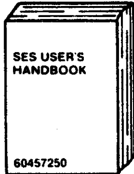
60462500

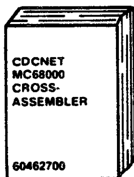
CDCNET
MC68000 UTILITIES

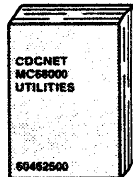
RELATED PUBLICATIONS

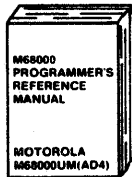
Background (Access as Needed):



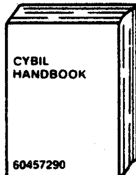
Software Development:

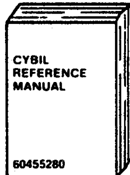


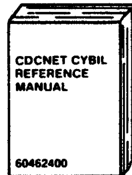




CYBIL References:







MANUAL HISTORY

This manual is Revision 01, printed 10/84. It is the Preliminary Release under NOS Version 2.

© 1984

Control Data Corporation. All rights reserved.
Printed in the United States of America.

CONTENTS

HOW TO USE THIS MANUAL 5
    Organization 5
    Audience 5

INTRODUCTION 1-1

SYMBOL MODULE PURGER 2-1

OBJECT MODULE BINDER 3-1

DI LIBRARY TRANSLATOR 4-1

ABSOLUTE LINKER 5-1
    Linker Parameter File 5-3
    LINK_OPTIONS Subcommand 5-3
    OBJECT_FILE and OBJECT_LIBRARY
      Subcommands 5-5
    DEFINE_SEGMENT Subcommand 5-6
    OBJECT_MODULE Subcommand 5-7
    INBOARD_SYMBOL_TABLE Subcommand .. 5-7
    INCLUDE_LINKED_SYMBOLS Subcommand 5-8
    END Subcommand 5-8
    Linker File Formats 5-9
    Input Files 5-9
    Output Files 5-10

OBJECT RECORD TRANSLATOR 6-1

MEMORY IMAGE BUILDER 7-1

DIAGNOSTIC MESSAGES A-1
    Linker Map File Diagnostic
      Messages A-1
    Linker Dayfile Messages A-6
    MC68000 Utility Dayfile Messages . A-8

OBJECT TEXT FORMATS FOR THE MC68000
  ABSOLUTE LINKER B-1

ABOUT THIS MANUAL

This manual describes the Motorola 68000 Utilities for CDCNET software development. These utilities are part of the Software Engineering Services (SES) tools package available under the Network Operating System (NOS).

AUDIENCE

This manual is written for the CDCNET system programmer. It assumes that the reader is familiar with the CYBIL programming language as described in the CDCNET CYBIL Reference Manual, and the Motorola MC68000 Cross-Assembler as described in the CDCNET MC68000 Cross-Assembler manual.

ORGANIZATION

Chapter 1 of this manual shows how the utilities described in this manual are used in the CDCNET software development process. Each of the remaining chapters describes one of the MC68000 utilities.

The appenxides contain supplementary information. Appendix A describes all of the diagnostic messages associated with the utilities. Appendix B presents the object text formats for the MC68000 Absolute Linker; this appendix is intended for programmers who wish to use object program modules other than those created by the CDCNET CYBIL compiler or the MC68000 Cross-Assembler as input to the Linker.

CONVENTIONS

Command formats in this manual follow the conventions generally used in NOS manuals for presenting NOS system commands.

All of the commands used to call the utilities are actually calls to a NOS procedure. The procedure allows you to use an order-dependent or an order-independent format.

- If you use the order-dependent format, you do not have to specify the reserved words that introduce parameter values in the commands. You must include a separator (comma or space) between each parameter you specify. If you omit a parameter from the parameter list and wish to specify a parameter that occurs later in the parameter list, you must include an additional separator for the parameter you omitted.

- If you use the order-independent format, you must specify the reserved words that introduce the parameter values. You must include a separator between each parameter you specify, but you do not need to include additional separators for parameters you do not specify.

Uppercase letters in the command formats represent reserved words; you must enter them exactly as shown. Lowercase letters indicate names and values that you supply.

Optional parameters are labeled as optional in the parameter list. You must specify values for all parameters that are not labeled as optional.

Permissible abbreviations and alternate forms of parameters are shown in parentheses nex to the parameter definitions.

The lowercase descriptions of parameters in the parameter lists give an indication of the type of the parameter. In general, the words are used in the same sense as in CYBIL (for example, name refers to a string of up to 31 characters). The following terms that are not standard CYBIL terms are also used:

- The term file name refers to a NOS file name.
- The term user name refers to a NOS user name.
- The term number refers to a decimal integer or a hexadecimal integer followed by the integer 16 in parentheses.

INTRODUCTION 1

The CDCNET MC68000 utilities are a set of software tools that allow you to create software for CDCNET device interfaces (DIs). These utilities are part of the Software Engineering Services (SES) tools package available under the Network Operating System (NOS).

Figure 1-1 shows how the utilities interact to process program modules written using the CDCNET CYBIL compiler or the MC68000 Cross-Assembler. The commands for the utilities are shown in parentheses. This figure shows three major paths in preparing program modules for a DI.

- Path (1) uses the DI Library Translator utility (TRANDILIB). This path is used for creating a relocatable object library to be loaded into a DI.
- Path (2) uses the Absolute Linker (LINK68K) and the Memory Image Builder (BLDMI68K). This path is used to create absolute memory images of program modules to be loaded into a DI.
- Path (3) uses the Absolute Linker (LINK68K) and the Object Record Translator (TRAN68K). This path is used in creating read-only memories (ROMs) to be installed in a DI.

