CDC - SOFTWARE ENGINEERING SERVICES

02/15/80

ERS for SES Virtual Environment LINKER REV: G

ARH2816

EXTERNAL REFERENCE SPECIFICATION

FOR

SES Virtual Environment LINKER

Submitted: _____

Approved: _____

_____

_____

DISCLAIMER:

This document is an internal working paper only. It is subject to change, and does not necessarily represent any official intent on the part of CDC.

REVISION DEFINITION SHEET

REV	DATE	DESCRIPTION
A	08/22/75	ERS (V1.0) ASL00365
B	10/04/76	ERS (V2.0)
C	11/15/76	ERS (V2.1)
D	02/06/78	Update for MIGDS Rev. M
E	10/02/78	ERS (V2.3)
F	05/29/79	(V2.4) Process V1.0 of Object Text
G	02/15/80	ERS (V2.5)

1.0 PREFACE

The Linker is a post compilation utility by which input object text is processed and written to various "hardware" segment files which can then be loaded on the SES CYBER 180 Simulator. The different object text records in a particular section are processed to form a set of hardware segment images. Data in the different segments is primarily referenced via pointers set up in the Binding segment which is initialized at Link time. Unsatisfied externals are satisfied from the specified libraries.

Users have the option of supplying the input via an object file (with/without library files) as generated by the compiler/assembler, or else (s)he may specify an entry point from a module on a specified library as the PEP parameter on the LINK command, and the Linker will extract the module from the library which contains the specified entry point and link it.

The input to the Linker consists of CYBER 180 (C180) object modules in CYBER 180 loader text format V1.1 generated by compilers and/or assemblers running on CYBER 170.

The output of the Linker is a series of files corresponding to CYBER 180 hardware segments; a header file which contains the segment descriptor attributes for each output segment; and potentially an outboard symbol table file.

Currently, the only mechanism available for taking Linker output and preparing it for execution in the CYBER 180 Simulator is the Virtual Environment Generator. It will take the segment files produced by the VE Linker, and build a Checkpoint File (CPF).

1.1 SCOPE

This document describes the external characteristics of the SES Virtual Environment Linker V2.5. The Linker is written in CYBIL.

This Linker is strictly intended as a transitional vehicle

1.0 PREFACE
1.1 SCOPE

from CYBER 170/SES to CYBER 180. It will not have a lifetime beyond the period of transition.

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 this specification.

1) SES User's Handbook (ARH 1833).
2) ERS for Virtual Environment Generator (ARH2591).
3) CYBER 180 Mainframe Model Independent GDS (ARH 1700).
4) ERS for Simulated NOS/VE I/O (ARH3125).
5) ERS for CYBER 180 Object Code Utilities (ARH 2922).
6) ERS for CYBER 180 Simulator (ARH 1729).

1.3 LINKER V2.5 AND LINKER V2.4 DIFFERENCES

1) The LINKER has been converted to CYBIL compiler language.

2) Updated to process V1.1 of the 180 Object Text.

3) Interface to the SES subsystem has been replaced with an internal linker parameter file.

1.4 DEFICIENCIES AND LIMITATIONS

The Linker performs only cursory checks to determine if the user has specified any duplicate file names in his/her parameterization.

The Linker performs no checks to determine if any file names generated from the NAME_SEED duplicate any file names the user has specified in his/her parameterization. Therefore, the Linker may abort or yield indeterminate results. Hence, the user must resolve any file naming conflicts prior to executing the LINKER command.

The Linker is sensitive to portions of the data on an object file. It uses some of the data for computations. Therefore, if an object file was generated by other than PASCAL-CI or the CYBER 180 Assembler; or modified with other than the CYBER 180 Object File Utilities, the Linker may abort or yield indeterminate results.

1.5 TERMINOLOGY

Offset linking: An optional mode of linking wherein the segments output by the Linker are to be loaded at addresses that are different than the addresses at which they will ultimately execute.

Inboard symbol table: A table of resolved entry points from a previous linkage provided as input to the current linkage. Use of the inboard symbol table allows the linkage of only a part of the code that will actually be present when the code is executed.

Outboard symbol table: A table of resolved entry points produced by the current linkage for inclusion in future linkages. Only entry points from modules possessing the "gated" attribute are included in the outboard symbol table. Use of the outboard symbol table allows entry definitions from the current linkage to be used in subsequent linkages without relinking the modules in which the entry points are defined.

2.0 ACCESSING THE LINKER

2.0 ACCESSING THE LINKER

The Linker is accessed via an SES procedure, which is described in a later section. The user can control the linkage by specifying his parameters on the procedure call, (GENCPF or VELINK), in the Linker Parameter File, or a combination of both. Parameters on the procedure call override the parameters specified in the Parameter File.

2.0 ACCESSING THE LINKER

LINKER PARAMETER FILE (LPF)

The Linker Parameter File (LPF) consists of a legible file built and maintained by an editor. LPF is an optional parameter to the LINK command. If one is not provided, defaults are as shown below. Several "subcommands" will be provided: LINK_OPTIONS, OBJECT_FILE, OBJECT_LIBRARY, DEFINE_SEGMENT, OBJECT_MODULE, INBOARD_SYMBOL_TABLE and END. Each subcommand is entered on a separate line; however, continuation lines of a subcommand are indicated by placing two or more periods at the end of the line.

LINK_OPTIONS

The LINK_OPTIONS command provides general miscellaneous parameters which establish defaults for the LINK command. Only one LINK_OPTIONS command is allowed.

```
LINK_OPTIONS,&MAP_FILENAME=filename,&..
MAP_OPTIONS=map_option,&REWIND_MAP,&NAME_SEED=name_seed,&..
MAX_EXTERNALS=maximum_externals,&HEAP_SIZE=heap_size,&..
PRIMARY_ENTRY_PT=primary_entry_point
```

FIELD DEFAULT	DESCRIPTION
MAP_FILENAME LINKMAP	(CHAR 7) The NOS filename of the LINKER output listing file on which the LINKER map will be written. This parameter is ignored if the MF (map file) parameter is specified on the LINK command.
MAP_OPTIONS M	(CHAR 1) The Linker map options which control the amount of information output on the Linker map. Values for this field are: N - no map information; diagnostics are output. S - section allocations for every section of every input object module

