ARH2922

EXTERNAL REFERENCE SPECIFICATION

for

SES OBJECT CODE UTILITIES

REVISION DEFINITION SHEET

REV	DATE	DESCRIPTION
1	12/13/83	Preliminary manual released.

1.0 PREFACE

1.1 INTRODUCTION

The Object Code Utilities are a collection of commands which can be 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 CDC object text format V1.4 generated by cross compilers and cross assemblers executing on CYBER 170. They run as stand-alone commands via the SES Processor.

1.2 APPLICABLE DOCUMENTS

The following is a list of documents that either are referred to in this specification, or are recommended to aid in understanding and using these utilities.

SES User's Handbook (60457250)

2.0 OBJECT CODE UTILITY COMMANDS

The Object Code Utilities use System Control Language (SCL) syntax as the parameter interface. In the descriptions that follow, optional parameters are enclosed in brackets, and all parameters of type 'name' can have a maximum length of 31 characters, except for parameters designated as file names (NOS file names cannot exceed 7 characters in length).

If the name of the generated file is the same as an input file name, the new file is generated on a scratch file and then copied over the old file when the generation is complete.

In all of the commands, the input parameter keywords (file ! library and base ! baselib) are set up such that if no keyword is specified or if the 'file' or 'base' keyword is specified, the input can be a collection of object modules formatted into a Library, or just a file containing one or more object modules. If, however, the 'library' or 'baselib' keyword is specified, the input must be formatted as a Library, or a diagnostic is issued. The output is determined by each particular command.

Most of the parameters described do not have defaults. When a parameter does have a default, the default value is documented in the parameter description.

The commands are described in alphabetical order.

2.1 COM_!_CHANGE_OBJECT_MODULE

This command allows the user to alter various characteristics of a module in the specified file.

```
com library=<local_file_name>
       module=<name>
       [new_name=<name>]
       [substitute=((<name>,<name>)[,(<name>,<name>)]...)]
       [omit=(<name>[,<name>]...)]
       [gate=(<name>[,<name>]...)]
       [not_gate=(<name>[,<name>]...)]
       [procedure=<name>]
       [comment=<string>]
       upon=<local_file_name>
```

library ! lib ! file ! f :
This parameter specifies the name of the local file containing the module to be changed. 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.

module ! mo :
This parameter specifies the name of the module in the library to be changed.

new_name ! nn :
This parameter specifies a name which replaces the name of the specified module.

substitute ! s :
This parameter specifies 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 ! o :
This parameter specifies entry point(s) whose definitions are to be removed from the output module.

gate ! g :
This parameter specifies 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.

2.0 OBJECT CODE UTILITY COMMANDS
2.1 COM ; CHANGE OBJECT MODULE

not_gate ; ng :
This parameter specifies entry point(s) for which the gated attribute is to be removed.

procedure ; pro :
This parameter specifies the entry point at which execution is to begin (transfer symbol).

comment ; co :
This parameter specifies the contents of the commentary field in the module header.

upon ; up :
This parameter specifies the name of the local file containing the new file. Whether the upon file is a library is determined by the format of the input file.

Examples:

```
SES.COM LIBRARY=LIB1 MODULE=MODX NEW_NAME=MODY UPON=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
```
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'.

```
SES.COM LIB1 MO=MODULEX D=ENT1 G=ENT2 NG=ENT3 PRO=ENT4 ..
  CO='CPU MIGDS REVIEW' UP=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.

2.2 DEOM ¦ DELETE OBJECT MODULE

This command allows the user to delete a module or range of modules from a specified file.

```
deom library=<local_file_name>
       module=(<name>[..<name>][,<name>[..<name>]]...)
       upon=<local_file_name>
```

library ¦ lib ¦ file ¦ f :
This parameter specifies the name of the local file containing the module(s) or module subrange(s) to be deleted. 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.

module ¦ mo :
This parameter specifies the 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 ¦ up :
This parameter specifies the name of the local file upon which the remaining modules are written. Whether the upon file is a library is determined by the format of the input file.

Examples:

```
SES.DEOM LIBRARY=LIB1 MODULE=MODX UPON=LIB2
```
This command creates a library LIB2 identical to LIB1 except that module MODX is deleted.

```
SES.DEOM FILE=FILE1 MO=(MODA..MODB,MODC) UP=FILE2
```
This command creates a file FILE2 (will be library if FILE1 is library) identical to FILE1 except that the modules between MODA and MODB inclusive and MODC are deleted.

2.3 DIOM ! DISPLAY OBJECT MODULE

This command allows the user to display information about all or part of the contents of a object file or library. The format of the list file produced is described in a later section of this document entitled "LISTING FILES".

