ARH4895

EXTERNAL REFERENCE SPECIFICATION

for

SES MC68000 Absolute 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	12/22/81	ERS (V3.0) Original
B	05/10/84	Miscellaneous corrections

1.0 PREFACE

The Linker is a post compilation utility by which input object text is processed and written to various segment files which can then be loaded on the Motorola 68000 (MC68000) Simulator or MC68000 hardware. The different object text sections are processed to form a set of memory 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 or without library files) as generated by a compiler or assembler, or else they may specify an entry point from a module on a specified library as the STARTING_PROCEDURE parameter on the LINK68K 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 MC68000 object modules in SFSloader text format V1.4 generated by compilers and/or assemblers running on CYBER 170.

The output of the Linker is a series of files corresponding to MC68000 memory; a header file which contains the segment attributes for each output segment; potentially an outboard symbol table file and a debug information file.

1.1 SCOPE

This document describes the external characteristics of the SES MC68000 Absolute Linker V3.0.

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 (ARH1833)

1.3 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.

1.4 DEFICIENCIES AND LIMITATIONS

The Linker performs only cursory checks to determine if users have specified any duplicate file names in their

parameterization.

The Linker performs no checks to determine if any file names generated from the NAME_SEED duplicate any file names users have specified in their parameterization. Therefore, the Linker may abort or yield indeterminate results. Hence, users 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, an object file that has been incorrectly generated may cause the Linker to abort or yield indeterminate results.

2.0 ACCESSING THE LINKER

The Linker is accessed via an SES procedure, which is described in a later section. Users can control the linkage by specifying their parameters on the procedure call (LINK68K), in the Linker Parameter File, or a combination of both. Parameters on the procedure call override the parameters specified in the Parameter File.

3.0 LINKER PARAMETER FILE (LPF)

3.0 LINKER_PARAMETER_FILE_(LPF)

The Linker Parameter File (LPF) consists of a legible file of subcommands, typically built and maintained by an editor, that controls the operation of the MC68000 linker. LPF is an optional parameter on the LINK68K command. If a linker parameter file is not provided, defaults are as shown below. Several subcommands are provided: LINK_OPTIONS, OBJECT_FILE, OBJECT_LIBRARY, DEFINE_SEGMENT, OBJECT_MODULE, INBOARD_SYMBOL_TABLE, INCLUDE_LINKED_SYMBOLS, and END. Each subcommand is entered on a separate line.

3.1 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,&..
STARTING__PROCEDURE=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

3.0 LINKER PARAMETER FILE (LPF)
3.1 LINK_OPTIONS

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
I - Section allocations, entry points, output segment and common block allocations, plus Inboard Symbol Table (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 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 LINK68K command.

MAX_EXTERNALS
300 — (INTEGER) The maximum number of externals allowed in this link.

STARTING_PROCEDURE
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 SP parameter is used on the LINK68K command. If not specified here or on the command, the first transfer symbol encountered is the primary entry point.

3.2 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. LINK68K command parameters 'OFL' and 'LFL' take precedence over these commands.

```
OBJECT_FILE, FILENAME=filename,..
DEFAULT_SECTION=list of (section_name, ..
list of section_attributes)

OBJECT_LIBRARY, FILENAME=filename,..
DEFAULT_SECTION=list of (section_name, ..
list of section_attributes)
```

FIELD	DEFAULT	DESCRIPTION
FILENAME	none	(CHAR 7) NOS file that name of a local file that contains object modules.
DEFAULT_SECTION	none	(CHAR 31, CHAR 1) The list of section names to be associated with any unnamed sections with the matching attributes which appear in any module in the file or library. Values for the section attribute are R - read W - write E - execute

