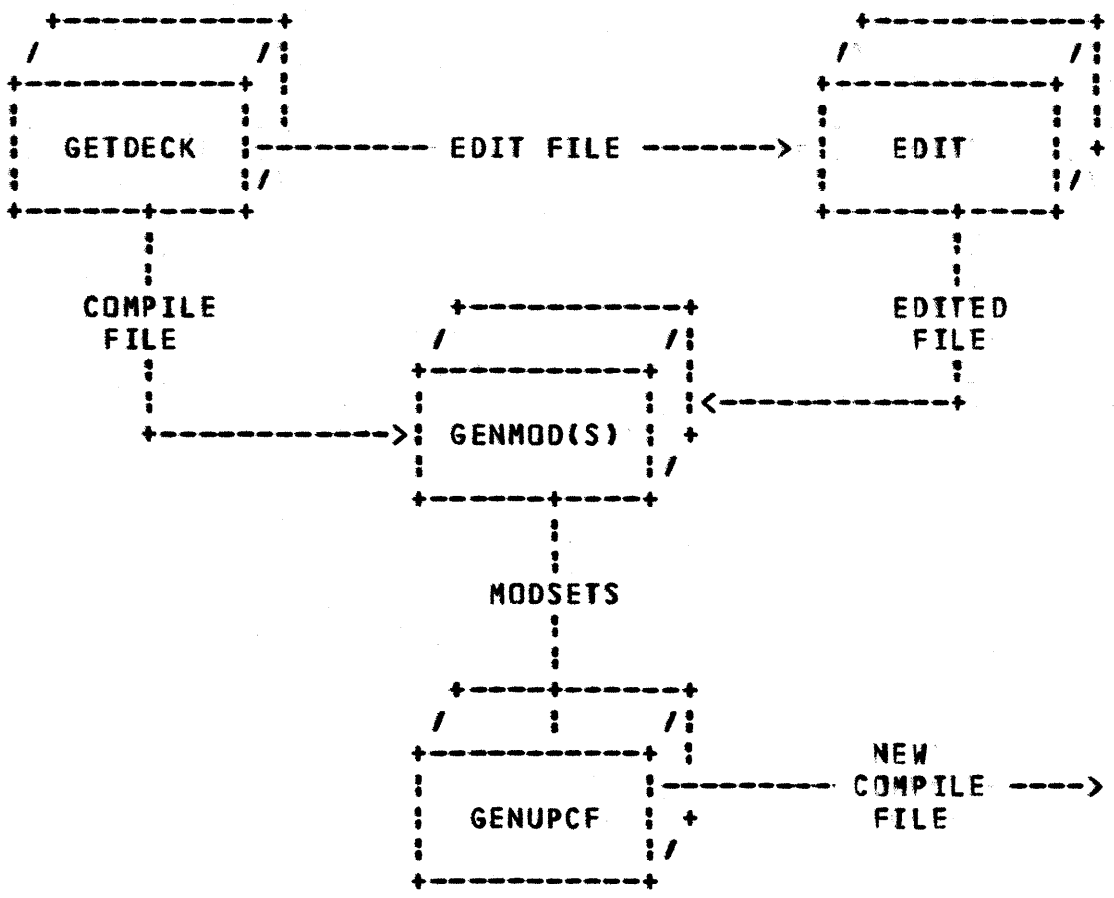
- Most of the procedures work on the concept of applying modsets to an existing program library, and only making a new program library at fairly infrequent intervals, say once every two weeks or so. This allows many people to be working on unconnected decks at a time, with one person controlling the building of new program libraries.

- To make modifications to a deck, use GETDECK to generate an EDIT FILE (plus a COMPILE FILE); edit the edit file, then use GENMOD(S) to generate modsets which can then be used in subsequent GENUPCF runs.

- To make major changes to decks, use GENUPSF to get the source of the deck out, and edit that.

The working scheme suggested here has been in use, successfully, for some time. It is far easier to comprehend and handle than it is to be 'manually' creating correction sets. Pictorially, the scheme looks something like this :

19.0 HANDLING UPDATE PROGRAM LIBRARIES



To show how this goes in practice, on the next page there's an example of a complete session of work involving the three widely used procedures GETDECK, GENMOD(S) and GENUPCF. We want to generate modsets for decks "delta", and "mu" through "phi". User typing is in lower case, and NOS messages in UPPER CASE. It is assumed that the program library - PROGLIB - is specified in your profile.

19.0 HANDLING UPDATE PROGRAM LIBRARIES

```
ses.getdeck d=(delta,mu..phi)
*    GENERATING COMPILE FILE COMPILE
*    GENERATING EDIT FILE EDTFILE
REVERT.    END GETDECK EDTFILE, COMPILE <- PROGLIB


edit,edtfile
 ...
   editing
      session
         ...
EDIT,EDTFILE


ses.genmods anymods jan0879 cd=(delta,mu..phi)
REVERT.    END GENMOD COMPILE:EDTFILE -> ANYMODS


ses.genupcf anymods
*    GENERATING COMPILE FILE COMPILE
REVERT.    END GENUPCF COMPILE <- PROGLIB
```

As you can see from the example, the amount of typing you do is small, there are in fact only four commands (excluding the actual edit session commands), the system does the rest.

19.0 HANDLING UPDATE PROGRAM LIBRARIES
19.1 GENUPCF - GENERATE UPDATE COMPILE FILE

19.1 GENUPCF - GENERATE UPDATE COMPILE FILE

GENUPCF generates a compile file for any specified decks on an UPDATE program library. Modsets can be applied also. Parameters to GENUPCF are :

m or mods :
is an (optional) list of names of files containing modsets to be applied against the PL when generating the compile file. One or both of the mods parameter and the d parameter must be specified.

d or all :
is an (optional) list of deck names to be placed on the compile file. If the all key is coded, this implies a full update in which case no deck names should be specified. One or both of the mods and the d parameter must be specified.

cf :
(optional) name of the file to receive the Compile File. If you don't code this parameter, the compile file appears on a file called compile.

pl or oldpl :
(optional) name of the UPDATE program library from which to generate the compile file. If you don't code the pl parameter, GENUPCF uses the value of profile variable oldpl as the name of the program library, and if there's no such profile variable defined, uses the name oldpl.

un :
(optional) name of user in whose catalog the program library resides. If you don't code the un parameter, GENUPCF uses the value of profile variable plowner (PL OWNER) as you name, and if there's no such profile variable defined, GENUPCF uses the name of the current user.

status or sts :
these (optional) keys are used for those cases where GENUPCF is being used as a building block of a larger procedure. The NOS job control register EFG is set to zero (0) in the event of a successful completion, and to non

zero if the UPDATE process bombs for any reason. If status is not specified and an error occurs, the procedure EXITs.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

mc :
(optional) Master Character to be used instead of the standard master character (*). If you don't code the mc parameter, GENUPCF uses the value of profile variable updtmcc, and if there's no such profile variable, uses the default master character asterisk (*).

Examples of GENUPCF Usage

```
ses.genupcf  modfile
*    GENERATING COMPILE FILE  COMPILE
REVERT.     END GENUPCF  COMPILE <- MOLDPL
```

```
ses.genupcf m=(mods1,mods2), d=(alldig,whiles,defines), pl=anypl
*    GENERATING COMPILE FILE  XPANDED
REVERT.     END GENUPCF  XPANDED <- ANYPL
```

The first example shows the simplest usage of GENUPCF to generate the default compile file compile from a source program library called moldpl (Master OLDPL) whose name is presumably defined in the user's profile. The compile file is generated by applying the modsets in the file modfile.

The second example shows a slightly more complicated situation where you are applying a list of modsets and specifying a list of decks to be placed on a compile file called xpanded. The compile file is to come from a source program library called anypl.

19.2 GETDECK - GET DECK(S) FOR EDITING

GETDECK is used to produce two files from an UPDATE program library : a compile file with sequence numbers and an edit file without sequence numbers. The edit file is altered using an editor or suchlike. The two files can then be used as inputs to GENMOD(S) to produce UPDATE correction sets. Parameters to GETDECK are :

m or mods :
is an (optional) list of names of files containing modsets to be applied against the PL when generating the compile and edit files. One or both of the **mods** parameter and the **d** parameter must be specified.

d or decks or c or coms or all :
is an (optional) list of deck or common deck names to be placed on the compile and edit files. If the **all** key is coded, this implies a full update in which case no deck names should be specified. One or both of the **mods** and the **d** parameter and the **mods** parameter must be specified.

ef :
(optional) name of the file to receive the Edit File. If you don't code this parameter, the edit file appears on a file called **edtfile**.

pl or oldpl :
(optional) name of the UPDATE program library from which to generate the compile file. If you don't code the **pl** parameter, GETDECK uses the value of profile variable **oldpl** as the name of the program library, and if there's no such profile variable defined, uses the name **oldpl**.

un :
(optional) name of user in whose catalog the program library resides. If you don't code the **un** parameter, GETDECK uses the value of profile variable **plowner** (PL OWNER) as you name, and if there's no such profile variable defined, GETDECK uses the name of the current user.

cf :
(optional) name of the file to receive the compile file. If you don't code this parameter, the compile file appears on a file called **compile**.

status or sts :
these (optional) keys are used for those cases where GETDECK is being used as a building block of a larger procedure. The NOS job control register EFG is set to zero (0) in the event of a successful completion, and to non zero if the UPDATE process bombs for any reason. If **status** is not specified and an error occurs, the procedure EXITs.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

mc :
(optional) Master Character to be used instead of the standard master character (*). If you don't code the **mc** parameter, GENUPSF uses the value of profile variable **updtmcc**, and if there's no such profile variable, uses the default master character asterisk (*).

Example of GETDECK Usage

```
ses.getdeck (list,of,todays,modsets,to,apply,against,the,pl)
*    GENERATING COMPILE FILE  COMPILE
*    GENERATING EDIT FILE  EDTFILE
REVERT.    END GETDECK  EDTFILE, COMPILE <- SOLDPL
```

This example shows a whole raft of modsets applied to a program library whose name you defined as SOLDPL (Super OLDPL) in the profile. The compile file and edit file names take their defaults of compile and edtfile. The edit file edtfile may now be edited to provide a basis for a GENMOD(S) run to generate some more modsets.

19.0 HANDLING UPDATE PROGRAM LIBRARIES
19.3 GENMOD(S) - GENERATE MODSETS FOR UPDATE

19.3 GENMOD(S) - GENERATE MODSETS FOR UPDATE

GENMOD(S) uses the SCOOP utility to compare two files - a new file and an old compile file with sequence numbers - and produce an UPDATE correction set which can subsequently be used by the other UPDATE interface procedures. Parameters to GENMOD(S) are :

m or mods :
(optional) name of file to receive the generated **modsets**. If you don't code the **m** parameter, GENMOD(S) places the generated modsets on a file called **modsets**.

i or id or ident :
(optional) name of the IDENT to appear as the first line of the generated **modsets**. If you don't code the **ident** parameter, the **ident** name defaults to that of the file specified by the **m** parameter.

c or cd :
(optional) Compile Directives are the name(s) of deck(s) to be compiled when the modsets are eventually applied in a compile file generation. Ranges of decks may be specified.

n or new :
(optional) name of the NEW file (the edited file without sequence idents). If you don't code the **new** parameter, GENMOD(S) uses **edtfile**, which is the default edit file name generated by GETDECK.

o or old :
(optional) name of the OLD file (compile file with sequence idents). If you don't code the **old** parameter, GENMOD(S) uses **compile**, which is the default compile file name generated by GETDECK.

mc :
(optional) Master Character to be used instead of the standard master character (*). If you don't code the **mc** parameter, GENMOD(S) uses the value of profile variable **updtmcc**, and if there's no such profile variable, uses the default master character asterisk (*).

Examples of GENMOD(S) Usage

```
ses.genmods m=today id=jan08 cd=(decka,dekalast)
REVERT.     END GENMODS COMPILE:EDTFILE -> TODAY


ses.genmods tuesday jan09 new=good old=broken
REVERT.     END GENMODS BROKEN:GOOD -> TUESDAY
```

19.4 GENUPSF - GENERATE UPDATE SOURCE FILE

GENUPSF generates a source file for any specified decks on an UPDATE program library. Modsets can be applied also. Parameters to GENUPSF are :

m or mods :
is an (optional) list of names of files containing modsets to be applied against the PL when generating the source file. One or both of the **mods** parameter and the **d** parameter must be specified.

d or decks or
c or coms or
allcom or allcom :
is an (optional) list of the things to be placed on the source file. **d or decks** is a list of DECKS to be placed on the source file. **c or coms** is a list of COMMON DECKS to be placed on the source file. Coding the **allcom** or the **allcom** keys indicates that a full mode update is to be performed, in which case no deck names should be specified. One or both of the **mods** and the **decks** parameter must be specified.

sf :
(optional) name of the file to receive the Source File. If you don't code this parameter, the source file appears on a file called **source**.

pl or oldpl :
(optional) name of the UPDATE program library from which to generate the source file. If you don't code the **pl** parameter, GENUPSF uses the value of profile variable **oldpl** as the name of the program library, and if there's no such profile variable defined, uses the name **oldpl**.

un :
(optional) name of user in whose catalog the program library resides. If you don't code the **un** parameter, GENUPSF uses the value of profile variable **plowner** (PL OWNER) as you name, and if there's no such profile variable defined, GENUPSF uses the name of the current user.

status or sts :
these (optional) keys are used for those cases where GENUPSF is being used as a building block of a larger procedure. The NOS job control register EFG is set to zero (0) in the event of a successful completion, and to non zero if the UPDATE process bombs for any reason. If status is not specified and an error occurs, the procedure EXITs.

msg or nomsg :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

mc :
(optional) Master Character to be used instead of the standard master character (*). If you don't code the mc parameter, GETDECK(S) uses the value of profile variable updtmcc, and if there's no such profile variable, uses the default master character asterisk (*).

Examples of GENUPSF Usage

```
ses.genupsf c='git$mcs', pl=mcspl, un=jej0102
*    GENERATING SOURCE FILE  SOURCE
REVERT.    END GENUPSF  SOURCE <- MCSPL
```

```
ses.genupsf (modsone,modstwo), d=deckone..decktwo, s=newfile
*    GENERATING SOURCE FILE  NEWFILE
REVERT.    END GENUPSF  NEWFILE <- OLDPL
```

The first example shows GENUPSF used to place a common deck onto the source file (which defaults to source since it wasn't specified). Note the use of string delimiters (') because the common deck name **git$mcs** has a $ sign in it. Also in this example both the program library name and user name where it is to be found are specified.

The second example shows a pair of modsets being applied to the program library called **oldpl** in order to generate a source file for a range of decks **deckone** thru **decktwo**, with the source file appearing on a file called **newfile**.

19.5 UPDATE - UPDATE PROGRAM LIBRARY WITH CORRECTION SET(S)

UPDATE applies a list of modsets against an old program library to produce a new program library in UPDATE format. The old and new program library names can be the same, in which case the old program library is rewritten. **Note** : that program library creation is the **only** function performed by this UPDATE procedure - simultaneous compile file generation is **not** done. Parameters to UPDATE are :

mods or m :
list of one or more file(s) containing the MODification Set(s) to be applied against the program library specified by the **pl** parameter.

pl or oldpl :
(optional) name of OLD Program Library to be updated from the correction set file. If you don't code the **pl** parameter, UPDATE uses the value of profile variable **oldpl** as the name of the program library, and if there's no such variable defined, UPDATE uses the default name **oldpl**.

npl or newpl :
(optional) name of New Program Library to be created when the update has been completed. If you don't code the **npl** parameter, UPDATE uses the value of profile variable **newpl** as the name of the new program library, and if there's no such variable defined, UPDATE writes the new program library over the old program library specified by the **pl** parameter.

un :
(optional) User Name in whose catalog the program library specified by **pl/npl** is to be found, if **pl/npl** is not in the catalog of the current user. If you don't code the **un** parameter, UPDATE uses the value of profile variable **plowner** as the user name from whose catalog the program library is to be obtained, and if there's no such variable defined, UPDATE uses the current user's catalog.

lock or nolock :
these (optional) parameters determine whether the program library update process is interlocked against simultaneous updates; coding a filename for the **lock** parameter determines the name of the interlock file. Interlocking is

the default action when the program library being updated is in another user's catalog. If you don't code either of the **lock** or **nolock** keys, interlocking is controlled by the **lokmode** profile variable. Refer to the introductory sections of this chapter for information on the interactions of the **lokmode** profile variable and the **lock** and **nolock** parameters. If you don't code a filename for the **lock** parameter, the contents of profile variable **intrlok** is used as the interlock filename; if there's no such profile variable, the name **intrlok** is used as the lock filename. The interlock file must be in the same catalog as the program library being updated. If the interlock file cannot be found, the procedure aborts.

status or **sts** :
these (optional) keys are used for those cases where UPDATE is being used as a building block of a larger procedure. The NOS job control register EFG is set to zero (0) in the event of a successful completion, and to non zero if the UPDATE process bombs for any reason. If **status** is not specified and an error occurs, the procedure EXITs.

msg or **nomsg** :
these (optional) keys control the generation of informative messages by this procedure and are fully described in the section entitled "INFORMATIVE MESSAGES FROM SES PROCEDURES".

mc :
(optional) Master Character to be used instead of the standard master character (*). If you don't code the **mc** parameter, UPDATE uses the value of profile variable **updtmcc**, and if there's no such profile variable, uses the default master character asterisk (*).

