HEWLETT PACKARD

RTE-A LINK

User's Manual

Software Technology Division
11000 Wolfe Road
Cupertino, CA 95014-9804

Manual Part No. 92077-90035
E1193

Printed in U.S.A. November 1993
Fifth Edition

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARs 252.227.7013.

Printing History

The Printing History below identifies the edition of this manual and any updates that are included. Periodically, update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this printing history page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past updates; however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all updates.

To determine what manual edition and update is compatible with your current software revision code, refer to the Manual Numbering File or the Computer User's Documentation Index. (The Manual Numbering File is included with your software. It consists of an "M" followed by a five digit product number.)

Edition 1	Jun 1983	
Edition 2	Jan 1985	Added IF, MS, NA, OR, PA, TR Commands
Update 1	Jan 1986	Added RM Command
Reprint	Jan 1986	Update 1 incorporated
Edition 3	Aug 1987	Rev. 5000 (Software Update 5.0)
Update 1	Jul 1990	Rev. 5020 (Software Update 5.2)
Edition 4	Dec 1992	Rev. 6000 (Software Update 6.0)
Edition 5	Nov 1993	Rev. 6100 (Software Update 6.1)

Preface

This manual is a tutorial guide and reference for users of the RTE-A linkage editor program LINK. It is assumed that you know how to edit and compile programs in the RTE-A Operating System.

Chapter 1 Provides a description of LINK, its features and capabilities, and how it fits into the RTE-A program development cycle. Several examples are given to illustrate how LINK works.

Chapter 2 Describes the various ways in which LINK can be run. It includes descriptions of the LINK runstring, runstring options, file type extensions, and command files.

Chapter 3 Describes the LINK commands. The command syntax and a brief description for each command are provided. The commands are given in alphabetical order. This chapter can be used as a reference section.

Chapter 4 Describes advanced LINK concepts for experienced LINK users. A section is included to show how to optimize the performance of LINK by merging and indexing libraries. Sample load maps are also provided and discussed.

Chapter 5 Describes how LINK and LINDX are installed.

Appendix A Provides a description of the LINK error messages.

Appendix B Provides information on running LINK under the File Manager Program (FMGR).

Table of Contents

Chapter 1
Introduction

Chapter 2
Running LINK

Chapter 3
LINK Commands

Chapter 4
Advanced Concepts

Chapter 5
Installation Guide

Appendix A
Error Messages

Appendix B
Using LINK with FMGR

Tables

1

Introduction

This chapter provides an overview for users who have little experience in linking programs on the RTE-A Operating System. It describes LINK, the linking process, and how LINK fits into the RTE-A program development cycle.

What is LINK?

LINK is the linkage editor for RTE-A. LINK collects relocatable files and library routines into a runnable program file. LINK allows you to change certain attributes of program files, such as program priority, program size, and Virtual Memory Area/Extended Memory Area (VMA/EMA) specifications.

Running LINK is an important step in the development cycle of all programs. The steps in the program development cycle for RTE-A are listed below. Note that LINK is not involved when you use the BASIC Interpreter because the interpreter does not produce relocatable code.

RTE-A Program Development Cycle

1. Plan your program.
2. Create or modify the files containing the source code for your program using an editor.
3. Compile or assemble the source code using the appropriate language processor.
4. Use LINK to create a program file containing executable code.
5. Execute your program. Test and debug the program as needed. Symbolic Debug can be used with FORTRAN, Pascal, Macro, HP C/1000, and compiled BASIC programs.
6. Repeat steps 2 through 5 until your program executes correctly.

What is a Library?

A library is a relocatable file containing a number of modules, such as subroutines. LINK searches libraries to locate all externals (items referenced but not defined by your program). You can use the libraries supplied with your RTE-A system and create your own libraries.

When the system is generated, the System Manager can designate any libraries that are used by most programs as system libraries. System libraries are automatically searched by LINK when your program is linked.

LINK searches other libraries if you specify the names of the libraries when running LINK. LINK searches the libraries you specify before searching system libraries. In general, you want LINK to search the user-specified libraries first because subroutines in these libraries may reference subroutines in the system libraries.

Indexing Libraries

Searching libraries is often the most time-consuming part of linking a program. You can reduce the amount of search time by indexing libraries. An indexed library contains an index at the beginning of the file that indicates the name and location of each subroutine in the library. Chapter 4 contains a discussion on indexing libraries.

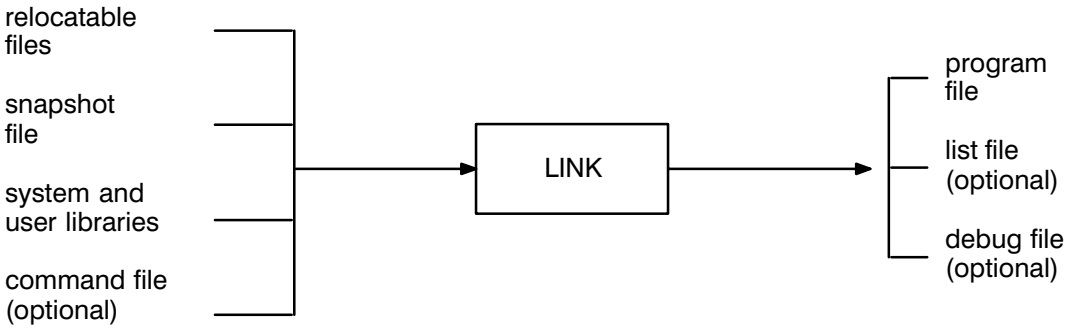
Using LINK

You can specify commands for LINK from three different sources:

- A CI (or FMGR) LINK runstring
- An interactive LINK session
- A LINK command file

You can use any combination of these three sources to run LINK. For example, a LINK command file can be specified as part of a LINK runstring or during an interactive LINK session.

Regardless of the method you choose, LINK requires the same input and gives you the same output. The following figure illustrates input to and output from LINK:



The snapshot file contains information about the operating system that LINK needs to produce the program file. If you do not specify a snapshot file, LINK uses the snapshot file for your system. Chapter 4 contains more information on the snapshot file.

LINK Examples

The easiest way to specify commands to LINK is to enter a runstring similar to the following:

```
CI> link myprog.rel
```

In this example, LINK relocates the file MYPROG.REL, searches the system libraries to resolve undefined external references and creates a program file containing executable code. The name of the program file is based on the PROGRAM statement in the source code (or NAM statement in Macro/1000). Therefore, if the name of the program is MYPROG, the name of the program file is MYPROG.RUN.

The following interactive LINK session gives the same result:

```
CI> link                          (Run LINK interactively)
link version 6100      Use ? for help
link: re myprog.rel               (Relocate file MYPROG.REL)
link: en                          (End LINK session)
```

If you want to link PROG1.REL, which calls subroutines in library MYLIB.LIB, you can enter the following LINK runstring:

```
CI> link prog1.rel mylib.lib
```

Or, you can run LINK interactively as follows:

```
CI> link                          (Run LINK interactively)
link version 6100      Use ? for help
link: re prog1.rel                (Relocate file PROG1.REL)
link: li mylib.lib                (Define library MYLIB.LIB)
link: en                          (End LINK session)
```

In both these examples, LINK searches library MYLIB.LIB in addition to the system libraries when creating the program file.

2

Running LINK

This chapter describes how to specify options and commands to LINK from a runstring, interactive session, or LINK command file. All examples are shown using the Command Interpreter (CI). Appendix B describes using LINK from FMGR.

Using LINK Runstring Options

The LINK runstring allows you to specify many LINK options in an easy to use format. To use the LINK runstring, enter the following:

```
CI> link <file name>.rel [parameters]
```

The optional parameters can be LINK runstring options and file names with a file type extension. Both are explained in detail in following sections.

The rules for using the LINK runstring are as follows:

- Delimiters between file names and parameters can be either commas or blanks. (Commas are required if LINK is run from FMGR.)
- A maximum of 255 characters, including delimiters, is allowed in the runstring.
- The file names specified in the runstring must include the file type extensions described in the section “LINK File Type Extensions”. (Optionally, the file names can use the prefixes shown in the LINK Runstring section in Appendix B.)
- Uppercase and lowercase letters can be used. All characters are converted to uppercase internally.

Runstring Options

LINK options entered in the runstring are prefixed by the plus (+) character. The options that can be used in the runstring are as follows (Chapter 3 contains detailed descriptions of the LINK commands that correspond to these options):

`+B` Batch mode; LINK never goes interactive. If LINK reaches a point at which it would normally go interactive, it aborts instead.

`+CR:`*file*`|`*crn* Specify scratch disk file or cartridge (for VMA backing store file).

`+DE` Create Symbolic Debug file for relocatable files included in the runstring.

`+DM` Set Debug Monitor mode.

`+DP` Prohibit purging of existing program file.

`+E` *cmd*`|`*cmd*... Execute LINK commands from the runstring. If given, this must be the last option in the runstring; the remainder of the runstring is a series of LINK commands, separated by vertical bars (|). These commands are executed before interactive commands or transfer file commands are executed.

`+EC` Echo commands, all input commands are sent to list file or device.

`+LC` Use labeled system common.

`+LL:`*file*`|`*lu* Specify list file or LU to which messages and the load map are sent.

`+MA` Display load map on terminal.

`+RO` Reorder modules in the data segment to reduce base page links.

`+SC` Use blank system common.

`+SP` Declare program as shareable (CDS programs only).

`+SU[:OF]` Specify a system utility. The optional OF parameter indicates that any user can remove the program from physical memory.

`+SZ:[+]`*pages* Set non-CDS program to specified number of pages in physical memory. Pages can be set to a value between 1 and 32, inclusive.

`+WD[:/`*dir*`[/@>`*dsinfo*`]]`
Set the LINK working directory for file references. See the WD command description in Chapter 3 for more information.

The colon is required when the CR, LL, SU:OF, or SZ option is used in the runstring. The LINK runstring options can be entered in any order, with the exception of the E option. Because the LC and SC options are mutually exclusive, only one should be entered in the runstring. If both are entered, the last one specified is used.

LINK File Type Extensions

In the runstring, LINK uses the file type extensions to identify the file types. When LINK is executing, various files are read, searched, and created. The file type extensions are as follows:

`.REL`	Relocatable file (input to LINK)
`.R`*nnn*	Relocatable file where *nnn* is 3 integers (input to LINK)
`.LIB`	Library file (input to LINK)
`.SNP`	Snapshot file (input to LINK)
`.LOD`	LINK command file (input to LINK)
`.MAP`	List file (output by LINK)
`.RUN`	Program file (output by LINK)
`.DBG`	Debug file (output by LINK)

Relocatable files (`.REL` and `.R`*nnn*) are files produced by the compiler or assembler that LINK uses to produce the program files.

Library files (`.LIB`) are searched to resolve undefined external references made in the relocatable files.

The snapshot file (`.SNP`) contains information about the operating system on which the program is to run.

The command file (`.LOD`) contains LINK commands and is used to direct the execution of LINK.

The list file (`.MAP`) contains a listing of the load map and LINK session, and is output by LINK, if specified.

The program file (`.RUN`) contains the executable code.

The debug file (`.DBG`) allows you to use Symbolic Debug when running the program.

Only one each of the following types is allowed: `.MAP`, `.RUN`, `.DBG`, and `.SNP`. Multiple relocatable, library, and command files are allowed. Multiple command files are processed in the order given. For example, consider the following runstring:

```
CI> link job.rel job2.lod job1.lod
```

In this example, the file JOB.REL is relocated. The order in which the command files appear in the runstring determines the order of execution; therefore, JOB2.LOD is executed before JOB1.LOD.

Examples of LINK Runstrings

The following are examples of the LINK runstring:

```
CI> link part.map +ec +sz:32 part1.rel part2.rel part3.rel part4.rel
```

This runstring relocates the four modules, PART1, PART2, PART3, and PART4, and lists the load map to file PART.MAP. The EC option echoes the LINK commands to file PART.MAP, and the SZ option sizes the program to 32 pages.

```
CI> link +lc /circle/area.rel /circle/area.lib /prog/area.run +dp
```

This runstring relocates file /CIRCLE/AREA.REL. LINK searches library /CIRCLE/AREA.LIB to resolve undefined external references and produces program file AREA.RUN on directory PROG. The LC option specifies that AREA.RUN uses labeled system common during execution. The DP option prohibits LINK from overwriting /PROG/AREA.RUN if the file exists. If the file does exist, LINK issues a warning and aborts.

```
CI> link +wd /invention/mother.lod>star +e vm,,l|ws 10
```

This runstring transfers to LINK command file MOTHER.LOD on directory /INVENTION of DS node STAR. All relocatable, library, and transfer files will be searched for in that directory on that node. The program will be a Large VMA program with a working set size of 10 pages.