3.3 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=(offset), ..
EXECUTE_ADDRESS=(offset),..
ATTRIBUTES=(attributes), SECTION_NAME=(section_names)
```

FIELD DEFAULT	DESCRIPTION
LOAD_ADDRESS 0	(ADDRESS) The load address (byte offset) of the segment; specifies where the segment will be loaded.
EXECUTE_ADDRESS 0	(ADDRESS) The execution address (byte offset) of the segment; specifies where the segment will ultimately execute. If both LOAD_ADDRESS and EXECUTE_ADDRESS are not specified, offset loading will not be performed.
ATTRIBUTES	(CHAR 2) Segment access attributes: any number of the following attributes can be specified. RD - read WT - write EX - executable ET - extensible
SECTION_NAME blanks	(CHAR 31) Name of Working_Storage sections to be mapped into this segment.

3.4 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.

3.5 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.

3.6 INCLUDE_LINKED_SYMBOLS

The INCLUDE_LINKED_SYMBOLS command copies the outboard symbol table into a previously defined segment file built during this linkage. All of these subcommands will be processed after all modules included in the linkage have been linked; i.e. when the outboard symbol table is complete.

INCLUDE_LINKED_SYMBOLS, POINTER=pointer name, ..
SECTION=section name

FIELD DEFAULT	DESCRIPTION
POINTER none	(CHAR 31) The externally declared variable name defined in the current linkage, of an adaptable pointer to the linked symbol table which will be initialized by this command.
SECTION none	(CHAR · 31) The section name of the segment in which the linked symbol table is to be included. The segment associated with the section name must have been allocated previously during the linkage.

3.7 END

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

4.0 LINKER FILE INTERFACE

4.0 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 object library and object file subcommands identify input object files and libraries by name and their associated attributes. The Inboard Symbol Table subcommand identifies by name input file(s) containing inboard symbol table(s).

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

The NAME_SEED parameter of the LINK_OPTIONS subcommand specifies the string used by the Linker to generate names for the output files: header, segment, and outboard symbol table files.

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).

4.1 OBJECT FILES

The Linker accepts, as input, object files containing object modules in CDC loader text format V1.4 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 (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.

4.2 LIBRARY FILES

The Linker also accepts as input MC68000 library files containing SES object modules formatted into a library by the MC68000 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.

4.3 HEADER_FILE

The Linker outputs a header file which describes the results of a linkage. The name of the header file is the concatenation of the NAME_SEED from the LINK_OPTIONS subcommand 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 section 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.

4.4 SEGMENT FILES

The Linker outputs segment files which are acceptable as input to the SES procedures TRAN68K and BLDMI68K. 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.

4.5 SYMBOL_TABLE_FILES

A symbol table file is a file conteining 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. The OST file name is the concatentation of the NAME_SEED from the LINK_OPTIONS subcommand with the string 'OST'.

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

5.0 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 LINK68K 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.

5.1 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 Section name or default section name if applicable

5.2 ENTRY POINT NAMES

The following informaton is printed for every entry point:

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

5.3 EXTERNAL REFERENCES

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

5.4 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 Section names, if any, associated with the segment

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)

6.0 LINKER PROCEDURAL INTERFACES

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

6.1 LINK68K - EXECUTE THE MC68000 ABSOLUTE LINKER

LINK68K executes the MC68000.absolute Linker, which links data from CYBIL 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 MC68000 absolute Linker subcommand 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 (V1.4). 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 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 Paremeter File specification of this field.

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
I - Section allocations, entry points, output segment and common block allocations, plus Inboard Symbol Table (full Linker map).

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

cybmlib, dioslib (Keyword, optional)
This parameter, when specified, will cause CYBMLIB

6.0 LINKER PROCEDURAL INTERFACES
6.1 LINK68K - EXECUTE THE MC68000 ABSOLUTE LINKER

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 CYBMLIB as part of the Link.

Example:

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

```
ses.link68k ofl=(lgo1,lgo2) lpf=mylpf
```

6.2 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,max_externals=500,name_seed=lnk1

object_file filename=sys default section((exec r e) (comn r w))

object_file filename=rel1 default_section((code1 r e))

object_file filename=rel2 default_section((code2 r e))

define_segment (00(16)) attributes=(rd ex wt) ALS$ORG_00000000

define_segment (30(16)) attributes=(rd ex wt) ALS$ORG_0000003C

define_segment (80(16)) attributes=(rd ex wt) ALS$ORG_00000080

define_segment (8800(16)) attributes=(rd ex) (exec code1)

define_segment (111400(16)) attributes=(rd ex) code2

define_segment (111000(16)) attributes=(rd wt) data

define_segment (8400(16)) attributes=(rd wt) comn
```

WARNING: If an ORG section is not loaded at the address specified in the assembly, problems will show up in the local symbol file when everything is taken down to the H-P development station.