2.0 ACCESSING THE LINKER

	E - Section allocations plus entry point names and address assignments M - Section allocations, entry points plus output segment and common block allocations (full Linker map).
REWIND_MAP REWIND_MAP	(KEYWORD) The option allowing you to specify whether to rewind the LINKER map file before it is written. REWIND_MAP - rewind map file NO_MAP_REWIND - don't rewind map file
NAME_SEED SEGM	(CHAR 4) Used as the first four characters of of the NOS file names for the header file, output segments, and outboard symbol table file. This field is ignored if the name_seed parameter is specified on the VELINK command.
MAX_EXTERNALS 300	(INTEGER) The maximum number of externals allowed in this link.
HEAPSIZE 0	(INTEGER) The size in bytes of the system heap. IF specified, this value always takes precedence over the values specified in the input object modules.
LOAD_SEGMENT 4095	(INTEGER) The starting segment number for load load address. The linker allocates segment numbers consecutively.
EXECUTE_SEGMENT 4095	(INTEGER) The starting segment number for execution_address. The Linker allocates segment numbers consecutively. If either but not both LOAD_SEGMENT and EXECUTE_SEGMENT are specified, offset loading will not be performed. A null specification for LOAD_SEGMENT or EXECUTE_SEGMENT is indicated by a value of 4095.
PRIMARY_ENTRY_PT blanks	(CHAR 31) Primary entry point of the program being linked. If specified, it overides all transfer symbols encountered in the input object modules. This parameter is ignored if the PEP parameter is used on the LINK command. If not specified here or on the command, the first transfer symbol encountered is the primary entry point.

2.0 ACCESSING THE LINKER

OBJECT_FILE/OBJECT_LIBRARY

One OBJECT_FILE command exists for each specified object file entry, and one OBJECT_LIBRARY command exists for each specified library entry. VELINK command parameters "OFL" and "LFL" take precedence over these commands.

```
OBJECT_FILE, FILENAME=filename, R1=ring, R2=ring, ..
R3=ring, GLOBAL_LOCAL_KEY=key, EXECUTE_PRIVILEGE=attribute

OBJECT_LIBRARY, FILENAME=filename, R1=ring, R2=ring, ..
R3=ring, GLOBAL_LOCAL_KEY=key, EXECUTE_PRIVILEGE=attribute
```

FIELD	DEFAULT	DESCRIPTION
FILENAME		(CHAR 7) NOS file that name of a local file that
	none	contains object modules.
R1	11	(INTEGER) The ring bracket to be used for all
R2	11	object modules contained on this file.
R3	11	
GLOBAL_LOCAL_KEY		(INTEGER) The global and local key values
	0	to be used for all object modules contained on this file.
EXECUTE_PRIVILEGE		(CHAR 1) The type of execute to be attribute to be
	E	associated with all object modules contained on this file. Values for this field are: E - executable - non privileged; L - executable - local privilege; G - executable - global privilege

2.0 ACCESSING THE LINKER

DEFINE_SEGMENT

One DEFINE_SEGMENT entry exists for each Preallocated Segment Descriptor. Either LOAD_ADDRESS or EXECUTE_ADDRESS and ATTRIBUTES must be specified.

```
DEFINE_SEGMENT, LOAD_ADDRESS=(ring, segment, offset), ..
EXECUTE_ADDRESS=(ring,segment,offset), R1=ring, R2=ring, ..
R3=ring, GLOBAL_LOCAL_KEY=key, ATTRIBURES=(attributes), ..
SECTION_NAME=section_name
```

FIELD	DEFAULT	DESCRIPTION
LOAD_ADDRESS	0	(IPL_PVA) Three component load address (ring, segment and byte offset) of the segment; specifies where the segment will be loaded. A null specification is indicated by a ring number of zero.
EXECUTE_ADDRESS	0	(IPL_PVA) Three component execution address (ring, segment and byte offset) of the segment; specifies where the segment will ultimately execute. A null specification is indicated by a ring number of zero. If both LOAD_ADDRESS and EXECUTE_ADDRESS are not specified, offset loading will not be performed.
R1	11	(INTEGER) Ring brackets for the segment.
R2	11	
R3	11	
GLOBAL_LOCAL_KEY	0	(INTEGER) Global and local key for segment.
ATTRIBUTES		(CHAR 2) Segment access attributes: any number of the following attributes can be specified. RD - read not controlled by key/lock RK - read controlled by key/lock BI - binding; read not controlled by key/lock WT - write not controlled by key/lock WK - write controlled by key/lock EX - executable non privileged LP - executable local privilege GL - executable global privilege ET - extensible CB - cache bypass
SECTION_NAME	blanks	(CHAR 31) Name of Working_Storage sections to be mapped into this segment.

2.0 ACCESSING THE LINKER

OBJECT_MODULE

The OBJECT_MODULE command specifies module names to be included in the link. Only one OBJECT_MODULE command is allowed.

OBJECT_MODULE, NAME=(mod_name, mod_name...)

FIELD	DESCRIPTION
NAME	(CHAR 31) Name(s) of modules to be included.

INBOARD_SYMBOL_TABLE

The INBOARD_SYMBOL_TABLE command specifies NOS file names which contain Inboard Symbol Tables to be introduced into the LINK. Only one INBOARD_SYMBOL_TABLE command is allowed.

INBOARD_SYMBOL_TABLE, NAME=(filename, filename...)

FIELD	DESCRIPTION
NAME	(CHAR 7) NOS file name(s) of files containing Inboard Symbol Tables.

END

The END command or end-of-file signifies end of the Linker Parameter File.

3.0 LINKER FILE INTERFACE

The Linker Parameter File interface, described in a preceding section, in its most specific sense prescribes the file interface of the Linker. The Linker File Descriptors (LFD's) identify input object files and libraries by name and their associated attributes. The Inboard Symbol Table List (ISTL) identifies by name input file(s) containing inboard symbol table(s).

NOTE: All files named in the LFD and ISTL must be local at the time the Linker is invoked.

The Linker Control Block (LCB), via the NAME_SEED, specifies the string used by the Linker to generate names for the output files: header, segment, and outboard symbol table files. The Linker Program Descriptor (LPD) and Segment Array specifically identifies the header and segment files by name.

The following sections are a brief description of the input files (object, library and inboard symbol table files) and the output files (header, segment, and outboard symbol table files).

3.0 LINKER FILE INTERFACE
3.1 OBJECT FILES

3.1 OBJECT FILES

The Linker accepts, as input, object files in containing object modules in CYBER 180 loader text format V1.1 generated by compilers and assemblers running on CYBER 170. Multiple object modules may reside on an object file.