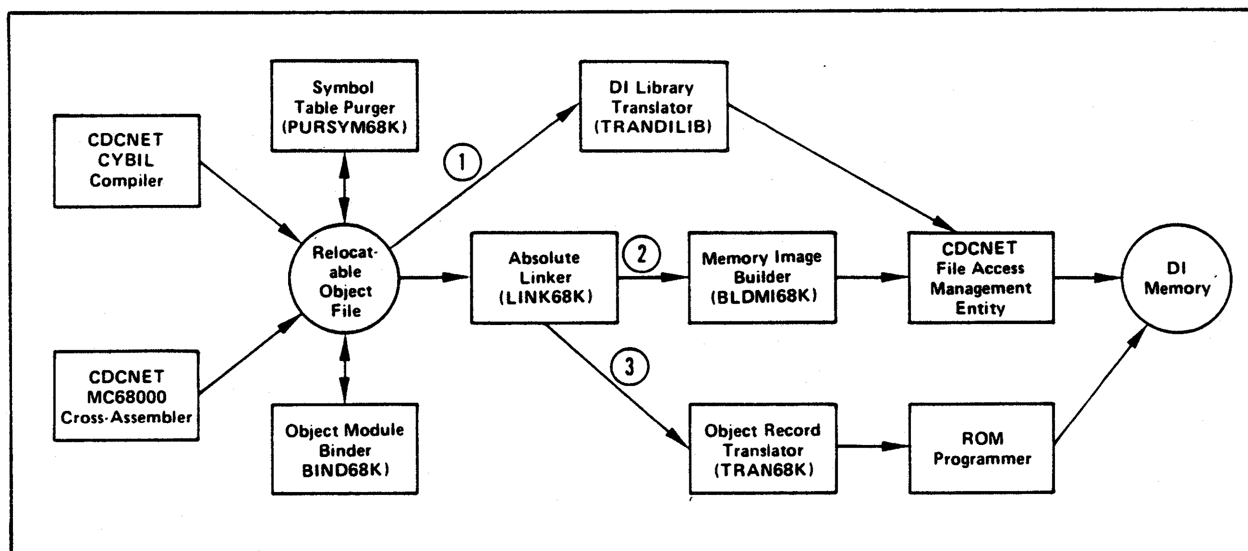


Figure 1-1. Interaction of the CDCNET MC68000 Utilities

The Symbol Table Purger (PURSYM68K) and the Object Module Binder (BIND68K) can be used in all paths.

- The Symbol Table Purger is used when the program modules generated by the CDCNET CYBIL compiler or the MC68000 Cross-Assembler have been created using the debug option. The Symbol Table Purger removes the debug symbol table from the file containing the program modules. The debug symbol table must be removed in path 1 before the DI Library Translator is used.

- The Object Module Binder reduces the size of an object library by combining code, data, and binding sections, by optimizing the combined binding section for minimum redundancy, and by removing unnecessary entry points. The use of this utility is always optional.

The SES Object Code Utilities, not shown in the figure, can be used to manipulate the object files at any point in the development of DI software. These utilities can be used to create object libraries from object files. The SES Object Code Utilities are described in the SES User's Handbook.

SYMBOL TABLE PURGER 2

The Symbol Table Purger utility (PURSYM68K) removes the debug symbol tables from program modules on object files or object libraries. A debug symbol table is generated when the program module is created using the CDCNET CYBIL compiler or the CDCNET MC68000 Cross-Assembler using the debug option. If you intend to make a relocatable object library from program modules that contain debug symbol tables, you must remove these tables before you use the DI Library Translator utility.

Format:

```
        SES.PURSYM68K
        INPUT=list of file name
        OUTPUT=list of file name
        UN=user name
optional
```

Parameters:

INPUT (I)

Names of the files from which debug symbol tables are to be removed. These files need not be local. The files can be object files or object libraries.

OUTPUT (O)

Names of the files to which the output is to be written. These files are written into the permanent file catalog of the user specified in the UN parameter. For all input files that are object files, the output files are object files; for all input files that are object libraries, the output files are object libraries.

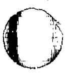
The number of files specified in the OUTPUT parameter must be the same as the number of files specified in the INPUT parameter.

UN

The user name of the permanent file catalog to be searched for input files and to which the output files are to be written. If you omit this parameter, your permanent file catalog is used.

Example: In the following example, debug sysmol tables are removed from the program modules on files EXECUTV and MONITOR, and the resulting modules are written to files EXEC and MNTR respectively.

```
SES.PURSYM68K (EXECUTV MONITOR) (EXEC MNTR)
```



OBJECT MODULE BINDER 3

The Object Module Binder utility (BIND68K) creates a single bound load module from input object modules. The code sections of the input modules are combined into a single code section, data sections with identical attributes are combined into a single section, and the binding sections are combined and reorganized for minimum redundancy. Entry point definitions that are referenced from one or more component modules are deleted unless they are explicitly retained in the BIND68K command or if they have been previously retained.

Format:

```
SES.BIND68K
FILE=list of (file name or (list of file name,user name))
NAME=name
MODULE=list or range of name                   optional
RETAIN=list of name                             optional
OMIT=list of name                               optional
STARTING_PROCEDURE=name                         optional
UPON=file name or (file name,user name)
BASE=file name or (file name,user name)         optional
LOCK=file name                                  optional
NOLOCK                                          optional
OUTPUT=file name                                optional
LPF=file name                                   optional
```

Parameters:

FILE (LIBRARY, LIB, F)

The names of the files containing program modules to be bound. Files that are not in your permanent file catalog must be referenced in the format (list of file name,user name).

NAME (N)

The name to be associated with the output module.

MODULE (MODULES, M)

The modules that are to be components of the output module. You can specify subranges of modules in the list of modules. If this parameter is omitted, all modules on the input files are used.

RETAIN (R)

The externally declared names in the component modules whose definitions are to be retained in the output module. The externally declared names are referenced by a component of the created module. The Object Module Binder automatically retains entry point names that have been retained previously.

OMIT (O)

The externally declared names in the component modules whose definitions are to be removed in creating the output module.

STARTING_PROCEDURE (SP)

The externally declared name of the procedure at which execution of the output module is to begin. If you omit this parameter, the last transfer symbol encountered is used to identify the starting procedure. The starting procedure is always retained in the new module.

UPON (UP)

The name of the file to which the output module is to be written. If you want the output to be written to another user's permanent file catalog, use the format (file name,user name) for this parameter.

BASE (B)

The name of a file of object code modules to which the output file is to be added. If this parameter is specified, the SES Object Code utilities GOF and GOL are used to append the output file to the base file. If you want the output to be written to another user's permanent file catalog, use the format (file name,user name) for this parameter.

LOCK

This parameter specifies that the file specified in the UPON parameter is to be interlocked while the Object Module Binder is writing to that file.

NOLOCK

This parameter indicates that the file specified in the UPON parameter is not interlocked while the Object Module Binder is writing to that file.

OUTPUT (OUT)

The name of the file to which an object map of the output module is to be written.

LPF

The name of the file containing a list of parameters for the Object Module Binder. The FILE, MODULE, OMIT, RETAIN, NAME, and STARTING_PROCEDURE parameters can be specified in this file. The parameter file must contain one parameter per line, and the last line must consist of the word END. This file must be local to your job.