6.3 EXAMPLE OF LINKER MAP

```
          LINKER V 3.0 OUTPUT LISTING     05/21/83   16.00.46

MODULE  = TEST                  LANGUAGE = ASSEMBLER
FILE    = LGO        05/21/83          16:00:06.000

SECTION TYPE/                   LOAD/
ACCESS ATTRIBUTES    LENGTH     EXECUTION ADDR
-----------------------------------------------------------
WORKING STORAGE
READ WRITE                28      000 00000000

WORKING STORAGE
READ                       8      000 00000030

CODE
READ EXECUTE              22      000 0000004C


      ENTRY POINT DEFINITIONS              ADDRESS
        ENT                    NOT GATED   000 0000004C
          LINKER V 3.0 OUTPUT LISTING     05/21/83   16.00.46

SES/MC68000 LINKER OUTPUT

         STARTING PROCEDURE = ENT1
                    ADDRESS =   000 0000004C

FILE NAME/                      LOAD/
ACCESS ATTRIBUTES    LENGTH     EXECUTION ADDR
-----------------------------------------------------------
SEGM1C1
READ WRITE                28    * 000 00000000

SEGM1C2
READ                       8    * 000 00000030

SEGM1C1
READ EXECUTE              22    * 000 0000004C

NO LINKER ERRORS WERE DETECTED
```

7.0 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.

7.1 SCL_MESSAGES

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

10000 LINKER TERMINATED NORMALLY
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 'ofi' 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.

7.0 ERROR MESSAGES
7.1 SCL MESSAGES

10200 LIBRARY FILE file_name NOT LOCAL
MEANING: File specified by the 'lfl' parameter of the LINK68K 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 LINK68K 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 LINK68K 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 LINK68K command is not a local file.
ACTION: Make the file local or remove the

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 INVALID MAP OPTION map_option SPECIFIED
MEANING: The Linker has detected an invalid map option specified by the 'mo' parameter of the LINK68K command or 'MAP_OPTIONS' specification of the Linker Parameter File 'LINK_OPTIONS' command.
ACTION: Correct the map option specification.

10303 INVALID NAME_SEED xxxx SPECIFIED
MEANING: The Linker has detected an invalid name seed as specified by the 'ns' parameter of the LINK68K command or 'NAME_SEED' specification of the Linker Parameter File 'LINK_OPTIONS' command.
ACTION: Correct the name seed specification.

10304 INVALID 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.

10306 INVALID SECTION ATTRIBUTE SPECIFIED
MEANING: The linker has detected conflicting access attributes on the specification of a default section name.
ACTION: Correct access attributes.

7.2 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.

7.0 ERROR MESSAGES
7.2 LINKER DIAGNOSTICS

7 BINDING SECTION ALIGNMENT ERROR
MEANING: The binding section must be aligned on a 16 bit MC68000 full word boundry.
ACTION: Correct object text.

8 BINDING SECTION ATTRIBUTE SPECIFICATION ERROR
MEANING: Binding sections must not be writable or executable.
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 seperate 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.

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.

21 POINTER IN BINDING SEGMENT WAS MISALIGNED
MEANING: Binding section entries must be right justified in an MC68000 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.

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

25 PREALLOCATED EXECUTABLE SEGMENT ATTRIBUTE ERROR
MEANING: Executable segments may not be writable.
ACTION: Correct DEFINE_SEGMENT command 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 SDC

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 program in one of your CYBIL modules or specify the primary entry point name on the LINK_OPTIONS command.

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 one of your modules with the 'XDCL' attribute.

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 check the file names of all members 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 SDC 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 2. To facilitate "binding", procedure offsets for indirect calls should be 0 mod 2.
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 63. 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 MC68000 Library created by the SES MC68000 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.