The object module structure is basically comprised of an Identification Record (IDR), Section Definition Records (SDC), and the object text associated with the sections. The IDR describes external characteristics (name, time and date created, version, creator, and a commentary) and internal characteristics (module attributes and the number of sections) of the object module. There is a SDC for each section type (code, binding, working storage, common, etc.) contained in the object module describing the various attributes of the section.

The specific structure of object modules is not of general concern to users of the Linker, but for those who must generate object modules (compiler, assembler and utility writers), a copy of the object module type definition may be found in Appendix A of this document.

3.2 LIBRARY FILES

The Linker also accepts as input C180 library files containing c180 object modules formatted into a library by the C180 Object Code Utilities. The definition of the object modules contained therein is the same as described in the section entitled "Object Files".

The specific structure of a library file is not of general concern to users of the Linker, but for those who are interested, the record definition may be found in Appendix A of this document.

3.3 HEADER FILE

The Linker outputs a header file which describes the results of a linkage. The name of the header file is the concatentation of the NAME_SEED from the LCB with the string "HDR".

The header file structure is comprised of one header variant and a segment descriptor variant for each segment file generated as the result of a linkage. The header variant contains the number of segment descriptors, the initial p-address and its key, and the binding address. The segment descriptor identifies by name the file on which the segment was written and its segment attributes.

The specific structure of the header file is not of general concern to the users of the Linker, but for those who must interface to the header file a copy of the type definition can be obtained by contacting an SES representative.

3.4 SEGMENT FILES

The Linker outputs segment files which are acceptable as input to the CYBER 180 Simulator or VE Generator. A segment file contains a direct I/O user information record. In the context of the Simulator the segment file is a load file and the user information record is a Load File Directory of type "empty". The Linker allocates and outputs a segment file for each section type encountered on object files during a linkage.

The segment file structure is comprised basically of a load file directory and the linked segment.

The specific structure of the segment file is not of general concern to the users of the Linker. A description of the file, however, may be found in the C180 Simulator ERS, in the appendix section entitled "Central Memory Format".

3.5 SYMBOL TABLE FILES

A symbol table file is a file containing a table of resolved entry points from an instance of Linker execution. The file is termed an Inboard Symbol Table file when used as input to the Linker. The file is termed an Outboard Symbol Table file when it is output as a result of Linker execution.

The Outboard Symbol Table (OST) file is generated by the Linker only if entry points which have the "gated" attribute are encountered during a Linker execution. (Only entry points with the "gated" attribute are externalized to modules in higher rings. This attribute is specifically assigned to a module via the Object Code Utility CHANGE command.) The OST file name is the concatentation of the NAME_SEED from the LCB with the string "OST".

The structure of a symbol table is pertinent only to the Linker.

4.0 LINKER MAP

The Linker map describes the address assignments made by the Linker. The NOS file name, onto which the Linker map is output, is specified either as a VELINK command parameter, or else in the Linker Parameter File. The Linker map contains four basic components which may be optionally listed. A sample map is included in a later section.

4.1 SECTION DEFINITIONS

The following information is printed for every section of every object module:

- o Section type
- o Access attributes
- o Length
- o Address (load and execution if different)
- o Ring brackets
- o Global/local key

4.2 ENTRY POINT NAMES

The following informaton is printed for every entry point:

- o name
- o address (load and execution if different)

4.3 EXTERNAL REFERENCES

A list of the external references is printed after the entry point list.

4.4 OUTPUT SEGMENTS AND COMMON BLOCKS

The following information is printed for every output segment allocated by the Linker:

- o NOS file name
- o Address (load and execution if different)
- o Length
- o Access attributes
- o Ring brackets
- o Global/local key

The following information is printed for every common block allocated by the Linker:

- o Name
- o Access attributes
- o Length
- o Address (load and execution if different)

5.0 LINKER PROCEDURAL INTERFACES

The following sections are also contained in the SES User's Handbook.

GENCPF - GENERATE A CHECKPOINT FILE (CPF)

GENCPF executes the SES VE Linker and the SES VE Generator and produces a Checkpoint File (CPF) which can be loaded and executed on the CYBER 180 system simulator. The user must provide a file containing 180 object text, and can optionally provide a Linker Parameter File (LPF), a set of Monitor Segment files, and/or a VE Generator DIRective file. The "MF" file will contain the map produced by the VE Linker and additional information produced by the VE Generator.

Note: most of the following parameters will not be used 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 180 object text (ver 1.0). This parameter does not have a default.

lfl (Filename(s), optional)
Library_File_List - list of up to 10 names of Library files containing 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)

5.0 LINKER PROCEDURAL INTERFACES

Name_Seed - This parameter specifies the "name seed" for the VE 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 (Char 1, optional)
Linker Map options - This parameter specifies the amount of information output on the Linker Map. The value of this parameter OVERRIDES any Linker Parameter File specification of this field. Values for this field are:
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, entry points plus output segment and common block allocations (full Linker map).

rewind (Keyword, optional)
Rewind_map_file - This parameter specified whether to rewind the Map file before it is written. Default is to rewind it. The value of this parameter OVERRIDES any Linker Parameter File specification of this field. Keyword value meanings are:
REWIND - rewind Map file
NOREW - don't rewind Map file

cpf (Filename, optional)
Checkpoint_File - This parameter specifies the name of the file containing the output from the VE Generator. The default is "CPFILE". The value of this parameter OVERRIDES any Linker Parameter File specification of this field.

lpf (Filename, optional)
Linker_Parameter_File - This parameter specifies a file

5.0 LINKER PROCEDURAL INTERFACES

that contains Linker parameters that affect the Link. If no lpf is specified, default values indicated in the LPF description apply.

cybilib (Keyword, optional)
This parameter, when specified, will cause CYBILIB to be used to satisfy externals during the linking process. The procedure will ACQUIRE the file from SES and add 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 will be a file containing default directives.

mtrns or mns (String(4), optional)
Monitor Name_Seed - This parameter specifies the "name seed" of the VE Linker Segment files that contain the Monitor code. This parameter is provided to allow the user to have his own Monitor program. The default for this parameter is "MTRX". The procedure will check the local files, the local PF catalog, and then the SES catalog to get the files.

Examples:

The following example illustrates how a user would compile a test program written in CYBIL, generate a Checkpoint File, and run it on the Simulator.

```
ses.cybil ci i=testprg l=testlst b=testlgo
ses.gencpf ofl=testlgo cpf=testcpf cybilib
ses.sim180 restart=testcpf
?  run
?  bye
```

The last two lines were Simulator commands.