Examples: In the following example, the input program modules are located on file MYLIB, which is either local or in the user's own permanent file catalog, and files SIN and TAN, which are in the permanent file catalog of user name FTNLIB. The Object Module Binder binds all program modules in the input files. The output module, named TRIG_FUNCT, has a starting procedure named START, and is written to file BOUND in the user's permanent file catalog.

```
SES.BIND68K F=(MYLIB,(SIN,TAN,FTNLIB)) M=TRIG_FUNCT SP=START UP=BOUND
```

In the following example, some of the parameters for the Object Module Binder are contained in local file BINDLPF. The input program modules are located on files FILE1 and FILE2, which are either local or in the user's own permanent file catalog. The Object Module Binder only binds program modules MOD1 through MOD4. Entry points ENT1 and ENT2 are retained in the binding process. The output module, named NEWMOD, has a starting procedure named TEST, and is written to file FILE3 in the user's permanent file catalog. An object map of program module NEWMOD is written to file LIST.

```
SES.BIND68K LPF=BINDLPF OUT=LIST UPON=FILE3
```

File BINDLPF contains:

```
FILE=(FILE1,FILE2)
MODULE=(MOD1..MOD4)
RETAIN=(ENT1,ENT2)
NAME=NEWMOD
SP=TEST
END
```

DI LIBRARY TRANSLATOR 4

The DI Library Translator utility (TRANDILIB) translates program modules on object files or object libraries into packed byte-oriented data mappings to create a relocatable object library that can be loaded into a DI.

Format:

```
SES.TRANDILIB
     INPUT=list of file name
     OUTPUT=list of file name
     UN=user name                          optional
     keyword                               optional
```

Parameters:

INPUT (I)

The names of the files to be translated. These files need not be local to your job.

OUTPUT (O)

The names of the files to which the translated files are to be written. The number of output files must be the same as the number of input files.

UN

The user name whose permanent file catalog is to be searched for input files and to which output files are to be written. If you omit this parameter, your permanent file catalog is used.

FILE (F)

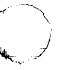
If you specify this keyword, or if you specify no keyword, the output files are formatted as files rather than as libraries.

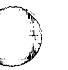
LIB (L)

If you specify this keyword, the output files are formatted as libraries. Each library file contains a program module directory and an entry point directory in addition to the program modules.

Example: In the following example, the program modules on files OBJFIL1 and OBJFIL2 are translated and written to files OBJLIB3 and OBJLIB4 in the user's permanent file catalog. Files OBJLIB3 and OBJLIB4 are written in library format.

```
SES.TRANDILIB I=(OBJFIL1 OBJFIL2) O=(OBJLIB3 OBJLIB4) L
```





MC68000 ABSOLUTE LINKER (LINK68K) 5

The MC68000 Absolute Linker (the Linker) creates a set of linked segment files from input object modules created by the CDCNET CYBIL compiler or the MC68000 Cross-Assembler. These linked segment files can be input to the Object Record Translator or Memory Image Builder utilities to create software for a DI.

The Linker forms a set of memory segment images from the object text sections of the input object modules. Data in the memory segments is primarily referenced via pointers set up in the binding segment, which is initialized during the link operation. Unsatisfied externals are satisfied from libraries specified in the call to the Linker.

You execute the Linker with the LINK68K command, which calls an SES procedure. Parameters for the call to the Linker can be specified in the LINK68K command, in a Linker Parameter File, or in both places. Values that you specify for parameters in the LINK68K command take precedence over values you specify in the Linker Parameter File.

Format:

```
SES.LINK68K
    OFL=list of file name      optional
    LFL=list of file name      optional
    SP=name                    optional
    NS=name                    optional
    MF=file name               optional
    MO=character               optional
    REWIND or NOREW            optional
    LPF=file name              optional
    CYBMLIB                    optional
```

Parameters:

OFL

The names of up to 10 files containing input object modules for the Linker.

LFL

The names of up to 10 library files containing input object modules for the Linker.

SP

The starting procedure for the linked modules. This parameter specifies the entry point at which execution of the linked modules is to begin. If you omit this parameter, execution begins at the first transfer symbol encountered.

NS

The four-character name seed for the files created by the Linker. The name seed is used as the first four characters of the NOS file names for the header file, output segment files, and the outboard symbol table file. This field is ignored if the NS parameter is specified on the LINK68K command. If you omit this parameter, the characters SEGM are used as the name seed.

MF

The name of the Linker map file. If you omit this parameter, the map file is written to file LINKMAP.

MO

This parameter specifies the amount of information to be included in the Linker map file. Values for this parameter are:

N   No map information.

S   Section allocations for each section of each input object module.

E   Section allocations, entry points names and address assignments.

M   Section allocations, entry points, and output segment and common block allocations.

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

REWIND

If you specify this parameter, the Linker map file is rewound before it is written. If you specify neither REWIND nor NOREW, the Linker map file is rewound.

NOREW

If you specify this parameter, the Linker map file is not rewound before it is written.

LPF

The name of a file containing a list of parameters for the Linker (see Linker Parameter File later in this chapter). If you omit this parameter, the default values for parameters described under Linker Parameter File are used.

CYBMLIB (DIOSLIB)

If you specify this parameter, the Linker uses SES library file CYBMLIB to satisfy external references during the linking process. The Linker accesses this file and adds it to the library file list. If you omit this parameter, CYBMLIB is not used to satisfy external references.

Example: In the following example, the object modules on files LG01 and LG01 are input to the Linker, and the remaining parameters for the Linker are specified in file MYLPF.

```
SES.LINK68K OFL=(LG01,LG02) LPF=MYLPF
```

LINKER PARAMETER FILE

A Linker parameter file is a file of subcommands that control the operation of the Linker. The Linker parameter file allows you to specify a number of parameters that can not be included in the LINK68K command. You specify the LPF parameter in the LINK68K command to indicate that you are including a Linker parameter file. The Linker parameter can contain the following subcommands:

- LINK_OPTIONS
- OBJECT_FILE
- OBJECT_LIBRARY
- DEFINE_SEGMENT
- OBJECT_MODULE
- INBOARD_SYMBOL_TABLE
- INCLUDE_LINKED_SYMBOLS
- END

The subcommands must be on separate lines in the Linker parameter file. If you need to continue a subcommand on a second or subsequent line, you must end every line in the subcomand except the last line with two periods (..) to indicate that a continuation line follows.

LINK_OPTIONS Subcommand

The LINK_OPTIONS subcommand provides parameters for the LINK68K command. With the exception of the MAX_EXTERNALS and HEAP_SIZE parameters, all parameters for this subcommand can be specified in the LINK68K command. Any values specified for these parameters in the LINK68K command override the values specified in the LINK_OPTIONS subcommand.

Only one LINK_OPTIONS subcommand may be included in the Linker parameter file. If you omit this subcommand, values for parameters not specified in the LINK68K command are the same as the values used when the individual parameters are omitted.