Running LINK Interactively

Running LINK interactively allows you to use the more powerful features of LINK. To begin an interactive LINK session, enter the following:

```
CI> link [parameters]
```

The optional parameters can be any valid LINK runstring command or file name with any file type extension except .REL.

LINK responds by displaying a one-line message followed by the LINK prompt:

```
link version 6100      Use ? for help
link:
```

You may enter only one LINK command per line. If you enter more than one command per line, only the first command is executed. Chapter 3 contains descriptions of all LINK commands.

If you do not specify the file type extensions, the extensions are defaulted to those recognized by LINK as described in the LINK File Type Extensions section of this chapter.

Getting Help

If you need help any time during an interactive LINK session, enter a question mark and LINK displays a syntax summary of the LINK commands. A question mark followed by a LINK command (?,*command*) displays a more detailed description of the specified LINK command.

Example of an Interactive LINK Session

The following is an example of an interactive LINK session (user input is underlined):

```
CI> link +cr:myscratch                    (Run LINK interactively and use
link version 6100  Use ? for help         MYSCRATCH as scratch file)
link: re area.rel                         (Relocate file AREA.REL)
   AREA
link: di                                  (Display undefined externals)
   Undefined symbols:
   PERIM   .NFEX   EXEC   .EIO   CIRCLE   .FION   TRIAN
   SQUARE   RADI   .DTA.
link: re square.rel                       (Relocate file SQUARE.REL)
   SQUARE
link: re circle.rel                       (Relocate file CIRCLE.REL)
   CIRCLE
link: se mylib.lib                        (Search library MYLIB.LIB)
   TRIAN   PERIM   RADI
link: di                                  (Display undefined externals)
   Undefined symbols:
   .EXIT   .NFEX   EXEC   .EIO.   .FION   .DTA.
link: en                                  (End LINK session)
```

In the above example, LINK relocates three modules and searches a user-specified library and the system libraries to resolve undefined external references. Assuming the name of the program is AREA, LINK writes the executable code to file AREA.RUN; the file is created if it does not exist or, the contents are overwritten if the file does exist. The CR command is used as an optional runstring parameter to specify a scratch file.

After you enter the EN command, LINK creates a load map that is displayed at your terminal. See Chapter 4 for detailed information on the load map. (You can use the LL command to direct the load map to a different device or to a file.)

Using the Command Stack

As command lines are entered at the terminal keyboard, they are saved in a stack for future reference or editing and reuse. The number of command lines saved depends on the number of characters in each saved line. You can expect approximately 40 lines to be saved. If the stack is full, the oldest commands in the stack are removed to make room for the new command. Duplicated commands are removed from the stack.

$VISUAL Command Editing

For VC+ systems, LINK supports the $VISUAL command editing modes (for example, EMACS or VI) through use of the CMNDO monitor. If CMNDO is used, LINK saves and restores its command stack from file LINK.STK in your home directory, if the file exists. To use this feature, make sure your home directory is set, using the PATH program to set UDSP #0, and create file LINK.STK in that directory as a type 3 or 4 file.

To use CMNDO, set the $CMNDO_LINK or $CMNDO environment variable by entering one of the following commands from CI (and restart LINK):

```
CI> set -x CMNDO_LINK = T
```
Use CMNDO from LINK, but not other utilities.

or

```
CI> set -x CMNDO = T
```
Use CMNDO from all utilities that support it.

If you have $CMNDO set to TRUE but do not wish to use CMNDO from LINK enter:

```
CI> set -x CMNDO_LINK = F
```

Refer to the "Command Editing" chapter in the *RTE-A User's Manual*, part number 92077-90002, for more information on the $VISUAL command editing modes and the CMNDO monitor.

CI-Style Command Editing

LINK contains built-in support for the CI-style command stack. Command stack accessing is performed using the / (slash) command. Entering a slash causes up to the last 20 commands to be displayed on your terminal. You can then use the terminal's cursor movement keys and local editing keys to move to one of the displayed commands and edit it. After selecting one of the commands, and possibly modifying it, pressing carriage return causes LINK to execute it. An example is below (user input is underlined):

```
link version 6100      Use ? for help
link: ws 32
link: debug
link: re text
  No such file TEXT.REL
  Working directory: /JDJ/CDS
  Module not relocated
link: //
--001/003-- Commands:
re test                          ← the cursor was moved to this line
TEST  TEST10                        and the TEXT was changed to TEST.
link:
```

There are additional ways to display and select commands from the command stack. For a complete description, see the *RTE-A User's Manual.* Note that LINK does not have the capability to save and restore commands across successive runs.

Using LINK Command Files

Using a command file to run LINK allows you to link your program continually without re-entering the same LINK commands. All LINK commands that are available in an interactive LINK session can be used in a LINK command file.

There are two ways to use LINK command files. One way is to specify the command file name in the LINK runstring:

```
CI> link <command file>.lod [parameters]
```

The .LOD file type extension is required and indicates to LINK the specified file contains LINK commands. You also can enter other runstring commands or file names with the command file name. All runstring options are processed, regardless of their order, before the command file is read.

The second way to use LINK command files is to use the TR command during an interactive LINK session:

```
link: tr <command file>.lod
```

LINK transfers control to the command file and processes all the LINK commands in the file. If the command file does not contain the EN command, control returns to the interactive session after LINK processes the last command in the file.

Examples of LINK Command Files

In the first example, file EMAJOB.REL is relocated, user-specified library VMALB.LIB is searched for undefined externals, the working set size is set to 100 pages, and VMA size is set to 1400 pages. The EC command is included to echo the LINK commands at the terminal. The program file is EMAJOB.RUN on directory MYDIR. If the file already exists, it is overwritten; otherwise, the file is created.

```
ec
re emajob.rel
li vmalb.lib
ws 100
vm 1400
en /mydir/emajob.run
```

The following LINK command file loads the four modules of PART. The result is identical to that of the first runstring given in the section, Examples of LINK Runstrings. The LL command is equivalent to specifying a file with type extension .MAP in the LINK runstring.

```
ec
11 part.map
sz 32
re part1.rel
re part2.rel
re part3.rel
re part4.rel
en
```

The result of the following command file is identical to the result of the second runstring given in the section "Examples of LINK Runstrings". The OU command prohibits LINK from overwriting an existing program file. If the program file specified already exists, LINK issues a warning and aborts. Other specifications are the same: labeled common is used, the library is AREA.LIB on directory CIRCLE, and the program file is AREA.RUN on directory PROG.

```
lc
re /circle/area.rel
ou /prog/area.run
li /circle/area.lib
en
```

Using the Default Directory Path

LINK can provide a default directory and node name to be used whenever LINK opens files. It can be used both in the runstring and interactively. For example, the runstring:

```
CI> link +wd:/utilities/util/@>96 util.lod
```

causes LINK to use the file /UTILITIES/UTIL/UTIL.LOD>96 as the LINK command file. Each RE command in this file also has this directory and node name appended to it, provided that the specified relocatable file name does not have a directory or node name.

The WD command allows you to create a LINK command file to load a program locally without setting the default directory path, or remotely from another system by setting the default directory path. Remotely refers to running LINK on a system different from the one on which the LINK command file resides. In this case, you would use the local snap file and library files, and remotely access the relocatable needed to link your program.

Using the $LINK Environment Variable

On VC+ systems, environment variable $LINK may be set to a runstring to automatically prepend to each LINK runstring. For example, the CI command:

```
CI> set –x link = ginger.snp +de
```

causes snap file GINGER.SNP to be used for all loading and for symbolic debug information files to automatically be produced. The CI command:

```
CI> set –x link = +wd:#3
```

causes LINK to use UDSP #3 for all file searches.

3

LINK Commands

This chapter contains descriptions of all LINK commands. Table 3-1 lists the LINK commands and indicates which commands can be used in a runstring, interactive session (or command file), and relinking session. The table also shows which commands can be used with CDS and non-CDS programs.

Descriptions of LINK Commands

Descriptions of the LINK commands are in alphabetical order.

Commas or blanks can be used as delimiters. (The syntax descriptions in this chapter use commas.)

Parameters described with two or more words are shown in angle brackets; for example, *<partition #>*.

In the syntax descriptions, brackets indicate optional parameters; for example, `SN[,`*filedescriptor*`]` means that the command can be specified with or without a file descriptor.

In the syntax descriptions, commands that can be used in the LINK runstring are shown with the + sign in the command syntax. Certain commands require the colon as a parameter delimiter in the runstring, but a comma or blank is required interactively or in a command file.

If you are running LINK interactively or from a LINK command file, you can enter only one command per line. If you enter two commands, only the first is executed; the second command is ignored. For example, if you enter DEEN, the DE command is executed but the EN command is ignored.

Table 3-1. LINK Command Summary

Command	Description	Run-string	Inter-active	Relink	CDS	Non-CDS
AB	Abort LINK		X	X	X	X
AL[,UN]	Lock or unlock all code segments in memory		X	X	X	
BP	Report Base Page links	X	X		X	X
AS,<partition #> [,C | D]	Assign partition number		X	X	X	X
CD,blocks	Set number of code segment blocks		X	X	X	
CR:filedescriptor [C | D]	Specify scratch file name or cartridge	X			X	X
DB	Append DBUGR subroutine		X			X
DE	Create Symbolic Debug/1000 file	X	X		X	X
DI	Display undefined external references		X		X	X
DM	Sets Debug Monitor mode	X	X	X	X	X
DP	Prohibit purging of program file	X			X	X
EC	Echo input to list file or list device	X	X	X	X	X
EM,pages [,model]	Specify number of pages for EMA		X	X	X	X
EN[,filedescriptor]	End command input, specify program file		X	X	X	X
ES,commonblock,emaseg	Relocate a common block to start of an EMA segment		X		X	X
FO	Force linking of a program		X		X	X
HE[,words]	Set number of words in heap area		X	X	X	
IF A | 6 <link command>	Conditionally execute a LINK command		X		X	X
LC	Specify use of labeled system common	X	X		X	X
LI,filedescriptor	Define a library file		X		X	X
LK,filedescriptor	Relink and change program attributes		X	X	X	X
LL,filedescriptor | lu	Specify list file or device for load map	X	X	X	X	X
LO	List program attributes during relinking			X	X	X
MA	List load map at terminal	X			X	X
ML,<segment #>[,UN]	Lock or unlock a code segment in memory		X	X	X	
MS,filedescriptor	Multiple search library file		X		X	X

Table 3-1. LINK Command Summary (continued)

Command	Description	Run-string	Inter-active	Relink	CDS	Non-CDS
NA[,filedescriptor]	Set name of program file and check validity		X		X	X
NS[,pages]	Start a new code segment		X		X	
OR,<ema area>	Order EMA area		X		X	X
OS	Operator suspend		X	X	X	X
OU, filedescriptor	Set name of program file		X		X	X
PA	Page align next EMA area		X		X	X
PC,progcplv[,rquscplv]	Specify a program's capability level		X	X	X	X
PR,priority	Set program priority		X	X	X	X
PS	Start overlays at page boundary		X			X
RE,filedescriptor	Relocate a file		X		X	X
RM,filedescriptor,symbol	Relocate a module		X		X	X
RO	Rearrange modules to reduce base page links	X	X		X	X
SC	Specify use of system common	X	X		X	X
SE[,filedescriptor]	Search file for external references		X		X	X
SH,label[,<partition#>]	Allow use of shareable EMA partition		X	X	X	X
SN[,filedescriptor]	Define or display snapshot file name		X		X	X
SP[,UN]	Declare program to be shareable		X	X	X	
ST,words	Set size of stack area		X	X	X	
SU[,OF]	Specify a system utility	X	X		X	X
SZ,[+]pages	Specify size of program file	X	X	X		X
TR,filedescriptor	Specify LINK command file		X	X	X	X
VM[,pages] [,model]	Specify size of virtual memory area		X	X	X	X
WS[,pages]	Specify working set size VMA		X	X	X	X
WD[,DIRECTORY]	Set default directory	X	X		X	X
*	Specify a comment line	Used to comment LINK command files				
?[,<link command>]	Display help information	Used during any interactive LINK session				
/[stack commands]	Display command stack	Used during any interactive LINK session				

AB (Abort)

Purpose: Aborts LINK immediately.