The next example illustrates how to generate a Checpoint File from several input files of 180 object text.

```
ses.gencpf ofl=(mylgo1,mylgo2) cpf=mycpf cybilib
```

5.0 LINKER PROCEDURAL INTERFACES

VELINK - EXECUTE THE VIRTUAL ENVIRONMENT LINKER

VELINK executes the VE Linker, which links data from 180 object files/libraries and produces a map file and a set of SEGMent files. The procedure is set up so that the user can specify his own Linker Parameter File (LPF) containing VE Linker variables that control the linkage. For more information on this, refer to the SES VE Linker ERS (ARH2816).

Note: this proc 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 180 object text (ver 1.0). This parameter does not have a default.

lfl (Filename(s), optional)
: Library_File_List - list of up to 10 names of Library files containing 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 VE 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.

5.0 LINKER PROCEDURAL INTERFACES

mo (Char 1, optional)
Linker Map options - This parameter specifies the amount of information output on the Linker Map. The value of this parameter OVERRIDES any Linker Parameter File specification of this field. Values for this field are:

- 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, entry points plus output segment and common block allocations (full Linker map).

rewind (Keyword, optional)
Rewind_map_file - This parameter specified whether to rewind the Map file before it is written. Default is to rewind it. The value of this parameter OVERRIDES any Linker Parameter File specification of this field. Keyword value meanings are:

- REWIND - rewind Map file
- NOREW - don't rewind Map file

lpf (Filename, optional)
Linker_Parameter_File - This parameter specifies a file that contains Linker parameters that affect the Link. If no lpf is specified, default values indicated in the LPF description apply.

cybilib (Keyword, optional)
This parameter, when specified, will cause CYBILIB to be used to satisfy externals during the linking process. The procedure will ACQUIRE the file from SES and add it to the Library_File_List. The default is not to use PASILIB as part of the Link.

Example:

In the following example the Linker is passed a Linker Parameter File containing the variable 'mylcb'.

```
ses.velink ofl=(lgo1,lgo2) lpf=mylpf lcb=mylcb vef=myvef
```

5.0 LINKER PROCEDURAL INTERFACES
5.1 EXAMPLE OF LINKER PARAMETER FILE

5.1 EXAMPLE OF LINKER PARAMETER FILE

The following is an example of a Linker Parameter File. Since the Linker ignores items in the list with blank or uninitialized file names, this allows you to include the declaration and access attribute initialization of the object file list in your command file, leaving only the specification of input file name until you are ready to issue the LINK command.

```
link_options,map_filename=linklist,map_options=m,no_map_rewind,
name_seed=lxxx,load_segment=6,execute_segment=6

object_file,kfn=" ",r1=11,r2=11,r3=11,global_local_key=0,
execute_privilege=e

object_file,kfn=" ",r1=11,r2=11,r3=11,global_local_key=0,
execute_privilege=e

object_file,kfn=" ",r1=11,r2=11,r3=11,global_local_key=0,
execute_privilege=e

object_file,kfn=" ",r1=11,r2=11,r3=11,global_local_key=0,
execute_privilege=e

define_segment,load_address=(11,0,4000),execute_address=(11,0,
4000),r1=11,r2=11,r3=11,global_local_key=0,attributes=(rd,ex)

define_segment,load_address=(11,0,5000),execute_address=(11,0,
5000),r1=11,r2=11,r3=11,global_local_key=0,attributes=(bi)

define_segment,load_address=(11,0,6000),execute_address=(11,0,
6000),r1=11,r2=11,r3=11,global_local_key=0,attributes=(rd,wt)

define_segment,load_address=(11,0,7000),execute_address=(11,0,
7000),r1=11,r2=11,r3=11,global_local_key=0,attributes=(rd,wt,et)

define_segment,load_address=(11,0,8000),execute_address=(11,0,
8000),r1=11,r2=11,r3=11,global_local_key=0,attributes=(rd)
```

5.2 EXAMPLE OF LINKER MAP

The following linkmap was produced by linking the sample program described in the ERS for the 180 CPU Assembler.

```
          LINKER V 2.5 OUTPUT LISTING     05/21/79        16.00.46

MODULE  = TEST                    LANGUAGE = CPU ASSEMBLER
FILE    = LGO         05/21/79          16:00:06.000

SECTION TYPE/                    LOAD/              RING
ACCESS ATTRIBUTES     LENGTH     EXECUTION ADDR     BRACKETS   G/L KEY
-------------------------------------------------------------------------
WORKING STORAGE
READ WRITE                 28    B 00D 00000000     (B,B,B)    (00,00)

WORKING STORAGE
READ                        8    B 00C 00000000     (B,B,B)    (00,00)

BINDING
READ BINDING               10    B 00B 00000000     (B,B,B)    (00,00)

CODE
READ EXECUTE               22    B 00A 00000000     (B,B,B)    (00,00)


      ENTRY POINT DEFINITIONS                ADDRESS
        ENT                    NOT GATED B 00A 00000000
          LINKER V 2.5 OUTPUT LISTING     05/21/79        16.00.46

SES/C80 LINKER OUTPUT : LCB = LINK_CONTROL_BLK
                        PCB = LINK_PCB
                        LPD = LINKED_PRG_DESC

        PRIMARY ENTRY POINT = ENT1
                    ADDRESS = B 00A 00000000

FILE NAME/                       LOAD/              RING
```

5.0 LINKER PROCEDURAL INTERFACES
5.2 EXAMPLE OF LINKER MAP

ACCESS ATTRIBUTES	LENGTH	EXECUTION ADDR	BRACKETS	G/L KEY
SEGM101				
READ EXECUTE	22	* 00A 00000000	(B,B,B)	(00,00)
SEGM102				
BINDING	10	* 00B 00000000	(B,B,B)	(00,00)
SEGM103				
READ	8	* 00C 00000000	(B,B,B)	(00,00)
SEGM104				
READ WRITE	28	* 00D 00000000	(B,B,B)	(00,00)

NO LINKER ERRORS WERE DETECTED

6.0 ERROR MESSAGES

6.0 ERROR MESSAGES

The Linker provides two varieties of error message: SES SCL error messages describing the disposition of the command, and linker diagnostics printed on the Linker map.

6.1 SCL MESSAGES

Linker SCL messages conform to the SES message standards as described in the Meggage Generator (MG) Interface ERS.

10000 LINKER NORMAL TERMINATED
MEANING: The Linker has terminated normally with no fatal or nonfatal errors.
ACTION: Be thankful.