SES/CYBIL OBJECT TEXT FORMAT

```
{ Date request return value. }

TYPE
  ost$date = record
    CASE date_format: ost$date_formats OF
    =osc$month_date=
      month: ost$month_date, { month DD, YYYY }
    =osc$mdy_date=
      mdy: ost$mdy_date, { MM/DD/YY }
    =osc$iso_date=
      iso: ost$iso_date, { YYYY-MM-DD }
    =osc$ordinal_date=
      ordinal: ost$ordinal_date, { YYYYDDD }
    CASEND,
  recend,

  ost$date_formats = (osc$default_date, osc$month_date, osc$mdy_date,
    osc$iso_date, osc$ordinal_date),

  ost$month_date = string (18),
  ost$mdy_date = string (8),
  ost$iso_date = string (10),
  ost$ordinal_date = string (7);

{ Time request return value. }

TYPE
  ost$time = record
    CASE time_format: ost$time_formats OF
    =osc$ampm_time=
      ampm: ost$ampm_time, { HH:MM: AM or PM }
    =osc$hms_time=
      hms: ost$hms_time, { HH:MM:SS }
    =osc$millisecond_time=
      millisecond: ost$millisecond_time, { HH:MM:SS.MMM }
    CASEND,
  recend,

  ost$time_formats = (osc$default_time, osc$ampm_time, osc$hms_time,
    osc$millisecond_time),
```

```
  ost$ampm_time = string (8),
  ost$hms_time = string (8),
  ost$millisecond_time = string (12);


TYPE
  pmt$program_name = ost$name;


  CONST
    osc$max_name_size = 31,
    osc$null_name = '                                ';

  TYPE
    ost$name_size = 1 .. osc$max_name_size;

  TYPE
    ost$name = string (osc$max_name_size);

{ * amdname}
  TYPE
    amt$file_name = string ( * ),
    amt$local_file_name = ost$name;



{  CYBER 80 PPU characteristic definition. }

CONST
  llc$max_ppu_number = 20 - 1, {maximum number of PPUs in a configuration.}
  llc$max_ppu_size = 0fff(16); {maximum number of words in a PPU.}

TYPE
  llt$ppu_address = 0 .. llc$max_ppu_size;
{
{
{    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.
```