Syntax: `AB`

Remarks: When aborted, LINK closes all open files and does not create a program file.

The AB command can be used during relinking.

AL (All Memory Locked)

Purpose: Declares that all code segments are locked into memory before the program is executed. (For CDS programs only.)

Syntax: `AL`

`AL[,UN]` (Relinking syntax)

Parameters: `UN` Unlocks memory.

Remarks: The AL command overrides the ML command.

You can use the AL command during relinking to set the memory-lock flag in the program file to either on or off. When the memory-lock flag is set to off, any code segment memory-lock flags, which are set using the ML command, become effective.

AS (Assign Partition)

Purpose: Assigns a partition where the program resides.

Syntax: `AS,<`*partition #*`>[,C|D]`

Parameters: *partition #* Specifies the number of the partition in which the program resides. The partition number can be set to a value between 1 and 1023, inclusive. If omitted, LINK uses the value of 0, which cancels any partition number defined previously in the LINK session.

`C` or `D` Optional parameter for CDS programs only. C indicates that the code partition is assigned to the specified partition number. D indicates that the data segment is assigned to the specified partition number.

Remarks: If you specify the optional parameter (C or D) for a non-CDS program, the value is ignored.

The AS command can be used during relinking.

BP (Report Base Page Usage)

Purpose: Report the number of base page links used by non-CDS modules.

Syntax:
```
BP
+BP    (in runstring)
```

Remarks: The BP command causes LINK to emit an additional line into the load map for some of the modules relocated. This line reports how many base page links are used by the module. The form of the line is:

```
*bp: rel=   2. int=   5. ext=   3.
```

This reports that the module relocated 2 (decimal) words onto base page, that 5 base page links are needed for addresses that are internal to the module itself, and that 3 new base page links are needed to reach addresses that are external to the module. If the module does not use any base page links, the *bp line is not reported.

CD (Code Segment)

Purpose: Sets the number of code segment blocks allocated to the code partition. (For CDS programs only.)

Syntax: `CD,`*blocks*

Parameters: *blocks* Specifies the number of code segment blocks. Blocks can be set to a value between 1 and 128, inclusive.

Remarks: If the number of code segment blocks is less than the number of memory locked code segments previously specified with the ML command, LINK issues an error. If you enter the CD command before any ML commands and specify fewer code segment blocks than memory locked code segments, LINK does not detect the error until you enter the EN command.

This command does not affect the behavior of the program, but can affect performance.

If this command is omitted, LINK places all program code segments in one code block.

The CD command can be used during relinking.

CR (Specify VMA Backing Store File)

Purpose: Allows you to specify the scratch file or cartridge to be used by LINK.

Syntax: `+CR:`*filedescriptor*`|`*crn* (in runstring)

Parameters:

filedescriptor	Specifies the name of the LINK backing store file.
crn	Specifies the FMGR cartridge where LINK puts the backing store file.

Remarks: The CR command is valid only in the LINK runstring. If you omit this command, LINK creates a default backing store file. LINK purges the backing store file before finishing the linking process.

DB (DBUGR)

Purpose: Appends the DBUGR subroutine to the program file.

Syntax: `DB`

Remarks: The DB command allows you to use DBUGR to debug your program. This command is provided for backward compatibility. Refer to the *RTE-6/VM Debug Subroutine Reference Manual*, part number 92084-90014, for information on the DBUGR program.

DE (Debug)

Purpose: Allows debugging of the program with the Symbolic Debug/1000 program available with purchase of the HP 92860A Symbolic Debug/1000 product.

Syntax:
```
DE
+DE    (in runstring)
```

Remarks: LINK creates a file with file type extension .DBG (for example, TEST.DBG) containing debug information. LINK places the debug file in the same directory as the program file. LINK overwrites an existing file with the same name.

DE must be specified before any modules are relocated.

DI (Display)

Purpose: Displays undefined external references.

Syntax: `DI`

Remarks: To satisfy undefined externals, use the LI, MS, RE, or SE command. To ignore undefined external references and force linking of the program, use the FO command.

DM (Debug Monitor)

Purpose: Turns on debug monitor mode.

Syntax:

```
DM
+DM              (in runstring)
DM[,OF]          (relink syntax)
```

Parameters: `OF` Turns off debug monitor mode. It is only valid during relinking.

Remarks: If the debug monitor mode is turned on, programs that are about to be aborted (due to some program violation) are operator suspended instead, allowing you to debug the program using Symbolic Debug/1000 and to examine the state of the program that caused the program violation.

The DM command can be issued independently of the DE command, but if you want to use the debugger, you must also issue the DE command.

For example, assume your program aborts. You can relink your program using the DM command and execute your program again. The program then becomes operator suspended. Now use Debug to adopt the program:

```
RU,DEBUG,-D,son.dbg son:ih
```

Remember, the DE command must have been issued when the program was initially linked.

See the discussion of the −D option in the *Symbolic Debug/1000 User's Manual*, part number 92860-90001, for additional information.

DP (Do Not Purge)

Purpose: Prohibits purging of an existing program file.

Syntax: `+DP` (in runstring)

Remarks: The DP command is valid only in the LINK runstring.

If there is an existing program file, LINK issues a warning and aborts. You should specify a different program file name, rename the existing program file, or remove the DP command and run LINK again.

The EN or NA command, with the *filedescriptor* parameter specified, overrides the DP command.

EC (Echo)

Purpose: Echoes input from LINK command file to the list file or list device.

Syntax: `EC`

`+EC` (in runstring)

Remarks: This command is useful for debugging LINK command files.

The EC command can be used during relinking.

EM (Extended Memory Access)

Purpose: Specifies the number of Extended Memory Area (EMA) pages.

Syntax: `EM[,`*pages*`][,`*model*`]`

Parameters:

pages Specifies the minimum number of pages in EMA. Pages can be set to a value between 2 and 1022, inclusive, for Normal and Large model programs; between 2 and 32733 for Extended model. Refer to the *RTE-A Programmer's Reference Manual*, part number 92077-90007, for a description of the EMA models.

model Specifies the EMA model as follows:

`L` Use Large EMA model.
`X` Use Extended EMA model.

If a model is not specified, the Normal EMA model is used.

Remarks: If the EM command is omitted and EMA is used by your program, LINK uses the number of EMA pages calculated by the compiler.

If your program uses EMA and you enter the EM command, LINK compares the number of EMA pages calculated by the compiler and the number of pages you specified with the EM command, and uses the larger value.

The EM command can be used during relinking. However, only the "pages" parameter may be specified; the EMA model may not be changed.

EN (End)

Purpose: Ends LINK.

Syntax: `EN[,`*filedescriptor*`]`

Parameters:

filedescriptor Specifies the name of the program file. If you omit this parameter, LINK uses either the file name previously specified by the OU or NA command, or the program name found in the PROGRAM statement in the source file (or NAM statement in Macro).

Remarks: The EN command indicates to LINK that the command mode has ended and the linking process can be completed. If you include the filedescriptor parameter, the specified file is used as the program file. If the file already exists as a program file, LINK overwrites the existing file.

A file name specified with the EN command overrides a file name specified with the NA and OU commands, or the program name specified in the program statement. A file name specified with the EN command does not override a file name with the .RUN file type extension specified in the LINK runstring.

If the EN command is used to end a relinking session in which changes were made, LINK modifies the program file. The filedescriptor parameter is ignored, if specified; LINK always modifies the program file specified in the LK command.

After entering EN and setting undefined externals, no new overlays can be relocated. They generate illegal relocatable messages.

ES (EMA Segment)

Purpose: This command relocates a common block to the start of an EMA segment, for use in programs that call the RteAllocShema routine to attach shared EMA areas.

Syntax: `ES,`*commonblock*`,`*emaseg*

Parameters:

commonblock The name of the common block that should be relocated to the given EMA segment.

emaseg An EMA segment number from 1 to 64.

Remarks: This command allows a FORTRAN program to declare a common block that defines the contents of a shared EMA area attached to the program via the RteAllocShema routine. LINK relocates the addresses of the variables in the named common block to the given EMA segment, that is, to the proper 1024-page boundary of the program's EMA address space. See the EMA/VMA Programming chapter of the *RTE-A Programmer's Reference Manual,* part number 92077-90007, for more information on the RteAllocShema routine.

The first variable in the common block can be passed as the "starting address" parameter to the RteAllocShema routine. Therefore, the program need not contain a "hard coded" starting EMA address; the choice of EMA segment number is determined in the LINK command file.

For example, a FORTRAN program may declare:

```
$ema /bat/
 :
common /bat/ chain, puller
```

The LINK command “es bat 2” relocates the EMA address of variable “chain” to the start of the second 1024-page range of the program’s EMA space, address 4000000 octal or 1048576 decimal. The address of variable “puller” is relocated to appear immediately after “chain”. This program can contain a statement of form:

```
error = rteallocshema('BAT',1,chain,0j)
```

This call possibly creates and attaches a shared EMA with label “BAT” to the program’s EMA address space. The point of attachment in the program’s EMA address space begins at the EMA address to which the variable named “chain” has been relocated by the “ES” command.

FO (Force)

Purpose: Forces linking of a program or an overlay.

Syntax: `FO`

Remarks: This command links a program or an overlay even if there are undefined external references. All undefined externals are set to zero and ignored, but are still listed to the list device or list file. Results are unpredictable if an undefined external is referenced during execution.

HE (Heap Area)

Purpose: Sets the size of the heap area. (For CDS programs only.)

Syntax: `HE[,`*words*`]`

Parameters:

words Specifies the number of words in the heap area. Words can be set to a value between 4 and the remainder of the data partition. If this parameter is omitted, LINK uses the value of 4.

Remarks: FORTRAN programs use the LIMEM system subroutine and Pascal programs use the HEAP 1 compiler directive to reference the heap area. (LIMEM is described in the *RTE-A/RTE-6/VM Relocatable Libraries Reference Manual*, part number 92077-90037.) The heap area is included in the data partition. If the data partition reaches the maximum of 32 pages, space is taken from the stack area if possible, to satisfy the HE request.

If the HE command is omitted, LINK uses the value of 4.

The HE command can be used during relinking. See the section on Relinking CDS Programs in Chapter 4 for more details.

IF (Conditional Execution of LINK)

Purpose: Allows a command in a LINK command file to be executed depending on whether the RTE-A or RTE-6/VM LINK program is being used.

Syntax: `IF A|6 <`*link command*`>`

Parameters: `A` or `6` Specifies RTE-A (A) or RTE-6/VM (6) LINK.

link command Specifies a valid RTE-A or RTE-6/VM LINK command.

Remarks: The IF command is useful in command files that are used to link programs for both RTE-A and RTE-6/VM systems. The command allows the correct system-dependent modules to be relocated. For example,

```
re testprog.rel
if a li liba.lib
if 6 li lib6.lib
en
```

RTE-A LINK searches LIBA.LIB and RTE-6/VM LINK searches LIB6.LIB.

LC (Labeled System Common)

Purpose: Specifies that the program uses labeled system common.

Syntax: `LC`

`+LC` (in runstring)

Remarks: You can use labeled system common to store subroutines or data that is shared by several programs. The LC command instructs LINK to search the snapshot file for labeled system common entry points before searching system libraries or libraries specified with the LI command. Labeled system common entry points do not supersede entry points previously resolved with the SE, MS, or RE commands; however, such conflicts are reported.

LI (Library)

Purpose: Defines a library file that LINK searches immediately preceding the search of the snapshot file and system libraries.

Syntax: `LI,`*filedescriptor*

Parameters: *filedescriptor* Specifies the library file to be searched.

Remarks: LINK searches the libraries specified with the LI command after you enter the EN command. These libraries are searched before the snapshot file and system libraries, and in the order in which you entered the names.

The libraries are searched linearly only one time; therefore, if the LI command is used for an unindexed library, backward references in the file are not resolved. It is recommended that you use the MS command for searching unindexed libraries.

The LI command and libraries specified in the LINK runstring can be used to search a maximum of 10 libraries during one LINK session.

LK (Relink)

Purpose: Relinks a program. (Allows you to change program attributes in a previously linked program file.)

Syntax: `LK,`*filedescriptor*

Parameters: *filedescriptor* Specifies the name of the program file to be relinked.