Example of UPDATE Usage

```
ses.update (monday,tuesday,friday), pl=cobolpl
*    APPLYING MONDAY..FRIDAY TO COBOLPL
*    NEW LIBRARY ON SESTMPL
*    NEW LIBRARY NOW ON COBOLPL
*    SESTMPL PURGED
REVERT.    END UPDATE  COBOLPL
```

This example shows a list of modifications in files monday, thursday and friday applied against the cobolpl program library.

20.0 STRUCTURED PROCESS TOOLS

20.0 STRUCTURED PROCESS TOOLS

SCAD - SOFTWARE COMPUTER AIDED DESIGN

The structured process tools are a set of Design Dictionary utilities, Computer Graphics Design tools and CYBIL to Structure Chart tools. They support the Structured Analysis/Structured Design (SASD) methodology used for development of software.

SCADG creates or updates DFD, SCT and DSD diagrams used in the SASD methodology.

SCADGEDIT a utility that enables a user to edit, copy, reorder, merge and list title information for diagrams in notebooks created by SCADG. SCADGEDIT does not required a graphics terminal.

DDCREATE creates a design dictionary.

DDDISPLAY displays the contents of a design dictionary.

DDMERGE merges two design dictionaries into a new, third design dictionary.

DDXREF prepares a cross reference listing of a design dictionary.

DDXCHK checks a design dictionary for consistency.

DDHREF produces hierarchical expansion reports for selected entries in a design dictionary.

DDEDIT provides for interactive editing of entries in a design dictionary.

20.0 STRUCTURED PROCESS TOOLS

CYTOSCT creates graphics notebook files of structure charts from CYBIL source code.

CYTODD is the first phase of CYTOSCT. CYTODD creates a design dictionary for input into DDTOSCT (see note below on advanced design dictionary).

DDTOSCT creates graphics notebook files of structure charts from design dictionary files produced by CYTODD (see note below on advanced design dictionary).

FTNTOSCT creates graphics notebook files of structure charts from FORTRAN source code.

FTNTODD is the first phase of FTNTOSCT. FTNTODD creates a design dictionary for input into DDTOSCT (see note below on advanced design dictionary).

20.0 STRUCTURED PROCESS TOOLS

The Computer Graphics Diagram tool supports the Data Flow Diagram (DFD), Structure Charts (SCT) and Data Structure Diagram (DSD). These diagrams can be created, printed, saved, recalled and updated using a CDC 721-30, IST-III or TEKTRONIX 4014 Graphics Terminal. The CDC 721-30 requires the 721-301 Graphics/Firmware option. See the "Applicable Documents" section of this document for the manual that supports the SCAD Graphics.

The Design Dictionary Utilities provide a set of creation, updating, checking and printing functions. The Design Dictionary consists of CREATE, DISPLAY, MERGE, XCHECK, XREF, HREF and EDIT functions. See the "Applicable Documents" section of this document for the manual that supports the Design Dictionary.

The CYBIL and FORTRAN to Structure Chart tools create graphics notebook files from CYBIL or FORTRAN source code. The process consists of two procedures, CYTODD or FTNTODD and DDTOSCT. The procedures can be run separately or as a single procedure, CYTOSCT or FTNTOSCT (see note below on advanced design dictionary). See the NOS GRAPH Graphics User's Handbook in the "Applicable Documents" section of this document for the manual that supports the CYBIL and FORTRAN to Structure Chart tools.

NOTE: The design dictionaries produced by the CYBIL and FORTRAN to Structure Chart tools are an advanced version of design dictionary. These advanced design dictionaries are <u>not</u> compatible with the current design dictionary tools, DDCREATE, DDDISPLAY, etc.

20.0 STRUCTURED PROCESS TOOLS
20.1 SCADG - CREATE OR UPDATE SASD DIAGRAMS

20.1 SCADG - CREATE OR UPDATE SASD DIAGRAMS

The SCAD Graphics tool, SCADG, supports the creation and maintenance of Data Flow Diagrams (DFD), Data Structure Diagrams (DSD) and Structure Charts (SCT). These diagrams are the primary diagrams used in the SASD technique. It is assumed you are familiar with SCAD. Parameters to SCADG are:

ntbk :
(optional) name of the initial file to be used as the source notebook. If this parameter is not coded, SCADG uses the value of the profile variable ntbk. If the ntbk parameter is not defined in the user's profile, SCADG uses the filename NOTEBK. The **ntbk** notebook is copied to a working file, SESTMNB, if the **ntbk** and **newntbk** notebooks are the same file; otherwise, the working notebook is the **newntbk** notebook. If the **ntbk** file does not exist, SCADG creates a new notebook.

newntbk :
(optional) name of the file to receive the updated **ntbk** notebook. If the **newntbk** parameter is omitted or is the same as **ntbk**, the new or updated notebook is copied back over the **ntbk** file.

term :
the **term** parameter specifies the user's terminal type. Valid terminal type entries are : 721, IST3 or TEK4014. If this parameter is not coded, SCADG uses the value of the profile variable **term**. If not defined in the user's profile, SCADG requests the terminal type.

baud :
represents the **baud** rate of your terminal. Acceptable values for each terminal type are as follows: 721 - 300, 1200, 2400, 4800 or 9600. IST3 - 300, 1200 or 2400. TEK4014 - 300, 1200, 2400, 4800 or 9600. If this parameter is not coded, SCADG uses the value of the profile variable **baud**. If not defined in the user's profile, SCADG requests the baud rate.

hardcopy :
indicates whether the terminal has a **hardcopy** device connected. This parameter may be given values of 'YES',

'Y', 'NO' or 'N'. If this parameter is not coded, the value of the profile variable **hardcopy** is used. If not defined in the user's profile, SCADG requests whether a **hardcopy** device is present.

egm :

refers to the Extended Graphics Module of the TEK4014. Acceptable values for the **egm** parameter are 'YES', 'Y', 'NO' or 'N'. The **egm** parameter does not apply to IST3 or 721 terminals. If this parameter is not coded, SCADG uses the value of the profile variable **egm**. If not defined in the user's profile, SCADG requests whether EGM is present.

sync :

refers to Synchronous or Asynchronous mode for a TEK4014 terminal. The **sync** parameter does not apply to IST3 or 721 terminals. Acceptable values for the **sync** parameter are 'A' or 'S'. If this parameter is not coded, SCADG uses the value of the profile variable **sync**. If not defined in the user's profile, SCADG requests whether the terminal is running in A or S mode.

Example of SCADG Usage

```
SES.SCADG NTBK=MYBOOK NEWNTBK=MYNEWBK
ENTER TERMINAL TYPE (IST3, 721 OR TEK4014)
? 721
WHAT BAUD RATE (300, 1200, 2400, 4800 OR 9600)
? 9600
WITH HARDCOPY, (YES OR NO)
? Y
*                                                              *
*                                                              *
*             PREPARING SCADG - PLEASE WAIT                    *
*                                                              *

BEGINNING SCADG 2.1 UPDATING MYBOOK  -> MYNEWBK


OLD NOTEBOOK = MYBOOK                NEW NOTEBOOK = MYNEWBK

TYPE IN COMMAND
?

    ( SCADG interactive session )

REVERT. END SCADG, NOTEBOOK PROCESSING COMPLETE
```

20.2 SCADGEDIT - DIAGRAM HEADER EDIT & LIST, NOTEBOOK UTIL.

The SCADGEDIT SES procedure provides a utility for editing and listing the diagram title blocks and notebook maintenance. Parameters to SCADGEDIT are:

ntbk :
(optional) name of the file containing the notebook. If this parameter is not coded, SCADGEDIT uses the filename NOTEBK. The **ntbk** notebook is copied to a working file SESTMNB if the **ntbk** and **newntbk** notebooks are the same file, otherwise the working notebook is the **newntbk** notebook. No changes are made to the **ntbk** notebook unless it is also the new notebook and then only after SCADGEDIT terminates.

newntbk :
(optional) name of the file to be used as the new notebook. If this parameter is not coded, disposition depends on the profile variable **oldeqnew**. If there is no such variable defined in the profile or **oldeqnew** has a value of FALSE, SCADGEDIT writes the notebok to the file NEWNTBK. If **oldeqnew** has a value of TRUE, SCADGEDIT writes the notebook back to the **ntbk** notebook.

l :
(optional) name of the file to receive the listing output from SCADGEDIT. If this parameter is not coded, then SCADGEDIT will use the filename LISTING for listing output.

dfile :
(optional) name of the file containing directives, in NOS 64 character set format, for the SCADGEDIT processor. If this parameter is not coded, then SCADGEDIT accepts directives from the terminal or, in the case of a batch job, from the filename INPUT. The **dfile** file is rewound before it is used.

print :
(optional) parameter to control the PRINTing of the output, **l** file. When the job runs in batch mode and there is no **l** file, one copy is printed automatically. If running the procedure in local mode, no copies are printed unless the user specifically codes some print parameter. For example,

you code **print=c=3** to obtain three copies of the **l** file. The format of the **print** parameter is that of the parameters for the PRINT procedure.

20.3 DDCREATE - CREATE A DESIGN DICTIONARY

The DDCREATE function takes a user text input file and creates a Design Dictionary (DD) file. Parameters to DDCREATE are :

i :
(optional) name of the file containing the source for creating the DD. If you don't code the i parameter, then DDCREATE uses the value of the profile variable ddi, and if there's no such variable defined in your profile, then DDCREATE uses the filename DDTEXT. The i file is rewound before it is read.

new :
(optional) name of the file onto which the new DD is to be written. If you don't code the new parameter, then DDCREATE uses the value of the profile variable ddnew, and if there's no such variable defined in your profile, then DDCREATE uses the file name NEWDD.

Example of DDCREATE Usage

```
SES.DDCREATE I=MYTEXT, NEW=MYNEWDD
*     BEGINNING DD CREATE
      ( Additional informative messages from DDCREATE )
*     MYTEXT ----> MYNEWDD
REVERT.   END DD CREATE
```

This example shows DDCREATE used to create a new design dictionary, MYNEWDD, from the text in the source file MYTEXT.

20.4 DDDISPLAY - DISPLAY A DESIGN DICTIONARY

The DDDISPLAY function extracts entries from a design dictionary and writes them to an output text file. Extraction criteria are specified by directives on a directives file or by specifying a **dlist** parameter. Parameters to DDDISPLAY are :

old :
(optional) name of an old design dictionary file from which entries are to be extracted. If you don't code this parameter, then DDDISPLAY uses the value of the profile variable **ddold**, and if there's no such variable defined in your profile, then DDDISPLAY uses the filename OLDDD.

o :
(optional) name of the file to receive the output from DDDISPLAY. If you don't code this parameter, then DDDISPLAY uses the value of the profile variable **ddo**, and if there's no such variable defined in your profile, then DDDISPLAY uses the filename DDTEXT. The **o** file is not rewound.

dfile :
(optional) name of the file containing directives, in NOS ASCII line-image format, to the DDDISPLAY processor. If you don't code this parameter, DDDISPLAY uses the value of the profile variable **dddfile**. If there's no such profile variable defined in your profile and a **dlist** parameter is not specified, then DDDISPLAY accepts directives from your terminal or, in the case of a batch job, from the filename **input**. The **dfile** is not rewound. If you code the **dfile** parameter, don't code a **dlist** parameter.

dlist :
(optional) parameter specifies a DDDISPLAY directive or directive list. Acceptable values for **dlist** include: **all**, **allh**, **data**, **file**, **flow**, **component**, **element**, **process**, **processh**, **module**, **dfd**, **dfdh**, **set**, **notype**, and **undef**. Directives of the form 'psn n' may also be included in the **dlist**. Coding the **dlist** parameter is equivalent to specifying directives on a **dfile** (directives) file. If you code the **dlist** parameter, don't code the **dfile** parameter.

b :

(optional) name of a Base Library to be updated. If you do not code the b or the m parameter, display output is not placed in a Base Library. If you code the m parameter, but not the b parameter, the b parameter defaults to BASE.

m :

(optional) name of the Common Module to be created from the display output. REPMOD replaces the m module on the b Base Library. If you code the b parameter and not the m parameter, the m parameter defaults fo DICT.

print :

(optional) parameter to control the PRINTing of the output, o file. When the job runs in batch mode and there is no o file, one copy is printed automatically. If you run the procedure in local mode, no copies are printed unless you specifically code some print parameter. For example, you code **print=c=3** to obtain three copies of the parameter o file. The format of the print parameter is that of the parameters for the PRINT procedure.

Example of DDDISPLAY Usage

```
SES.DDDISPLAY, OLD=MYOLDDD, O=LISTIT, DFILE=DOWHAT
*    BEGINNING DD DISPLAY
 ACCEPTING DISPLAY DIRECTIVES

( Additional informative messages from DDDISPLAY )

*    MYOLDDD ----> LISTIT
REVERT.   END DD DISPLAY
```

This example shows DDDISPLAY used to display selected entries from the design dictionary, MYOLDDD. The directives for selecting entries were read from the file DOWHAT. If the dfile parameter had been omitted, directives would have been read from the file INPUT. The listing of formatted design dictionary entries was written to the file LISTIT in the order selected by the directives read from the file DOWHAT.

The b and m parameters are useful for including sections of the design dictionary as sections of an Analysis and Design Specifications document.

20.5 DDMERGE - MERGE TWO DESIGN DICTIONARIES

The DDMERGE function combines two design dictionaries to produce a new, third DD. Parameters to DDMERGE are :

old :
(optional) name of the old design dictionary file with which you wish to merge another design dictionary. If you don't code the old parameter, then DDMERGE uses the value of the profile variable **ddold**, and if there's no such variable defined in your profile, then DDMERGE uses the filename OLDDD.

new :
(optional) name of the file onto which the new DD is to be written. If you don't code the new parameter, then DDMERGE uses the value of the profile variable **ddnew**, and if there's no such variable defined in your profile, then DDMERGE uses the file name NEWDD.

merge :
(optional) name of the design dictionary file you wish to merge with the old design dictionary. If you don't code the merge parameter, DDMERGE uses the value of the profile variable **ddmerge**, and if there's no such variable defined in your profile, then DDMERGE uses the filename MERDD.

replace :
(optional) key causes entries contained in the old design dictionary whose names are duplicated by entries contained in the merge design dictionary to be REPLACE'd by the merge design dictionary entries. A file named DDLOG will contain a list of duplicate entries and the action taken in each case.

Example of DDMERGE Usage

```
SES.DDMERGE, OLD=MYOLDDD, MERGE=MERGEDD, NEW=MYNEWDD, REPLACE
*    BEGINNING DD MERGE
*    MYOLDD + MERGEDD ----> MYNEWDD
REVERT.   END DD MERGE
```

This example shows the use of DDMERGE to combine entries from the

design dictionaries **MYOLDDD** and **MERGEDD** to form a new design dictionary **MYNEWDD**. If an entry name is defined in both **MYOLDDD** and **MERGEDD** then the definition used in creating **MYNEWDD** is taken from **MERGEDD**. If the **REPLACE** parameter had been omitted, the definition from **MYOLDDD** would have been used. In either case an explanatory record is written to filename DDLOG.

20.6 DDXREF - CROSS REFERENCE A DESIGN DICTIONARY

The DDXREF function produces a formatted cross reference listing of part or all of a design dictionary. Parameters to DDXREF are :

old :
(optional) name of the old design dictionary file from which you want a cross reference listing. If you don't code the old parameter, then DDXREF uses the value of the profile variable ddold, and if there's no such variable defined in your profile, then DDXREF uses the filename OLDDD.

l :
(optional) name of the file to receive the output of DDXREF in a form suitable for printing on the ASCII printer. If you don't code the l parameter, then DDXREF uses the value of the profile variable ddl. If there's no such variable defined in your profile, and you're running DDXREF LOCAL at your terminal, the output appears on a file called LISTING. The l file is not rewound.

ref :
(optional) parameter selects the type of entries you want to be included in the cross reference listing. Acceptable values for ref are: flow, component, element, file, data, process, module, dfd, sct, notype, undef, and all.

width :
(optional) parameter selects the page width of the output listing written to the l file. Acceptable values for width are printer or term (for terminal width). If you don't code the width parameter, then DDXREF uses the value of the profile variable ddwidth, and if there's no such variable defined in your profile, your listing will be formatted for a line printer.

print :
(optional) parameter to control the PRINTing of the output, l file. When the job runs in batch mode and there is no l file, one copy is printed automatically. If you run the procedure in local mode, no copies are printed unless you specifically code some print parameter. For example, you code print=c=3 to obtain three copies of the parameter l

file. The format of the print parameter is that of the parameters for the PRINT procedure.

Example of DDXREF Usage

```
SES.DDXREF, OLD=MYOLDDD, L=LISTIT, REF=ALL, WIDTH=TERM
*    BEGINNING DD XREF
 (Additional informative messages from DDXREF)
*    MYOLDDD ----> LISTIT
REVERT.   END DD XREF
```

This example shows the use of DDXREF to obtain a terminal width cross reference listing of all sections of the design dictionary MYOLDDD. The cross reference listing will be on the local file LISTIT. Printed copies could have been obtained by adding a PRINT parameter. For example, PRINT=(C=3,H='XLIST') would have caused 3 copies with a header XLIST to be printed.