Format: LINK_OPTIONS.&MAP_FILENAME=file name,&MAP_OPTIONS=character,&REWIND_MAP,..
&NAME_SEED=name,&MAX_EXTERNALS=integer,&HEAP_SIZE=integer,&STARTING_PROCEDURE=name

Parameters:

MAP_FILENAME

The name of the Linker map file. If you omit this parameter, the map file is written to file LINKMAP.

MAP_OPTIONS

This parameter specifies the amount of information to be included in the Linker map file. Values for this parameter are:

N No map information.

S Section allocations for each section of each input object module.

E Section allocations, entry points names and address assignments.

M Section allocations, entry points, and output segment and common block allocations.

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

REWIND_MAP

If you specify this parameter, the Linker map file is rewound before it is written. If you specify neither REWIND_MAP nor NO_MAP_REWIND, the Linker map file is rewound.

NO_MAP_REWIND

If you specify this parameter, the Linker map file is not rewound before it is written.

NAME_SEED

The four-character name seed for the files created by the Linker (the header file, the segment files, and the Outboard Symbol Table file). The name seed is used as the first four characters of the NOS file names for the header file, output segment files, and the outboard symbol table file. This field is ignored if the NS parameter is specified on the LINK68K command. If you omit this parameter, the characters SEGM are used as the name seed.

MAX_EXTERNALS

The maximum number of external references to be allowed in the link operation. If you omit this parameter, a value of 300 is used.

HEAP_SIZE

The size, in bytes, of the system heap for object modules generated by the CDCNET CYBIL compiler. This value overrides the value specified in the object modules. If you omit this parameter, the heap sizes specified in the individual object modules are used for each module; if no object module specifies a heap size, a value of zero is used. For further information about the system heap, refer to the CDCNET CYBIL Reference Manual.

STARTING_PROCEDURE

The starting procedure for the linked modules. This parameter specifies the entry point at which execution of the linked modules is to begin. If you omit this parameter, execution begins at the first transfer symbol encountered.

OBJECT_FILE and OBJECT_LIBRARY Subcommands

The OBJECT_FILE and OBJECT_LIBRARY subcommands specify the input files to the Linker and provides names for sections of the input files. Any values specified for the OFL and LFL parameters in the LINK68K command override the values specified in the OBJECT_FILE and OBJECT_LIBRARY subcommands. You should include one OBJECT_FILE subcommand for each input file that is not in library format and one one OBJECT_LIBRARY subcommand for each input file that is in library format.

Format:

```
OBJECT_FILE,FILENAME=file name,DEFAULT_SECTION=list of (name,list of character)

OBJECT_LIBRARY,FILENAME=file name,DEFAULT_SECTION=list of (name,list of character)
```

Parameters:

FILENAME

Name of the local file containing input object modules for the Linker. This file must be local to your job.

DEFAULT_SECTION

The names to be associated with any unnamed sections of the input file, and the attributes to be assigned to those sections. Values for the section attributes are:

R Read.

W Write.

E Execute.

DEFINE_SEGMENT Subcommand

The DEFINE_SEGMENT subcommand allows you to define an absolute program segment. You can include any number of DEFINE_SEGMENT subcommands in the Linker parameter file. You must specify either the LOAD_ADDRESS parameter or the EXECUTE_ADDRESS and ATTRIBUTES parameters for each DEFINE_SEGMENT subcommand.

Format:

```
DEFINE_SEGMENT,LOAD_ADDRESS=(number),EXECUTE_ADDRESS=(number),..
ATTRIBUTES=(list of name),SECTION_NAME=(list of name)
```

Parameters:

LOAD_ADDRESS

The absolute address (or byte offset) at which the segment is to be loaded. This number may be a decimal integer, or a hexadecimal integer in the format (hexadeximal integer(16)).

EXECUTE_ADDRESS

The absolute address (or byte address) at which the segment is to executecifies where the segment will ultimately execute. This number may be a decimal integer, or a hexadecimal integer in the format (hexadeximal integer(16)). If you specify neither LOAD_ADDRESS nor EXECUTE_ADDRESS, offset loading is not performed.

ATTRIBUTES

The access attributes for the segment. You can specify any combination of the following attributes:

RD Read.

WT Write.

EX Execute.

ET Extend.

SECTION_NAME

The names of Working_Storage sections to be mapped into this section.

OBJECT_MODULE Subcommand

The OBJECT_MODULE subcommand specifies names of modules to be included in the link operation. You may include only one OBJECT_MODULE subcommand in the Linker parameter file.

Format: OBJECT_MODULE,NAME=list of name

Parameters:

NAME Names of modules to be included in the link operation.

INBOARD_SYMBOL_TABLE Subcommand

The INBOARD_SYMBOL_TABLE subcommand specifies files that contain Inboard Symbol Tables to be passed to the Linker. An Inboard Symbol Table is a table of entry points resolved in a previous link operation. An Inboard Symbol Table allows you to link only the code that will be executed. You may include only one INBOARD_SYMBOL_TABLE subcommand in the Linker parameter file.

Format: INBOARD_SYMBOL_TABLE,NAME=list of file name

Parameters:

NAME Names of files containing Inboard Symbol Tables.

INCLUDE_LINKED_SYMBOLS Subcommand

The INCLUDE_LINKED_SYMBOLS subcommand copies the Outboard Symbol Table into a segment file defined in a DEFINE_SEGMENT subcommand for the current link operation. This subcommand is processed after the link operation is complete.

Format: INCLUDE_LINKED_SYMBOLS,POINTER=name,SECTION=name

Parameters:

POINTER

The name of a pointer to the adaptable array to which the Outboard Symbol Table is to be written. This subcommand causes the adaptable pointer to be initialized. The variable name must be defined in the current link operation.

SECTION

The section name of the segment in which the linked symbol table is to be included. You must define this segment with a DEFINE_SEGMENT subcommand in the current link operation.

END Subcommand

The END subcommand indicates end of the Linker parameter file. If you omit this subcommand, the Linker uses the end-of-information of the file to determine the end of the Linker parameter file.

Format: END

The following example shows a call to the Linker that uses a Linker parameter file. The Linker parameter file specifies three input object files and defines seven object segments.

```
SES.LINK68K LPF=LPARAM1
```

File LPARAM1 contains:

```
LINK_OPTIONS,MAX_EXTERNALS=500,NAME_SEED=LNK1
OBJECT_FILE FILENAMR=SYS DEFAULT SECTION((EXEC R R) (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_00000030
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
END
```

LINKER FILE FORMATS

The following sections describe the formats of the files that are input to the Linker and the output files created by the the Linker.