Remarks: The LK command is used to specify a program file in which various program attributes are to be changed. Commands that can be used are as follows:

`AB`	Abort relinking, do not modify program file.
`AL`	Lock or unlock code segments in memory.
`AS`	Assign partition number.
`CD`	Specify number of code blocks.
`DM`	Set/reset Debug Monitor mode.
`EC`	Echo command file to list device or file.
`EM`	Set EMA size.
`EN`	End relinking, modify program file.
`HE`	Set size of heap area.
`LL`	Set list device or file.
`LO`	Examine program attributes.
`ML`	Memory lock specified code segment.
`OS`	Operator suspends LINK.
`PC`	Set a program's capabilities.
`PR`	Set program priority.
`SH`	Specify SHEMA block.
`SP`	Specify program as shareable.
`ST`	Set size of stack area.
`SZ`	Set program partition size.
`TR`	Transfer control to command file.
`VM`	Specify VMA backing store file size.
`WS`	Set VMA working set size.

LL (List Option)

Purpose: Specifies the list file or the LU number of the list device to which LINK sends messages and the load map.

Syntax: `LL,`*filedescriptor*`|`*lu*

`+LL:`*filedescriptor*`|`*lu* (in runstring)

Parameters:

filedescriptor Specifies the name of the list file.

lu Specifies the LU number of the list device.

Remarks: If you specify the name of an existing file, the existing name must include type extension .MAP; otherwise, an error occurs. Also, if you specify the name of an existing file, the contents of the file are overwritten.

If you specify a new file name and do not include a type extension, LINK appends the type extension .MAP to the file name.

The LL command can be used during relinking.

LO (List Program Attributes)

Purpose: Lists program attributes during relinking.

Syntax: `LO`

Remarks: The LO command can be used to list the current values of the program attributes only when you are relinking the program.

MA (Send Load Map to Terminal)

Purpose: Directs LINK to list the load map at your terminal.

Syntax: `+MA` (in runstring)

Remarks: The MA command can be used only in the LINK runstring.

ML (Memory Locked)

Purpose: Declares that the specified code segment is to be memory locked or unlocked. (For CDS programs only.)

Syntax: `ML`

`ML,<`*segment #*`>[,UN]` (Relinking syntax)

Parameters:

segment #	Specifies the number of the segment to be locked or unlocked in memory.
`UN`	Unlocks the segment specified.

Remarks: This command does not affect program behavior but can affect program performance.

The ML command can be used during relinking.

MS (Multiple Search)

Purpose: Searches a library file.

Syntax: `MS,`*filedescriptor*

Parameters:

filedescriptor	Specifies the name of the library file to be searched.

Remarks: Each subroutine in the library file is searched sequentially. If an external reference can be resolved, the subroutine is included in the program file.

If any external references are resolved during the search, the file is searched again to resolve any backward references from the subroutine that was just included in the program file. This process is repeated until no additional references can be resolved.

A more efficient approach to searching unindexed libraries is to use LINDX to index the file, and then use the LI or SE command to search the file.

NA (Name)

Purpose: Sets the program file name. The name is tested immediately for validity.

Syntax: `NA[,`*filedescriptor*`]`

Parameters:

filedescriptor Specifies the name to be tested for validity and used for the program file name. If you omit this parameter, LINK tests for validity the name specified in the PROGRAM statement in the source file (or NAM statement in Macro), in a previously entered OU command, or in the LINK runstring (.RUN file type extension).

Remarks: If you enter a name that is not valid, a warning message is displayed. If the name is valid, it is used to create the program file. A file name specified in the EN command overrides the name specified in the NA command.

NS (New Segment)

Purpose: Starts a new code segment. (For CDS programs only.)

Syntax: `NS[,`*pages*`]`

Parameters:

pages Specifies the maximum number of pages in a code segment. Pages can be set to a value between 1 and 31 pages, inclusive. If this parameter is omitted, LINK uses the value of 30.

Remarks: LINK starts a new segment every time the specified number of pages is reached.

The command does not affect program behavior but can affect program performance.

OR (Order EMA Area)

Purpose: Allows you to specify the order in which EMA areas are allocated.

Syntax: `OR,<`*ema area1*`>,<`*ema area2*`>. . . .`

Parameters:

ema area Specifies the name of an EMA area declared in the source file.

Remarks: Multiple OR commands can be specified in one LINK session. The EMA area names listed in the first OR command are placed before the names listed in the second OR command, and so forth.

If an EMA area name is specified more than once, the first occurrence is used. If you specify the name of an EMA area that does not exist, LINK issues a warning and ignores the name.

If the command is not specified, LINK chooses the order in which the EMA areas are allocated. You cannot order EMA common blocks that are declared in a block data subprogram.

You can use the PA command with the OR command to specify that an EMA area is to begin on a page boundary. The first EMA area always begins on a page boundary.

OS (Operator Suspend)

Purpose: Allows you to suspend LINK.

Syntax: `OS`

Remarks: LINK remains suspended until you enter the system GO command. While LINK is suspended, you have access to CM. You may want to suspend LINK and use CM to display a directory list or run another program.

The OS command can be used during relinking.

OU (Output)

Purpose: Specifies the name of the program file.

Syntax: `OU,`*filedescriptor*

Parameters: *filedescriptor* Specifies the name of the program file.

Remarks: The OU command prevents LINK from automatically purging an existing program file. After you enter the EN command, LINK issues a warning and aborts if the file already exists. You can either rename the existing program file or use a different program file name.

The name entered with the OU command is not checked for validity until you end LINK. If the name is invalid, LINK issues an error message. You have to run LINK, re-enter your LINK commands, and use a valid file name. If you want to check the validity of the name before ending LINK, enter the NA command without parameters.

Specifying a file descriptor with the EN command overrides the name specified with the OU command.

PA (Page Align EMA Area)

Purpose: Starts the next Extended Memory Area (EMA) specified with an OR command on an even page boundary.

Syntax: `PA`

Remarks: The PA command is useful if a single VMA/EMA transfer of more than 31 pages is done in the program. In this case, the area should start on a page boundary to avoid mapping problems.

The first EMA area always begins on a page boundary; therefore, this command is useful only if used with the OR command. For example,

```
or,ema1,ema2,ema3
pa
or,ema4
pa
or,ema5
```

In this example, the PA and OR commands are used to force EMA4 and EMA5 to begin on a page boundary. EMA1 also begins on a page boundary because it is the first EMA area relocated.

PC (Set Program Capability)

Purpose: Sets the capability of the program to be linked and, optionally, sets the capability needed by a user to run the program.

Syntax: `PC,`*progcplv*`[,`*rquscplv*`]`

Parameters:

progcplv Specifies the capability level assigned to the program.

rquscplv Specifies the capability level required by a user to run a program. If omitted, LINK defaults *rquscplv* to 0.

Remarks: This command is used in conjunction with Security/1000. Security need not be on for this command to be issued. If this command is omitted, LINK sets the program capability level and the user capability level to zero.

PR (Priority)

Purpose: Sets the priority of the program.

Syntax: `PR,`*priority*

Parameters:

priority	Specifies program priority. The priority can be set to a value between 1 and 32767, inclusive, with 1 being the highest priority and 32767 the lowest.

Remarks: This command overrides the priority set in the PROGRAM statement in the source file. If this command is omitted and the priority was not set in the source file, LINK sets the program priority to 99.

The PR command can be used during relinking.

PS (Page Align Overlays)

Purpose: Starts overlays at a page boundary. (For non-CDS programs only.)

Syntax: `PS`

Remarks: The PS command may reduce the number of pages that must be swapped when an overlay is loaded. The command may increase the program partition size.

RE (Relocate)

Purpose: Relocates a file as part of the program.

Syntax: `RE,`*filedescriptor*

Parameters:

filedescriptor	Specifies the file to be included as part of the program.

Remarks: The relocatable file specified may contain a program, program overlays, procedures, or block data (FORTRAN).

RM (Relocate Module)

Purpose: Extracts a single module from a file and relocates the module as part of a program.

Syntax: RM,*filedescriptor*,*symbolname*

Parameters:

filedescriptor Specifies the file that contains the module to be relocated. The file can contain one or more modules and should be indexed for optimum performance.

symbolname Specifies an entry point for the module to be relocated. Often the entry point name is the same as the module name; however, a module can have several entry points.

Remarks: If the symbol is not found, LINK reports "module not relocated".

This command gives you more control over the selection of modules than the SE or LI commands. The RM command is most useful in the two following specific applications:

1. The RM command can be used when you want to use a software version of a machine instruction. For example, the following command instructs LINK to search /LIBRARIES/$MATH for the entry point .ENTR, and relocate this module into the program (note that this produces Warning 141: RPL replaced).

```
RM /LIBRARIES/$MATH .ENTR
```

2. The RM command can be used when you have a file that contains two or more modules that share a common entry point but also have individual entry points different from the shared entry points. For example, if the file LIB contains module RTA with entry points SUBR and SUBA, and module RT6 with entry points SUBR and SUB6, the following LINK commands can be included in a LINK command file to relocate the proper module and associated entry point.

```
RE PROG
IF A RM LIB SUBA
IF 6 RM LIB SUB6
EN
```

RO (Reorder)

Purpose: Rearranges the order of the modules in the program.

Syntax: `RO`

`+RO` (in runstring)

Remarks: For non-CDS programs, the RO command reorders the modules to reduce the number of base page links.

For CDS programs, the RO command reorders the non-CDS modules in the data segment. CDS modules in the code segment are not reordered because they use current page linking.

SC (System Common)

Purpose: Specifies that all references to blank common in the program are placed in blank system common.

Syntax: `SC`

`+SC` (in runstring)

Remarks: This command allows programs to share data.

SE (Search)

Purpose: Searches a library file to satisfy undefined external references.

Syntax: `SE[,`*filedescriptor*`]`

Parameters: *filedescriptor* Specifies name of the library file to be searched. If you omit this parameter, LINK searches the system library files.

Remarks: If the *filedescriptor* parameter is specified, LINK searches that particular file immediately. If an external reference can be resolved, the procedure is included in the program file. All backward references in the library are satisfied if the library is indexed. A warning is issued when an unindexed library is searched. The SE command cannot be used without a parameter to search the system library after an overlay is relocated. To do so would cause all system library routines to be loaded in the overlay.

The SE command can be used to search unindexed libraries; however, the search does not resolve any backward references in the file. It is recommended that you use LINDX to index the unindexed library, and then use the LI or SE command to search the library.

SH (Shareable EMA)

Purpose: Specifies that the EMA data resides in the shareable EMA partition specified. Use of this command makes the program a "SHEMA-only" program.

Syntax: `SH,`*label*`[,<`*partition #*`>]`

Parameters:

label Specifies the name of the partition (maximum of sixteen characters).

partition # Specifies a reserved partition number. Partition number can be set to a value between 0 and 1023. If this parameter is omitted, LINK uses the value of 0 (not reserved).

Remarks: The shareable partition is labeled with the specified label when the program is loaded into memory. All programs sharing the EMA partition must use the same label.

If a partition number is not specified but another program already has allocated the partition, the existing partition is used. If a partition number is not specified and the EMA partition does not exist, a partition is allocated in dynamic memory.

Only one shareable EMA area can be specified per program. The last SH,<label> command specified overrides any previous ones.

The SH command can be used during relinking.

SN (Snapshot)

Purpose: Defines or displays the name of the snapshot file.

Syntax: `SN[,`*filedescriptor*`]`

Parameters:

filedescriptor Specifies the snapshot file. If this parameter is omitted, the snapshot file name is listed to the list device or list file.

Remarks: The snapshot file is used to resolve system labels and entry points.

The SN command must be entered before any RE commands.

If you omit the SN command, LINK searches for a snapshot file to use as a default. LINK first searches for file SNAP.SNP::SYSTEM. LINK next searches for file SNAP::0, and, finally, file SNAP.SNP on your working directory (if you have a working directory).

SP (Shareable Program)

Purpose: Declares a program to be shareable. (For CDS programs only.)

Syntax:

```
SP
SP[,UN]          Relinking syntax
+SP              (in runstring)
```

Parameters:

`UN` Declares the program to be unshareable during relinking.

Remarks: The SP command sets a flag in the program file that indicates to the system that the program is shareable.

During relinking, you can use this command to set or unset the shareable program flag in the program file.

ST (Stack)

Purpose: Sets the size of the stack area. (For CDS programs only.)

Syntax: `ST,[+|-]`*words*

Parameters:

words Specifies the number of words in the stack area. Words can be set to any value between 6 and a maximum number that is dependent on the heap size and MSEG usage.