```
diom library=<local_file_name>
    [module=(<name>[..<name>][,<name>[..<name>]]...)]
    [listing=<local_file_name>]
    [on=(<display_option>[,<display_option>]...)]
```

library ! lib ! file ! f :
This parameter specifies the name of the local file whose contents are to be displayed. The specified file may or may not be a library, but if the keyword used is 'lib' or 'library', it must be a library.

module ! mo :
This parameter specifies the 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 ! list :
This parameter specifies the local file name of the file on which the display information is to be written. Omission causes the information to be written to the job output file.

on :
This parameter specifies the 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)

2.0 OBJECT CODE UTILITY COMMANDS
2.3 DIOM | DISPLAY OBJECT MODULE

Examples:

SES.DIOM LIBRARY=LIB1 MODULE=MODX LIST=LISTX ON=A
This command lists all information about module MODX of library LIB1 on file LISTX.

SES.DIOM FILE=FILEX MO=(MODA..MODB,MODC) ON=(D,H,X)
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).

2.4 GOF ! GENERATE OBJECT FILE

This command allows a user to generate or update an object file. Also, several object files or libraries can be combined into one object file. Parameters for this command are described under 'GOL - Generate Object Library' which follows below.

2.5 GOL ! GENERATE OBJECT LIBRARY

This command allows a user to generate or update a object library. Also, several object files or libraries can be combined into one object library.

Description for calling either the 'GOF' or 'GOL' commands follows below:

```
gof ! gol
        [file=(<local_file_name>[,<local_file_name>]...)]
        [combine=(<name>[..<name>][,<name>[..<name>]]...)]
        [add=(<name>[..<name>][,<name>[..<name>]]...)]
        [replace=(<name>[..<name>][,<name>)..<name>]]...)]
        [after=<name>]
        [base=(<local_file_name>[,<local_file_name>]...)]
        upon=<local_file_name>
        [listing=<local_file_name>]
```

file ! f ! library ! lib :
This parameter specifies the names of the files from which object modules specified by the following three parameters are to be obtained for the new 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).

The next three parameters ('combine', 'add', and 'replace') specify module(s) or module subrange(s) to be included on the new file from the files specified by the 'file/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/library' parameter will be included on the new file. Only the first occurrance of duplicate modules are included in the new file. As explained below, the particular parameter(s) used give the user

control over whether the given modules should exist on the 'base' library(s).

combine ! co :
This parameter specifies module(s) to be included in the new file. The specified module(s) may or may not exist on the 'base' file(s). If they exist, they are replaced.

add :
This parameter specifies module(s) to be added on the new file. If a specified module duplicates a module already on a file specified by the 'base' parameter, a fatal error is issued.

replace ! rep :
This parameter specifies module(s) to be replaced on the new file. If a specified module does not already exist in a file specified by the 'base' parameter, a fatal error is issued.

after ! af ! before ! be :
This parameter specifies a module on a base file after or before which to position the new modules. Default position is after the last module on the last base library.

base ! b ! baselib ! bl :
This parameter specifies a list of files to be included in the new file. All modules from the base files become part of the new file except duplicate modules (the first occurrance of the module takes precedence). The specified file(s) may or may not be library(s), but if the keyword used is 'bl' or 'baselib', they must be librery(s).

upon ! up :
This parameter specifies the local file name of the new file. If the filename specified duplicates an existing local file, the file is generated and then copied to the specified file.

listing ! list :
This parameter specifies the name of the listing file on which the names of the modules on the new file 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. The format of this file is described in a later section of

this document entitled "LISTING FILES".

The utility requires that there be an input file specified either by the 'file' parameter, or 'base' parameter. By utilizing the 'add', 'replace', and 'module' parameters, the user can control exactly which modules from the file(s) specified by the 'file' parameter he wants to include on the new file, while the 'after' and 'before' parameters allow him to position the new modules. The keyword specified for the 'upon' parameter controls whether the resulting new file is a library. For a more detailed listing of the modules on the library, the user can use the DIOM command.

Examples:

SES.GOL FILE=FILEX LIST=LISTX ULON=NEW
This command creates an object library NEWLIB which contains all the modules from file FILEX (may be library). Display information appears on the file LISTX.

SES.GOL LIB=INPLIB ADD=MOD1 REP=MOD2 BL=BASELIB UP=NEWLIB
This command creates an object library NEWLIB identical to library BASELIB except that module MOD2 from library INPLIB replaces MOD2 and module MOD1 from INPLIB is added at the end.

SES.GOF BASE=(FILE1,FILE2,FILE3) UPON=NEWFILE LISTING=LISTX
This command creates an object file (not library) NEWFILE which is the combination of files FILE1, FILE2 and FILE3. Only the first occurrance of duplicate modules appears on the new file. Display information appears on file LISTX.

SES.GOF FILEA,FILEB MO=(MODA..MODB,MODC) AFTER=MODX ..
BASE=BASEFILE UP=NEWFILE
This command creates an object file NEWFILE using file BASEFILE as a base. Modules MODA thru MODB inclusive and MODC from files FILEA and FILEB (may be libraries) are added after module MODX.

3.0 MESSAGES

The following messages are output at the termination of commands. All errors cause command to abort.

13001 FILE file_name NOT LOCAL
SEVERITY: Error
MEANING: Self expanatory.

13002 MISSING IDENTIFICATION RECORD IN FILE file_name
SEVERITY: Error
MEANING: Data at the beginning of a module does not match expected format.

13003 UNEXPECTED IDENTIFICATION RECORD IN FILE file_name
SEVERITY: Error
MEANING: Second identification record found before end of module.

13004 UNKNOWN OBJECT TEXT INFILE file_name
SEVERITY: Error
MEANING: Self explanatory.

13005 FILE file_name IS NOT LIBRARY
SEVERITY: Error
MEANING: Source or base file declared as library is not one.

13010 MODULE module_name NOT FOUND
SEVERITY: Error
MEANING: Self explanatory.

13011 DUPLICATE MODULE module_name DECLARED
SEVERITY: Error
MEANING: Module specified more than once.

13012 END OF MODULE RANGE module_name1 .. module_name2 NOT FOUND
SEVERITY: Error
MEANING: Self explanatory.

13100 GENERATION OF type file_name COMPLETE
SEVERITY: Informational

3.0 MESSAGES

MEANING: Generate command normal termination (type specifies if library).

13101 MODULE module_name LOCATION TO ADD NOT FOUND
SEVERITY: Error
MEANING: Module declared as position for new modules (before or after) not found on base file(s) for generate command.

13102 MODULE module_name TO ADD ALREADY EXISTS ON BASE file_name
SEVERITY: Error
MEANING: New module specified by 'ADD' parameter of generate command is already present on a base library.

13103 MODULE module_name TO BE REPLACED NOT FOUND
SEVERITY: Error
MEANING: New module specified by 'REPLACE' paremeter of generate command does dot exist on any of the base file(s).

13105 FILE file_name HAS THE SAME NAME AS A SOURCE FILE
SEVERITY: Error
MEANING: A base file has the same name as a source file.

13200 MODULES DELETED ON FILE file_name
SEVERITY: Informational
MEANING: Delete command normal terminate.

13201 ALL MODULES DELETED ON FILE file_name
SEVERITY: Warning
MEANING: 'UPON' file of this delete command contains no modules because they were all deleted.

13300 MODULES CHANGED ON FILE file_name
SEVERITY: Informational
MEANING Change command normal terminate

3.0 MESSAGES

13301 ENTRY POINT entry_point_name TO CHANGE NOT FOUND
SEVERITY: Error
MEANING: Entry point to substitute, omit, gate, or not_gate using change command not found. This message can also indicate that no entry point was found to match the new tranfer symbol (PROCEDURE parameter).

13400 MODULES FROM FILE file_name DISPLAYED
SEVERITY: Informational
MEANING: Display commend normal terminate.

4.0 LISTING FILES

4.0 LISTING_FILES_

The generate and display commands generate listings of the following formats.

4.1 GOE/GOL_LISTING_FORMAT

The Generate Object File Library commands produces an output file ('LIST' parameter) to show the disposition of modules from the source and base files in the order they are encountered. The format of the listing is as follows:

```
GENERATE_OBJECT_file/library                          PAGE xx

MODULE_NAME          FILE_NAME        STATUS
~~~~~~~~~~~          ~~~~~~~~~        ~~~~~~

module_name_1        file_name        ADDED
module_name_2        file_name        REPLACED
module_name_3        base_name
module_name_4        base_name        DELETED
  .                    .                .
  .                    .                .
  .                    .                .
```

In the listing, 'file/library' indicates whether a library was generated. The module_names show the modules on the new library (except if STATUS=DELETED). File_names indicated the source or base file the module came from. Status indicates the following:

-blank-	module copied from base library
ADDED	module added from source file
REPLACE	module from source file replaced one from a base library
DELETED	module from a base library was deleted since it duplicated one from a prevoius base library.

4.2 DIOM_LISTING_FORMAT

The Display Object Module command produces an output file (LIST parameter) to show the requested information about the specified modules. The format of the printout requesting all information (parameter ON=A) is shown below.

```
DISPLAY OF OBJECT_fil/lib - file_name dat/tim      PAGE xx

MODULE: mod_name    CREATED: date/time  KIND: mod_kind
GENERATOR NAME VERS: gen_name
COMMENTARY: com_text

ENTRY POINT DEFINITIONS
~~~~~~~~~~~~~~~~~~~~~~~
ep_name                    ...       (2 per line)
 .
 .
 .

EXTERNAL REFERENCES
~~~~~~~~~~~~~~~~~~~
ext_name                   ...       (2 per line)
 .
 .
 .
```

In the printout, fil/lib indicates whether the file is a library. The filename and date/time of the display appear in the page header. For each module:

mod_name	name of module being DISPLAYed
date/time	date and time created
mod_kind	module kind (MVS, VVS, IOU, MC68000, or P_CODE)
gen_name	generator name and version
com_text	commentary text from identification record
ep_name	name of entry point; If this entry point is gated, "GATED" precedes the name; Multiple entry points are allowed.
ext_name	name of external; Multiple externals are allowed.

Table of Contents