20.7 DDXCHK - CHECK A DESIGN DICTIONARY FOR CONSISTENCY

The DDXCHK function runs a cross check against a design dictionary. A formatted listing is produced which contains the names of entries which meet criteria selected by the check parameter. Parameters to DDXCHK are :

old :
(optional) name of the design dictionary file you want cross checked. If you don't code the **old** parameter, then DDXCHK uses the value of the profile variable **ddold**, and if there's no such variable defined in your profile, then DDXCHK uses the filename OLDDD.

l :
(optional) name of the file to receive the output of DDXCHK in a form suitable for printing on the ASCII printer. If you don't code the l parameter, then DDXCHK uses the value of the profile variable **ddl**. If there's no such variable defined in your profile, and you're running DDXCHK local at your terminal, the output appears on a file called LISTING. The l file is not rewound.

check :
(optional) parameter specifies the kind of consistency check you want run against the **old** design dictionary. Acceptable values for **check** are: **undef**, **unref**, **noref**, **recur**, and **full**.

print :
(optional) parameter to control the PRINTing of the output, l file. When the job runs in batch mode and there is no l file, one copy is printed automatically. If you run the DDXCHK procedure in local mode, no copies are printed unless you specifically code some **print** parameter. For example, you code **print=c=2** to obtain two copies of the parameter l file. The format of the **print** parameter is that of the parameters for the PRINT procedure.

Example of DDXCHK Usage

```
SES.DDXCHK, OLD=MYOLDDD, L=LISTIT, CHECK=FULL
*    BEGINNING DD XCHECK
     ( Additional informative messages from DDXCHK )
*    MYOLDDD ----> LISTIT
REVERT.   END DD XCHECK
```

This example shows the use of DDXCHK to perform a full cross check of the design dictionary MYOLDDD. The cross check listing will be on the local file LISTIT. Printed copies could have been obtained by adding a PRINT parameter.

20.8 DDHREF - HIERARCHICAL REF EXPANSIONS FOR A DESIGN DICT

The DDHREF procedure produces Hierarchical REFerence expansion reports for selected entries in a design dictionary.

old :
(optional) name of the design dictionary file from which you want hierarchical reference expansions. If you don't code the **old** parameter, then DDHREF uses the value of the profile variable **ddold**, and if there's no such variable defined in your profile, then DDHREF assumes the design dictionary is on a file called OLDDD.

l :
(optional) name of the file to receive the output of DDHREF in a form suitable for printing on the ASCII printer. If you don't code the **l** parameter, then DDHREF uses the value of the profile variable **ddl**. If there is no such variable defined in your profile, then DDHREF output appears on a file called LISTING. The **l** file is not rewound.

dfile :
(optional) name of file containing directives to DDHREF in NOS ASCII line-image format.

enter/noenter :
(optional) **key** controlling selection of interactive directive input to DDHREF. **Enter** selects interactive directive input via your terminal keyboard. The default is **noenter** (i.e., no terminal interaction). If a **dfile** was specified, interaction would occur after DDHREF reads the **dfile**.

levels :
(optional) maximum number of hierarchical levels to which DDHREF will expand design dictionary entries. If specified, the value must be an integer from 1 to 10; the default value is 5.

droots/nodroots :
(optional) **key** controlling selection of the DDHREF derived roots entry name expansions. Specifying **droots** causes these entry names to be expanded at the end of your DDHREF

session. Specifying **nodroots** causes DDHREF to only expand the user supplied entries in the supplied root list. The default value is **nodroots** if the enter key is specified, otherwise the default value is **droots**.

print :
(optional) parameter to control the printing of the **i** (output) file. When this job runs in batch mode and there is no **i** file, one copy is printed automatically. If you run the DDHREF procedure in local mode, no copies are printed unless you specifically code some **print** parameter. For example, you code **print=c=2** to obtain two copies of the parameter **i** file. The format of the **print** parameter is that of the parameters for the PRINT procedure.

20.9 DDEDIT - EDIT A DESIGN DICTIONARY

DDEDIT allows you to interactively examine and alter the contents of a design dictionary on an entry-by-entry basis via directives that you enter from your terminal. Parameters to DDEDIT are :

old :
(optional) name of an existing design dictionary that you want to edit. If you don't code this parameter and also don't specify the **create** key (described below), then DDEDIT uses the value of the profile variable **ddold**, and if there's no such variable defined in your profile, then the filename OLDDD is used.

new :
(optional) name of the file onto which the new edited DD is to be written. If you don't code the new parameter, then DDEDIT uses the value of the profile variable **ddnew**, and if there's no such variable defined in your profile, then the filename NEWDD is used.

scr :
(optional) name of the file that contains or will contain entries you wish to add or replace in your design dictionary. If you don't code the **scr** parameter, then DDEDIT uses the value of the profile variable **ddscr**, and if there's no such variable defined in your profile, then the filename SCRATCH is used.

e :
(optional) name of the text editor you want to use to edit entries on the **scr** file. Acceptable values for **e** are **XEDIT**, **EDIT**, **EDT** or **FSE**. If you don't code this parameter, DDEDIT uses the value of the profile variable **dde**, and if there's no such variable defined in your profile, you'll get XEDIT as your editor.

create :
(optional) key when specified indicates to DDEDIT that you want to begin a new DD. If you code the create key, you can't code the **old** parameter.

Example of DDEDIT Usage

```
SES.DDEDIT, OLD=MYOLDDD, NEW=MYNEWDD, SCR=MYSCR, E=FSE
*    BEGINNING DD EDIT
ENTER DDEDIT DIRECTIVE.

     (Additional DDEDIT/User Interaction)

REVERT.   END DD EDIT
```

This example shows DDEDIT being used to edit an existing design dictionary, MYOLDDD. During the DDEDIT session, text editor FSE is used to edit DD entries contained on the file MYSCR. When the DDEDIT session is terminated, the edited DD is to be written to file MYNEWDD.

20.0 STRUCTURED PROCESS TOOLS
20.10 CYTOSCT - CYBIL TO STRUCTURE CHARTS

20.10 CYTOSCT - CYBIL TO STRUCTURE CHARTS

The CYBIL to Structure Chart tool, CYTOSCT, creates an SCAD graphics notebook from CYBIL source code. CYTOSCT performs the combined functions of the CYTODD and DDTOSCT tools. Input to CYTOSCT is syntax error free CYBIL source code. The CYTOSCT tool creates:

1) a graphics notebook.
2) a table of contents listing and an index listing of the graphics notebook.
3) a cross reference listing of the design dictionary (if selected).

   (See note on advanced design dictionary at the beginning of this section.)

Parameters to CYTOSCT are :

input or i :
(optional) name of the file containing error free CYBIL source code. If you don't code the i parameter, CYTOSCT uses the value of the profile variable cdi. If there is no such variable defined in your profile, CYTOSCT assumes the CYBIL source is contained on a file named COMPILE.

ntbk :
(optional) name of the file onto which the new notebook is to be written. If you don't code the ntbk parameter, CYTOSCT uses the value of the profile variable ntbk. If there is no such variable defined in your profile, CYTOSCT uses the filename NOTEBK.

nlist :
(optional) name of the file onto which the table of contents and index listings for the notebook are to be written. If you don't code the nlist parameter, CYTOSCT uses the value of the profile variable nlist. If there is no such variable defined in your profile, CYTOSCT uses the filename NLIST.

order :
(optional) parameter to control the order of the structure charts that are generated. If the order parameter is coded as A, the structure charts are produced in alphabetical

order. If the order parameter is coded as H, the structure charts are produced in hierarchical order. The default value for the order parameter is A.

newdd :
(optional) name of the file onto which the new design dictionary is to be written. If you don't code the newdd parameter, CYTOSCT uses the value of the profile variable ddnew. If there is no such variable defined in your profile, CYTOSCT uses the filename DDNEW.

xref :
(optional) name of the file onto which a cross reference listing of the new design dictionary is to be written. If you don't code the xref parameter, CYTOSCT uses the value of the profile variable ddxref. If there is no such variable defined in your profile, no cross reference listing is generated.

print :
(optional) parameter to control printing of the nlist and xref files. If you code the print keyword, one copy of each of the files is printed. You code print=c=2 to obtain two copies of each file. The format of the print parameter is that of parameters for the SES.PRINT procedure.

un :
(optional) user name associated with files used by CYTOSCT. If you don't code the un parameter, CYTOSCT uses the value of the profile variable cyun. If there is no such variable defined in your profile, CYTOSCT assumes the files are in the current user's catalog. Note: If un is not the current user's catalog, the ntbk and newdd files must exist with write permission. In local mode CYTOSCT allows the use of local files for nlist and xref files, but in batch mode these files must exist with write permission.

Example of CYTOSCT Usage

```
SES.CYTOSCT I=COMPILE NEWDD=DDFILE NTBK=NBFILE ..
NLIST=NTBKLST
*    BEGINNING CYTOSCT V 1 ON COMPILE -> DDFILE, NBFILE
*    BEGINNING CYTODD V 1 ON COMPILE -> DDFILE

    . . .

*    END CYTODD COMPILE ----> DDFILE
*    BEGINNING DDTOSCT V 1 ON DDFILE -> NBFILE

    . . .

*    CREATING UNFORMATTED NOTEBOOK
*    CONSOLIDATING NOTEBOOK
*    FORMATTING NOTEBOOK
*    END CYTOSCT COMPILE -> NBFILE, NTBKLST
REVERT.
```

This example shows CYTOSCT creating a new DD file, DDFILE, and a new graphics notebook, NBFILE, from the CYBIL source code found on file COMPILE. The table of contents and index listings of the notebook are on the permanent file NTBKLST.

20.11 CYTODD - CYBIL TO DESIGN DICTIONARY

The CYBIL to Design Dictionary tool, CYTODD, is the first phase of the CYBIL to Structure Chart tool, CYTOSCT. The CYTODD tool creates a design dictionary that is input to the DDTOSCT tool (see note on advanced design dictionary at the beginning of this section). Parameters to CYTODD are :

input or i :
(optional) name of the compile file containing error free CYBIL source code. If you don't code the i parameter, CYTODD uses the value of the profile variable cdi. If there is no such variable defined in your profile, CYTODD uses the filename COMPILE.

newdd :
(optional) name of the file onto which the new DD is to be written. If you don't code the newdd parameter, CYTODD uses the value of the profile variable ddnew. If there is no such variable defined in your profile, CYTODD uses the filename DDNEW.

xref :
(optional) name of the file onto which a cross reference listing of the new design dictionary is to be written. If you don't code the xref parameter, CYTODD uses the value of the profile variable ddxref. If there is no such variable defined in your profile, no cross reference listing is generated.

print :
(optional) parameter to control printing of the xref file. If you code the print keyword, one copy of the file is printed. You code print=c=2 to obtain two copies of the xref file. The format of the print parameter is that of the parameters for the SES.PRINT procedure.

un :
(optional) user name associated with files used by CYTODD. If you don't code the un parameter, CYTODD uses the value of the profile variable cyun. If there is no such variable defined in your profile, CYTODD assumes files are in the current user's catalog. Note: If un is not the current user's catalog, the newdd file must exist with write

permission. In local mode CYTODD allows the use of a local file for xref, but in batch mode xref must exist with write permission.

Example of CYTODD Usage

```
SES.CYTODD I=COMPILE NEWDD=DDFILE
*    BEGINNING CYTODD V 1 ON COMPILE -> DDFILE
*    CYBIL COMPILING COMPILE
*    BEGINNING CREATION OF DDFILE
*    PROCESSING TEXT FOR MODULE MOD1
*    PROCESSING TEXT FOR MODULE MOD2
*    END CYTODD COMPILE ----> DDFILE
REVERT.
```

This example shows CYTODD compiling source code found on file COMPILE and creating a design dictionary on file DDFILE.

20.12 DDTOSCT - DESIGN DICTIONARY TO STRUCTURE CHARTS

The Design Dictionary to Structure Chart tool, DDTOSCT, is the second and last phase of the Source Code to Structure Chart tools, CYTOSCT and FTNTOSCT. The DDTOSCT tool creates an SCAD graphics notebook from a design dictionary created by CYTODD (see note on advanced design dictionary at the beginning of this section). Parameters to DDTOSCT are :

olddd :
(optional) name of the file containing the design dictionary created by CYTODD. If you don't code the **olddd** parameter, DDTOSCT uses the value of the profile variable **ddold**. If there is no such variable defined in your profile, DDTOSCT uses the filename OLDDD.

ntbk :
(optional) name of the file onto which the new notebook is to be written. If you don't code the **ntbk** parameter, DDTOSCT uses the value of the profile variable **ntbk**. If there is no such variable defined in your profile, DDTOSCT uses the filename NOTEBK.

nlist :
(optional) name of the file onto which the table of contents and the index information for the notebook is to be written. If you don't code the **nlist** parameter, DDTOSCT uses the value of the profile variable **nlist**. If there is no such variable defined in your profile, DDTOSCT uses the filename NLIST.

order :
(optional) parameter to control the order of the structure charts that are generated. If the **order** parameter is coded as A, the structure charts are produced in alphabetical order. If the **order** parameter is coded as H, the structure charts are produced in hierarchical order. The default value for the **order** parameter is A.

print :
(optional) parameter to control the printing of the **nlist** file. If you code the **print** keyword, one copy of the file is printed. You code **print=c=2** to obtain two copies of the **nlist** file. The format of the **print** parameter is that of

parameters for the SES.PRINT procedure.

un :
(optional) user name associated with files used by DDTOSCT. If you don't code the un parameter, DDTOSCT uses the value of the profile variable ddun. If there is no such variable defined in your profile, DDTOSCT assumes that the olddd parameter file is in the current user's catalog. Note: If un is not the current user's catalog, the ntbk file must exist with write permission. In local mode DDTOSCT allows use of a local file for nlist, but in batch mode nlist must exist with write permission.

Example of DDTOSCT Usage

```
SES.DDTOSCT OLDDD=DDFILE NTBK=NBFILE
*    BEGINNING DDTOSCT V 1 ON DDFILE -> NBFILE
*    CREATING UNFORMATTED NOTEBOOK
*    CONSOLIDATING NOTEBOOK
*    FORMATTING NOTEBOOK
*    END DDTOSCT DDFILE -> NBFILE,NLIST
REVERT.
```

This example shows DDTOSCT creating a new graphics notebook, NBFILE, from the file DDFILE. The notebook's table of contents and index listings are written to file NLIST.

20.13 FTNTOSCT - FORTRAN TO STRUCTURE CHARTS

The FORTRAN to Structure Chart tool, FTNTOSCT, creates an SCAD graphics notebook from FORTRAN 5 source code. FTNTOSCT performs the combined functions of the FTNTODD and DDTOSCT tools. Input to FTNOSCT is syntax error free FORTRAN 5 source code. Other versions of FORTRAN source code or FORTRAN 5 source code containing errors may be input to FTNTOSCT, but some code may not be recognized and will be bypassed. The CYTOSCT tool creates:

1) a graphics notebook.
2) a table of contents listing and an index listing of the graphics notebook.
3) a list file of the source code with line numbers. Any source code diagnostics will appear at the end of the list file.
4) a cross reference listing of the design dictionary (if selected).

   (See note on advanced design dictionary at the beginning of this section.)

Parameters to FTNTOSCT are :

input or i :
(optional) name of the file containing FORTRAN source code. If you don't code the i parameter, FTNTOSCT uses the value of the profile variable **ftndi**. If there is no such variable defined in your profile, FTNTOSCT assumes the FORTRAN source is contained on a file named COMPILE.

ntbk :
(optional) name of the file onto which the new notebook is to be written. If you don't code the ntbk parameter, FTNTOSCT uses the value of the profile variable **ntbk**. If there is no such variable defined in your profile, FTNTOSCT uses the filename NOTEBK.

nlist :
(optional) name of the file onto which the table of contents and index listings for the notebook are to be written. If you don't code the nlist parameter, FTNTOSCT uses the value of the profile variable nlist. If there is no such variable defined in your profile, FTNTOSCT uses the filename NLIST.

order :
(optional) parameter to control the order of the structure charts that are generated. If the order parameter is coded as A, the structure charts are produced in alphabetical order. If the order parameter is coded as H, the structure charts are produced in hierarchical order. The default value for the order parameter is A.

mode or m :
(optional) paramter to indicate the mode of the source input. If the mode parameter is nonseq, all the source is considered to be written in nonsequenced mode. The mode parameter can also be seq which indicates that the source line is preceded by a sequence number. The default value for the mode parameter is nonseq.

newdd :
(optional) name of the file onto which the new design dictionary is to be written. If you don't code the newdd parameter, FTNTOSCT uses the value of the profile variable ddnew. If there is no such variable defined in your profile, FTNTOSCT uses the filename DDNEW.

list or l :
(optional) name of a file onto which the source and any error messages will be listed. If you don't code the list parameter, FTNTOSCT uses the filename LISTING.

xref :
(optional) name of the file onto which a cross reference listing of the new design dictionary is to be written. If you don't code the xref parameter, FTNTOSCT uses the value of the profile variable ddxref. If there is no such variable defined in your profile, no cross reference listing is generated.

print :
(optional) parameter to control printing of the nlist, list, and xref files. If you code the print keyword, one copy of each of the files is printed. You code print=c=2 to obtain two copies of each file. The format of the print parameter is that of the parameters for the SES.PRINT procedure.

un :

(optional) user name associated with files used by FTNTOSCT. If you don't code the un parameter, FTNTOSCT uses the value of the profile variable ftnun. If there is no such variable defined in your profile, FTNTOSCT assumes the files are in the current user's catalog. Note: If un is not the current user's catalog, the newdd and ntbk files must exist with write permission. In local mode FTNTOSCT allows the use of local files, for list, nlist and xref files but in batch mode these files must exist with write permission.