If the number of words specified is preceded by a plus (+) or a minus (−), then *words* is added to or subtracted from the default number of words estimated by LINK.

Remarks: The stack area is included in the data partition. If the maximum of 32 pages in the data partition has been reached, increasing the stack area decreases the amount of heap area, if possible.

If you omit this command, LINK estimates the amount of stack space needed by your program based on compiler generated information in the relocatable files. However, this estimate may be insufficient. If your program incurs a CS06 violation, this may indicate that LINK's estimate was too small.

The ST command can be used during relinking.

SU (System Utility)

Purpose: Declares the program to be a system utility.

Syntax: `SU[,OF]`

`+SU[:OF]` (in runstring)

Parameters: `OF` Specifies that any user can remove the program. If this parameter is omitted, only a superuser can remove the program. If the program is RP'd but not active at logoff, the program's ID segment is removed.

Remarks: The SU command is used for frequently used system utilities. A system utility cannot be cloned.

This command is useful for programs that use a lot of system resources or monopolize one particular system resource because the restriction on cloning prevents multiple copies of the program from running simultaneously.

If a program is loaded as a system utility, it should normally be RP'd as a permanent program, that is, the "T" option should not be specified in the RP command or FmpRpProgram call. This should be done because if the system utility is executing while another program attempts to schedule it, that program will queue-suspend on the system utility. If the system utility is not permanently RP'd, its ID segment is removed as soon as it terminates. This will cause the queue-suspended scheduler to receive an SC05 violation because the program to be scheduled no longer exists.

SZ (Size)

Purpose: Specifies the number of physical memory pages required to run the program. For CDS programs, this command specifies the number of data partition pages.

Syntax: `SZ,`*pages* or `SZ,+`*pages*

`+SZ:`*pages* or `+SZ:+`*pages* (in runstring)

Parameters: *pages* Specifies either the number of pages in the program partition or, if you include the plus sign, the number of pages to be added to the minimum number of pages required by the program. Pages can be set to a value between 2 and 32, inclusive.

Remarks: This command allows the program to use space beyond its program boundary. The size of system common area is not counted in the size specification of the program. If the number of pages specified is larger than the number of pages available, the available pages are used. If this command is omitted, the minimum number of pages required by the program is used.

The SZ command can be used during relinking on non-CDS programs only.

TR (Transfer)

Purpose: Transfers control to a file containing LINK commands.

Syntax: `TR,`*filedescriptor*

Parameters: *filedescriptor* Specifies the name of the command file. If you do not specify the file type extension, LINK assumes the .LOD file type extension.

Remarks: Control is transferred to the specified file. Control returns to the point of transfer unless LINK aborts or is terminated while the command file is executing.

A maximum of 128 command files can be used in one LINK session. Command files can be nested.

The TR command can be used during relinking.

VM (Virtual Memory Size)

Purpose: Indicates that the program is a Virtual Memory Area (VMA) program and specifies the size of the VMA backing store file. For backward compatibility, this command may also be entered as VS.

Syntax: `VM[,`*pages*`][,`*model*`]`

Parameters: *pages* Specifies the maximum number of pages in the virtual backing store file. Pages can be set to a value between 32 and 65536, inclusive. If this parameter is omitted, LINK uses the value of 8192.

model Specifies the VMA model as follows:

- `L` Use Large VMA model. Refer to the *RTE-A Programmer's Reference Manual*, part number 92077-90007, for a description of the VMA models.
- `X` Use Extended VMA model.

If a model is not specified, the Normal VMA model is used.

Remarks: The file is allocated on disk when the VMA working set cannot hold more data. The file grows in size as needed, but does not exceed the limit imposed by the VM command.

Although LINK allows 65536 pages (128 Mbytes) of virtual space, the volume on which the backing store file is created may limit the maximum size.

The VM command can be used during relinking. However, the VMA model cannot be changed during relinking.

WD (Default Working Directory)

Purpose: Provide a default directory and node name to be used when LINK opens files.

Syntax:
```
WD directory[/@>dsinfo]
+WD[:directory[/@>dsinfo]]    (in runstring)
```

Parameters:

directory Specifies the default directory path to use when opening a file. If this parameter starts with a / (slash), then the path starts at the global directory. Otherwise the path is relative to the current session's working directory.

`@` A placeholder for the file name.

dsinfo The DS transparency specifier of a remote node and optional user logon of the following form:

node[*user*/*password*]

Remarks: The default working directory is applied to the file open operation for all RE, RM, SE, LI, and TR commands. The directory can contain subdirectories, and can be relative (that is, "../") path. The directory and node name parts are defaulted separately. For LI library files, the defaulted values used are those in effect when the EN (end) command is issued, and not the values in effect when the LI command is issued.

In the runstring, if the +WD option is used in front of the file descriptor of a .LOD file, the default directory path becomes the same as the one used by the .LOD file. For example, the runstring:

```
CI> link +wd /utilities/util/util.lod>96
```

causes LINK to set its default directory path to:

```
/UTILITIES/UTIL/@>96
```

There is no nesting of default working directories. That is, if one is specified in a TR file, then it continues to be used even after the command file returns.

WS (Working Set Size of VMA)

Purpose: Specifies the working set size of VMA.

Syntax: `WS[,`*pages*`]`

Parameters:

pages Specifies the number of pages in the working set. Pages can be set to a value between 2 and 1022, inclusive, for Normal and Large model programs, and between 2 and 32733 for Extended model programs. If this parameter is omitted, LINK uses the value of 32. (Refer to the *RTE-A User's Manual* for descriptions of EMA models.)

Remarks: The number of pages specified in the WS command does not include the number of page table (PTE) pages required:

Normal	1
Large	2
Extended	2 + 1 per 1,024 data pages

The WS command can be used during relinking.

* (Comment)

Purpose: Allows entry of comments.

Syntax: *

Remarks: When the asterisk (*) is the first character in a line, LINK ignores the entire line. The asterisk can be used to add comments to a LINK command file.

? (Help)

Purpose: Displays help information.

Syntax: `?[,`*link command*`]`

Parameters:

link command Specifies the LINK command for which you want information. If this parameter is omitted, LINK displays a summary of all commands.

Remarks: The question mark (?) can be entered during an interactive LINK session any time you need help with a LINK command or want to see a syntax summary of all LINK commands.

4

Advanced Concepts

This chapter describes some features and functions of LINK that are useful for experienced LINK users. Two sample load maps are provided at the end of this chapter to illustrate LINK features.

Linking Files for Different Target Systems

In RTE-A, the system snapshot file, which is created when the system is generated, contains a description of the operating system (defined system libraries, boundaries, and entry points). Each RTE-A system has a unique snapshot file. LINK uses the snapshot file to reference operating system related information and for automatic searching of system libraries.

To specify the name of the snapshot file to be referenced, use the SN command, or specify the name of the snapshot file in the runstring.

Reducing Base Page Links

The number of links on the base page can be reduced by reordering the sequence in which the modules are relocated. Use the RO (reorder) command to reorder the modules. Reordering is necessary when the base page link area overflows. Base page links are used by modules in the data segment in CDS programs and by non-CDS programs. See the *RTE-A System Design Manual*, part number 92077-90013, for more information on base page links.

System Common Allocation

System common is an area of memory that is accessible to more than one program. Therefore, information can be stored in this area by one program, then used later by other programs in a predetermined manner. The LC and SC commands can be used to access system common. Refer to the *RTE-A System Design Manual* for more details on using system common.

VMA/EMA Allocation

When you specify Virtual Memory Area (VMA) or Extended Memory Area (EMA), LINK allocates a minimum of two pages for the EMA area (or VMA working set) plus the required number of pages for the page table. The default order for mapping labeled common blocks into a program is not specified. It is not necessarily dependent on the order of the common statements declared in the program. The ordering may change with different revisions of LINK or the compiler. If the ordering is important, as with Shared EMA, use LINK's OR (Order EMA Area) command to specify the EMA common block order.

MSEG Allocation

The mapping segment (MSEG) is the part of the EMA area or VMA working set that is logically visible (mapped in) to your program at one time. If you use a compiler option to specify a size for MSEG, LINK allocates the specified number of pages plus an additional spillover page. If you do not specify a size for MSEG (and do not specify a program size), LINK allocates all available remaining program (data segment) space to MSEG.

CDS Program Space Allocation Considerations

The data segment contains the following:

- Base page
- System common
- Stack area
- Heap area
- Global data
- Non-CDS code
- MSEG

A maximum of 32 pages is allowed. If the data segment has reached the maximum of 32 pages, none of the above areas can be increased unless the size of another area is decreased. LINK issues an error message and aborts if it cannot decrease the size of any area in the data segment.

Using EMA and Libraries

If your program uses library modules that use EMA and are the only modules in the program that use EMA, you must inform LINK that the program requires EMA. Because LINK needs to know whether or not a program requires EMA before it starts the search sequence, you must use the EM (or VM) command before the EN command. Otherwise, when LINK encounters a library module that uses EMA during the library search sequence, the message "Library routine uses EMA" (fatal error 120) is issued.

Merging and Indexing Libraries

For high-speed performance using LINK, it is recommended that you always merge and index your library files and relocatable files containing external modules into large indexed library files. If you index your libraries, LINK finds all backward references.

When generating a new system, the system manager should merge any commonly used libraries and current system libraries into one library, and index the merged library.

If using modules from many different libraries, you should merge the modules into one library, then index the library to eliminate repeated searches for backward references.

When you run MERGE, the resulting merged library is not indexed, even if the individual libraries that form the input to MERGE are indexed. After merging, you must run LINDX on the merged file in order to reindex them. See the *RTE-A Utilities Manual*, 92077-90004, for a detailed description of the MERGE utility.

LINDX

LINDX indexes libraries. The format of the LINDX command is as follows:

LINDX,<*input file*>,<*output file*>[,+NL][,+L *listfile*]

where:

input file is the file to be indexed.

output file is the file to which the indexed library is written. It must be different from the input file.

+NL specifies no list of entry points.

+L *listfile* sends the entry point listing to the specified file.

LINDX reads the specified input file, then copies the contents to the output file and includes an index of the modules at the beginning. When searching the library, LINK uses the index to quickly access the names and locations of modules in the library file. If there is no type extension specified in the input file name, LINDX uses .LIB.

When indexing a file, LINDX indicates if duplicate entries occur and indexes only one of the entry points. The other module is written to the indexed library but cannot be accessed because it is not listed in the index.

File Naming Defaults

You may choose to default all or part of a file name. The defaults LINK uses depend on whether you have a working directory, whether you have specified a directory or FMGR cartridge, whether you have specified a file type extension, and what command the file is associated with. LINK chooses the first one that works from the naming defaults given below.

Snapshot File

If a snapshot file is not specified, LINK searches for the snapshot file in the following sequence:

`SNAP.SNP::SYSTEM`	on directory /SYSTEM
`SNAP::0`	on a FMGR cartridge
`SNAP.SNP`	on working directory (if you have a working directory)

Relocatable Files

If a file type extension or directory is not specified, LINK searches for relocatable files in the following sequence:

<*filename*>	
<*filename*>`.REL`	on working directory

LINK Command Files

If a file type extension or directory is not specified, LINK searches for the LINK command file in the following sequence:

<*filename*>	
<*filename*>`.LOD`	on working directory

Library Files

If a file type extension or directory is not specified, LINK searches for library files in the following sequence:

<*filename*>	
<*filename*>`.LIB`	on working directory
<*filename*>`.LIB::LIBRARIES`	on directory /LIBRARIES

Program Files

If the program name is not specified, LINK takes the name from the relocatable module that contains the primary entry point.

If a directory is not specified and you have a working directory, the file is placed on the working directory. If a directory is not specified and you do not have a working directory, the file is placed on the first FMGR cartridge.

If a file type extension is not specified and the file is being placed on a new directory, the file type extension .RUN is given to the file. If a program file with the same name already exists, LINK, by default, purges the file before creating the new one. You can use either the OU or DP command to inhibit purging of an existing program file.

List Files

If a directory is not specified, the list file is placed on the working directory. If there is no working directory, the list file is placed on the first FMGR cartridge.

If a file type extension is not given and the list file is to reside in a new directory, the file type extension .MAP is used.

Runstring Defaults

If you have a working directory, any file in the runstring without a directory specification is assumed to be in the working directory. Any file in a different directory must include the directory name in the file descriptor.

If you do not have a working directory, any file in the runstring without a directory specification are searched for on the FMGR cartridges in the order the cartridges are mounted.