10100 LINKER NORMAL TERMINATE WITH NONFATAL ERRORS - SEE MAP LISTING
MEANING: One or more non fatal errors were encountered during the Linker Command. Linker output is probably valid.
ACTION: Check Linker map for diagnostic or diagnostics.

10101 LINKER ABNORMAL TERMINATE - SEE MAP LISTING
MEANING: The Linker has detected a fatal error and gracefully aborted. Linker output is undefined.
ACTION: Check Linker map for diagnostic: correct problem and rerun

10102 LINKER ABNORMAL TERMINATE - NO MODULES PROVIDED
MEANING: No object module input was encountered by the Linker.
ACTION: Check the "of" parameter on the LINK command, if specified, or else the OBJECT_FILE in the Linker Parameter File; check the KFN fields of all elements of the object file list; check all files for input.

6.0 ERROR MESSAGES
6.1 SCL MESSAGES

10200 LIBRARY FILE file_name NOT LOCAL
MEANING: File specified by the "lfl"parameter of the VELINK command or the "OBJECT_LIBRARY" command of the Linker Parameter File is not a local file.
ACTION: Make the file local or remove the parameter.

10201 OBJECT FILE file_name NOT LOCAL
MEANING: File specified by the "ofl"parameter of the VELINK command or the "OBJECT_FILE" command of the Linker Parameter File is not a local file.
ACTION: Make the file local or remove the parameter.

10202 OBJECT FILENAME file_name DUPLICATES EXISTING FILE
MEANING: Filename specified as and object file or library ("ofl" or "lfl" of the VELINK command or "OBJECT_FILE" or "OBJECT_LIBRARY" of the Linker Parameter File duplicates another specifed filename.
ACTION: Remove the specification of this file. Make the file local or remove the parameter.

10203 IST FILE file_name NOT LOCAL
MEANING: File specified by the "INBOARD_SYMBOL_TABLE"command of the Linker Parameter File is not a local file.
ACTION: Make the file local or remove the parameter.

10204 IST FILENAME file_name DUPLICATES EXISTING FILE
MEANING: Filename specified by the "INBOARD_SYMBOL_TABLE" command of the Linker Parameter File duplicates another specifed filename.
ACTION: Remove the specification of this file. Make the file local or remove the parameter.

10300 LPF FILE file_name NOT LOCAL
MEANING: File specified by the "lpf" parameter of the VELINK command is not a local file.
ACTION: Make the file local or remove the

6.0 ERROR MESSAGES
6.1 SCL MESSAGES

parameter.

10301 UNKNOWN LPF COMMAND command SPECIFIED
MEANING: The Linker has detected an invalid Linker Parameter File command.
ACTION: Correct the Linker parameter file.

10302 INVALAD MAP OPTION map_option SPECIFIED
MEANING: The Linker has detected an invalid map option specified by the "mo" parameter of the VELINK command or "MAP_OPTIONS" specification of the Linker Parameter File "LINK_OPTIONS" command.
ACTION: Correct the map option specification.

10303 INVALAD NAME_SEED xxxx SPECIFIED
MEANING: The Linker has detected an invalid name seed as specified by the "ns" parameter of the VELINK command or "NAME_SEED" specification of the Linker Parameter File "LINK_OPTIONS" command.
ACTION: Correct the name seed specification.

10304 INVALAD SEGMENT ATTRIBUTE segment_attribute SPECIFIED
MEANING: The Linker has detected an invalid segment attribute specified by the "ATTRIBUTES" specification of the Linker Parameter File "DEFINE_SEGMENT" command.
ACTION: Correct the attribute specification.

10305 INVALAD EXECUTE PRIVILEGE executive_privilege SPECIFIED
MEANING: The Linker has detected an invalid executive privilege specified by the "EXECUTIVE_PRIVILEGE" specification of the Linker Parameter File "OBJECT_FILE" or "OBJECT_LIBRARY" command.
ACTION: Correct the executive privilege specification.

6.2 LINKER DIAGNOSTICS

The Linker prints formatted messages on the Linker map. The template of all messages is as follows:

```
* * * LINKER ERROR NNNNN [*FATAL*] <error message text>
        MODULE       = <module name if appropriate>
        FILE         = <file name if appropriate>
        NAME         = <entry point name if appropriate>
        RECORD COUNT = <record count if appropriate>
```

NNNNN	ERROR MESSAGE TEXT
1	IMPROPER RELOCATION ADDRESS SPECIFICATION MEANING: RIF item is being incorrectly generated; Linker output unaffected. ACTION: Correct object text.
2	UNANTICIPATED EOR MEANING: The Linker encountered an EOR somewhere other than the appropriate end of an object module ACTION: Correct object text
3	MORE THAN ONE CODE SECTION IN A MODULE MEANING: A single object module had more than one code section; only one is permitted. ACTION: Correct object text.
4	SDO GREATER THAN IDR SPECIFICATON ENCOUNTERED MEANING: The number of sections in the IDR is incorrect or section ordinals do not start at zero or are not contiguous. ACTION: Correct object text.
5	CODE SECTION ATTRIBUTE SPECIFICATION ERROR MEANING: Code sections must not have write or binding attributes. ACTION: Correct object text.
6	MORE THAN ONE BINDING SECTION PER MODULE MEANING: A single object module had more than one binding section; only one is permitted. ACTION: Correct object text.
7	BINDING SECTION ATTRIBUTE SPECIFICATION ERROR MEANING: Binding sections must not be writable or

executable.
ACTION: Correct object text.

8 BINDING SECTION ALIGNMENT ERROR
MEANING: The binding section must be aligned on a 64 bit CYBER 180 full word boundary.
ACTION: Correct object text.

9 DUPLICATE SECTION DEFINITION ORDINAL
MEANING: The same ordinal has been used for two sections in a single object module.
ACTION Correct object text.

10 BINDING ATTRIBUTE SPECIFIED FOR A NON BINDING SECTION
MEANING: The binding attribute may only be specified for the binding section.
ACTION: Correct object text.

11 CONFLICTING PROTECTION ATTRIBUTE FOR COMMON BLOCK
MEANING: Different protection has been specified in separate common block declarations.
ACTION: Correct source program.

12 CONFLICTING LENGTH SPECIFICATION FOR COMMON BLOCK
MEANING: Unequal lengths were specified in separate common block declarations.
ACTION: Correct source program.

13 COMMON TABLE OVERFLOW - RECOMPILE LINKER
MEANING: Linker internal table size exceeded.
ACTION: Call SES representative.