INPUT FILES

The files that constitute the input to the Linker are object files, object libraries, and files containing Inboard Symbol Tables. You must specify at least one object file or object library as input to the Linker; an Inboard Symbol Table is not required.

Object Files

Object files contain object modules created by the CDCNET CYBIL compiler or the CDCNET MC68000 Cross-Assembler. Several object modules may reside on a single object file.

Each object module consists of an identification record, a section definition record, and the object text associated with the sections. The identification record describes the external characteristics of the object module (module attributes and the number of sections). Each section in the object module contains a section definition record that describes the attributes of the section (code, binding, working storage, or common).

Appendix B contains the object module type definition for object files that are input to the Linker.

Library Files

In addition to object files, the Linker accepts MC68000 library files as input. MC68000 library files contain object modules formatted into a library by the SES Object Code Utilities (refer to the SES User's Handbook for a description of these utilities). The object module structure for library files is identical to the object module structure for object files.

Appendix B contains the library record definition for library files that are input to the Linker.

Inboard Symbol Table Files

The Inboard Symbol Table file contains a table of gated entry points created in one execution of the Linker. A symbol table is called an Inboard Symbol Table when used as input to the Linker; the same symbol table is called an Outboard Symbol Table when it is created by the Linker. An Outboard Symbol Table file is generated by the Linker if entry points that have the gated attribute are encountered during a link operation. Gated entry points are associated with gated variables, functions, and procedures in CYBIL object modules (refer to the CDCNET CYBIL Reference Manual).

OUTPUT FILES

The Linker creates a header file and a number of segment files during each successful link operation. Depending on the parameters specified in the call to the Linker, an Outboard Symbol Table and a Linker map may also be created.

Header File

The Linker creates a header file that describes the results of the link operation. The name of the header file is the name seed, specified in the NAME_SEED parameter, followed by the characters HDR. For example, if the name seed is SEGM, the header file is named SEGMHDR.

The header file contains a header variant and a segment descriptor variant for each segment file created in the link operation. The header variant contains the number of segment descriptors, the initial program address and its key, and the binding section address. The segment descriptor contains the name of the file on which the segment was written and its segment attributes.

Segment Files

The Linker outputs segment files that can be input to the Memory Image Builder and Object Record Translator utilities. A segment file contains a load file directory and a linked segment.

The names of the segment files are generated by adding a three-digit number to the name seed specified in the NAME_SEED parameter. A unique three-digit number, starting with 101, is used for each segment file. For example, if the name seed is SEGM and four segment files are created, their names are SEGM101, SEGM102, SEGM103, and SEGM104.

The Linker allocates and creates segment files during a link operation. You can explicitly allocate a segment using the DEFINE_SEGMENT subcommand in the Linker parameter file.

Outboard Symbol Table File

The Outboard Symbol Table file contains a table of gated entry points created in one execution of the Linker. An Outboard Symbol Table file is generated by the Linker if entry points that have the gated attribute are encountered during a link operation. Gated entry points are associated with gated variables, functions, and procedures in CYBIL object modules (refer to the CDCNET CYBIL Reference Manual). The Outboard Symbol Table file name is the name seed specified in the NAME_SEED parameter followed by the string OST.

Linker Map

The Linker map shows the address assignments made by the Linker. You specify the name of the file onto which the linker map is written either as a parameter in the LINK68K command or as parameter of the LINK_OPTIONS subcommand in the Linker Parameter File. The linker map contains four parts; you have the option of selecting, all, some, or none of these parts to be included in the Linker map. The four parts are:

- Section Definitions
- Entry Point Names
- External References
- Output Segments and Common Blocks

Section Definitions

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

- Section type (working storage or code).
- Access attributes (read, write, or execute).
- Length, in bytes.
- Address (load address and execution address if different).
- Section name or default section name if applicable.

Entry Point Names

The following information is printed for every entry point:

- Name.
- An indication as to whether the entry point is gated or not gated.
- Address (load address and execution address if different).

External References

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

Output Segments and Common Blocks

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

- File name.
- Address (load address and execution address if different).
- Length, in bytes.
- Access attributes (read, write, or execute).
- Section names associated with the segment.

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

- Name.
- Access attributes.
- Length, in bytes.
- Address (load address and execution address if different).

Figure 5-1 shows an example of a full Linker map. The section definitions portion of the map shows that the three input object modules T01MONT, T01CYBM, and T01ASM1 were involved in a link operation. The entry point names portion shows that a single entry point TEST was present. The absence of an external references portion indicates that no external references were encountered. The output segments and common blocks section shows that the four segment modules T001101, T001102, T001103, and T001104 were created by the Linker.

```
                    LINKER V 4.1 OUTPUT LISTING      9/24/84      16.00.46

MODULE   = T01MONT                         LANGUAGE = ASSEMBLER
FILE     = T01MONR          July 11, 1984    5:12 PM


SECTION TYPE/                                LOAD/
ACCESS ATTRIBUTES                   LENGTH   EXECUTION ADDR
--------------------------------------------------------------------------------------------
WORKING STORAGE - ALS$ORG_00000000
READ WRITE EXECUTE                      30     000 000000000

WORKING STORAGE - ALS4ORG_0000007C
READ WRITE EXECUTE                       4     000 00000007C

WORKING STORAGE - ALS4ORG_0000007C
READ WRITE EXECUTE                       4     000 00000008C

CODE - MONR                              - PROG
READ EXECUTE                           578     000 000000400


       ENTRY POINT DEFINITIONS                        ADDRESS
         MONITOR_CONTROL                  NOT GATED   000 0000086C
         MONITOR_REGISTE                  NOT GATED   000 00000802
         XFR_BUF                          NOT GATED   000 00000878
         MONITOR_MESSAGE                  NOT GATED   000 00000850
         MONITOR_ENTRY                    NOT GATED   000 00000400
         RESET_ENTRY                      NOT GATED   000 00000634
         SPECIAL_ENTRY                    NOT GATED   000 00000498


---- TOTAL MODULE LENGTH ----          580
```