Any default file type extensions must conform to the standard convention described in the LINK File Type Extension section in Chapter 2. Otherwise, LINK may misinterpret the file contents; for example, file type extension .RUN indicates a program file and .REL indicates a relocatable file.

Remote Files

LINK does not supply default file type extensions or directory paths when accessing files using DS transparency. You must specify the entire file descriptor.

For example (user input is underlined):

```
link: re,area.rel::mydir>5003
   AREA
link: se,libry>5003
File not found:  LIBRY>5003
link: se,libry.lib::system>5003
   SUB1 SUB2 SUB3
```

In this example, LINK issued an error message for the first SE command because the entire file descriptor for the remote file was not specified.

Error Reporting on Default Names

If LINK creates default names when looking for a file and an error occurs, the error is reported on the last name LINK tried. This most commonly occurs when the file is not found.

For example (user input is underlined):

```
link: re,area
   AREA
link: se,libry
File not found:  LIBRY.LIB::LIBRARIES
```

In this example, LINK searched first for LIBRY, then LIBRY.LIB on the working directory, then for LIBRY.LIB::LIBRARIES. Because LINK could not find any of the names, it reported the error as having occurred on the last one it tried.

Relinking

During relinking, you can permanently change certain attributes of programs that have already been linked. Because relinking operates directly on the program file, this process cannot be used if the program file is being used (for example, the program is running).

Relinking is faster than linking the program again. The following LINK commands can be used during relinking:

AB	Abort relinking, do not modify program file
AL	Lock or unlock all code segments in memory
AS	Assign the program to a partition
CD	Specify the number of code segment blocks
DM	Set/reset Debug Monitor mode
EC	Echo command file to list device or file
EM	Set EMA size
EN	End relinking, modify program file
HE	Set size of heap area
LL	Set list device or file
LO	Examine program attributes
ML	Memory lock specified code segment
OS	Operator suspends LINK
PC	Set program's capability level
PR	Set program priority
SH	Specify SHEMA block
SP	Specify program as shareable
ST	Set size of stack area
SZ	Set program as shareable
TR	Transfer control to LINK command file
VM	Set VMA backing store file size
WS	Set VMA working set size

Chapter 3 contains detailed information on these commands.

To relink a program, run LINK interactively or from a command file and use the LK command to begin the relinking session. The LK command gives you access to the commands listed above. During relinking you can use the LO command to examine the current program attributes.

When you finish making changes, enter the EN command to end the relinking session with LINK modifying the program file, or enter the AB command to end the relinking session without LINK modifying the program file.

The following is an example of starting a relinking session:

```
link: lk area
Ready to modify AREA
Use LO to examine program attributes
link:
```

If you enter a command other than a valid relink command, LINK displays one of the following messages:

```
Not a legal command in relink mode:  <command entered>
```

or

```
Unknown command; use ? for help
```

Relinking CDS Programs

While relinking CDS programs, an ST (stack size) command that increases the stack size first makes the data partition larger to accommodate the larger stack space. After the data partition reaches its maximum size, the heap space is reduced to accommodate the larger stack. An ST command that reduces the stack size may reduce the partition size.

After an ST command, any extra space required to round up the partition to a page boundary is allocated to the heap. However, if the heap is the minimum size of 4 words, then any extra space is allocated to the stack.

An HE (heap size) command that increases the heap space first makes the data partition larger to accommodate the larger heap space. After the data partition reaches its maximum size, the stack space is reduced to accommodate the larger heap. An HE command that reduces the heap size may reduced the partition size. After an HE command, any extra space required to round up the partition to a page boundary is allocated to the stack.

As a relinking example, to specify as much stack as possible and 4K words of heap, use:

```
link: lk
link: st 32000
link: he 4096
```

The ST command first increases the data partition to the maximum size, and reduce the heap to its minimum size. Then the HE command causes the stack to be reduced to accommodate the required 4K heap size.

Note that LINK does not issue warnings when it reduces the sizes you specify. You should use the LO command to check that the resulting stack and heap sizes are what you expect.

Searching for Block Data Subroutines

Block data subprograms (FORTRAN) can be picked up in library searches. A block data subprogram matches named common in an explicitly relocated module. However, if both a block data subprogram and a subroutine that uses common blocks initialized by that subprogram are placed into a library with LINDX, the block data subprogram and the subroutine require the following FORTRAN declaration:

```
$ALIAS /COMMON_BLOCK_NAME/, NOALLOCATE
```

Using CI Command Files to Run LINK

During program development, you can use a command file to link and run the application program after each change to the program. By using CI positional variables ($1 through $9), you can write one command file for all occasions.

When LINK finishes, it places information into the predefined variables, $RETURN1 through $RETURN5 and $RETURN_S, and indicates whether an error occurred. At completion, the predefined variables contain the following:

$RETURN1: characters 1 and 2 of the program name

$RETURN2: characters 3 and 4 of the program name

$RETURN3: characters 5 and 6 of the program name

$RETURN4:

0	if the link was successful
−1000	if the user aborted the link
nnn	if the link was unsuccessful, where *nnn* is the fatal error code as summarized in Appendix A

$RETURN5: unused (blanks)

$RETURN_S: file descriptor for the program file

The following is a sample command file for linking and, if the link is successful, running the program:

```
LINK $1.REL
IF is $return4 eq 0
THEN $1
FI
RETURN
```

If the name of the command file is LINKR, the following command executes LINKR (EXAMPLE is the name of the relocatable file):

```
CI> linkr example
```

Relocation Sequence

LINK uses the following sequence when relocating modules:

1. Before any relocatable modules are processed, all RPLs in the snapshot file are entered into LINK's symbol table as defined symbols. These RPLs specify the machine instructions needed to run the program created by LINK.

2. Relocatable modules are then processed in the order they are encountered. The RE, RM, SE, or MS commands can be used to relocate modules.

   RE command: All modules in the specified file are immediately relocated.

   RM command: Only the module with the specified entry point is relocated.

   SE command: All modules in the specified file that satisfy undefined external references are relocated. If the file is indexed, all backward references within the file are resolved; otherwise, the file is searched linearly only once.

   MS command: Similar to the SE command, except that the file is searched multiple times until no more modules are relocated (all backward references have been resolved).

After LINK has finished relocating modules and the EN command is given, LINK attempts to satisfy undefined external references. Files are searched in the following sequence:

1. System labeled common is searched, if it has been specified using the LC command.

2. Libraries specified with the LI command are searched in the order given. If the file is indexed, all backward references within the file are resolved; otherwise, the file is searched linearly only once.

3. Each system library is searched to resolve backward references. If a CDS program is being linked and CDS system libraries exist for the target system, the CDS libraries are searched; otherwise, the non-CDS system libraries are searched.

4. The snapshot file is searched to resolve system references.

For non-CDS overlay programs, the sequence described above has several implications. Keep the following in mind when linking an overlay program:

- A subroutine called by more than one overlay must be relocated with either the main or each overlay that calls it, or it must reside in a library to be searched.

- Subroutines relocated with the main are not linked into program overlays.

- Any subroutines relocated before the first program overlay are linked into the main program.

- Any subroutines relocated after the first overlay and before the second overlay are linked into the first overlay.

- Similarly, any subroutines relocated between overlays are linked into the preceding overlay, and subroutines relocated after the last overlay are linked into the last overlay.

If undefined externals are reported after EN is entered, a new overlay cannot be relocated to satisfy the undefined externals. This generates fatal error 108.

Subroutines must be relocated with the overlays (or the main) to which they belong, or in a library to be searched.

Note If you have used other RTE loaders, this may not be the search sequence to which you are accustomed. The most important differences are that when using LINK, subroutines are loaded with the overlay and libraries cannot be concatenated to the end of the overlay relocatable file.

If the relocatable file you are linking was produced for another RTE system and contains an overlay relocatable program, you may have to change it if any of the following is true:

- Subroutines called from an overlay are not relocated directly after that overlay, nor in the main, nor in a library. That is, the relocatable file resides with the relocatable for another overlay or in a library appended to the file to be linked.
- Subroutines called from the main are not relocated in an overlay or directly after the main. That is, the relocatable resides in a library appended to the file to be linked.

The most important class of file that meets these requirements is an overlay program with libraries concatenated to the end of the file. Such files must be changed in some combination of the following ways to be linked:

- Each subroutine called by more than one overlay must have a copy immediately following each calling overlay in the file, or must reside in a library searched by LINK, or must reside in the main.
- Any subroutine called by the main must reside directly after the main or in an overlay or in a library searched by LINK. Note that subroutines residing in an overlay are linked into that overlay and not with the main.

CDS Library Considerations

In VC+ systems, two sets of libraries can be specified; CDS libraries and non-CDS libraries. The CDS libraries are searched only when the program being relocated is a CDS program.

If a program has a CDS main and a non-CDS subroutine that calls an FMP subroutine, you must explicitly search library $BIGLB (LI,$BIGLB). If $BIGLB, which contains non-CDS versions of the FMP subroutines, is not searched explicitly, the CDS version of the FMP subroutine is taken from library $BGCDS. LINK issues an error because a non-CDS subroutine is illegally calling a CDS FMP subroutine.

Separate versions of the FORTRAN formatter routines with different names exist for CDS and non-CDS code. As a result, searching $BIGLB does not pull in non-CDS versions of the FORTRAN formatter when CDS is on.

Manual Program Segmentation

LINK provides automatic segmentation for CDS programs, which provides good performance for the majority of programs. Most users do not need to be concerned with program segmentation. However, in some cases, manual segmentation can improve user program performance. This section is for users who want to gain as much performance as possible by manually segmenting their programs.

The goal in manual segmentation is to reduce the number of procedure calls from one segment to another. This is because cross-segment calls require more time than in-segment calls. Cross-segment calls to segments residing on the disk are very costly, because the segment must be brought in from the disk; therefore, the number of calls made to segments on the disk should be reduced. Under automatic segmentation LINK sets the code partition size so that all segments can reside in memory simultaneously. Here are some general rules for achieving higher program performance by manual segmentation:

- Group routines that call each other frequently into the same segment using the NS command.
- Create as few segments as possible.
- Allow as many code segment blocks in the code partition as possible with the CD command.

The ML command can be used to keep a particular segment in memory. There is no advantage to this unless the CD command has been used to reduce the number of segment blocks to less than the number of segments.

Example Non-CDS Program LINK Session

A non-CDS program called EMATX that has two overlays, OVER1 and OVER2, is being linked with a command file. The command file and the load map produced by LINK are shown below. Lines have been deleted from the listing; only the pertinent sections of the maps are displayed.

The names and addresses of each EMA area declared in the program are listed before the load map. This information is useful when trying to trace VMA/EMA problems.

The load map shows name, size, and address of each module linked with the program. This information is useful if the system aborts the program and displays the address of the offending instruction.

As each module is relocated, the module name is printed to the screen or list file, with up to ten names per line. Each of the module names is printed. After the EN command is entered, LINK searches libraries and displays the names of the required modules it finds. A sample display format is shown below:

```
(name) (start address)        (length) (comment)
 EIO.    3337                 43. 24998-1X329 REV.xxxx 910303
```

The module name is .EIO, with a starting address of 3337 octal. The number 43 is the decimal representation of the length of the module in words. LINK uses a period at the end of all numbers to indicate decimal representation. The rest of the line is reserved for comments supplied by the module.

LINK concludes the load map with information showing the program name, size, and other information.

In the following example, LINK displays information in the load map indicating that this program needs six pages for the program and 64 pages to hold the EMA area. Therefore, 71 pages (6+64+1) are needed for execution. The extra page is added because every EMA program requires one page of overhead.

When LINK reports the program size in the summary line, it is reporting the number of pages needed for the program's data partition. This number is not the same as the page number of the highest used data space address when system common is used.