Example of FTNTOSCT Usage

```
SES.FTNTOSCT I=COMPILE NEWDD=DDFILE NTBK=NBFILE ..
NLIST=NTBKLST
*    BEGINNING FTNTOSCT V 1 ON COMPILE -> DDFILE, NBFILE
*    BEGINNING FTNTODD V 1 ON COMPILE -> DDFILE

    . . .

*    END FTNTODD COMPILE ----> DDFILE, LISTING
*    BEGINNING FTNTOSCT V 1 ON DDFILE -> NBFILE

    . . .

*    CREATING UNFORMATTED NOTEBOOK
*    CONSOLIDATING NOTEBOOK
*    FORMATTING NOTEBOOK
*    END FTNTOSCT COMPILE -> NBFILE, NTBKLST
REVERT.
```

This example shows FTNTOSCT creating a new DD file, DDFILE, and a new graphics notebook, NBFILE, from the FORTRAN source code found on COMPILE file. The table of contents and index listings of the notebook are on the permanent file NTBKLST. A listing of the FORTRAN source and any error messages are on the file LISTING.

20.14 FTNTODD - FORTRAN TO DESIGN DICTIONARY

The FORTRAN to Design Dictionary tool, FTNTODD, is the first phase of the FORTRAN to Structure Chart tool, FTNTOSCT. The FTNTODD tool creates a design dictionary that is input to the DDTOSCT tool (see note on advanced design dictionary at the beginning of this section). Parameters to FTNTODD are :

input or i :
(optional) name of the compile file containing FORTRAN source code. If you don't code the i parameter, FTNTODD uses the value of the profile variable ftndi. If there is no such variable defined in your profile, FTNTODD uses the filename COMPILE.

newdd :
(optional) name of the file onto which the new DD is to be written. If you don't code the newdd parameter, FTNTODD uses the value of the profile variable ddnew. If there is no such variable defined in your profile, FTNTODD uses the filename DDNEW.

mode or m :
(optional) parameter to indicate the mode of the source input. If the mode parameter is nonseq, all the source is considered to be written in nonsequenced mode. The mode parameter can also be seq which indicates that the source line is preceded by a sequence number. The default value for the mode parameter is nonseq.

list or l :
(optional) name of a file onto which the source and any error messages will be listed. If you don't code the list parameter, FTNTODD uses the filename LISTING.

xref :
(optional) name of the file onto which a cross reference listing of the new design dictionary is to be written. If you don't code the xref parameter, FTNTODD uses the value of the profile variable ddxref. If there is no such variable defined in your profile and you don't code the xref parameter, no cross reference listing is generated.

print :

(optional) parameter to control printing of the list and xref parameter files. If you code the print keyword, one copy of the file is printed. You code print=c=2 to obtain two copies of each file. The format of the print parameter is that of the parameters for the SES.PRINT procedure.

un :

(optional) user name associated with files used by FTNTODD. If you don't code the un parameter, FTNTODD uses the value of the profile variable ftnun. If there is no such variable defined in your profile, FTNTODD assumes files are in the current user's catalog. Note: If un is not the current user's catalog the newdd file must exist with write permission. In local mode FTNTODD allows the use of local files for list and xref, but in batch mode these files must exist with write permission.

Example of FTNTODD Usage

```
SES.FTNTODD I=COMPILE NEWDD=DDFILE
*    BEGINNING FTNTODD V 1 ON COMPILE -> DDFILE
*    GENERATING DD TEXT FROM COMPILE
*    CREATING DD DDFILE FROM DD TEXT
*    END FTNTODD COMPILE ----> DDFILE, LISTING
REVERT.
```

This example shows FTNTODD using source code found on COMPILE to create a design dictionary on file DDFILE. A listing of the FORTRAN source and any error messages are on the file LISTING.

A1.0 PROFILES

A PROFILE is an important, albeit optional, component of SES usage. Any user may choose to establish a PROFILE in their catalog or procedure library. PROFILE follows the same rules as any SES procedure, that is, the name of the file must be PROFILE, and the first line of the profile must be the word PROFILE. From there on, the profile may contain just about any SES command. The most important aspect of the profile is the SEARCH directive, explained in the next section.

Typically, the types of things that a user may place in the profile would be :

- a command to set a variable called PASSWOR to the user's password. Procedures which optionally run as batch jobs can then get the user's password without having to be told it on the SES control statement.
- commands to establish defaults for library names (for the source code and library maintenance procedures), tape numbers (and related information for the tape management procedure), and other data for various procedures.
- SEARCH directives to establish a search order for procedures.

It is possible for a user to have more than one PROFILE, and select which one to use by coding the PN or P parameter on the SES control statement. For example, by coding :

```
ses,pn=alternate_profile.procedure_name list_of_parameters
```

a user can use the file "alternate_profile" as the PROFILE for the duration of that procedure call. Also, a user may use someone else's profile by coding the PUN or PU parameter. For example, by coding :

```
ses,pun=profile_owner.procedure_name list_of_parameters
```

a user can access the profile belonging to "profile_owner". Of course, the PN and PUN parameters may be used together.

A1.0 PROFILES
A1.0.1 SEARCH DIRECTIVE - ESTABLISH LIBRARY SEARCH ORDER

A1.0.1 SEARCH DIRECTIVE - ESTABLISH LIBRARY SEARCH ORDER

Using the SEARCH directive, a user can establish, within PROFILE, the names of libraries to search when locating a procedure, and also the user names in whose catalogs those procedure libraries reside. The general form of SEARCH is :

\ SEARCH search_spec, search_spec.....

where "search_spec" is in the form :

user_name

or

(library_name, library_name....., user_name)

The first form indicates that the library name contained in the predefined variable SESLNAM is to be searched for in the catalog of the user specified by "user_name". The second form gives a list of library names, with the last item in the list being the user name in whose catalog those libraries may be found.

Examples of SEARCH Directives

\ SEARCH USER, SSS, SES

\ SEARCH (SESLNAM, CCG), USER, SSS, SES

The first example shows a simple SEARCH directive. The search order for procedures is first, the currently logged in user's catalog, then the SSS catalog, then the SES catalog. The second example searches first in the library given by SESLNAM in the CCG catalog, then in the currently logged in user's catalog, then in the SSS catalog, and finally in the SES catalog.

Note : only one SEARCH directive is processed per SES processor call. Second and subsequent SEARCH directives are ignored.

A1.0 PROFILES
A1.1 LOCATING A PROCEDURE

A1.1 LOCAIING_A_PROCEDURE

The SES processor performs its search for a given procedure according to well defined and consistent rules. Basically SES has three methods of specifying how a procedure is to be located. The SES processor has an internal table which contains the following data :

library_name	user_name
library_name	user_name
etc.	
etc.	
etc.	

Given that the table may be set up by one of three different methods which are explained in more detail in the sections following, the procedure that SES follows to locate a procedure is :

1. If there's a local file of the "procedure_name", whose first line is "procedure_name", that file is used as the procedure.

2. SES searches the catalog of the user whose user name appears as the first entry in the table, for a file of name "procedure_name", whose first line is "procedure_name". If such a file is found, it is used as the procedure.

3. For each entry in the table, a search is made for a library of name "library_name" in the catalog of the corresponding "user_name", and searches that library (which must have a directory) for a TEXT record of name "procedure_name". If such a record is found, it is used as the procedure.

4. If the search is unsuccessful, an error message is issued

-PROCEDURE procedure_name NOT FOUND-

The next three sections provide a more detailed explanation of the methods by which the SES processor has its search table set up. The methods are basically the default, the user name specified on the SES control statement, and the SEARCH directive.

A1.0 PROFILES
A1.1.1 DEFAULT ORDER OF SEARCH

A1.1.1 DEFAULT ORDER OF SEARCH

When the SES processor is called, it sets up the following data in its search table :

&SESLNAM&	user_name
&SESLNAM&	&SESUNAM&

This table is the normal default for SES. "user_name" is the user name of the current user.

"SESLNAM" is a predefined variable which contains the name of the SES Library NAMe. "SESUNAM" is a predefined variable which contains the SES User NAMe.

A1.1.2 SEARCH SPECIFIED ON CONTROL STATEMENT

When the user types the SES control statement, he may specify via the UN or U parameter of the SES program, which user's catalog to look in for the procedure specified by the call. For example :

```
ses,un=user_name.procedure_name list_of_parameters
```

specifies that the procedure "procedure_name" is to be searched for only in the catalog of the user "user_name" (if the procedure is not already local). In this case the SES processor modifies its search table to contain only the following data :

&SESLNAM&	user_name

where "SESLNAM" contains the SES Library NAMe as before, and "user_name" is the user name specified on the SES control statement. It is also possible to tell the SES processor, via the LIBPFN or LPFN parameter, the name of the library to be searched for the procedure. For example :

A1.0 PROFILES
A1.1.2 SEARCH SPECIFIED ON CONTROL STATEMENT

```
ses,lpfn=lib_name.procedure_name list_of_parameters
```

specifies that the procedure "procedure_name" is to be searched for only in the library "lib_name". In this case the SES processor modifies its search table to reflect the following data :

lib_name	user_name

where "user_name" is the user name of the current user, and "lib_name" is the library name specified on the SES control statement. Of course, the UN and LIBPFN parameters may be used together.

A1.1.3 SEARCH ORDER SPECIFIED VIA SEARCH DIRECTIVES

The third method of specifying the order in which to look for the procedure is via SEARCH directives in the user's PROFILE. For example, supposing that the user's PROFILE contains the following SEARCH directive :

```
\ SEARCH (HOLMLIB,JIMLIB,HG74), AM74, JF03, (ANDYLIB,ED73)
```

In this case the SES processor would modify its search table to look like this :

HOLMLIB	HG74
JIMLIB	HG74
&SESLNAM&	AM74
&SESLNAM&	JF03
ANDYLIB	ED73

Note : SEARCH directives are ignored if the UN or LIBPFN parameters were specified on the SES control statement.

B1.0 OPERATING MODES OF THE SES PROCESSOR

B1.0 OPERATING MODES OF THE SES PROCESSOR

The SES processor processes procedures in one of four modes :

RUN This is the normal mode. The procedure is processed, presumably generating control statements, and then these control statements are executed.

TEST In this mode the procedure is processed in the normal manner, but the generated control statements are not executed, instead they are placed on a designated file for possible inspection by the user. This mode is meant as an aid in debugging new procedures.

HELP This mode is similar to test mode, however instead of generating control statements, a procedure set up for HELP mode produces some documentation on its purpose and usage.

STATUS This mode is similar to test mode, however instead of generating control statements, a procedure set up for STATUS mode produces any change information available about the procedure.

The modes are selectable by the user by means of parameters to the SES processor; and the procedure can determine in which of the modes it was called by means of predefined variables set up by the SES program. These variables are :

MODE This variable may be compared with the variables RUN, TEST, HELP or STATUS to determine which of the modes is in effect; for example :

```
\  IF    MODE = HELP THEN
         " code for HELP mode "
\  ORIF MODE = STATUS THEN
         " code for STATUS mode "
\  ORIF MODE = TEST THEN
         " code for TEST mode "
\  ELSE
         " code for RUN mode "
\  IFEND
```

PRIMOUT This variable contains the name of the PRIMary OUTput file. In RUN mode this is the new control statement file; in TEST mode this is the file designated by the test mode parameter on the SES call (default is SESTEST); and in

B1.0 OPERATING MODES OF THE SES PROCESSOR

HELP or STATUS mode this is the file designated by the help or status mode parameter on the SES call (default is OUTPUT). PRIMOUT is particularly useful in HELP mode for directing the descriptive information about the procedure to the file selected by the user on the SES call. This may be accomplished as follows :

```
\  IF MODE = HELP THEN
\    ROUT FA=PRIMOUT
       " descriptive information about called procedure "
\    ROUTEND PRIMOUT
\    STOP
\  IFEND
```

Note : in HELP or STATUS mode, a PARMEND directive is interpreted as a STOP directive, to prevent a procedure not set up for HELP or STATUS mode from doing strange or undesirable things.

B1.1 SELECTING MODE OF OPERATION

As stated above, the mode of operation for a procedure is selected by a parameter to the SES processor.

TEST mode may be selected by one of the keywords : TEST, TM, or T. For example :

```
ses,test.procedure_name list_of_parameters
```

processes procedure "procedure_name" in TEST mode, and place the generated control statements on file SESTEST; whereas :

```
ses,t=my_file.procedure_name list_of_parameters
```

processes procedure "procedure_name" in TEST mode, but places the generated control statements on file "my_file".

HELP mode may be selected by one of the keywords : HELP or H. For example :

```
ses,help.procedure_name
```

causes procedure "procedure_name" to be processed in HELP mode, and any descriptive information available is placed on file OUTPUT; whereas :

```
ses,h=my_info.procedure_name
```

causes procedure "procedure_name" to be processed in HELP mode, but any descriptive information available is placed on file "my_info".

Note : that when calling a procedure in HELP mode, a list of parameters should **not** be given. HELP for a group of procedures may be obtained by one call to SES, as follows :

```
ses,help.proc_1; proc_2; proc_3
```

STATUS mode may be selected by one of the keywords : STATUS or S. For example :

```
ses,status.procedure_name
```

causes procedure "procedure_name" to be processed in STATUS mode, and change information available is placed on file OUTPUT; whereas :

```
ses,s=my_change.procedure_name
```

causes procedure "procedure_name" to be processed in STATUS mode, but any change information available is placed on file "my_change".

Note : that when calling a procedure in STATUS mode, a list of parameters should **not** be given. STATUS for a group of procedures may be obtained by one call to SES, as follows :

```
ses,status.proc_1; proc_2, proc_3
```

C1.0 ERROR AND INFORMATIVE MESSAGES

C1.0 ERROR_AND_INFORMATIVE_MESSAGES

This appendix describes the messages produced by the SES processor when errors are detected. SES error messages have been made as self-explanatory as possible. When an error is detected by SES, a message is printed in the form:

```
** E CL 11001: EXPECTING "name found integer for parameter I" ON COMMAND STATEMENT
```

The E at the beginning of the line indicates this is an error message.
The CL is an abbreviation for the System Command Language used by SES to do syntax processing.
The number 11001 is an error code assigned to this error condition.
The text which follows the error code describes the error in detail. Appended to the end of the text is the line number of the line being processed by SES. In this first example, it is the command statement which is in error.

After the error message, SES outputs the line it was processing when the error was detected, followed by a line containing an up_arrow at the point in the line where the error was detected.

```
SES.REWRITE  I=123  O=ABC
             ^
```

Usually the error actually occurred on the token just before the up_arrow.

Here are two more typical examples of error messages:

```
** F CL 11007: REQUIRED PARAMETER MISSING "I" ON COMMAND STATEMENT
   SES.FORMAT
             ^


** E CL 11011: UNKNOWN KEYWORD "NVLS" ON LINE # 7 OF PROC SEGMENT MYPROC
   PARM  KEY = ('group', 'g')  NVLS = 1   NAM
                               ^
```

Other abbreviations used in SES error messages are SE, which means the error was detected in the processor itself, and UT, which

means the error was detected by a utility routine called by SES.

D1.0 SEMI-FORMAL SYNTAX DESCRIPTION

D1.0 SEMI-FORMAL SYNTAX DESCRIPTION

> "...What is the use of repeating all that stuff", the Mock Turtle interrupted, "if you don't explain it as you go on? It's by far the most confusing thing I ever heard!"
>
> Lewis Carroll

This section gives a semi-formal description of the syntax used when writing procedures for and calling the SES. The description is not intended to be rigorous. First we introduce the "meta-language" used to describe the syntax; second the character set used by SES is defined; and finally the syntax description itself is given.

D1.1 THE META LANGUAGE

This section describes the symbols used in the description of the SES syntax.

Symbol	Interpretation
::=	This symbol should be read as "is defined to be".
!	This symbol is used to indicate alternatives, for example: A ! B means that either A or B is allowed.
<item>	This group of symbols denotes that item is to be treated as a syntactic unit in relation to surrounding meta symbols.
[item]	This group of symbols denotes that item is optional, i.e. zero or one occurences of item are allowed.
{item}	This group of symbols denotes that item may be used zero or more times.

Spaces are used in the syntax description to improve its readability, however they are not part of what's being defined unless otherwise noted.

D1.0 SEMI-FORMAL SYNTAX DESCRIPTION
D1.1 THE META LANGUAGE

There are a few instances where some of the meta symbols themselves are part of the syntax definition, and when this occurs the meta symbol is underlined, for example: ¦ means the ¦ character and not the meta symbol. When an _ appear's alone, it means itself.