14 MISPLACED IDR OR SDC
MEANING: Object text structure is incorrect.
ACTION: Correct object text.

15 MODULE DID NOT CONTAIN A CODE SECTION
MEANING: No code section encountered; Linker output unaffected.
ACTION: Correct object text if necessary.

16 MODULE DID NOT CONTAIN A BINDING SECTION
MEANING: No binding section encountered; Linker output unaffected.
ACTION: Correct object text if necessary.

17 SDOS NOT CONTIGUOUSLY NUMBERED
MEANING: Object text structure is incorrect.

ACTION: Correct object text.

18 SEGMENT TABLE OVERFLOW - RECOMPILE LINKER
MEANING: Linker internal table size exceeded.
ACTION: Call SES representative.

19 ZEROIZE SECTION INTERNAL LOGIC ERROR
MEANING: Linker has aborted.
ACTION: Call SES representative.

20 PROCEDURE DESCRIPTOR ALLOCATED IN NON BINDING SEGMENT
MEANING: Procedure descriptors that are to be used with the hardware CALL instructions must be in binding segments.
ACTION: Correct object text if necessary.

21 POINTER IN BINDING SEGMENT WAS MISALIGNED
MEANING: Binding section entries must be right justified in an CYBER 180 full word.
ACTION: Correct object text.

22 ATTEMPTED TO PLACE DATA IN A BINDING SECTION
MEANING: Binding section entries may only be pointers or procedure descriptors.
ACTION: Correct object text.

23 LFD RING BRACKET SPECIFICATION ERROR
MEANING: An illegal ring number was encountered or R1>R2>R3.
ACTION: Correct LFD entry and retry LINK command.

24 PREALLOCATED BINDING SEGMENT ATTRIBUTE ERROR
MEANING: Binding segments may not be writable or executable or readable under key lock control.
ACTION: Correct PSA entry and retry LINK command.

25 PREALLOCATED EXECUTABLE SEGMENT ATTRIBUTE ERROR
MEANING: Executable segments may not be writable.
ACTION: Correct PSA entry and retry LINK command.

26 PREALLOCATED SEGMENT RING BRACKET INCONSISTENCY
MEANING: The relationship $R1 \leq R2 \leq R3$ did not apply.
ACTION: Correct RSA entry and retry LINK command.

27 PREALLOCATED SEGMENT ADDRESS (RING) ERROR
MEANING: An illegal ring number was specified.
ACTION: Correct PSA entry and retry LINK command.

28 FIRST RECORD OF AN OBJECT MODULE WASNT AN IDR
MEANING: Improper object text structure or Linker input file was inappropriate.
ACTION: Correct input file.

29 DUPLICATE ENTRY POINT WAS DETECTED
MEANING: A symbol has been XDCLed as an entry point more than once. First definition is used; output is unaffected.
ACTION: Correct object file list if necessary.

30 LST OVERFLOW - TOO MANY ENTRY POINTS
MEANING: Internal table overflow - Linker must be recompiled.
ACTION: Call SES representative.

31 EXTERNAL ARRAY OVERFLOW - TOO MANY EXTERNALS
MEANING: Internal Table overflow - Linker must be recompiled. (Maximum is 200 entries.)
ACTION: Call SES representative.

32 RECORD CONTAINS IMPROPER SDO
MEANING: An object text record referenced an undefined object text section.
ACTION: Correct object text.

33 INPUT RECORD CONTAINS AN IMPROPER SECTION OFFSET
MEANING: An object text record referenced an offset outside the range specified in the section definition.
ACTION: Correct object text.

34 NO PRIMARY ENTRY POINT ENCOUNTERED
MEANING: No primary entry point was specified.
ACTION: Declare a procedure in your PASCAL-X program with the *MAIN* attribute or place the primary entry point name in the PEP field of the PCB.

35 PRIMARY ENTRY POINT NOT XDCLED
MEANING: The name encountered by the Linker for the primary entry point was not XDCLed in any object module encountered in the linkage.
ACTION: Declare a procedure in your ISWL program with the *XDCL* attribute.

6.0 ERROR MESSAGES
6.2 LINKER DIAGNOSTICS

36 NO OBJECT FILE INPUT
MEANING: No object module input was encountered by the Linker.
ACTION: Check the OBJ_FILE_LIST parameter on the LINK command if specified, or else the PCB.OBJ_FILE_LIST_NAME for an LNS name; check the KFN fields of all elements of the object file list; check all elements of the object file list; check all files for input.

37 UNSATISFIED EXTERNAL REFERENCE
MEANING: An XREFed declaration was not XDCLed in any module encounterd in the linkage, or on any of the modules on the specified libraries.
ACTION: Check input file list to assure that you specified all the files you intended to; check your program for a missing 'XDCL'.

38 LFD CONTAINS IMPROPER EXECUTE ATTRIBUTE
MEANING: Unknown execute attribute was specified in LFD.
ACTION: Correct LFD and retry LINK command.

39 UNKNOWN OBJECT TEXT RECORD TYPE
MEANING: Object text structure is incorrect.
ACTION: Correct object text.

40 UNKNOWN EXTERNAL REFERENCE INSERTION TYPE
MEANING: Object text structure is correct.
ACTION: Correct object text.

41 UNKNOWN SECTION DEFINITION TYPE
MEANING: Object text structure is correct.
ACTION: Correct object text.

42 RIF DOES NOT PERTAIN TO CODE OR BINDING SECTION
MEANING: Relocation information is being incorrectly generated. Linker output is unaffected.
ACTION: Correct object text.

43 IMPROPER RELOCATION CONTAINER SPECIFICATION
MEANING: Relocation information is being incorrectly generated. Linker output is unaffected.
ACTION: Correct object text.

44 EXPECTED SCD RECORD
MEANING: Object text structure is incorrect.
ACTION: Correct object text.

45 INVALID PROCEDURE OFFSET FOR INDIRECT CALL
MEANING: The procedure offset for an indirect call is not 0 mod 8. To facilitate "binding", procedure offsets for indirect calls should be 0 mod 8.
ACTION: None necessary at this time, but the object text generator should be modified to allocate all procedures on a word boundary.

46 INVALID BIT STRING INSERTION RECORD
MEANING: The Linker encountered a bit string insertion record with a bit offset greater than 7 or bit length greater than 57. No bit string insertion has taken place.
ACTION: The object text generator has caused the error and must be corrected.