Figure 5-1. Example Linker Map

```
MODULE   = TO1CYBM                              LANGUAGE = CYBIL
FILE     = TO1CYBR     1984/07/11     17:14:02


SECTION TYPE/                                 LOAD/
ACCESS ATTRIBUTES                    LENGTH   EXECUTION ADDR
--------------------------------------------------------------------------------
CODE - CYBER
READ EXECUTE                           60       000 000000978

WORKING STORAGE - DATA1
READ                                   2A       000 0000009D8

WORKING STORAGE - DATA2
READ WRITE                             20       000 000000A02


       ENTRY POINT DEFINITIONS                               ADDRESS
         TEST                                 NOT GATED    000 00000978


       EXTERNAL ENTRY POINTS REFERENCED
         SCAN


---- TOTAL MODULE LENTH ----            AA


MODULE   = TO1ASM1                              LANGUAGE = ASSEMBLER
FILE     = TO1ASMR     July 11, 1984      5:12 PM


SECTION TYPE/                                 LOAD/
ACCESS ATTRIBUTES                    LENGTH   EXECUTION ADDR
--------------------------------------------------------------------------------
CODE - ASMR                               - PROG
READ EXECUTE                           46       000 000000A22


       ENTRY POINT DEFINITIONS                               ADDRESS
         SCAN                                 NOT GATED    000 00000A22


---- TOTAL MODULE LENTH ----            46
```

Figure 5-1. Example Linker Map (Contd.)

```
SES/MC68000 LINKER OUTPUT

        PRIMARY ENTRY POINT = TEST

ID = $0000000000000000000C003SRXMDO


FILE NAME/                        LOAD/
ACCESS ATTRIBUTES                 LENGTH   EXECUTION ADDR
  SECTION NAMES
-------------------------------------------------------------------------------------------
TOO1101
READ WRITE EXECUTE                     4   * 000 0000008C
  ALS$ORG_000000BC

TOO1102
READ WRITE EXECUTE                     4   * 000 0000007C
  ALS$ORG_0000007C

TOO1103
READ WRITE EXECUTE                    30   * 000 00000000
  ALS$ORG_00000000


TOO1104
READ WRITE EXECUTE                   668   * 000 00004000
  MONR
  ASMR
  CYBR
  DATA1
  DATA2



---- TOTAL LENGTH ----              6a0


NO LINKER ERRORS WERE DETECTED
```

Figure 5-1. Example Linker Map (Contd.)

LIMITATIONS OF THE LINKER

The Linker performs only cursory checks to determine if you have specified any duplicate input file names.

The Linker performs no checks to determine if any file names generated using the NAME_SEED parameter duplicate any file names you have specified in the call to the Linker. You must resolve any file naming conflicts resulting from the use of the name seed prior to executing the Linker command. Creation of duplicate file names may cause the Linker to abort or yield unpredictable results.

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 unpredictable results.

OBJECT RECORD TRANSLATOR 6

The Object Record Translator utility (TRAN68K) translates files created by the MC68000 Absolute Linker into files of Motorola type S records. Output files can be used as input to a ROM programmer to create ROMs that can be installed in DIs.

Format: SES.TRAN68K
HDR=file name
SREC=file name
UN=user name optional

Parameters:

HDR (H)

The name of the header file generated by the MC68000 Absolute Linker for the program modules to be translated. The header file name has the form seedHDR, where seed is the name seed specified in the call to the Linker for the program modules. The header file contains information about additional files needed to create the type S records.

SREC (S)

The name of the file to which the translated program modules are to be written.

UN

The user name whose permanent file catalog is to be searched for input files and to which output files are to be written. If you omit this parameter, your permanent file catalog is used.

Example: In the following example, the program modules referenced in file SEGMHDR are translated into Motorola type S records. The output is written to file SRECFIL in the user's permanent file catalog.

```
SES.TRAN68K SEGMHDR SRECFIL
```

MEMORY IMAGE BUILDER 7

The Memory Image Builder utility (BLDMI68K) creates an absolute module from output files of the MC68000 Absolute Linker. The output module can be loaded directly into a DI.

Format:

```
SES.BLDMI68K
        NAME=file name
        HDR=file name
        OUTPUT=file name        optional
        UN=user name            optional
```

Parameters:

NAME

The name to be associated with the absolute module created.

HDR (H)

The name of the header file generated by the MC68000 Absolute Linker for the program modules to be processed. The header file name has the form seedHDR, where seed is the name seed specified in the call to the Linker for the program modules (refer to chapter 5). The header file contains information additional files needed to create the absolute module.

OUTPUT (O)

The name of the file to which the absolute module is to be written. If you omit this parameter, the absolute module is written to a file named MIBMOD.

UN

The user name whose permanent file catalog is to be searched for input files and to which output files are to be written. If you omit this parameter, your permanent file catalog is used.

Example: In the following example, an absolute module named ABSMOD is created from the files referenced in file SEGMHDR and written to file NEWMOD in the user's permanent file catalog.

```
SES.BLDMI68K ABSMOD SEGMHDR NEWMOD
```

DIAGNOSTIC MESSAGES A

The CDCNET MC68000 Utilities generate the following three types of diagnostic messages:

- Diagnostic messages written by the MC68000 Absolute Linker to the Linker map file.
- Diagnostic messages issued by the Linker only that are displayed at the interactive terminal and written to the job dayfile.
- Diagnostic messages issued by all of the CDCNET MC68000 Utilities that are displayed at the interactive terminal and written to the job dayfile.

LINKER MAP FILE DIAGNOSTIC MESSAGES

The Linker writes all messages to the Linker map file in the following format:

```
* * * LINKER ERROR NNNNN *FATAL* error message text

    MODULE =name

    FILE =file name

    NAME =name

    RECORD COUNT =integer
```

The string *FATAL* appears only if the error being reported was fatal to the link operation. NAME indicates the entry point at which the error occurred.

1 IMPROPER RELOCATION ADDRESS SPECIFICATION

Description: Relocation information is being incorrectly generated; Linker output is unaffected.
Action: Correct the object text.

2 UNANTICIPATED EOR

Description: The Linker encountered an EOR somewhere other than at the end of an object module.
Action: Correct the object text.

3 MORE THAN ONE CODE SECTION IN A MODULE

Description: A single object module had more than one code section; only one is permited.
Action: Correct the object text.

4 SDO GREATER THAN IDR SPECIFICATION ENCOUNTERED

Description:	The number of sections in the IDR is incorrect or the section ordinals do not start at zero or are not contiguous.
Action:	Correct the object text.

5 CODE SECTION ATTRIBUTE SPECIFICATION ERROR

Description:	A code section was found to have a write or binding section attribute.
Action:	Correct the object text.

6 MORE THAN ONE BINDING SECTION PER MODULE

Description:	An object module had more than one binding section; only one is permitted.
Action:	Correct the object text.

7 BINDING SECTION ALIGNMENT ERROR

Description:	The binding section for a module was not aligned on a 16-bit word boundary.
Action:	Correct the object text.

8 BINDING SECTION ATTRIBUTE SPECIFICATION ERROR

Description:	The binding section for a module had a write or execute attribute.
Action:	Correct the object text.

9 DUPLICATE SECTION DEFINITION ORDINAL

Description:	The same ordinal has been used for two sections in a single object module.
Action:	Correct the object text.