SES/CYBIL OBJECT TEXT FORMAT

```
{
{     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:
{
{         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);


  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 =
        unused: llt$section_length, {must be zero}
      = llc$libraries =
        number_of_libraries: 1 .. llc$max_libraries,
      = llc$text, llc$replication =
        number_of_bytes: 1 .. llc$max_section_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,
```

```
      llc$cybil_symbol_table_fragment, llc$symbol_table,
        llc$line_table_fragment, llc$symbol_table_fragment =
      sequence_length: llt$section_length, {REP sequence_length OF CELL}
    = llc$ppu_absolute =
      number_of_words: llt$ppu_address,
    = llc$allotted_section_definition =
      allotted_section: ost$relative_pointer, { REL ^seq(*) }
    = oct$module_directory, oct$entry_point_directory =
      number_of_directory_entries: integer,
    = llc$68000_absolute =
      number_of_68000_bytes: 1 .. llc$maximum_68000_address,
    = llc$line_table, llc$obsolete_line_table =
      number_of_line_items: 1 .. llc$max_line_adr_table_size,
    casend,
  recend;


TYPE
  llt$section_ordinal = 0 .. llc$max_section_ordinal,
  llt$section_offset = 0 .. llc$max_section_offset,
  llt$section_length = 0 .. llc$max_section_length,
  llt$section_length_in_bits = 0 .. (llc$max_section_length *
    llc$bits_per_byte),
  llt$section_address_range = - (llc$max_section_offset + 1) ..
    llc$max_section_offset;

CONST
  llc$max_section_ordinal = 0ffff(16),
  llc$max_section_offset = 7fffffff(16),
  llc$max_section_length = llc$max_section_offset + 1,
  llc$bits_per_byte = 8;

TYPE
  llt$object_record_kind = (llc$identification, llc$libraries,
    llc$section_definition, llc$text, llc$replication,
    llc$bit_string_insertion, llc$entry_definition, llc$relocation,
    llc$address_formulation, llc$external_linkage, llc$formal_parameters,
    llc$actual_parameters, llc$binding_template, llc$ppu_absolute,
    llc$obsolete_line_table, llc$cybil_symbol_table_fragment,
    llc$allotted_section_definition, llc$symbol_table, llc$transfer_symbol,
    oct$library_header, oct$module_directory, oct$entry_point_directory,
    llc$68000_absolute, llc$line_table, llc$line_table_fragment,
    llc$symbol_table_fragment);

TYPE
  llt$line_address_table_size = 0 .. llc$max_line_adr_table_size;

CONST
  llc$max_line_adr_table_size = 0ffffff(16);
```

```
  TYPE
    llt$68000_address = 0 .. llc$maximum_68000_address;

  CONST
    llc$maximum_68000_address = 0ffffffff(16);


{ NOS/180 address constants. }


  CONST


    { Ring names. }

    osc$min_ring = 1, { Lowest ring number (most privledged). }
    osc$max_ring = 15, { Highest ring number (least privledged). }
    osc$invalid_ring = 0,
    osc$os_ring_1 = 1, { Reserved for Operating System. }
    osc$tmtr_ring = 2, { Task Monitor. }
    osc$tsrv_ring = 3, { Task services. }
    osc$sj_ring_1 = 4, { Reserved for system job. }
    osc$sj_ring_2 = 5,
    osc$sj_ring_3 = 6,
    osc$application_ring_1 = 7, { Reserved for application subsystems.}
    osc$application_ring_2 = 8,
    osc$application_ring_3 = 9,
    osc$application_ring_4 = 10,
    osc$user_ring = 11, { Standard user task. }
    osc$user_ring_1 = 12, { Reserved for user...O.S. requests available.}
    osc$user_ring_2 = 13,
    osc$user_ring_3 = 14, { Reserved for user...O.S. requests not available. }
    osc$user_ring_4 = 15;


{ Virtual address space dimensions. }

  CONST
    osc$maximum_segment = 0fff(16),
    osc$maximum_offset = 7fffffff(16),
    osc$max_segment_length = osc$maximum_offset + 1;


{ Global-local key lock definition. }

  TYPE
    ost$key_lock = packed record
      global: boolean, { True if value is global key. }
```

```
    local: boolean, { True if value is local key. }
    value: ost$key_lock_value, { Key or lock value. }
  recend,

  ost$key_lock_value = 0 .. 3f(16),


  { CYBER 180 forty eight bit PVA definition. }

  ost$ring = osc$invalid_ring .. osc$max_ring, { Ring number. }
  ost$valid_ring = osc$min_ring .. osc$max_ring, { Valid Ring Number. }
  ost$segment = 0 .. osc$maximum_segment, { Segment number. }
  ost$segment_offset = - osc$maximum_offset .. osc$maximum_offset,

  ost$segment_length = 0 .. osc$max_segment_length,

  ost$relative_pointer = - 7fffffff(16) .. 7fffffff(16),

  ost$pva = packed record
    ring: ost$ring,
    seg: ost$segment,
    offset: ost$segment_offset,
  recend;


{ Identification record. }

  TYPE
    llt$identification = record
      name: pmt$program_name,
      object_text_version: string (4),
      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;

  CONST
    llc$object_text_version = 'V1.4';

  TYPE
    llt$module_kind = (llc$mi_virtual_state, llc$vector_virtual_state, llc$iou,
      llc$motorola_68000, llc$p_code, llc$motorola_68000_absolute);

  TYPE
```