D1.2 CHARACTER SET

Characters used for NAMES

A .. Z	a .. z	Letters
0 .. 9		Decimal Digits
_		Underline
$		Dollar Sign
#		Pound
@		Commercial At

Characters used for INTEGER CONSTANTS

0 .. 9		Decimal Digits
A .. F	a .. f	Hexadecimal Digits
(		Open Parenthesis
)		Close Parenthesis

Characters used for OPERATORS

+		Plus Sign
-		Minus Sign
*		Asterisk
/		Slash (Slant)
=		Equal Sign
>		Greater Than Sign
<		Less Than Sign

Characters used for PUNCTUATION

		Blank (Space)
,		Comma
(		Open Parenthesis
)		CLose Parenthesis
.		Period

Character used for STRING DELIMITER

'		Apostrophe (Single Quote)

Character used for COMMMENT DELIMITER

"		(Double) Quote

D1.0 SEMI-FORMAL SYNTAX DESCRIPTION
D1.2 CHARACTER SET

(Default) Character used for SUBSTITUTION DELIMITER

& ..... Ampersand

(Default) Character used for DIRECTIVE HEADER

\ ..... Reverse Slash (Slant)

Note: Any ASCII character not listed in the above character set has no meaning to the SES processor. These characters may however be used in strings, comments, or as data characters.

D1.3 SYNTAX

D1.3.1 BASIC DEFINITIONS

```
<upper case letter> ::= A ¦ B ¦ C ¦ D ¦ E ¦ F ¦ G ¦ H
                      ¦ I ¦ J ¦ K ¦ L ¦ M ¦ N ¦ O ¦ P
                      ¦ Q ¦ R ¦ S ¦ T ¦ U ¦ V ¦ W ¦ X
                      ¦ Y ¦ Z

<lower case letter> ::= a ¦ b ¦ c ¦ d ¦ e ¦ f ¦ g ¦ h
                      ¦ i ¦ j ¦ k ¦ l ¦ m ¦ n ¦ o ¦ p
                      ¦ q ¦ r ¦ s ¦ t ¦ u ¦ v ¦ w ¦ x
                      ¦ y ¦ z

<letter> ::= <upper case letter>
           ¦ <lower case letter>

<decimal digit> ::= 0 ¦ 1 ¦ 2 ¦ 3 ¦ 4 ¦ 5 ¦ 6 ¦ 7 ¦ 8 ¦ 9

<hexadecimal digit> ::= A ¦ B ¦ C ¦ D ¦ E ¦ F
                      ¦ a ¦ b ¦ c ¦ d ¦ e ¦ f

<digit> ::= <decimal digit>
          ¦ <hexadecimal digit>

<base> ::= 2 ¦ 3 ¦ 4 ¦ 5 ¦ 6 ¦ 7 ¦ 8 ¦ 9 ¦ 10
         ¦ 11 ¦ 12 ¦ 13 ¦ 14 ¦ 15 ¦ 16
```

D1.3.2 TOKENS

This section defines the building blocks of SES syntax, collectively referred to as tokens. The internal token scanner of the SES processor is made available to the procedure writer by means of the built-in function TOKEN.

```
<token> ::= <name> ¦ <number> ¦ <string>
          ¦ <delimiter> ¦ <operator>

<name> ::= <alphabetic char> {<alphabetic char> ¦ <decimal digit>}

<alphabetic char> ::= <letter> ¦ _ ¦ $ ¦ # ¦ @

<upper case name> ::=
    <upper case letter> {<upper case letter> ¦ <decimal digit>}
```

All names are limited to thirty-one characters in length, except

procedure names and procedure identifiers, which are limited to ten characters. With the exception of <upper case name>s, any name may be specified with either upper or lower case letters, but before a name is used all letters in it are converted to upper case. For instance the names: ABC, abc, aBc, and so on, are all equivalent. (This includes any of the "special" names, such as DO, THEN, WHEN, etc. In this description, however, these names are always spelled out in upper case letters.)

```
<variable name> ::= <name>

<function name> ::= <name>

<parameter name> ::= <name>

<directive name> ::= <name>

<assignee> ::= <parameter name> ! <variable name>

<procedure name> ::= <name>

<procedure identifier> ::= <upper case name>


<number> ::= <decimal digit> {<digit>} [(<base>)]


<string character> ::= ''
     ! <any ASCII character except '>

<string> ::= '{<string character>}'


<constant> ::= <string> ! <number> ! <name>


<delimiter> ::= , ! ( ! ) ! = ! . ! ..{.}
              ! <end of line>

<operator> ::= <graphic operator> ! <mnemonic operator>

<graphic operator> ::= ** ! * ! / ! // ! + ! - ! ++
                     ! = ! /= ! <> ! ≤ ! ≤= ! ≥ ! ≥=

<mnemonic operator> ::= AND ! OR ! XOR ! NOT
```

D1.3.3 USE OF SPACES

Before discussing when and how spaces can be used we will first define the syntax of comments.

<comment> ::= "{<any ASCII character except ">}"

In almost all cases a comment is treated identically to a single blank character, and 2 or more contiguous blank characters (or comments) are treated as a single blank character. Blank characters and comments treated in this manner are known as spaces.

Spaces may be used between tokens to improve readability and in general may be used to replace commas when used as argument, value, or parameter separators. Spaces must be used to separate tokens when no <delimiter> or <graphic operator> can be used to separate them. For example the spaces between the tokens on the following line must be present:

```
V1 AND V2
```

whereas the following two expressions are equivalent:

```
V1 + V2
V1+V2
```

Further, the following value list contains 2 values:

```
( X, -3 )
```

whereas the next contains only 1 value:

```
( X -3 )
```

namely the value of the expression X-3.

Spaces within character strings represent themselves, and comments may not be used in front of the \ which occurs at the beginning of directive lines, nor following the continuation signal at the end of directive or call lines. Lines within procedures which are not directives or continuations of directives or lines which are read using the ACCEPT directive, are treated as unquoted strings, and therefore spaces are significant in them. Whenever a line is read by the SES processor, trailing blank characters are deleted. Also, it is legal to precede the \ of a directive line by one or more blank characters.

D1.3.4 EXPRESSIONS

```
<expr> ::= <lterm> {<or> <lterm>}
    <or> ::= OR ¦ XOR

<lterm> ::= <lfactor> {AND <lfactor>}

<lfactor> ::= [NOT] <lprimary>

<lprimary> ::= <sterm> {<rel op> <sterm>}
    <rel op> ::= = ¦ /= ¦ ≤ ¦ ≤= ¦ ≥ ¦ ≥=

<sterm> ::= <term> {++ <term>}

<term> ::= [<term op>] <factor> {<term op> <factor>}
    <term op> ::= + ¦ -

<factor> ::= <primary> {<factor op> <primary>}
    <factor op> ::= * ¦ / ¦ //

<primary> ::= <operand> {** <operand>}

<operand> ::= <variable reference>
            ¦ <function reference>
            ¦ ( <expr> )
            ¦ <constant>
            ¦ <null>

<null> ::=

<variable reference> ::= <variable name>

<function reference> ::= <function name> <arguments>

<arguments> ::= ( [<arg> {, <arg>}] )
              ¦ <null>

<arg> ::= <name> ¦ <expr>


<integer expr> ::= <expr>    " must resolve to an integer "
<string  expr> ::= <expr>    " must resolve to a string   "
<boolean expr> ::= <expr>    " must resolve to an integer "
                             "   if the value is zero, it "
                             "     is taken to be FALSE   "
                             "   if non-zero, it's taken  "
                             "     to be TRJE             "
```

D1.3.5 FOREIGN TEXT

Foreign text is primarily used for parameter values which are to be in turn used as parameter lists (e.g., to secondary procedures) or simply to prevent the SES processor from evaluating an expression.

The scanning of foreign text is totally different from scanning "normal" text. The characteristics of this special scanning are

- parentheses are "balanced"
- single and double quotes are "matched"
- if not contained within parentheses, single quotes, or double quotes, the tokens: comma, period, ellipsis (..{.}), and close parenthesis will terminate the scanning (and thus the foreign text value). In addition, spaces which are used to separate names, numbers, or strings from names numbers or strings will terminate scanning; as will an "unenclosed" open parenthesis which follows a string or number (Note, that an open parenthesis following a name does not terminate scanning - this is because function references are allowed in foreign text but the foreign text scanner doesn't evaluate what it scans, and thus does not know if the name is indeed the name of a function).

Foreign text may also be described as having the general format of an expression, but the expression is not evaluated when scanned as foreign text. During scanning comments and blanks not contained within single quotes are "thrown away" and single blank characters are inserted between tokens which would otherwise not be separated.

The following example illustrates some of the idiosyncracies of foreign text:

```
\  vlist = '(a b c (d e) ''p q''''r, s'' 123(8) (x,(y+3)) )'
\  count = VCNT (vlist)                          " 2 "
\  value = VALS (vlist, 3)                       " 3 "
\  slist = GENLIST (vlist, index)                " 4 "
```

The first line defines a value list in the variable vlist. Line 2 sets the variable count to the value 6. Line 3 sets the the variable value to the value:

```
C(D E)
```

and line 4 sets the variable slist to the value:

```
A,B,C(D E),'p q''r, s',123(8),(X,(Y+3))
```

D1.0 SEMI-FORMAL SYNTAX DESCRIPTION
D1.3.5 FOREIGN TEXT

The next example illustrates how a parameter list may be passed as a foreign text parameter:

```
\  plist = '( i=infile  "columns"  cols=1..80 o=out )'
\  count = VCNT (plist)
\  low   = VALS (plist, 2, LOV)
\  high  = VALS (plist, 2, HIV)
\  slist = GENLIST (plist, index)
```

Count is set to 3; low is set to:

```
COLS=1
```

high is set to 80; and slist is set to:

```
I=INFILE,COLS=1..80,O=OUT
```

D1.3.6 PARAMETER LISTS

```
<parameter list> ::= [<parameter> {[,] <parameter>}]

<parameter> ::= [<parameter name> [=]] <value list>
              ¦ <parameter name>
              ¦ <null>

<value list> ::= <value>
               ¦ ( [<value> {[,] <value>}] )

<value> ::= <value side> [..{.} <value side>]

<value side> ::= <expr> ¦ <foreign text>
```

D1.3.7 SES PROCESSOR CALL

```
<csep> ::= [;] <end of line>
         ! ;

<proc call> ::= <procedure name> [,] <parameter list> <csep>

<control statement> ::= <string>

<control statements> ::= <control statement> [<csep>]
                         {[,] [<control statement>] [<csep>]}

<call element> ::= <proc call> ! <control statements>

<SES call> ::= SES [, <parameter list>] .
                   <call element> {<call element>}
```

Because of operating system restrictions, a <parameter list> following the SES (processor name) must have explicit punctuation. That is to say, commas must be used to separate parameters (and values) and equal signs must be used to separate parameter names (keywords) from their value lists.

Also, the operating system is not well acquainted with lower case letters, so only upper case should be used; however, NAM/IAF (or TELEX) and the SES processor alleviate this problem by converting lower case letters to upper case on command and continuation lines.

When <control statements> are used in a <SES call>, the SES processor insures that they are all "properly" terminated, i.e., each <control statement> string is scanned for a right parenthesis or period and if neither of these characters is found, a period will be appended at the end of the string; if however, a right parenthesis or period is found, the string will be left alone. NOTE that this is the only validity checking of the <control statement> done by the SES processor.

D1.3.8 SUBSTITUTION

```
<substitute> ::= <parameter name> ! <variable name>

<default substitution character> ::= &

<alternate substitution character> ::=
     !!"!#!$!%!'!(!)!*!=!+!-!/!~!^!`!@!,!.!;!:!?
    !\!_!<!>![!]!{!}!|

<substitution char> ::=
      <default substitution character>
    ! <alternate substitution character>

<substitution> ::=
    <substitution char> <substitute> <substitution char>
```

D1.3.9 PROCEDURES

```
<procedure> ::= <procedure identifier> {<procedure line>}

<procedure line> ::= {<procedure line element>}

<procedure line element> ::= <substitution>
                           ! <any ASCII character>
```

The process of substitution applied to a <procedure line> yields an <object line>.

```
<object line> ::= <directive line>
                ! <empty line>
                ! <data line>

<default directive character> ::= \

<alternate directive character> ::=
     !!"!#!$!%!'!(!)!*!=!+!-!/!~!^!`!@!,!.!;!:!?
    !&!_!<!>![!]!{!}!|

<directive header> ::=
      <default directive character>
    ! <alternate substitution character>

<directive line> ::= <directive header> <directive>

<empty line> ::=

<data line> ::= <any line which is not "empty" and does
                 not begin with a directive header>
```

Note: <empty line>s may contain comments enclosed in double quotes.

D1.3.10 DIRECTIVES

```
<directive> ::= <assignment>
    ¦ <if while> <boolean expr> [<then do>]
    ¦ <exit cycle> [WHEN <boolean expr>]
    ¦ <directive name> <parameter list>

<assignment> ::= <assignee> = <expr>

<if while> ::= IF ¦ ORIF ¦ WHILE

<then do> ::= THEN
            ¦ DO

<exit cycle> ::= EXIT ¦ CYCLE
```

D1.4 LINES_AND_THEIR_CONTINUATION

It is sometimes necessary to pass more parameters to a procedure (or give more parameters to a directive) than will fit on one line. To handle this problem, SES processes continuation lines.

The effective net result of using continuation lines is to construct an unbroken line of up to 2000 characters for calls to SES, or 256 characters for an SES directive.

Continuation may only be used in conjunction with SES directives and when calling SES to process a procedure. Continuation is signalled on the line which is to be continued, not the continuation line itself. Note that the <continuation signal> is not considered to be part of the line. The mechanism for doing this is defined as follows:

```
<whole line> ::=
    <line starter> <stuf 1> [ <continuation signal>
                   <stuf 2> { <continuation signal>

                   <stuf n> } ]


<continuation signal> ::= ..{.}

<line starter> ::= <directive header> <name>
                 ! SES <parameter list>

<stuf i> ::= <whatever belongs with the line starter>
      " 1 <= i <= n "
```

The effect of this is as if <whole line> had been specified as:

```
    <line starter> <stuf 1> {<stuf i>}
```

Note: Syntactic units (tokens) may cross line boundaries.

E1.0 ACQUIRE UTILITY

E1.0 ACQUIRE UTILITY

ACQUIRE is a program that enables easy retrieval (acquisition) of permanent files.

ACQUIRE combines the functions of the NOS "ATTACH" and "GET" control statements. For each file specified ACQUIRE determines if the file is already local to the job (unless suppressed by the PO parameter, see below), if so it is rewound; if not, then for each one of a list of user names, an ATTACH is attempted (waiting, if necessary, until the file is not busy), and if that fails a GET is tried. If, after all this, the file is still not local, an appropriate dayfile message is issued.

Unless the A (abort) parameter is specified, ACQUIRE aborts only because of control statement format or argument errors, or because of a permanent file manager (PFM) detected error; and not because one (or more) of the specified files could not be found.

The control statement format for ACQUIRE is :

```
ACQUIRE(lfn1=pfn1,lfn2=pfn2,.../op1,op2,...)
```

lfni is the (local) name of the file once it has been ACQUIREd (note that this is the name used in making the "is the file already local?" test)

pfni is the permanent file name for the file (if =pfni is omitted, pfni is assumed to be the same as lfni)

opi specify options used for acquiring the file(s) :

- A — specifies that if a file is not found, ACQUIRE should abort
- NA — is the opposite of A (and is the default)
- PO — specifies Permanent Only, i.e., that if a file is already local, it is returned and then the ATTACH and GET are attempted
- UN=users — specifies a list of user names to be searched for each file (the user names are separated from each other by commas)
- M=mode — specifies the access mode desired for the file

E1.0 ACQUIRE UTILITY

(READ or R -- the default, WRITE or W, or EXECUTE or E). If the file is local, the mode will not be changed. To insure the mode will be changed, the PO parameter should also be specified.

PW=pw specifies the permanent file's password

PN=pn specifies the packname for the permanent file

When ACQUIRE is attempting an ATTACH or GET, if the file is busy or if a permanent file utility is active, the following message is issued and the request is retried :

- WAITING FOR PFN=permanent_file_name UN=user_name

When ACQUIRE is attempting an ATTACH or GET, if an error is detected by PFM the following message is issued and ACQUIRE aborts :

- ERROR WITH PFN=permanent_file_name UN=user_name

In both of the above cases, the designated message is preceded by a more specific message generated by PFM.

F1.0 EDT - ENHANCED VERSION OF NOS TEXT EDITOR

F1.0 EDT - ENHANCED VERSION OF NOS TEXT EDITOR

EDT is an enhanced version of the standard NOS text editor (EDIT). EDT incorporates column searches, modified FIND and FINDS commands, Input and Output file and TAB specification on the EDT call, control over terminal interrupts, INPUT MODE options, multiple entries on command lines, command buffers, permanent file functions, and miscellaneous other additions.

User documentation can be found on TOOLDOC (SESD009).

F1.1 EDT CONTROL STATEMENT FORMAT

EDT,[lfn1],[lfn2],[lfn3],[tab].[cmdstr]

lfn1 - name of the file to be edited

lfn2 - name of the file containing the editor commands (default = INPUT)
- if 0, an END command is executed after processing the commands in cmdstr
- if 1, commands are read from file INPUT after processing the commands in cmdstr

lfn3 - name of the output file (default = OUTPUT)

tab - tabset to be used by EDT

cmdstr - editor Command String to be executed if lfn2 = 0 or 1

G1.0 EXTRACT UTILITY

G1.0 EXTRACT UTILITY

EXTRACT is a program that enables easy retrieval of records from permanent file (or local) libraries.