47 BAD LIBRARY FORMAT
MEANING: A file in the Library File List was not a recognizable 180 Library created by the SES CYBER 180 Object Code Utilities.
ACTION: Review the Libraries specified in the *Library File List*.

48 REQUIRED LIBRARY MISSING
MEANING: The Linker encountered a Libraries record that specified a library that was not present in the *Library File List*.
ACTION: ACQUIRE the library and specify it in the *Library File List*.

49 ERROR IN PARAMETER VERIFICATION
MEANING: The type declarations for the variable do not match on the XDCL and the XREF.
ACTION: Check the type declarations for the variable, they must be word for word.

A1.0 SES/C180 OBJECT TEXT FORMAT

A1.0 SES/C180 OBJECT TEXT FORMAT

The general form of an object module is a file of binary records with the following topology:

```
< object text descriptor # 1 >
  < object text record # 1 >
< object text descriptor # 2 >
  < object text record # 2 >
            ...
< object text descriptor # n >
  <object text record # n >
```

For the sake of simplicity the record descriptor - record pairs will be referred to as records hereafter.

For a CPU program, the object text records must be arranged in the following order:

1). Identification record
2.) Library, section definition, text, bit string insertion, address formulation, external linkage, entry definition, relocation, formal parameter specification, actual parameter specification and binding template records in arbitrary order with the one stipulation that a section definition record must precede any other object text records that refer to the section.
3). Transfer symbol record.

For a PPU program or overlay, the object text records must be arranged in the following order:

A1.0 SES/C180 OBJECT TEXT FORMAT

```
{
{       1.) Identification record
{       2.) PPU absolute record
{
{


{ Constants that pertain to both the object and load module. }

  CONST
    llc$max_adr_items = 0ffff(16),
    llc$max_ext_items = 0ffff(16),
    llc$max_libraries = 0ffff(16),
    llc$max_rel_items = 0ffff(16),
    llc$max_section_ordinal = 0ffff(16) - 1;


  TYPE
    llt$object_text_descriptor = record
      case kind: llt$object_record_kind of
      = llc$identification, llc$section_definition, llc$bit_string_insertion,
        llc$entry_definition, llc$binding_template, llc$transfer_symbol,
        oct$library_header =
        unused: ost$segment_offset, {must be zero}
      = llc$libraries =
        number_of_libraries: 1 .. llc$max_libraries,
      = llc$text, llc$replication =
        number_of_bytes: 1 .. osc$max_segment_length,
      = llc$relocation =
        number_of_rel_items: 1 .. llc$max_rel_items,
      = llc$address_formulation =
        number_of_adr_items: 1 .. llc$max_adr_items,
      = llc$external_linkage =
        number_of_ext_items: 1 .. llc$max_ext_items,
      = llc$formal_parameters, llc$actual_parameters =
        sequence_length: ost$segment_length, {REP sequence_length OF CELL}
      = llc$ppu_absolute =
        number_of_words: llt$ppu_address,
      = oct$module_directory, oct$entry_point_directory =
        number_of_directory_entries: integer,
      casend,
    recend;


  TYPE
    llt$object_record_kind = (llc$identification, llc$libraries,
      llc$section_definition, llc$text, llc$replication,
      llc$bit_string_insertion, llc$entry_definition, llc$relocation,
```

A1.0 SES/C180 OBJECT TEXT FORMAT

```
    llc$address_formulation, llc$external_linkage, llc$formal_parameters,
    llc$actual_parameters, llc$binding_template, llc$ppu_absolute,
    llc$reserved_1, llc$reserved_2, llc$reserved_3, llc$reserved_4,
    llc$transfer_symbol, oct$library_header, oct$module_directory,
    oct$entry_point_directory);


TYPE
  llt$identification = record
    name: pmt$program_name,
    object_text_version: string (4), { 'V1.1' }
    kind: llt$module_kind,
    time_created: ost$time,
    date_created: ost$date,
    attributes: llt$module_attributes,
    greatest_section_ordinal: llt$section_ordinal,
    generator_id: llt$module_generator,
    generator_name_vers: string (40),
    commentary: string (40),
  recend,

  llt$module_kind = (llc$mi_virtual_state, llc$vector_virtual_state,
    llc$iou);


TYPE
  llt$module_generator = (llc$algol, llc$apl, llc$basic, llc$cobol,
    llc$assembler, llc$fortran, llc$object_library_generator,
    llc$pascal, llc$cybil, llc$pl_i, llc$sympl),

  llt$module_attributes = set of (llc$nonbindable, llc$nonexecutable);



TYPE
  llt$libraries = array [ * ] of amt$local_file_name;

TYPE
  llt$section_definition = record
    kind: llt$section_kind,
    access_attributes: llt$section_access_attributes,
    section_ordinal: llt$section_ordinal,
    length: ost$segment_length,
    allocation_alignment: ost$segment_offset,
    allocation_offset: ost$segment_offset,
    name: pmt$program_name,
  recend,
```

A1.0 SES/C180 OBJECT TEXT FORMAT

```
    llt$section_ordinal = 0 .. llc$max_section_ordinal,
    llt$section_offset = 0 .. osc$maximum_offset;

  TYPE
    llt$section_kind = (llc$code_section, llc$binding_section,
      llc$working_storage_section, llc$common_block,
      llc$extensible_working_storage, llc$extensible_common_block,
      llc$invalid_section),

    llt$section_access_attributes = set of (llc$read, llc$write, llc$execute,
      llc$binding);


{ Text record. }

  TYPE
    llt$text = record
      section_ordinal: llt$section_ordinal,
      offset: llt$section_offset,
      byte: array [ * ] of 0 .. 255,
    recend;


{ Replication record. }

  TYPE
    llt$replication = record
      section_ordinal: llt$section_ordinal,
      offset: llt$section_offset,
      increment: 1 .. osc$max_segment_length,
      count: 1 .. osc$max_segment_length,
      byte: array [ * ] of 0 .. 255,
    recend;


{ Bit insertion record. }

  TYPE
    llt$bit_string_insertion = record
      section_ordinal: llt$section_ordinal,
      offset: llt$section_offset,
      bit_offset: 0 .. 7,
      bit_length: 1 .. 63,
      bit_string: packed array [1 .. 63] of 0 .. 1,
    recend;


{ Address formulation record. }
```