The contents of the command file are as follows:

```
*
*   LINK command file to link EMATX,
*   which is an overlay EMA program
*
ec
*   Specify list file
11  ematx.map
*   Specify 'Reorder' command
ro
*   Relocate
re  ematx.rel
*   Terminate relocation and specify program file
en  ematx.run::dougl
```

The list file produced is as follows:

```
       9:04 am Mar 22, 1991

*   Specify 'Reorder' command
ro
*   Relocate
re  ematx.rel

   EMATS
   OVER1
   OVER2
*   Terminate relocation and specify program file
en  ematx.run::dougl
   $EMA$  $INIT   VMAST  PNAME SEGLD  XREIO  REIO   LOGLU  $CVT1
   $CVT3  .EIO.   .FMIN  .FMCN .FMCV  .FMFP  .FMIO  .FMUI  .FMO?
   .FMER  .FMGB   .OPN?  .FION .UFMP  .IOER  PAU.E  ERO.E
   IFBRK  .IIO.   .EXIT  .IOCL .IOCM
   IFBRK  .IIO.   .EXIT  .IOCL .IOCM

EMA areas   Starting Address
AREA2       0
AREA4       41740
AREA3       62760
AREA1       110200
```

```
Load map:
EMATS    2000    99.  EMA test for LINK  <920222.1049>
$EMA$    2143   101.  92077-1X039  REV.2326     <850222.1049>
$INIT    2310    86.  92077-1X040  REV.2213     811120
VMAST    2436    35.  92077-1X041  REV.2213     811105
PNAME    2501    24.  92071-1X210  REV.2041     800409
SEGLD    2531    47.  92077-1X105  REV.2226     <850222.1049>
XREIO    2610   139.  92077-1X422  REV.2326     <850222.1049>
REIO     3023   131.  92071-1X212  REV.2041     800501
LOGLU    3226    20.  92077-1X205  REV.2326     <850222.1049>
$CVT1    3252     7.  92071-1X321  REV.2041     800530
$CVT3    3261    46.  92071-1X322  REV.2041     800530
.EIO.    3337    43.  24998-1X329  REV.2226     820503
.FMIN    3412   176.  24998-1X344  REV.xxxx     830316
.FMCN    3672    50.  24998-1X345  REV.2226     820107
.FMCV    4000   667.  24998-1X333  REV.XXXX     821014
.FMFP    6000   676.  24998-1X346  REV.2226     820426
.FMIO    5233   147.  24998-1X348  REV.xxxx     830316
.FMUI   10000   603.  24998-1X349  REV.2140     810416
.FMO?    5456    50.  24998-1X351  REV.2140     810415
.FMER    5540    42.  24998-1X352  REV.2226     820412
.FMGB    7244   175.  24998-1X353  REV.xxxx     830316
.OPN?    3754    20.  24998-1X325  REV.2101     800803
.FION    5612    22.  24998-1X355  REV.2226     820426
.UFMP    5640    14.  24998-1X296  REV.2226     820426
.IOER    7523   102.  24998-1X321  REV.2140     810506
PAU.E    5656     1.  24998-1X254  REV.2001     750701
ERO.E    5657     1.  24998-1X249  REV.2001     750701

Overlay OVER1
OVER1   12000    64.
IFBRK   12100    23.  92071-1X199  REV.2041     800409
.IIO.   12127   129.  24998-1X343  REV.XXXX     821213
.EXIT   12330    44.  24998-1X320  REV.2101     800731
.IOCL   12404    66.  24998-1X305  REV.2101     800731
.IOCM   12506    36.  24998-1X327  REV.2101     801007

Overlay OVER2
OVER2   12000    82.
IFBRK   12122    23.  92071-1X199  REV.2041     800409
.IIO.   12151   129.  24998-1X343  REV.XXXX     821213
.EXIT   12352    44.  24998-1X320  REV.2101     800731
.IOCL   12426    66.  24998-1X305  REV.2101     800731
.IOCM   12530    36.  24998-1X327  REV.2101     801007

Main                 2000 - 11777      4096. words
Overlay OVER1       12000 - 12551       362. words
Overlay OVER2       12000 - 12573       380. words


Program EMATX.RUN::DOUGL:6:63 ready; 6 pages, 64 pages EMA
9:04 am Mar 22, 1991
Runnable only on an RTE-A system
```

Non-CDS Program Load Map Discussion

The following paragraphs explain some features of LINK that may cause some confusion for users of other RTE loaders. These areas are covered in the form of questions and answers.

Why are the addresses not in order?

The addresses are often in order: However, in this case, the RO command was used, which caused LINK to reorder the modules in an attempt to reduce base page links. LINK displays the modules in the order processed.

What is the significance of the overlays starting at 12000?

Address 12000 is a page boundary. Overlays always start on the next page boundary following the main when the RO command is used.

With overlays starting at a page boundary, is the space between the end of the main and the beginning of the overlays wasted?

Yes. Some memory is unused on almost every page with reordering. However, the amount lost is small and is almost always less than previous loaders lost with current page linking.

Why is there an overlap of the two overlay addresses?

The overlays use the same address space. They are mutually exclusive. When one is in memory, the other cannot be in memory. When the user program calls SEGLD (overlay loader), the code of the appropriate overlay is brought in and overlays whatever is in memory at those addresses. Each overlay always has the same starting address as every other overlay in that program.

Why isn't there any free memory displayed in the load map?

Free memory is the memory left between the end of the user program and the end of the partition. This memory cannot be accessed unless the SZ command has been entered. LINK, therefore, does not report it unless this command is used.

Example CDS Program LINK Session

An example CDS program is linked with the command file shown below. The load map is discussed in the following paragraphs.

The LINK command file is as follows:

```
*
*    LINK command file to link
*        an EMA CDS program
*
*    Specify list file
11 emats.map
*    Specify relocatable file
re emats.rel
*    Specify program file
en /dougl/emats.run
```

The resulting load map is as follows:

```
        9:23 am Mar 22, 1991
    Purging old file:  EMATS.RUN::DOUGL:6:65
    Load map:
    Code segment 0:  2 pages
    Data            Code
Address  size  Address      size
EMATS                              EMA test for LINK <850222.1049>
 2000            7.   2601   249.
DISKSIZE                           92077-1X505 REV.2326 <850222.1049>
 2007           13.   <no code>
.ASKD                              92077-1X512 REV.2326 <850222.1049>
 2024           40.   <no code>
$EMA$                              92077-1X039 REV.2326 <850222.1049>
 2074          101.   <no code>
$INIT                              92077-1X040 REV.2213 811120
 2241           86.   <no code>
VMAST                              92077-1X041 REV.2213 811105
 2367           35.   <no code>
(Lines omitted)                    :
    :                              :
IFBRK                              92071-1X199 REV.2041 800409
 3146           23.   <no code>
LOGLU                              92077-1X205 REV.2326 <850222.1049>
 3175           20.   <no code>
$CVT1                              92071-1X321 REV.2041 800530
 3221            7.   <no code>
$CVT3                              92071-1X322 REV.2041 800530
 3230           46.   <no code>
!EIO.                              24998-1Xyyy REV.zzzz 821216
 3306           45.   <no code>
!NFEX                              24998-1Xyyy rev.xxxx 830201
 3363           42.   3172          96.
TRACE_BACK
 3435           48.   3332         676.
```

```
SNUMO
 3515            2.   5225         219.
.EIO                                24998-1X329 REV.2226 820503
 3517           43.   <no code>
(Lines omitted)
     :


Number of segments      1         Segment size              2
Code partition size     3
Data segment size       7         Data partition size      72
Stack area          12015         15363   1767.
Heap area           15774         15777   4.
EMA area   64 pages

Program EMATS.RUN::DOUGL:7:64 ready; 7 pages, 64 pages EMA
  9:24 am Mar 22, 1991
Runnable only on an RTE-A system
```

Example CDS Program Load Map Discussion

A load map for a CDS program consists of segment headers, module entries, and the attribute summary.

The segment header is a single line that gives the segment number and the pages of code linked into that segment. Here is the corresponding line from the above example:

```
Code segment 0:  2 pages
```

A module entry normally consists of two lines. The first line gives the module name, and any comment associated with that module. The second line gives the data address where the module starts, the amount of data there, the code address where the module starts, and the amount of code there. The amount of code does not include any current page link areas. A module entry can also include a current page link area descriptor. This occurs only in multi-page modules that include current page areas within their boundaries. Following is an example of a module entry with current page link descriptor:

```
TMPUPD                      92091-16001  REV.2201 <850222.1049>
    3162 82.   5466         4645.
    CP Link address information (Physical address – transform =
    Physical  Xform       Module           Module relative address)
        5466   5466            0
        6047   5610          361
       10003   5732         2173
       12037   6054         4105
       14034   6176         5760
```

The column labeled “Module” gives module relative code addresses. The column labeled “Physical” gives segment relative code addresses. The information in this table can be useful if your program should unexpectedly abort because the system returns a code address as the point of violation. For example, assume that a MP violation occurred at code address 10007 in the segment

containing TMPUPD. To determine your module relative address you would subtract the transform 5732; for example, 10007 − 5732 = 2055 octal. This module relative address can be used in combination with a listing of the module (for example, a "Q" listing in FTN7X) to determine the offending instruction.

The link summary occurs at the end of the load map and includes information on the total amount of data space used, the size of the data partition needed, the number and size of the code segments, and the size of the code partition needed. The summary also includes the starting and ending points of the stack and heap areas. The gap between the stack and the heap areas is a 264-word overflow buffer created by LINK.

Security/1000

Certain LINK commands are protected by Security/1000. A user's capability level must be greater than some value set by the system manager in the security tables to issue the command. The system manager sets the capability level for the following commands:

PR	Set priority
OS	Operator suspend
LC	Use labeled system common
SC	Use system common
SH	Use shareable EMA

The PR command has two additional levels of protection, L1 and L2. A capability level greater than the value of L2 allows a user to set any priority. A capability level between the value of L1 and L2 allows a user to set priorities greater than or equal to 51. A capability level with a value less than L1 allows the user to set priorities greater than or equal to 99. An attempt to set priorities higher than allowed results in an error message indicating the user has insufficient capability.

For example, assume your capability level is 19 and L1 and L2 are set to 10 and 20, respectively. You can set a program's priority to be greater than or equal to 51 when linking or relinking it.

If, for another example, your capability is 10 and L1 and L2 are set to 11 and to 16, respectively, you can set a program's priority when linking or relinking to be greater than or equal to 99.

The PC command is also protected. You can set the capability level of the program (progcplv) or that needed by a user (rquscplv) to be only as high as your capability level. For example, if 15 is your capability level, 15 is the highest you can set either the program's capability level or the needed user capability level.

Security/1000 and the %SECON security feature may co-exist, but Security/1000 checks are performed first.

Time and Space Considerations

LINK is a VMA program; its working set size (WS) dictates its performance. LINK runs fast with a large WS and slow with a small WS. If LINK is running too slow, relink LINK, giving it a larger WS.

If necessary, LINK also creates a VMA backing store file. It first attempts to place it on the FMGR cartridge specified with the SC command during bootup. If no SC command was given, it tries to place it in the /SCRATCH directory. If /SCRATCH does not exist, it places it in the working directory. If no WD is set, it then attempts to place it in the first cartridge on the cartridge list with room enough for the first extent. You can use the CR command to explicitly specify a location for the backing store file.

5

Installation Guide

This chapter describes how to install LINK and LINDX, and how to increase system security. LINK and LINDX are supplied as type 6 files on the Primary System. Relocatables are supplied also.

Installing LINK and LINDX

File /RTE-A/LINK.LOD is used to install LINK. The default WS size is 128 pages and the default VMA size is 8192 pages. The minimum value that WS can be set to is 21. The minimum value that VM can be set to is 2046.

If LINK is too big to fit in memory, as indicated by an SC09 error when you run it, then reduce the WS size using the following steps:

1. Make a copy of LINK.RUN (it can be any name; we will refer to it as FOO.RUN).

2. RP FOO.

3. Reduce WS size of FOO using WS FOO <*n*>, where <*n*> is some value that allows FOO to run.

4. Run FOO and re-link LINK.RUN. Set the WS size of LINK.RUN to a value that allows it to run.

Increasing System Security

The System Manager can install LINK with the SECURITY ON feature. This feature allows only superusers to link programs having priorities between 1 and 98.

To set the SECURITY ON feature, relocate %SECON (92077-16783) before %LINKA as shown in the command file for installing LINK.

Using RTE-A LINK on RTE-6/VM

To install RTE-A LINK on an RTE-6/VM system:

Modify the LINK.LOD file to relocate the RTE-A file $COMPT, which contains compatibility versions of several RTE-A routines. Then use the installation information given on the preceding page and rename the program (for example, LINKA) to avoid having two files named LINK on your system.

Copy the system snapshot file produced by the RTE-A generator to your RTE-6/VM system. You must explicitly specify the RTE-A system snapshot file whenever you run RTE-A LINK on RTE-6/VM. Otherwise, RTE-A LINK uses the system snapshot default file defined for your RTE-6/VM system.