EXTRACT is similar in function to the NOS "GTR" statement. It differs from "GTR" in the following ways:

- EXTRACT insists that the library to be searched has a directory (a directory can be built using the NOS utility "LIBEDIT").
- The record type parameter for EXTRACT, if given, applies to all records to be extracted, and if not given, only the names of the records are used when searching the library.
- Each extracted record is copied to its own local file by EXTRACT, rather than all to the same file.
- EXTRACT does not insist that the library to be searched be local to the job when it's called, but ACQUIRE's the library from a permanent file catalog.

The control statement format is:

```
EXTRACT(lfn1=rn1,lfn2=rn2,.../op1,op2,...)
```

lfni — Is the local file name given to the record once it's extracted (lfni is REWOUND before and after the extraction takes place).

rni — Is the name of the record to be extracted (if omitted, it is assumed to be the same as lfni).

opi — These parameters specify options that control the extraction process :

- A — Specifies that if a record is not found, EXTRACT should abort.
- NA — Is the opposite of A (and is the default).
- T=rt — Specifies the record type (if given, it applies to all records being extracted; if omitted, only

the record names are used when searching the library).

L=libname — Specifies the name of the library to be searched for the records (if omitted, "PROCLIB" is assumed).

LFN=libifn — Specifies the local file name for the library (if omitted, the "libname" from the L paraeter is used). Note that this is the name used to make the "is file local?" test when ACQUIRing the library.

UN=un — Specifies the user name of the permanent file catalog to be searched for "libname" if it's not already local (if omitted, the current user is assumed).

PW=pw — Specifies the library's permanent file password.

PN=pn — Specifies the library's permanent file packname.

Valid record type designators are documented under the description of the "CATALOG" control statement in the NOS Reference Manual.

In addition to these standard types, there's one more "type" processed by EXTRACT, which is designated by "TXT". This "type" is used to denote "TEXT" records that, when extracted, are to have their first line (which contains the record's name) "stripped off". This is useful if, for example, one has records containing directives for a NOS utility, in which case the name of such a record is in all likelihood an illegal directive to the utility program.

EXTRACT aborts under any of the following conditions:

- format or argument error(s) on the control statement
- the specified library could not be AQUIREd
- the library file does not have a directory as the last record before end-of-information

Note, however, that EXTRACT doesn't abort if it does not find any of the requested records (only an informative dayfile message is issued), unless the Abort parameter is coded on the call.

G1.0 EXTRACT UTILITY

If the library file is not local to the job when EXTRACT is called, it is RETURNed when EXTRACT terminates normally; but, if the library file is local, EXTRACT REWIND's it prior to normal termination.

H1.0 SESMSG UTILITY

H1.0 SESMSG UTILITY

SESMSG is a program which copies the comment field of its call line to a file. The control statement format is:

SESMSG,file.message

file is the name of the file to receive the message (if omitted, OUTPUT is assumed)

message is the message to be written to the file

The message is written to the file only if the file is a terminal file, or if "file" is explicitly quoted on the call line.

SESMSG is used in procedure files to inform the user about what the procedure is currently doing. It is also used for creating files of input directives to utility programs when such directives are dependent on execution time considerations.

I1.0 JOBID - PRODUCE LARGE PRINT BANNER

I1.0 JOBID - PRODUCE LARGE PRINT BANNER

JOBID is the program used by the SES PRINTID procedure to produce "large print" banner pages.

The output produced is one or more pages of 5 lines of block characters at 10 characters per line. All empty character positions and lines are blank filled. Additional pages of block lettering is produced if any information is specified following a fifth delimiter character, and so on.

The input to JOBID is a string of characters in the following form :

```
line1/line2/line3/line4/line5
```

or

```
/////+line1+line2+line3+line4+line5
```

In the first form the slant (/) character is taken to be the delimiter between lines (i.e., it is the default delimiter). The second form must be used when you want one or more of the lines to contain the slant character. In the example above, the delimiter has been changed to a plus (+) character.

Note : that only characters with a 6-bit display code representation can be given to JOBID. To produce a colon (:) on output you should use a percent sign (%) on input. Unnecessary delimiters after the message data should be omitted, otherwise a trailing delimiter may be printed.

Automatic Date and Time Substitution

Once JOBID has broken the message field into message lines based on the delimiter character, the front of each field is examined for the following sequences of characters :

```
)DATE
)TIME
)ETAD
)AMPM
```

Whenever these character sequences are found, the message line on which they occur is replaced with the current value of the indicated

I1.0 JOBID - PRODUCE LARGE PRINT BANNER

parameter. (NOTE that the right parenthesis must be the first character of the message line on which it occurs).

The above "commands" produce the following types of results :

```
)DATE       17 SEPT 74
)TIME        14:18:21
)ETAD        74/09/17          (date reversed)
)AMPM        3:05  PM
```

JOBID Examples

```
GRAPL/OLDPL/TAPE =/ 7777

TEXTJAB/COMPILE/ OF/)DATE/)TIME

/////+CREATED ON+ 17/09/74++LATEST RUN+)DATE

SEND TO/MEADOWVALE/ROOM 888

/DAYFILE/DUMP/)ETAD
```

The above calls produce the results shown below :

```
GRAPL        TEXTJAB      CREATED ON  SEND TO
OLDPL        COMPILE      17/09/74    MEADOWVALE  DAYFILE
TAPE =        OF                      ROOM 888    DUMP
 7777        17 SEPT 74   LATEST RUN               74/09/17
              13:35:26     18 SEPT 74
```

J1.0 TXTHEAD DOCUMENT (HEADINGS) POST-PROCESSOR

J1.0 TXTHEAD DOCUMENT (HEADINGS) POST-PROCESSOR

Sometimes the header material generated by TXTFORM is not desired or TXTFORM is unable to produce a needed header format. The TXTHEAD utility processes a TXTFORM output file, which contains special TXTHEAD instruction pages, and removes the old header material and/or inserts the new heading material. Additionally, an incrementing page number can be produced at any position within the header text, or the TXTFORM page number may be passed to TXTHEAD for placement into the header region.

COMMANDS	ACTION
///HEAD	New heading material follows until another /// is encountered in columns 2,3,4.
///END	Usually marks end of special heading material.
///PAGE[-]nn	Set page number for next page to nn. If negative, the page number is not printed until it increments to 1.
///LINEnn	Line nn of header on which to put a page number.
///COLUMNnn	Column position for units digit of page number is nn, for TXTHEAD generated page nos., and for TXTFORM generated page numbers the column position specifies the location of the left hand side of the page number. (Count includes carriage control column).
///ADDNOnn	Number of header lines to be added is nn. Default is add all lines given.
///DELETEnn	Number of lines to be deleted from top of each page is nn.
///OLDNO	Use the TXTFORM generated page nos.
///NEWNO	Use the sequentially generated TXTHEAD page nos. This is the default condition.

J1.0 TXTHEAD DOCUMENT (HEADINGS) POST-PROCESSOR

To give these commands to TXTHEAD, the following type of format should be used in a TXTCODE document at the place where the change is to occur.

```
\asis
\margin1
\length80
\page
///head
1       ******************
        *                *
        *  HEADER TEXT   *
        *                *                          Page
        ******************
///end
///page1
///line3
///column65
///delete3
\margin10
\length62
\block
\page
```

Notes :

Header information as well as any specifiable information may be defined or redefined anywhere. Old values remain set until replaced. The page number may be eliminated by specifying ///line0 or any out of range value.

Any size header is permissible. It is also possible to supply folio information for pages by putting the page eject at some point after the first line of header material. Column one of the header information is assumed to contain printer control characters such as "blank", +, -, 0, etc.

A great deal of flexibility is available for special page formats.

K1.0 JABFORM CONVERSIONS

K1.0 JABFORM CONVERSIONS

The following is a table of conversion performed by JABFORM :

IEXIJAB	IXIEORM
-A-	\A
-Bxx,...,zz-	\Bxx,...,zz
-BBxx,...,zz-	\BBxx,...,zz
-C-	\C
-E-	\E
-F-	\F
-Gxx,...,zz-	\Gxx,...,zz
-HN-Y.YC...C	\HN C...C
-IN-	\IN
-JN-	\JN
-KN-	\KN
-L-	\L
-M-	\M
-Nx-	\Nx
-O-	\O
-P-	\P
-Q-	\Q
-R-	\R
-SN-	\SN
-T-	\T
-.TN-	\.TN
-U-	\U
-Vxx,...,zz-	\Vxx,...,zz
-VVxx,...,zz-	\Vxx,...,zz
-XN-	\XN
-Z-	\Z
-- (MINUS MINUS)	\
-ANY COMBINATIONS OF EDIT CODES-	\COMBINATION OF CODES

K1.0 JABFORM CONVERSIONS

IEXIJAB	IXIEORM
+ARABIC+	discarded
+AUTO+	discarded
+CARDLNXX+	\RECORD1,XX
+CHANGE,C+	\CHANGE,C
+DATE+XXXXXX	\DATE XXXXXX
+DEPTHXX+	\FORMAT,,XX
+DOUBLE+	\SPACE2
+EIGHT+	discarded
+ENDT+	\ (See +TABLE+)
+FOLIO+XXXXXX	\FOLIO XXXXXX
+HEADA+	\HEADA
+HEADA=XX+	\HEADA (=XX discarded)
+HEADB+	\HEADB
+HEADB=XX+	\HEADB (=XX discarded)
+HEADC+	\HEADC
+HEADC=XX+	\HEADC (=XX discarded)
+JUST+	\just
+LENGTHNN+	\LENGTHNN (After this, JABFORM only processes column 1 thru NN; NN+1 to EOL is passed thru.)
+LIST(X,...,X)+TITLE	\LISTX,...,X TITLE (discards P= and R=)
+MARGINXX+	\MARGINXX
+NOAUTO+	discarded
+NOJUST+	\NOJUST
+NOPTAB+	discarded
+NOSEQ+	\NOSEQ
+NOSOURCE+	discarded
+PTAB+	discarded
+RECORDXX,YY+	\RECORDXX,YY (After this, JABFORM only processes columns XX thru YY; column 1 thru XX-1 and YY+1 to EOL will be passed thru.)
+ROMAN+	discarded
+SEQ+	\SEQ

K1.0 JABFORM CONVERSIONS

IEXIJAB	IXIFORM
+SETA+C	discarded (JABFORM recognizes the C as an alter code from now on.)
+SETC+C	discarded
+SETE+C	discarded (JABFORM recognizes the C as an edit code from now on.)
+SETQ+C	discarded (JABFORM recognizes the C as capitalize string code from now on.)
+SETS+C	discarded (JABFORM recognizes the C as the special character code from now on.)
+SETU+C	discarded (JABFORM recognizes the C as the underline string code from now on.)
+SETTAB1=XX,...,15=ZZ+	\SETTAB1XX,...,15ZZ
+SINGLE+	\SPACE1
+SIX+	discarded
+SOURCE+	discarded
+SUBTITLE+XXXXXX	\TITLE2 XXXXXX
+TABLE+	\COMMENT +TABLE+ (Input thru next endt is not processed, but passed to TXTFORM as a comment shifted to the right one space.) +ENDT+ \
+TITLE+XXXXXX	\TITLE1 XXXXXX
+TODAY+	\TODAY
*C	*C (C is not capitalized - ASCII 6/12 TEXTJAB input does not use this feature.)
'C...C' (apostrophies)	C...C (capitalized)
$C...C$	\UNDER C...C \UOFF

K1.0 JABFORM CONVERSIONS

IEXIJAB	IXIEORM
/A	+ (plus)
/B	] (closing bracket)
/C	' (single quote, grave accent, or closing quote)
/D	$ (dollar sign)
/E	; (semicolon)
/F	> (greater than)
/G	< (less than)
/H	+ (plus)
/I	\BLANK1
/J	_ (underline)
/K	% (percent)
/L	(unknown - blank)
/M	& (ampersand)
/N	!(exclamation)
/O	= (equal)
/P	* (asterisk)
/Q	@(commercial at)
/R	' (apostrophe or opening quote)
/S	- (minus)
/T	: (colon)
/U	# (pound)
/V	[(opening bracket)
/W	? (question)
/X	* (asterisk)
/Y	" (quotation marks)
/Z	/ (slant)
/0	0
/1	1
/2	2
/3	3
/4	4
/5	5
/6	6
/7	7
/8	8
/9	9

K1.0 JABFORM CONVERSIONS

IEXIJAB	IXIEORM
/(space)C..C	\COMMENT C..C
/*	* (asterisk)
/'	' (single quote, grave accent, or closing quote)
/-	- (minus)
/+	+ (plus)
/$	$ (dollar sign)
/[	[(opening bracket)
/]	] (closing bracket)
/"	" (quotation marks)
//A	+ (plus)
//B	} (closing brace)
//C	^ (circumflex)
//D	' (single quote, grave accent, or closing quote)
//E	' (apostrophe or opening quote)
//F	\(reverse slant)
//G	(commercial at)
//H	+ (plus)
//I	(space)
//J	+ (plus)
//K	¦ (vertical line)
//L	- (minus)
//M	¦ (vertical line)
//N	~ (tilde)
//O	" (quotation marks)
//P	(space)
//Q	(space)
//R	(space)
//S	(space)
//T	(space)
//U	+ (plus)
//V	{ (opening brace)
//W	* (asterisk)
//X	* (asterisk)
//Y	+ (plus)
//Z	\(reverse slant)
///	/ (slant)
//&	& (ampersand)

NOTES :

1) If JABFORM is passing columns, unprocessed data in the passed columns on input is in the same columns on output. Blank padding, when required, is done to accomplish this.

2) All TXTFORM codes are put on a line by themselves.

3) All TEXTJAB codes (except +TABLE+) that have no TXTFORM

K1.0 JABFORM CONVERSIONS

conversion is converted to EOL and a message sent to ERRFILE.

CAUTION :

1) The TEXTJAB source XXX 'YYY' converts to :
   XXX
   \UNDER
   YYY
   \UOFF
   Which lists as XXXYYY. To get a space before the first Y, the TXTFORM source must be changed to have a space before the Y.

2) TXTFORM can only handle line length up thru 100 characters on output and TEXTJAB allows 120. This means the +LENGTH+ code will convert OK, but TXTFORM will not process correctly.

3) TXTFORM can only handle tab settings up thru 99 and TEXTJAB will handle 119. This means a TEXTJAB SETTAB alter code will convert OK, but TXTFORM will not be able to handle it correctly.

   Example :

   +SETTAB7=103,8=113+
   This will convert to :
   \SETTAB7103,8113 - with tab number 71 and 81 with columns of 3 and 13.

L1.0 CHARACTER SET TABLE (USED BY CONV)

L1.0 CHARACTER SET TABLE (USED BY CONV)

The following table shows the ASCII characters and the internal octal codes that represent them within the various character sets that are processed by the CONV utility.

L1.0 CHARACTER SET TABLE (USED BY CONV)

ASCII GRAPHIC	N63 NBE63	N64 NBE64	N612	NBE63A	NBE64A	ASCII4 ASCII5	N612U	N612L
: (COLON)	63	00	7404	072	072	072	7404	7404
A	01	01	01	101	101	101	01	-
B	02	02	02	102	102	102	02	-
C	03	03	03	103	103	103	03	-
D	04	04	04	104	104	104	04	-
E	05	05	05	105	105	105	05	-
F	06	06	06	106	106	106	06	-
G	07	07	07	107	107	107	07	-
H	10	10	10	110	110	110	10	-
I	11	11	11	111	111	111	11	-
J	12	12	12	112	112	112	12	-
K	13	13	13	113	113	113	13	-
L	14	14	14	114	114	114	14	-
M	15	15	15	115	115	115	15	-
N	16	16	16	116	116	116	16	-
O	17	17	17	117	117	117	17	-
P	20	20	20	120	120	120	20	-
Q	21	21	21	121	121	121	21	-
R	22	22	22	122	122	122	22	-
S	23	23	23	123	123	123	23	-
T	24	24	24	124	124	124	24	-
U	25	25	25	125	125	125	25	-
V	26	26	26	126	126	126	26	-
W	27	27	27	127	127	127	27	-
X	30	30	30	130	130	130	30	-
Y	31	31	31	131	131	131	31	-
Z	32	32	32	132	132	132	32	-
0	33	33	33	060	060	060	33	33
1	34	34	34	061	061	061	34	34
2	35	35	35	062	062	062	35	35
3	36	36	36	063	063	063	36	36
4	37	37	37	064	064	064	37	37
5	40	40	40	065	065	065	40	40
6	41	41	41	066	066	066	41	41
7	42	42	42	067	067	067	42	42
8	43	43	43	070	070	070	43	43
9	44	44	44	071	071	071	44	44

L1.0 CHARACTER SET TABLE (USED BY CONV)