10 BINDING ATTRIBUTE SPECIFIED FOR A NON BINDING SECTION

Description:	The binding attribute was specified for a section other than the binding section.
Action:	Correct the object text.

11 CONFLICTING PROTECTION ATTRIBUTE FOR COMMON BLOCK

Description:	Different protection has been specified in separate common block declarations.
Action:	Correct source program.

12 CONFLICTING LENGTH SPECIFICATION FOR COMMON BLOCK

Description:	Unequal lengths were specified in separate common block declarations.
Action:	Correct source program.

13 COMMON TABLE OVERFLOW - RECOMPILE LINKER

Description:	Linker internal table size was exceeded.
Action:	Correct the object text.

14 MISPLACED IDR OR SDC

Description: Object text structure is incorrect.
Action: Correct the object text.

17 SDOS NOT CONTIGUOUSLY NUMBERED

Description: Object text structure is incorrect.
Action: Correct the object text.

18 SEGMENT TABLE OVERFLOW - RECOMPILE LINKER

Description: Linker internal table size was exceeded.
Action: Contact your CDC customer representative.

19 ZEROIZE SECTION INTERNAL LOGIC ERROR

Description: The Linker has aborted.
Action: Contact your CDC customer representative.

21 POINTER IN BINDING SEGMENT WAS MISALIGNED

Description: A binding section entry was not right-justified in a 16-bit word boundary.
Action: Correct the object text.

22 ATTEMPTED TO PLACE DATA IN A BINDING SECTION

Description: A binding section entry was not a pointer or a procedure descriptor.
Action: Correct the object text.

24 PREALLOCATED BINDING SEGMENT ATTRIBUTE ERROR

Description: A binding segment was defined as writable, executable, or readable under key lock control.
Action: Correct the DEFINE_SEGMENT subcommand for the segment.

25 PREALLOCATED EXECUTABLE SEGMENT ATTRIBUTE ERROR

Description: An executable segment was defined as writable.
Action: Correct the DEFINE_SEGMENT subcommand for the segment.

28 FIRST RECORD OF AN OBJECT MODULE WASNT AN IDR

Description: An input file to the Linker did not have the correct format.
Action: Correct input file.

29 DUPLICATE ENTRY POINT WAS DETECTED

Description: A symbol contains more than one XDCL as an entry point. The Linker used the first definition; Linker output is unaffected.
Action: Correct object file list if necessary.

30 LST OVERFLOW - TOO MANY ENTRY POINTS

Description: Linker internal table size was exceeded.
Action: Contact your CDC customer representative.

31 EXTERNAL ARRAY OVERFLOW - TOO MANY EXTERNALS

Description:	Linker internal table size was exceeded (maximum size is 200 entries.)
Action:	Contact your CDC customer representative.

32 RECORD CONTAINS IMPROPER SDO

Description:	An object text record referenced an undefined object text section.
Action:	Correct the object text.

33 INPUT RECORD CONTAINS AN IMPROPER SECTION OFFSET

Description:	An object text record referenced an offset outside the range specified in the section definition.
Action:	Correct the object text.

34 NO PRIMARY ENTRY POINT ENCOUNTERED

Description:	No primary entry point was specified.
Action:	Correct the object text.

35 PRIMARY ENTRY POINT NOT XDCLED

Description:	The XDCL attribute was not assigned to the object module containing the primary entry point.
Action:	Assign the XDCL attribute to the appropriate program module.

36 NO OBJECT FILE INPUT

Description:	No object module input was encounted by the Linker.
Action:	Check the input files for content and check the format of the input file parameters in the call to the Linker.

37 UNSATISFIED EXTERNAL REFERENCE

Description:	An externally referenced (XREF) declaration was not externally declared (XDCL) in any input module or in any of the modules on the referenced libraries.
Action:	Check your source program for a missing XDCL, and check the format of the input file parameters in the call to the Linker.

38 LFD CONTAINS IMPROPER EXECUTE ATTRIBUTE

Description:	An unknown execute attribute was specified in LFD.
Action:	Correct LFD.

39 UNKNOWN OBJECT TEXT RECORD TYPE

Description:	Object text structure is incorrect.
Action:	Correct the object text.

40 UNKNOWN EXTERNAL REFERENCE INSERTION TYPE

Description:	Object text structure is incorrect.
Action:	Correct the object text.

41 UNKNOWN SECTION DEFINITION TYPE

Description: Object text structure is incorrect.
Action: Correct the object text.

42 RIF DOES NOT PERTAIN TO CODE OR BINDING SECTION

Description: Relocation information is being incorrectly generated. Linker output is unaffected.
Action: Correct the object text.

43 IMPROPER RELOCATION CONTAINER SPECIFICATION

Description: Relocation information is being incorrectly generated. Linker output is unaffected.
Action: Correct the object text.

44 EXPECTED SDC RECORD

Description: Object text structure is incorrect.
Action: Correct the object text.

45 INVALID PROCEDURE OFFSET FOR INDIRECT CALL

Description: The byte offset for an indirect procedure call is not divisible by 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

Description: The Linker encountered a bit string insertion record with a bit offset greater than 7 or a bit length greater than 63. No bit string insertion has taken place.
Action: The utility that generated the object text has caused the error and must be corrected.

47 BAD LIBRARY FORMAT

Description: An input file that was specified as a library was not in library format.
Action: Check the input files that were specified as libraries.

48 REQUIRED LIBRARY MISSING

Description: The Linker encountered a libraries record that specified a library not present in the list of input library files.
Action: Correct the list of input library files in the call to the Linker to include the missing library.

49 ERROR IN PARAMETER VERIFICATION

Description: The type declarations for the variable do not match on the XDCL and XREF statements.
Action: Check the type declarations for the variable and correct them so that they match.

LINKER DAYFILE MESSAGES

The following messages are issued for the Linker only. They are displayed at the interactive terminal and written to the job dayfile.

10000 LINKER TERMINATED NORMALLY

Description: The Linker has terminated normally with no fatal or nonfatal errors.
Action: None.

10100 LINKER NORMAL TERMINATE WITH NONFATAL ERRORS - SEE MAP LISTING

Description: One or more non-fatal errors were encountered during the link operation. Linker output may or may not be valid.
Action: Check Linker map for further diagnostic messages.

10101 LINKER ABNORMAL TERMINATE - SEE MAP LISTING

Description: The Linker detected a fatal error and aborted. Linker output is undefined.
Action: Check Linker map for diagnostic message.

10102 LINKER ABNORMAL TERMINATE - NO MODULES PROVIDED

Description: The Linker found no input object modules.
Action: Check the file input parameters on the call to the Linker and check all input files for content.