SES/CYBIL OBJECT TEXT FORMAT

```
    llt$module_generator = (llc$algol, llc$apl, llc$basic, llc$cobol,
      llc$assembler, llc$fortran, llc$object_library_generator, llc$pascal,
      llc$cybil, llc$pl_i, llc$unknown_generator, llc$the_c_language, llc$ada,
      llc$real_memory_builder);

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


{  Library record. }

  TYPE
    llt$libraries = array [ 1 .. * ] of amt$local_file_name;


{ Section definition record. }

  TYPE
    llt$section_definition = record
      kind: llt$section_kind,
      access_attributes: llt$section_access_attributes,
      section_ordinal: llt$section_ordinal,
      length: llt$section_length,
      allocation_alignment: llt$section_address_range,
      allocation_offset: llt$section_address_range,
      name: amt$program_name,
    recend;

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

  TYPE
    llt$section_access_attributes = set of llt$section_access_attribute,

    llt$section_access_attribute = (llc$read, llc$write, llc$execute,
      llc$binding);


{ Text record. }

  TYPE
    llt$text = record
      section_ordinal: llt$section_ordinal,
      offset: llt$section_offset,
      byte: array [ 1 .. * ] of 0 .. 255,
    recend;
```

```
{ Replication record. }

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


{ Bit insertion record. }

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

    llt$bit_string_length = 1 .. llc$max_bit_string_length;

  CONST
    llc$max_bit_string_length = 63;


{ Address formulation record. }

  TYPE
    llt$address_formulation = record
      value_section: llt$section_ordinal,
      dest_section: llt$section_ordinal,
      item: array [ 1 .. * ] of llt$address_formulation_item,
    recend,

    llt$address_formulation_item = record
      kind: llt$internal_address_kind,
      value_offset: llt$section_address_range, { only llc$address can be negative. }
      dest_offset: llt$section_offset,
    recend;

  TYPE
    llt$address_kind = (llc$address, llc$internal_proc, llc$short_address,
      llc$external_proc, llc$address_addition, llc$address_subtraction);
```

```
  TYPE
    llt$internal_address_kind = llc$address .. llc$external_proc;


{ External reference record. }

  TYPE
    llt$external_linkage = record
      name: pmt$program_name,
      language: llt$module_generator,
      declaration_matching_required: boolean,
      declaration_matching_value: integer,
      item: array [ 1 .. * ] of llt$external_linkage_item,
    recend,

    llt$external_linkage_item = record
      section_ordinal: llt$section_ordinal,
      offset: llt$section_offset,
      kind: llt$address_kind,
      offset_operand: llt$section_address_range,
    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;

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


{ Relocation record. }

  TYPE
    llt$relocation = array [ 1 .. * ] of llt$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;

  TYPE
    llt$relocation_container = (llc$two_bytes, llc$three_bytes, llc$four_bytes,
      llc$eight_bytes, llc$180_d_field, llc$180_q_field, llc$180_long_d_field);

  TYPE
    llt$address_type = (llc$byte_positive, llc$two_byte_positive,
      llc$four_byte_positive, llc$eight_byte_positive, llc$byte_signed,
      llc$two_byte_signed, llc$four_byte_signed, llc$eight_byte_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;

  TYPE
    llt$source_line_number = 0 .. 999999;


{ 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$fortran_array_size, { only used for kind ARRAY }
      unknown_argument_ordinal: 1 .. llc$max_fortran_arguments, { only used }
```

```
    { for actual argument kind of UNKNOWN. Points back to formal parameter }
    { passed on by this call. }
    mode: llt$argument_usage,
  recend;

CONST
  llc$max_fortran_arguments = 500;


TYPE
  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$fortran_null_type,
    llc$fortran_statement_label);

TYPE
  llt$fortran_string_length = record
    attributes: llt$fortran_string_attributes,
    number_of_characters: llt$fortran_string_size,
  recend;

TYPE
  llt$fortran_string_size = 0 .. llc$max_fortran_string_size;

TYPE
  llt$fortran_string_attributes = set of llt$fortran_string_attribute,

  llt$fortran_string_attribute = (llc$fortran_assumed_len_string,
    llc$fsa_reserved_7, llc$fsa_reserved_6, llc$fsa_reserved_5,
    llc$fsa_reserved_4, llc$fsa_reserved_3, llc$fsa_reserved_2,
    llc$fsa_reserved_1);

CONST
  llc$max_fortran_string_size = 0ffff(16);

TYPE
  llt$fortran_argument_kind = (llc$fortran_variable, llc$fortran_array,
    llc$fortran_external, llc$fortran_array_element,
    llc$fortran_unknown_arg_kind);

TYPE
  llt$fortran_array_size = record
    attributes: llt$fortran_array_attributes,
    rank: llt$fortran_array_rank,
    number_of_elements: llt$section_length,
  recend;

TYPE
  llt$fortran_array_attributes =set of llt$fortran_array_attribute,
```

```
    llt$fortran_array_attribute = (llc$fortran_assumed_len_array,
      llc$fortran_adaptable_array, llc$faa_reserved_6, llc$faa_reserved_5,
      llc$faa_reserved_4, llc$faa_reserved_3, llc$faa_reserved_2,
      llc$faa_reserved_1);

  TYPE
    llt$fortran_array_rank = 0 .. llc$max_fortran_array_rank;

  CONST
    llc$max_fortran_array_rank = 7;


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


{ Binding template record }

  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_address_range,
        internal_address: llt$internal_address_kind,
      = llc$external_reference =
        name: pmt$program_name,
        address: llt$address_kind,
      casend,
    recend;

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

{ Symbol table record }

  TYPE
    llt$symbol_table = record
      language: llt$module_generator,
      text: SEQ ( * ),
    recend;


{ Debug table record used for emitting line tables and symbol tables }
{ in fragments rather than all together.  Not used by II compilers and }
{ simply passed over by any object text processors operating on NOS/VE. }
```