ASCII GRAPHIC	N63 NBE63	N64 NBE64	N612	NBE63A	NBE64A	ASCII4 ASCII5	N612U	N612L
+ (PLUS)	45	45	45	053	053	053	45	45
- (MINUS)	46	46	46	055	055	055	46	46
* (ASTERISK	47	47	47	052	052	052	47	47
/ (SLASH)	50	50	50	057	057	057	50	50
((LEFT PAREN)	51	51	51	050	050	050	51	51
) (RIGHT PAREN)	52	52	52	051	051	051	52	52
$ (DOLLAR SIGN)	53	53	53	044	044	044	53	53
= (EQUAL)	54	54	54	075	075	075	54	54
(SPACE)	55	55	55	040	040	040	55	55
, (COMMA)	56	56	56	054	054	054	56	56
. (PERIOD)	57	57	57	056	056	056	57	57
# (POUND)	60	60	60	043	043	043	60	60
[(LEFT BRACKET)	61	61	61	133	133	133	61	61
] (RIGHT BRACKET)	62	62	62	135	135	135	62	62
% (PERCENT)	-	63	63	-	045	045	63	63
" (DOUBLE QUOTE)	64	64	64	042	042	042	64	64
_ (UNDERLINE)	65	65	65	137	137	137	65	65
! (EXCLAMATION)	66	66	66	041	041	041	66	66
& (AMPERSAND)	67	67	67	046	046	046	67	67
' (APOSTROPHE)	70	70	70	047	047	047	70	70
? (QUESTION)	71	71	71	077	077	077	71	71
< (LESS THAN)	72	72	72	074	074	074	72	72
> (GREATER THAN)	73	73	73	076	076	076	73	73
@ (COMMERCIAL AT)	74	74	7401	100	100	100	7401	7401
\ (REV. SLASH)	75	75	75	134	134	134	75	75
^ (CIRCUMFLEX)	76	76	7402	136	136	136	7402	7402
; (SEMICOLON)	77	77	77	073	073	073	77	77

L1.0 CHARACTER SET TABLE (USED BY CONV)

ASCII GRAPHIC	N63 NBE63	N64 NBE64	N612	NBE63A	NBE64A	ASCII4 ASCII5	N612U	N612L
a			7601			141	-	7601
b			7602			142	-	7602
c			7603			143	-	7603
d			7604			144	-	7604
e			7605			145	-	7605
f			7606			146	-	7606
g			7607			147	-	7607
h			7610			150	-	7610
i			7611			151	-	7611
j			7612			152	-	7612
k			7613			153	-	7613
l			7614			154	-	7614
m			7615			155	-	7615
n			7616			156	-	7616
o			7617			157	-	7617
p			7620			160	-	7620
q			7621			161	-	7621
r			7622			162	-	7622
s			7623			163	-	7623
t			7624			164	-	7624
u			7625			165	-	7625
v			7626			166	-	7626
w			7627			167	-	7627
x			7630			170	-	7630
y			7631			171	-	7631
z			7632			172	-	7632

L1.0 CHARACTER SET TABLE (USED BY CONV)

ASCII GRAPHIC	N63 NBE63	N64 NBE64	N612	NBE63A	NBE64A	ASCII4 ASCII5	N612U	N612L
{ (LEFT BRACE)			7633			173	7633	7633
! (VERTICAL BAR)			7634			174	7634	7634
} (RIGHT BRACE)			7635			175	7635	7635
~ (TILDE)			7636			176	7636	7636
DEL			7637			177	7637	7637
NUL			7640			000	7640	7640
SOH			7641			001	7641	7641
STX			7642			002	7642	7642
ETX			7643			003	7643	7643
EOT			7644			004	7644	7644
ENQ			7645			005	7645	7645
ACK			7646			006	7646	7646
BELL			7647			007	7647	7647
BS			7650			010	7650	7650
HT			7651			011	7651	7651
LF			7652			012	7652	7652
VT			7653			013	7653	7653
FF			7654			014	7654	7654
CR			7655			015	7655	7655
SO			7656			016	7656	7656
SI			7657			017	7657	7657
DLE			7660			020	7660	7660
DC1			7661			021	7661	7661
DC2			7662			022	7662	7662
DC3			7663			023	7663	7663
DC4			7664			024	7664	7664
NAK			7665			025	7665	7665
SYN			7666			026	7666	7666
ETB			7667			027	7667	7667
CAN			7670			030	7670	7670
EM			7671			031	7671	7671
SUB			7672			032	7672	7672
ESC			7673			033	7673	7673
FS			7674			034	7674	7674
GS			7675			035	7675	7675
RS			7676			036	7676	7676
US			7677			037	7677	7677
` (GRAVE ACCENT)			7407			140	7407	7407

M1.0 DIFFERENCES BETWEEN MODIFY AND MADIFY

M1.0 DIFFERENCES BETWEEN MODIFY AND MADIFY

General	MODIFY	MADIFY
6/12 character set support	no	yes
Field Length management	yes	no
Nested CALLs	no	yes
Default line length	72	110
Maximum line width	100	120
Compile file sequenced by default	yes	no
SEQ directive allowed in input stream	no	yes
Value can be specified with DEFINE directive	yes	no
Control Statement		
Product Set Format	yes	no (/required)
CV option	yes	no
S1 through S5 options	no	yes
NN option (no-nesting of CALLS)	no (default)	yes
Compile file		
IF/ELSE/ENDIF directives	yes	no
IGNORE directive	yes	no
CALLC option (conditional CALL)	no	yes

Other Differences

1. MADIFY includes the following additional options -
   - S1 - Do not write the "deckname" or "COMMON" on the SOURCE file.
   - S2 - Do not expand CALL directives on the COMPILE files unless EDIT is specified.
   - S3 - Allow empty INPUT and OPLFILE directive files.
   - S4 - (Specified as DE=name). Write only the deck specified by "name" and all COMMON decks onto the new program library.
   - S5 - Allow *CREATE directive without the first image of the CREATE file being a deck name. Deck name defaults to "SESOPT".
   - NN - Do not process nested CALLs. That is, if a directive which would ordinarily require expansion of a COMMON deck appears in a COMMON deck, do not perform the expansion.

2. The compile file directive, CALLC, inhibits the expansion of

M1.0 DIFFERENCES BETWEEN MODIFY AND MADIFY

the specified deck if the deck specified has already been expanded within the outermost deck of the CALL nest.

3. MADIFY, although processing 6/12 characters, is based upon a 63-character set MODIFY and therefore does not handle display-code colons properly.

4. MODIFY and MADIFY new program libraries can be processed only by the program which generated them.

N1.0 MECHANISM FOR COLLECTING USER SUPPLIED SOFTWARE

N1.0 MECHANISM_FOR_COLLECTING_USER_SUPPLIED_SOFTWARE

This mechanism defines when and how SES makes user developed software tools available to the entire SES user community. The mechanism is based on the assumption that it is beneficial to gather tools developed by people outside the SES organization into a common, known location.

N1.1 OBTAINING_USER_SOFTWARE

The user supplied software which is assembled and made available by SES falls into the area generally considered as "tools". It is sufficiently general purpose so it is useful to people other than the originator. It is comprised of SES procedures, programs or modules.

Software supplied by a user is totally transferable to another catalog. It is not dependent on any developer's catalog, or any catalog other than the one the tool itself resides in. If the software being supplied is a program or module, the source code is supplied, along with a build procedure that is also totally transferable to another catalog. The author can request the source remain unaccessible to other users.

In addition to the actual software, the user submits the following documentation:

A) A short description of the tool, which is kept in an online directory of user supplied tools along with the supplier's name, location and extension.

B) Documentation on how to use the software. The more complete it is, the fewer phone calls the supplier receives.

C) SES procedures contain HELP documentation that assist someone in using the procedure.

N1.2 MAKING_USER_SOFTWARE_AVAILABLE_THROUGH_SES

When SES obtains user developed software, it is handled in a way similar to SES developed software. Procedures are collected on a procedure library called USRPLIB in the SES catalog. Binary files are assigned USS numbered files, and are kept in the SES catalog. The source for user supplied software is collected onto a

semi-private SCU PL called **USSPL** and kept in the SES catalog. If the author requests that it not be accessible, the source is kept in a private SES file.

The major advantage of maintaining user supplied software is the SES catalog is than any changes or new software are automatically sent to the sites receiving the regular SES updates.

N1.3 COMMUNICATING AVAILABILITY OF USER SUPPLIED SOFTWARE

An online directory (**USSINFO**) of user supplied tools is maintained on the SES catalog. This file contains a short description of the tool provided by the developer. This description is also circulated via the Advanced Systems Development Productivity Circle and/or the SES Tools Bulletin.

Hard copy documentation supplied by the user is available from SES on request or online via **TOOLDOC**.

N1.4 SUPPORT OF USER SUPPLIED SOFTWARE

All corrections, improvements or other modification to user supplied software are the responsibility of the person supplying the software. SES does not support any user supplied software.

If software supplied by a user is standardized (it becomes a SES supported product) it is removed from the user supplied category.

Table of Contents

1.0 PREFACE 1-1
1.1 INTRODUCTION 1-2
1.1.1 USING AN SES PROCEDURE 1-5
1.1.2 PARAMETERS FOR SES PROCEDURES 1-5
1.1.2.1 Types of Parameter Values 1-8
1.1.2.1.1 NAMES AS PARAMETER VALUES 1-8
1.1.2.1.2 LONG_NAMES AS PARAMETER VALUES 1-8
1.1.2.1.3 NUMBERS AS PARAMETER VALUES 1-8
1.1.2.1.4 CHARACTER STRINGS AS PARAMETER VALUES 1-9
1.1.2.2 Ranges of Parameter Values 1-10
1.1.2.3 Lists of Values for Parameters 1-11
1.1.2.4 Parameter Lists as Parameter Values 1-13
1.1.2.5 Parameter Keywords as Procedure Options 1-13
1.1.3 USING MORE THAN ONE SES PROCEDURE AT A TIME 1-14
1.1.4 ORDINARY CONTROL STATEMENTS ON A CALL TO SES 1-14
1.1.5 CONTINUING SES STATEMENTS OVER MORE THAN ONE LINE . 1-15
1.1.6 PROFILE - SETTING UP YOUR LOCAL ENVIRONMENT 1-16
1.1.7 SES PROCEDURES RUN AS BATCH JOBS 1-18
1.1.7.1 PROFILE Variables for Controlling Batch Mode . . 1-22
1.1.8 INFORMATIVE MESSAGES FROM SES PROCEDURES 1-24
1.1.9 GENERAL NOTES ON SES PROCEDURES 1-25
ACQUIRING Files in SES Procedures 1-26
Using SES Procedures in Batch 1-27
Unique Names in SES procedures 1-27
1.1.10 PROCEDURE DESCRIPTION CONVENTIONS 1-29
1.1.11 PARAMETERS TO THE SES PROCESSOR 1-30
1.1.11.1 UN 1-30
1.1.11.2 LPFN 1-30
1.1.11.3 PUN 1-30
1.1.11.4 P 1-30
1.1.11.5 DUN 1-31
1.1.11.6 DPN 1-31
1.1.11.7 Processor Operating Mode Parameter 1-31
1.1.11.7.1 RUN 1-32
1.1.11.7.2 TEST 1-32
1.1.11.7.3 HELP OR H 1-33
1.1.11.7.4 STATUS OR S 1-34
1.1.12 PRE-RELEASE VERSIONS OF TOOLS 1-35
1.1.13 PROBLEM REPORTING 1-36
1.1.14 APPLICABLE DOCUMENTS 1-37

2.0 SES PROCEDURE LIBRARY MANAGEMENT 2-1
Summary Of SES Procedure Library Management Procedures . 2-2
Parameter Naming Convention for SES Procedure Library Management 2-3
PROFILE Variables for SES Procedure Library Management . 2-4
SES Procedure Library Updating Process 2-5
Interlock Process for Updating a Library 2-6
2.1 GETPROC - EXTRACT PROCEDURE(S) FROM PROCEDURE LIBRARY . 2-8
2.2 REPPROC - ADD OR REPLACE PROCEDURE(S) IN PROC. LIB. . 2-10
Creating a New Library with REPPROC 2-12

2.3 CATPLIB - PRODUCE LIST OF PROCEDURES IN A PROC. LIB. . 2-13
2.4 LISTPROC - LIST CONTENTS OF PROCEDURE LIBRARY 2-15
2.5 WIPEPROC - DELETE PROCEDURE(S) FROM PROCEDURE LIBRARY . 2-17

3.0 PRINTING FILES 3-1
3.1 PRINT - PRINT FILE(S) 3-2
PROFILE Variables for PRINT 3-6
3.2 PRINTID - PRINT "LARGE PRINT" HEADING BANNERS 3-8
PROFILE Variables for PRINTID 3-10
3.3 COPYSAF - COPY SHIFTED ASCII FILE TO PREPARE FOR PRINTER 3-11
3.4 BANNER - WRITE BIN NUMBER ON LARGE PRINT BANNER PAGE . . 3-14
PROFILE Variables for BANNER 3-15
3.5 FICHE - SET UP LIST FILE FOR MICROFICHE PROCESSING . . . 3-16

4.0 DOCUMENT FORMATTING SYSTEM 4-1
4.1 FORMAT - RUN DOCUMENT FORMATTING SYSTEM 4-3
Data Flow of FORMAT Procedures 4-8
4.2 FORMREV - FORMAT A REVISION PACKAGE FROM DOCUMENT SOURCE 4-11
Data flow of FORMREV Procedures 4-14
4.3 SPELL - CHECK FILE FOR SPELLING MISTAKES 4-17
4.4 GRADLVL - DETERMINES READING LEVEL OF A DOCUMENT 4-19
4.5 TWOPAGE - PRINT TWO DOCUMENT PAGES SIDE BY SIDE 4-21
4.6 DIAGRAM - DIAGRAM DRAWING AID 4-23
4.7 MEMO - GENERATE STANDARD MEMO HEADER 4-25
4.8 TXTCODE - RUN TXTCODE DOCUMENT PREPROCESSOR 4-28
4.9 TXTFORM - RUN TXTFORM DOCUMENT PROCESSOR 4-29
4.10 TXTHEAD - PAGE HEADING PROCESSOR 4-30
4.11 GENREVB - GENERATE REVISION BARS FOR DOCUMENTS 4-31
4.12 GENREVP - GENERATE A REVISION PACKAGE FOR A DOCUMENT . 4-33

5.0 SOURCE TEXT MAINTENANCE 5-1
Summary Of Source Text Maintenance Procedures 5-2
Data flow of the source text maintenance procedures . . 5-4
Parameter Naming Convention for Source Text Maintenance 5-5
PROFILE Variables for Source Text Maintenance 5-6
Base Updating Process 5-8
Interlock Process for Updating a Base or Library 5-9
5.1 GETMOD OR GETMODS - EXTRACT MODULE GROUP FROM BASE . . . 5-11
Data flow of GETMOD(S) usage 5-14
5.2 REPMOD OR REPMODS - ADD OR REPLACE MODULES 5-15
Creating a New Base with REPMOD(S) 5-18
Data flow of REPMOD(S) process. 5-19
5.3 COLLECT - COLLECT MODULE(S) TO BUILD A GROUP FILE . . . 5-20
Data flow of COLLECT processing 5-22
5.4 GENCOMP - GENERATE COMPILE FILE FOR MODULE(S) 5-23
Data flow of GENCOMP usage 5-28
5.5 GENCOR OR GENCORS - GENERATE CORRECTION SETS 5-29
Data flow of GENCOR(S) 5-31
5.6 TEMPCOR - MAKE TEMPORARY CORRECTIONS TO BASE 5-32
5.7 MODIFY - UPDATE BASE WITH CORRECTION SET(S) 5-34
5.8 CATBASE - PRODUCE LIST OF MODULES IN A BASE LIBRARY . . 5-37
5.9 LISTMOD - LIST CONTENTS OF BASE 5-39
5.10 SORTMOD - SORT BASE INTO ALPHABETICAL ORDER 5-41
5.11 WIPEMOD - DELETE MODULE(S) FROM BASE 5-43

5.12 XREFMOD - CROSS REFERENCE OF A BASE 5-45
5.13 GETCOMN - ACQUIRE CYBIL COMMON DECK LIBRARY 5-49
5.14 SYSTEMX - GLOBALLY CROSS REFERENCES A CYBIL SYSTEM . . 5-51
5.15 SCOOP - COMPARES MADIFY OR UPDATE SOURCE FILES 5-62

6.0 LIBRARY OR MULTI RECORD FILE MANAGEMENT USING LIBEDIT . 6-1
Summary Of Library Management Procedures 6-2
Parameter Naming Convention for Library Management . . . 6-3
PROFILE Variables for Library Management 6-4
Library Updating Process 6-5
Interlock Process for Updating a Library 6-6
6.1 GETMEM OR GETMEMS - EXTRACT MEMBER(S) FROM LIBRARY . . . 6-8
6.2 REPMEM OR REPMEMS - ADD OR REPLACE MEMBER(S) ON LIBRARY 6-10
Creating a New Library with REPMEM(S) 6-12
6.3 REPULIB - ADD OR REPLACE MEMBER(S) ON USER LIBRARY . . . 6-13
6.4 COLLECT - COLLECT MEMBER(S) TO BUILD A GROUP FILE . . . 6-16
6.5 CATLIB - PRODUCE LIST OF MEMBERS IN A LIBRARY 6-18
6.6 LISTMEM - LIST CONTENTS OF LIBRARY 6-20
6.7 SORTMEM - SORT LIBRARY INTO ALPHABETICAL ORDER 6-22
6.8 SRTULIB - SORT USER LIBRARY INTO ALPHABETICAL ORDER . . 6-25
6.9 WIPEMEM - DELETE MEMBER(S) FROM LIBRARY 6-27
6.10 WIPULIB - DELETE MEMBER(S) FROM USER LIBRARY 6-30
6.11 LIBEDIT - RUN LIBEDIT UTILITY 6-32