A1.0 SES/C180 OBJECT TEXT FORMAT

```
  TYPE
    llt$address_formulation = record
      value_section: llt$section_ordinal,
      dest_section: llt$section_ordinal,
      item: array [ * ] of llt$address_formulation_item,
    recend,

    llt$address_formulation_item = record
      kind: llt$internal_address_kind,
      value_offset: llt$section_offset,
      dest_offset: llt$section_offset,
    recend,

    llt$address_kind = (llc$pva, llc$internal_proc, llc$one_word_external_proc
      llc$external_proc, llc$address_addition, llc$address_subtraction),

    llt$internal_address_kind = llc$pva .. llc$external_proc;


{ External reference record. }

  TYPE
    llt$external_linkage = record
      name: pmt$program_name,
      language: llt$module_generator,
      declaration_matching_required: boolean,
      declaration_matching_value: integer,
      item: array [ * ] of llt$external_linkage_item,
    recend,

    llt$external_linkage_item = record
      section_ordinal: llt$section_ordinal,
      offset: llt$section_offset,
      kind: llt$address_kind,
      offset_operand: ost$segment_offset,
    recend;


{ Entry point definition record. }

  TYPE
    llt$entry_definition = record
      section_ordinal: llt$section_ordinal,
      offset: llt$section_offset,
      attributes: llt$entry_point_attributes,
      name: pmt$program_name,
      language: llt$module_generator,
```

```
    declaration_matching_required: boolean,
    declaration_matching_value: integer,
  recend,

  llt$entry_point_attributes = set of (llc$retain_entry_point,
    llc$gated_entry_point);


TYPE
  llt$relocation = array [ * ] of llt$relocation_item,

  llt$relocation_item = record
    section_ordinal: llt$section_ordinal,
    offset: llt$section_offset,
    relocating_section: llt$section_ordinal,
    container: llt$relocation_container,
    address: llt$address_type,
  recend,

  llt$relocation_container = (llc$parcel, llc$three_bytes, llc$halfword,
    llc$word, llc$d_field, llc$q_field, llc$long_d_field),

  llt$address_type = (llc$byte_positive, llc$parcel_positive,
    llc$halfword_positive, llc$word_positive, llc$byte_signed,
    llc$parcel_signed, llc$halfword_signed, llc$word_signed);


{ Procedure formal parameter description record. }

TYPE
  llt$formal_parameters = record
    procedure_name: pmt$program_name,
    specification: SEQ ( * ),
  recend;


{ Procedure call actual parameters record. }

TYPE
  llt$actual_parameters = record
    callee_name: pmt$program_name,
    language: llt$module_generator,
    line_number_of_call: llt$source_line_number,
    specification: SEQ ( * ),
  recend,

  llt$source_line_number = string (18);
```

A1.0 SES/C180 OBJECT TEXT FORMAT

```
{ FORTRAN argument description: used to describe a single actual or }
{ formal parameter. }

  TYPE
    llt$fortran_argument_desc = record
      argument_type: llt$fortran_argument_type,
      string_length: llt$fortran_string_length, { only used for type CHAR}
      argument_kind: llt$fortran_argument_kind,
      array_size: llt$fortran_array_size, { only used for kind DIMENSION}
      mode: llt$argument_usage,
    recend,

    llt$fortran_argument_type = (llc$fortran_logical, llc$fortran_integer,
      llc$fortran_real, llc$fortran_double_real, llc$fortran_complex,
      llc$fortran_char, llc$fortran_boolean, llc$fortran_null_type),

    llt$fortran_string_length = record
      adaptable: boolean,
      number: 0 .. 0ffff(16),
    recend,

    llt$fortran_argument_kind = (llc$fortran_simple_variable,
      llc$fortran_dimension, llc$fortran_external,
      llc$fortran_subscripted_var),

    llt$fortran_array_size = record
      adaptable: boolean,
      dimension: ost$segment_length,
    recend,

    llt$argument_usage = (llc$argument_written, llc$argument_not_written);


  TYPE
    llt$binding_template = record
      binding_offset: llt$section_offset,
      case kind: llt$binding_template_kind of
      = llc$current_module =
        section_ordinal: llt$section_ordinal,
        offset: llt$section_offset,
        internal_address: llt$internal_address_kind,
      = llc$external_reference =
        name: pmt$program_name,
        address: llt$address_kind,
      casend,
    recend;
```

A1.0 SES/C180 OBJECT TEXT FORMAT

```
  TYPE
    llt$binding_template_kind = (llc$current_module, llc$external_reference);

  TYPE
    llt$transfer_symbol = record
      name: pmt$program_name,
    recend;


TYPE
  llt$ppu_absolute = record
    executes_on_any_ppu: boolean,
    ppu_number: 0 .. llc$max_ppu_number,
    load_address: llt$ppu_address,
    entry_address: llt$ppu_address,
    text: array[ * ] OF 0 .. 0ffff(16),
  recend;


{ library header record

  TYPE
    llt$library_record = record
      name: pmt$program_name, {name of library
      time_created: ost$time,
      date_created: ost$date,
      module_directory_index: integer,
      entry_point_directory_index: integer,
    recend;


{module or entry point directory record

  TYPE
    llt$directory_item = record
      name: pmt$program_name, {name of module or entry point
      module_size: integer, {size of module (170 words)
      module_index: integer, {random index of module
    recend;
```

Table of Contents

1.0 PREFACE 1-1
1.1 SCOPE 1-1
1.2 APPLICABLE DOCUMENTS 1-3
1.3 LINKER V2.5 AND LINKER V2.4 DIFFERENCES 1-4
1.4 DEFICIENCIES AND LIMITATIONS 1-5
1.5 TERMINOLOGY 1-6

2.0 ACCESSING THE LINKER 2-1

3.0 LINKER FILE INTERFACE 3-1
3.1 OBJECT FILES 3-2
3.2 LIBRARY FILES 3-3
3.3 HEADER FILE 3-4
3.4 SEGMENT FILES 3-5
3.5 SYMBOL TABLE FILES 3-6

4.0 LINKER MAP 4-1
4.1 SECTION DEFINITIONS 4-1
4.2 ENTRY POINT NAMES 4-1
4.3 EXTERNAL REFERENCES 4-1
4.4 OUTPUT SEGMENTS AND COMMON BLOCKS 4-2

5.0 LINKER PROCEDURAL INTERFACES 5-1
5.1 EXAMPLE OF LINKER PARAMETER FILE 5-6
5.2 EXAMPLE OF LINKER MAP 5-7

6.0 ERROR MESSAGES 6-1
6.1 SCL MESSAGES 6-1
6.2 LINKER DIAGNOSTICS 6-4

Appendix Object Text Definitions A1

A1.0 SES/C180 OBJECT TEXT FORMAT A1-1