SES/CYBIL OBJECT TEXT FORMAT

```
{ Intended for use by compilers producing this loader text on machines }
{ other than 180.  For example CYBIL C/M. }

  TYPE
    llt$debug_table_fragment = record
      offset: llt$section_offset,
      text: SEQ ( * ),
    recend;



{ Transfer record. }

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


{ PPU absolute record. }

  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 [ 0 .. * ] of 0 .. 0ffff(16),
    recend;

  TYPE
    llt$68000_absolute = record
      load_address: llt$68000_address,
      transfer_address: llt$68000_address,
      text: SEQ ( * ), { REP n OF byte }
    recend;
```

Table of Contents

1.0 PREFACE 1-1
1.1 SCOPE 1-1
1.2 APPLICABLE DOCUMENTS 1-2
1.3 TERMINOLOGY 1-2
1.4 DEFICIENCIES AND LIMITATIONS 1-2

2.0 ACCESSING THE LINKER 2-1

3.0 LINKER PARAMETER FILE (LPF) 3-1
3.1 LINK_OPTIONS 3-1
3.2 OBJECT_FILE/OBJECT_LIBRARY 3-3
3.3 DEFINE_SEGMENT 3-4
3.4 OBJECT_MODULE 3-5
3.5 INBOARD_SYMBOL_TABLE 3-5
3.6 INCLUDE_LINKED_SYMBOLS 3-5
3.7 END 3-6

4.0 LINKER FILE INTERFACE 4-1
4.1 OBJECT FILES 4-2
4.2 LIBRARY FILES 4-3
4.3 HEADER FILE 4-4
4.4 SEGMENT FILES 4-5
4.5 SYMBOL TABLE FILES 4-6

5.0 LINKER MAP 5-1
5.1 SECTION DEFINITIONS 5-1
5.2 ENTRY POINT NAMES 5-1
5.3 EXTERNAL REFERENCES 5-1
5.4 OUTPUT SEGMENTS AND COMMON BLOCKS 5-2

6.0 LINKER PROCEDURAL INTERFACES 6-1
6.1 LINK68K - EXECUTE THE MC68000 ABSOLUTE LINKER 6-1
6.2 EXAMPLE OF LINKER PARAMETER FILE 6-4
6.3 EXAMPLE OF LINKER MAP 6-5

7.0 ERROR MESSAGES 7-1
7.1 SCL MESSAGES 7-1
7.2 LINKER DIAGNOSTICS 7-4

Appendix Object Text.Definitions . . . A-1

SES/CYBIL OBJECT TEXT FORMAT A1