7.0 GETTING INFORMATION 7-1
7.1 CATLIST - DISPLAY PERMANENT FILE INFORMATION 7-3
7.2 DAYFILE - DISPLAY SELECTED PORTIONS OF DAYFILE 7-5
7.3 CATALOG - SHOW LIST OF RECORDS IN A FILE 7-6
7.4 FILES - DISPLAY LOCAL FILE INFORMATION 7-8
7.5 PERMIT - OBTAIN FULL LIST OF PERMISSION INFORMATION . . 7-9
7.6 LIMITS - DISPLAY VALIDATION LIMITS 7-10
7.7 TIME - DISPLAY CURRENT TIME OF DAY 7-11
7.8 DISPLAY - DISPLAY VARIOUS USEFUL INFORMATION 7-12
7.9 DAYWEEK - DISPLAY DAY OF THE WEEK 7-14
7.10 EXPLAIN - DISPLAY ONLINE MANUALS 7-15
7.11 CONVOLM - CONVERT ONLINE MANUAL SOURCE TO NOS/VE FORMAT 7-17
7.12 TOOLDOC - PRINT TOOL DOCUMENT 7-18
7.13 INFO - ACCESS SES INFORMATION 7-20
7.14 USSINFO - ACCESS USS INFORMATION 7-21
7.15 SESPROC - LIST SES PROCEDURE NAMES 7-22
7.16 SESPARM - PRINT PARAMETER REQUIREMENTS FOR SES PROCS . 7-23

8.0 FILESPACE MANAGEMENT 8-1
8.1 RETAIN - ACCESS ALL FILES IN USER'S CATALOG 8-2
8.2 DUMPPF / LOADPF - DUMP / LOAD PERMANENT FILES 8-3
8.3 REWRITE - REWRITE FILE 8-6

9.0 COMPILING, LINKING, AND DEBUGGING 9-1
9.1 CYBIL - RUN CYBIL COMPILER 9-3
9.2 ISWL - RUN ISWL CC COMPILER 9-7
9.3 SYMPL - RUN THE SYMPL COMPILER 9-9
9.4 FTN - RUN THE FTN (FORTRAN 4) COMPILER 9-11
9.5 FTN5 - RUN THE FTN5 (FORTRAN-5) COMPILER 9-13
9.6 COBOL5 - RUN THE COBOL5 COMPILER 9-15

9.7 COMPASS - RUN THE COMPASS ASSEMBLER 9-17
9.8 ASM180 - EXECUTE THE 180 ASSEMBLER ON 170 9-19
9.9 CPAS180 - RUN THE CPU ASSEMBLER FOR THE CYBER 180 . . . 9-21
9.10 PPAS180 - RUN THE PPU ASSEMBLER FOR THE CYBER 180 . . . 9-23
9.11 LINK170 - LINK RELOCATABLE BINARIES 9-25
9.12 XREFMAP - GENERATE A CROSS_REFERENCE FROM A LOAD MAP . 9-29
9.13 GETCCDB - GET CYBIL INTERACTIVE DEBUG 9-30
9.14 GETLIB OR GETLIBS - ACQUIRE LIBRARY FOR LINKING 9-31
9.15 CYBENV - RUN CYBIL ENVIRONMENT 9-32

10.0 CYBER 180 VIRTUAL ENVIRONMENT CREATION AND SIMULATION . 10-1
10.1 SIM180 - RUN THE CYBER 180 HARDWARE SYSTEM SIMULATOR . 10-3
10.2 TO180 - NOS/170 TO SIMULATED NOS/VE FILE CONVERSION . . 10-6
10.3 TO170 - NOS/170 TO NOS/170 INTERFACE FILE CONVERSION . 10-7
10.4 FROM180 - SIMULATED NOS/VE TO NOS/170 FILE CONVERSION . 10-8
10.5 FROM170 - NOS/170 INTERFACE TO NOS/170 FILE CONVERSION 10-9
10.6 DUMP180 - SIMULATED NOS/VE FILE DUMP 10-10
10.7 DUMP170 - NOS/170 INTERFACE FILE DUMP 10-11
10.8 GENCPF - GENERATE A CHECKPOINT FILE (CPF) 10-12
10.9 VELINK - EXECUTE THE VIRTUAL ENVIRONMENT LINKER 10-15
10.10 VEGEN - EXECUTE THE SES VIRTUAL ENVIRONMENT GENERATOR 10-18
10.11 GETLIB OR GETLIBS - ACQUIRE LIBRARY FOR LINKING . . . 10-20
10.12 GETDSI - ACQUIRE BINARY FOR DEADSTART TAPE GENERATOR . 10-21
10.13 GETLDB - ACQUIRE BINARY FOR ENVIR. INT. LOADER . . . 10-22

11.0 MOTOROLA 68000 ENVIRONMENT 11-1
11.1 ASM68K - EXECUTE THE MOTOROLA 68000 ASSEMBLER ON 170 . 11-2
11.2 LINK68K - EXECUTE THE MC68000 ABSOLUTE LINKER 11-4
11.3 BIND68K - BIND MC68000 MODULES 11-7
11.4 TRAN68K - GENERATE S RECORDS 11-10
11.5 REFORM68K - REFORMAT MODULE TO H-P OBJECT TEXT FORMAT . 11-11
11.6 BLDMI68K - MEMORY IMAGE BUILDER 11-13
11.7 GENGS68K - TRANSLATES GLOBAL SYM. FILE INTO H-P FORMAT 11-14
11.8 PURSYM68K - PURGE SYMBOL TABLES 11-15
11.9 TRANDILIB - TRANSLATE DI LIBRARIES 11-16

12.0 UCSD PCODE ENVIRONMENT 12-1
12.1 REFORMP - REFORMAT MOD TO UCSD PCODE OBJ TEXT FORMAT . 12-2

13.0 SES OBJECT CODE UTILITIES 13-1
Parameter Naming Convention for Object Code Utilities . 13-2
PROFILE Variables for Object Code Utilities 13-3
Interlock Process for Updating a Library 13-4
13.1 COM - CHANGE OBJECT MODULE 13-6
13.2 DEOM - DELETE OBJECT MODULE(S) 13-9
13.3 DIOM - DISPLAY OBJECT MODULE INFORMATION 13-12
13.4 GOL - GENERATE OBJECT LIBRARY 13-14
13.5 GOF - GENERATE OBJECT FILE 13-18
13.6 OBJLIST - LIST CDC OBJECT TEXT 13-23

14.0 SOURCE CODE UTILITY ON NOS 170 14-1
14.1 SCU - ACQUIRE EXECUTABLE BINARY FOR SCU 14-2
14.2 SOLO - STAND ALONE SCU EDITOR 14-3
14.3 SCUCOMP - GENERATE SCU CORRECTION SET 14-4

14.4 EDSCU - EDIT A DECK FROM AN SCU LIBRARY 14-6

15.0 SOURCE TEXT PREPROCESSORS 15-1
15.1 CYBFORM - CYBIL SOURCE TEXT REFORMATTER 15-2
15.2 ISWLFRM - ISWL SOURCE TEXT REFORMATTER 15-3
15.3 PSEUDO - RUN PSEUDO PREPROCESSOR 15-5
15.4 F5FORM - FORTRAN 5 SOURCE TEXT REFORMATTER 15-6
15.5 PASTOCYB - PASCAL TO CYBIL TRANSLATOR 15-7

16.0 SES COMMUNICATIONS 16-1
User to User Communications 16-1
Remote Mainframe Communications 16-3
16.1 SEND - TRANSMIT FILES BETWEEN NOS SITES 16-5
16.2 MAIL - SEND MAIL TO OTHER USERS 16-7
MAIL Addressees who do not have a MAILBOX 16-7
16.3 GETMAIL - DISPLAY MAIL FROM MAILBOX 16-9
16.4 NEWMAIL - CREATE A NEW MAILBOX / CLEAR EXISTING MAILBOX 16-10
16.5 ANYMAIL - COUNT NUMBER OF ITEMS IN MAILBOX 16-11
16.6 WHOMAIL - DISPLAY LIST OF USERS WHO HAVE SENT MAIL . . 16-12
16.7 SAVMAIL - SAVE MAILBOX 16-13
16.8 CHKMAIL - CHECK YOUR MAILBOX 16-14
16.9 FEEDBACK - SES USER FEEDBACK MECHANISM 16-15
16.10 SUBMIT - SUBMIT A JOB TO BE PROCESSED AT A REMOTE SITE 16-16
16.11 GETPF - GET A PERMANENT FILE FROM A REMOTE SITE . . . 16-17
16.12 SENDPF - SEND A PERMANENT FILE TO A REMOTE SITE . . . 16-18
16.13 CHANGPF - CHANGE PARAMETERS OF A PERMANENT FILE . . . 16-19
16.14 PURGPF - REMOVE (PURGE) A PERMANENT FILE 16-20
16.15 PERMPF - PERMIT ACCESS TO A PERMANENT FILE 16-21
16.16 CATPF - DISPLAYS INFORMATION ABOUT A FILE 16-22
16.17 DISPOSE - PRINT A FILE AT A REMOTE SITE 16-23
Alphabetical Summary of Remote Link Procedures 16-24

17.0 TEXT MANIPULATION AND CONVERSION UTILITIES 17-1
17.1 EDT - RUN THE EDT TEXT EDITOR 17-3
17.2 LOWTOUP - CONVERT LOWER CASE TO UPPER CASE 17-4
17.3 UPTOLOW - CONVERT UPPER CASE TO LOWER CASE 17-5
17.4 COPYACR - COPY ASCII CODED RECORD(S) 17-6
17.5 ASORT - SORT ASCII FILES 17-8
17.6 UNIQUE - REMOVE ADJACENT DUPLICATE LINES FROM A FILE . 17-11
17.7 MERGE - MERGE UP TO FIVE FILES INTO ONE FILE 17-13
17.8 DEMERGE - SPLIT FILE APART BY COLUMNS 17-14
17.9 COMPARE - COMPARE TEXT FILES 17-15
17.10 COUNT - COUNT THINGS IN A FILE 17-17
17.11 SELECT - COPY SELECTED LINE RANGES FROM A FILE 17-18
17.12 MULTED - EDIT MULTI RECORD FILE 17-20
17.13 PACK - PACK MULTI RECORD FILE 17-22
17.14 UNPACK - UNPACK TEXT FILE TO MULTI RECORD FILE 17-24
17.15 CONV - CONVERT CHARACTER SET 17-26
17.16 JABFORM - CONVERT TEXTJAB SOURCE TO TXTFORM SOURCE . . 17-28
17.17 WANG2CYBER - SEND TEXT FROM WANG DOC. TO CYBER FILE . 17-29
17.18 CYBER2WANG - SEND TEXT FROM CYBER FILE TO WANG DOC. . 17-30
17.19 SNOBOL - 8-BIT SNOBOL INTERPRETER 17-31
17.20 FIND - FIND PATTERNS IN A FILE 17-32
Text Patterns and Regular Expressions 17-33

Summary of Regular Expressions 17-36
17.21 CHANGE - CHANGE LINES THAT MATCH SPECIFIED PATTERNS . 17-38
17.22 XLIT - TRANSLITERATE CHARACTERS 17-40
Summary of XLIT Capabilities 17-41

18.0 MISCELLANEOUS USEFUL GOODIES 18-1
18.1 IAF - SET UP INITIAL IAF TERMINAL PARAMETERS 18-2
18.2 AUTOMOD - AUTOMATIC MODIFY TO SCU NAME CHANGE GENERATOR 18-7
18.3 DO - CONTROL STATEMENT GENERATOR 18-8
18.4 MATH - INTERACTIVE KEYBOARD CALCULATOR 18-10
18.5 BELL - ATTENTION GETTER OR WAKE UP 18-11
18.6 BYE - LOG OUT 18-12
18.7 HEXDMP - GENERATE HEXADECIMAL DUMP 18-13
18.8 CONCAT - CONCATENATE FILES 18-14
18.9 SCATTER - SCATTER A MULTI RECORD FILE TO OTHER FILES . 18-15

19.0 HANDLING UPDATE PROGRAM LIBRARIES 19-1
PROFILE Variable for UPDATE Facilities 19-2
Working with the UPDATE Procedures 19-3
19.1 GENUPCF - GENERATE UPDATE COMPILE FILE 19-6
19.2 GETDECK - GET DECK(S) FOR EDITING 19-8
19.3 GENMOD(S) - GENERATE MODSETS FOR UPDATE 19-10
19.4 GENUPSF - GENERATE UPDATE SOURCE FILE 19-12
19.5 UPDATE - UPDATE PROGRAM LIBRARY WITH CORRECTION SET(S) 19-14

20.0 STRUCTURED PROCESS TOOLS 20-1
SCAD - SOFTWARE COMPUTER AIDED DESIGN 20-1
20.1 SCADG - CREATE OR UPDATE SASD DIAGRAMS 20-4
20.2 SCADGEDIT - DIAGRAM HEADER EDIT & LIST, NOTEBOOK UTIL. 20-7
20.3 DDCREATE - CREATE A DESIGN DICTIONARY 20-9
20.4 DDDISPLAY - DISPLAY A DESIGN DICTIONARY 20-10
20.5 DDMERGE - MERGE TWO DESIGN DICTIONARIES 20-12
20.6 DDXREF - CROSS REFERENCE A DESIGN DICTIONARY 20-14
20.7 DDXCHK - CHECK A DESIGN DICTIONARY FOR CONSISTENCY . . 20-16
20.8 DDHREF - HIERARCHICAL REF EXPANSIONS FOR A DESIGN DICT 20-18
20.9 DDEDIT - EDIT A DESIGN DICTIONARY 20-20
20.10 CYTOSCT - CYBIL TO STRUCTURE CHARTS 20-22
20.11 CYTODD - CYBIL TO DESIGN DICTIONARY 20-25
20.12 DDTOSCT - DESIGN DICTIONARY TO STRUCTURE CHARTS . . . 20-27
20.13 FTNTOSCT - FORTRAN TO STRUCTURE CHARTS 20-29
20.14 FTNTODD - FORTRAN TO DESIGN DICTIONARY 20-32

APPENDIX A PROFILES A-1

A1.0 PROFILES A1-1
A1.0.1 SEARCH DIRECTIVE - ESTABLISH LIBRARY SEARCH ORDER . A1-2
A1.1 LOCATING A PROCEDURE A1-3
A1.1.1 DEFAULT ORDER OF SEARCH A1-4
A1.1.2 SEARCH SPECIFIED ON CONTROL STATEMENT A1-4
A1.1.3 SEARCH ORDER SPECIFIED VIA SEARCH DIRECTIVES . . . A1-5

APPENDIX B Operating Modes of the SES Processor B-1

B1.0 OPERATING MODES OF THE SES PROCESSOR B1-1

B1.1 SELECTING MODE OF OPERATION B1-2

APPENDIX C Error and Informative Messages C-1

C1.0 ERROR AND INFORMATIVE MESSAGES C1-1

APPENDIX D SYNTAX D-1

D1.0 SEMI-FORMAL SYNTAX DESCRIPTION D1-1
D1.1 THE META LANGUAGE D1-1
D1.2 CHARACTER SET D1-3
D1.3 SYNTAX D1-5
D1.3.1 BASIC DEFINITIONS D1-5
D1.3.2 TOKENS D1-5
D1.3.3 USE OF SPACES D1-7
D1.3.4 EXPRESSIONS D1-8
D1.3.5 FOREIGN TEXT D1-9
D1.3.6 PARAMETER LISTS D1-11
D1.3.7 SES PROCESSOR CALL D1-12
D1.3.8 SUBSTITUTION D1-13
D1.3.9 PROCEDURES D1-14
D1.3.10 DIRECTIVES D1-15
D1.4 LINES AND THEIR CONTINUATION D1-16

APPENDIX E ACQUIRE Utility E-1

E1.0 ACQUIRE UTILITY E1-1

APPENDIX F EDT - Enhanced Version of NOS Text EDITor . . F-1

F1.0 EDT - ENHANCED VERSION OF NOS TEXT EDITOR F1-1
F1.1 EDT CONTROL STATEMENT FORMAT F1-1

APPENDIX G EXTRACT Utility G-1

G1.0 EXTRACT UTILITY G1-1

APPENDIX H SESMSG Utility H-1

H1.0 SESMSG UTILITY H1-1

APPENDIX I JOBID - Produce Large Print Banner I-1

I1.0 JOBID - PRODUCE LARGE PRINT BANNER I1-1

APPENDIX J TXTHEAD Document (Headings) Post-Processor . . J-1

J1.0 TXTHEAD DOCUMENT (HEADINGS) POST-PROCESSOR J1-1

APPENDIX K JABFORM Conversions K-1

K1.0 JABFORM CONVERSIONS K1-1

APPENDIX L Character Set Table (used by CONV) L-1

L1.0 CHARACTER SET TABLE (USED BY CONV) L1-1

APPENDIX M Differences Between MODIFY and MADIFY M-1

M1.0 DIFFERENCES BETWEEN MODIFY AND MADIFY M1-1

APPENDIX N User Supplied Software N-1

N1.0 MECHANISM FOR COLLECTING USER SUPPLIED SOFTWARE N1-1
N1.1 OBTAINING USER SOFTWARE N1-1
N1.2 MAKING USER SOFTWARE AVAILABLE THROUGH SES N1-1
N1.3 COMMUNICATING AVAILABILITY OF USER SUPPLIED SOFTWARE . N1-2
N1.4 SUPPORT OF USER SUPPLIED SOFTWARE N1-2