10200 LIBRARY FILE file name NOT LOCAL

Description: The file specified in the LFL parameter of the LINK68K command or the OBJECT_LIBRARY subcommand of the Linker Parameter File is not a local file.
Action: Make the file local.

10201 OBJECT FILE file_name NOT LOCAL

Description: A file specified in the OFL parameter of the LINK68K command or the OBJECT_FILE subcommand of the Linker Parameter File is not a local file.
Action: Make the file local.

10202 OBJECT FILENAME file_name DUPLICATES EXISTING FILE

Description: A file name specified as an input object file or library is identical to the name of another file used by the Linker.
Action: Change the name of one of the file names involved in the duplication.

10203 IST FILE file_name NOT LOCAL

Description: A file specified in the INBOARD_SYMBOL_TABLE subcommand of the Linker Parameter File is not a local file.
Action: Make the file local.

10204 IST FILENAME file_name DUPLICATES EXISTING FILE

Description: A file name specified in the INBOARD_SYMBOL_TABLE subcommand of the Linker Parameter File is identical to the name of another file used by the Linker.

Action: Change the name of one of the file names involved in the duplication.

10300 LPF FILE file_name NOT LOCAL

Description: A file specified in the LPF parameter of the LINK68K command is not a local file.

Action: Make the file local.

10301 UNKNOWN LPF COMMAND command SPECIFIED

Description: The Linker detected an invalid Linker Parameter File subcommand.

Action: Correct the Linker Parameter File subcommand.

10302 INVALID MAP OPTION map_option SPECIFIED

Description: The Linker detected an invalid map option specified in the the MO parameter of the LINK68K command or in the MAP_OPTIONS specification of the LINK_OPTIONS subcommand in the Linker Parameter File.

Action: Correct the map option specification.

10303 INVALID NAME_SEED xxxx SPECIFIED

Description: The Linker detected an invalid name seed specified in the NS parameter of the LINK68K command or in the NAME_SEED specification of the LINK_OPTIONS subcommand in the Linker Parameter File .

Action: Correct the name seed specification.

10304 INVALID SEGMENT ATTRIBUTE segment_attribute SPECIFIED

Description: The Linker detected an invalid segment attribute specified in the ATTRIBUTES specification of DEFINE_SEGMENT subcommand in the Linker Parameter File .

Action: Correct the attribute specification.

10306 INVALID SECTION ATTRIBUTE SPECIFIED

Description: The Linker detected conflicting access attributes in the specification of a default section name.

Action: Correct the access attributes.

MC68000 UTILITY DAYFILE MESSAGES

The following messages are issued for all the MC68000 Utilities. The message indicated as being error messages reflect conditions that cause the utility to abort. These messages are displayed at the interactive terminal and written to the job dayfile.

20001 HEADER FILE file name IS BAD

Severity: Error.
Description: The specified file does not have the header file format.

20002 SEGMENT FILE file_name NOT FOUND

Severity: Error.
Description: Self-explanatory.

20003 ACQUIRE ERROR file_name

Severity: Error.
Description: Internal error trying to acquire file_name.

20004 USER ID long_user_id IS TOO LONG, NOW short_user_id

Severity: Informational
Description: Self-explanatory.

20005 SYMBOL long_symbol HAS BEEN SHORTENED TO short_symbol

Severity: Informational
Description: Self-explanatory.

20006 UNKNOWN OBJECT TEXT IN file_name

Severity: Error.
Description: Self-explanatory.

20007 UNEXPECTED EOF OR EOR FOUND

Severity: Error.
Description: Self-explanatory.

20008 TOO MANY EXTERNAL REFERENCES

Severity: Error.
Description: Self-explanatory.

20009 MODULE module_name IS NOT A MC68000 MODULE

Severity: Error.
Description: Self-explanatory.

20010 DUPLICATE ENTRY POINT NAME

Severity: Error.
Description: Entry point name specified more than once.

20011 MODULE module_name IS ALREADY ABSOLUTE AND CANNOT BE REFORMATTED

Severity: Error.
Description: Input modules must be relocatable.

20012 ORG ADDRESS FOR section_name IS INVALID

Severity: Error.
Description: The new module is made up of too many modules.

20201 MULTIPLE IDENTIFICATION RECORDS FOUND ON MODULE module_name

Severity: Error.
Description: Self-explanatory.

20202 SECTION OF NEW MODULE IS TOO LONG

Severity: Error.
Description: The new module is made up of too many modules.

20204 UNKNOWN SECTION ORDINAL FOUND ON MODULE module_name

Severity: Error.
Description: Recompile the module.

20205 MISSING SECTION DEFINITION ON MODULE module_name

Severity: Error.
Description: Recompile the module.

20206 REFERENCING OUTSIDE SECTION ON MODULE module_name

Severity: Error.
Description: Recomplied module.

20207 TOO MANY LIBRARIES ENCOUNTERED

Severity: Error.
Description: Self-explanatory.

20211 STARTING PROCEDURE start_proc_name NOT IN CODE SECTION

Severity: Error.
Description: Self-explanatory.

20217 ATTEMPTING TO BIND module_name, AN ABSOLUTE MODULE

Severity: Error.
Description: Module_name cannot be bound.

20218 COMMON BLOCK common_block_name HAS 2 DIFFERENT LENGTHS, SECOND FOUND IN MODULE module_name

Severity: Error.
Description: Self-explanatory.

20219 ERROR ENCOUNTERED IN SYMBOL TABLE IN MODULE module_name

Severity: Error.
Description: Self-explanatory, recompile module.

20220 ERROR ENCOUNTERED IN LINE TABLE IN MODULE module_name

Severity: Error.
Description: Self-explanatory, recompile module.

20301 NUMBER OF INPUT FILES DOES NOT EQUAL NUMBER OF OUTPUT FILES

Severity: Error.
Description: Self-explanatory.

20302 MISSING IDENTIFICATION RECORD ON FILE file_name

Severity: Error.
Description: Self-explanatory.

20302 FOUND A TEXT RECORD THAT IS NOT SUPPORTED FOR MC68000 ON FILE file_name

Severity: Error.
Description: Self-explanatory.

OBJECT TEXT FORMATS FOR THE MC68000 ABSOLUTE LINKER

B

This appendix contains the CYBIL type and constant declarations that define the format for files of object modules that are input to the MC68000 Absolute Linker. These formats are consistent with object files created by the CDCNET CYBIL compiler and the MC68000 Cross-Assembler.

```
{ 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 180 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.
{
{   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
  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$section_address_range,
    name: pmt$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$extensible_common_block,
    llc$lts_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. }
{ 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;
```