All libraries and files that RTE-A LINK needs must be present on your RTE-6/VM system. Copy these libraries and files to your RTE-6/VM system.

Now you can use RTE-A LINK on your RTE-6/VM system to produce program files that can be copied over to and run on an RTE-A system (the program files produced by RTE-A LINK do not run on your RTE-6/VM system).

The LINDX program is available on both RTE-A and RTE-6/VM. If LINDX already exists on your RTE-6/VM system, you can use this copy of the program.

A

Error Messages

When you run LINK, error messages can come from either LINK or from the file management package (FMP). Refer to the *RTE-A Programmer's Reference Manual*, part number 92077-90007, for the FMP error message format.

Fatal error messages from LINK have the following form:

message
<*additional information***>**
Fatal error *nnn* **– Link terminated**

message	A brief description of the error.
additional information	A supplementary explanation of the error.
nnn	A three-digit number, 100 or higher.

When a fatal error occurs, LINK displays an error message and terminates. Some errors are recoverable. In these cases, LINK displays a warning message. The message format is similar to that of the error format described above with the word "Warning" preceding the error message.

Warning *nnn***: message**

However, certain warnings can become fatal if the corrective action cannot be taken, such as in a non-interactive situation. If such errors occur while processing a command file, LINK considers them as fatal errors and terminates.

LINK also displays messages in explanation of some FMP errors in addition to the error or warning code message.

The rest of this section contains expanded descriptions of each LINK error message.

Unexpected eof from command file
Last module relocated: <*module name***>**
Fatal Error 100 – Link terminated

More commands are required to link the program than are in the command file. This often happens in combination with some other error. For example, your program may have undefined externals (for which an error is generated), and then LINK goes back to the command file for more commands. If there are none, LINK aborts with error 100.

Error 101 – Not used

Error 102 − Not used

Warning 103: Too many libraries

The maximum number of user libraries that can be specified using the LI command or specifying libraries in the LINK runstring is ten (10).

Warning 104: Can't change snap now

Once the RE command is given, the snapshot file cannot be changed.

Error 105 − Not used

Warning 106: Illegal name

Names cannot be numeric.

Legal:	A324	FORTY
Not legal:	1234	−18

No modules relocated
Fatal Error 107 − Link terminated

The EN command was given, but there is nothing to relocate.

Module not relocated
Last module relocated: <*module name***>**
Fatal Error 108 − Link terminated

LINK was unable to relocate the given module. This often happens because the given module was not a relocatable module (for example, the given file is a program file). This also occurs when attempting to relocate a new segment after EN has been entered.

Warning 109: Illegal sum of checksums

This relocatable has a checksum error; the source should be recompiled to repair the error.

Illegal relocatable (Ext)
Last module relocated: <*module name***>**
Fatal Error 110 − Link terminated

A relocatable record contains a reference to a non-existent external.

Illegal relocatable (EMA)
Last module relocated: <*module name***>**
Fatal Error 111 − Link terminated

A relocatable record contains an illegal reference to EMA.

Illegal relocatable (MR)
Last module relocated: <*module name***>**
Fatal Error 112 − Link terminated

A relocatable record contains a reference to a non-existent relocatable space. See the *Macro/1000 Reference Manual*, part number 92059-90001, for information on relocatable space.

Illegal relocatable (RPL)
Last module relocated: <*module name***>**
Fatal Error 113 – Link terminated

A relocatable record contains an RPL longer than one word.

Warning 114: Record ignored

LINK has encountered a relocatable it recognizes but does not use.

Error 115 – Not used

Out of table space
Last module relocated: <*module name***>**
Fatal Error 116 – Link terminated

Size up LINK as large as possible (preferably 30 pages). It needs more room for an internal table. Otherwise, reduce the size of indexed libraries.

Allocate type mismatch
Last module relocated: <*module name***>**
Last reference: <*symbol name***>**
Fatal Error 117 – Link terminated

The specified symbol has been assigned two different data area allocation types; for example, EMA in one program unit and named common in another. Types must match.

Symbol table overflow – reduce number of symbols
Last module relocated: <*module name***>**
Fatal Error 118 – Link terminated

LINK has run out of space in which to put all the symbols in your program. There is a maximum of 64K symbols allowed.

Illegal EMA record combination
Last module relocated: <*module name***>**
Fatal Error 119 – Link terminated

Two relocatable records have given conflicting EMA directives. Usually this occurs when some of the relocatables were generated by an (old) language processor not supported on RTE-A. Recompile the source, generating all new relocatables.

Library routine needs EMA
Fatal Error 120 – Link terminated

The EM command was not specified and the main program did not declare any EMA arrays, but a library routine uses EMA. The program should be linked again, this time specifying the EM (or VM) command.

Relocation into system common illegal
Last module relocated: <*module name***> Address: <***address of violation***>**
Fatal Error 121 – Link terminated

An attempt was made to relocate code or data into the system common area. LINK cannot do this. Your source code must be corrected, recompiled, and linked again.

Illegal system reference
Last module relocated: <*module name***> Address: <***address of violation***>**
Fatal Error 122 – Link terminated

An attempt was made to reference a system entry point with a one-word instruction. Because a system entry point cannot be mapped into a user program, this cannot work. A two-word instruction (for example, XLA) must be used.

Legal:	XLA $CON	DEF $CON
Illegal:	LDA $CON	JSB $CON

Unimplemented relocatable record type
Last module relocated: <*module name***>**
Fatal Error 124 – Link terminated

Future versions of LINK may accept these relocatables, but currently they are not processed by LINK.

Ran out of base page links
Last module relocated: <*module name***>**
Fatal Error 125 – Link terminated

This problem can be caused by either relocation of excessive amounts of data on base page, or by the creation of too many base page links by LINK.

There are three things to try when solving this problem. First, try the RO command in your LINK command file. This does not require recompilation; LINK attempts to reduce the number of base page links it generates. When using RO, it is best to relocate the large modules first to reduce the amount of wasted memory.

Second, you can try reorganizing your source code into smaller modules (non-CDS only). This type of structure tends to generate fewer base page links, and also allows the RO command to proceed more efficiently.

Third, if you are relocating data on the base page, consider moving all or as much of it as possible off of the base page.

Error 126 – Not used

Warning 127: There are undefined symbols

Your program has asked for symbols (for example, subroutines) which you have not relocated with the program.

You must relocate the appropriate modules or search the appropriate libraries to find these modules.

More common declared than system common available
Last module relocated: <*module name***>**
Fatal Error 128 – Link terminated

You have specified to LINK that this program uses system common. However, there is less system common than this program needs.

Overlay base page entry
Last module relocated: <*module name***>**
Fatal Error 129 − Link terminated

Relocation onto base page was attempted in an overlay. All relocation onto base page must be done in the main.

Data segment is too large
Last module relocated: <*module name***>**
Fatal Error 130 − Link terminated

CDS programs: Your program needs more logical address space in the data segment than is available. Generated data, non-CDS routines, heap, stack, and MSEG total to more than 32 pages. You must reduce the size of one of these quantities. This can be done by: converting non-CDS routines to CDS, moving static data to EMA, or reducing the size of MSEG, heap, or the stack.

Non-CDS programs: Your program needs more logical address space than is available. Generated code of (main + longest overlay + MSEG + system common specified) is greater than 32 pages. You must reduce the size of one of these quantities. This can be done by: moving code from one overlay to another, moving arrays to EMA, using a smaller MSEG, converting a non-overlay program to an overlay program or removing system common. Often the best solution is to convert your non-CDS program to CDS.

This error may occur if you use LINK with an unsupported compiler, such as FTN4 or ALGOL. In this case, the error occurs because the module being relocated has a length value (word 7 of the NAM record) of 177777B, which indicates that the module length was unknown when the source file was compiled. LINK does not support this type of NAM record.

Warning 131: Snap file must be type 3

The snapshot file must be type 3.

Too many system libraries
Last module relocated: <*module name***>**
Fatal Error 132 − Link terminated

Too many system libraries have been generated into your system. The maximum number allowed is 64.

This program has no main
Last module relocated: <*module name***>**
Fatal Error 133 − Link terminated

You must relocate a main with every program.

Warning 134: Program file not type 6

The given program name is not a program file: therefore LINK does not purge it. You must give LINK a different file name in which to place the program.

System has too many BP links
Last module relocated: <*module name***>**
Fatal Error 135 – Link terminated

Your snapshot file is corrupt. You must get a new copy and link again.

Bad labeled common links in snap file
Last module relocated: <*module name***>**
Fatal Error 136 – Link terminated

Your snapshot file is corrupt. You must get a new copy of your snapshot file and link again.

Error 137 – Not used

Corrupt snap file
Last module relocated: <*module name***>**
Fatal Error 138 – Link terminated

Your snapshot file is corrupt. You must get a new copy of your snapshot file and link again.

Warning 139: Duplicate entry point: <*entry point name***>**

Relocation of modules containing identical entry points was attempted. Either the same module was relocated twice, or two modules have identical entry points. Either eliminate the redundant module or change the entry point names.

Warning 140: Conflict with system labeled common: <*entry point name***>**

A module that contains an entry point with the same name as an entry point in system labeled common has been relocated. In this case, portions of system labeled common are inaccessible to the program, even if it was linked using the LC command.

Warning 141: RPL value replaced: <*RPL name***>**

This is a special case of duplicate entry points. A module that contains an RPL or entry point with the same name as a previously defined RPL has been relocated.

This warning usually occurs when a machine instruction, that is defined as an RPL in the snapshot file, is replaced by a software routine with the same name.

The *Macro/1000 Reference Manual*, part number 92059-90001, contains a description of RPLs.

Error 142 – Not used

Error 143 – Not used

Error 144 – Not used

Error 145 – Not used

Warning 146: Program name already exists

The OU command was used to specify the program file or the DP runstring command was used, which prevents automatic purging of the program file.

Error loading overlay LINKn
Last module relocated: <*module name***>**
Fatal Error 147 − Link terminated

LINK is unable to run because it cannot draw the given overlay into memory. LINK is probably corrupt, or was installed with an incorrect version of SEGLD.

Warning 148: Reducing MSEG from X to Y pages to allow program to fit.

The size of your program (data segment if a CDS program is being loaded) plus the specified MSEG size plus one for the spillover page is too large. However, by reducing the MSEG size it is possible to load your program. The MSEG size is never reduced to less than one page. If your program can run with the reported MSEG size than no action is necessary. If your program requires more than the reported MSEG size then you must reduce the size of your program and reload it.

Warning 149: System labeled common matched an allocated symbol

A module which contains named common was relocated. The named common matched system labeled common and the system labeled common was used. There is no problem if this was the intent. To eliminate the warnings, use $ALIAS /A1/, noallocate, where A1 is the named system common label.

The following example shows the correct approach for accessing system common:

```
$ALIAS /A1/,noallocate
      Program main
      implicit integer (a-z)
      common /A1/v1,v2,v3
      .
      .
      .
      end
```

and assume the module relocated in system common looks like:

```
      NAM DATA, 30
      ENT A1

A1    BSS 3
      .
      .
      .
      END
```

In this case /A1/ in your program matches with A1 in system common and no warning is generated.

Warning 150: File not indexed, search proceeding on <*filename*>

This file was specified as library. but it is not indexed. LINK is proceeding to search it anyway. Searches of indexed and unindexed libraries do NOT always produce the same results.

Searches are order dependent. and backward references are not picked up in unindexed library searches.

The indexing utility is called LINDX, and is discussed in this manual.

Attempted to assign a label to a non-code space address
Last Module relocated: <*module name*>
Fatal Error 151 – Link terminated

The given module has attempted to generate a code label to a location in non-code space. This can be caused by passing a procedure as a parameter, where the passing procedure is in code, and the passed procedure is in data space.

The solution is to move the passed procedure to code space or move the passing procedure to data space.

Unrecognized label type
Last module relocated: <*module name*>
Fatal Error 152 – Link terminated

The named module has attempted to generate a label of undefined type. From Macro/1000 this can be done by an incorrect LABEL or PCAL statement.

Ran out of STT entries
Last module relocated: <*module name*>
Fatal Error 153 – Link terminated

Each code segment can have a maximum of 255 entry points in the STT (Segment Transfer Table).

Use the NS command in the LINK command file immediately before the given module to solve this problem. This causes LINK to generate a new segment, with space for 255 new STT entries.

Unimplemented PCAL type
Last module relocated: <*module name*>
Fatal Error 154 – Link terminated

A non-existent PCAL type was specified. In Macro/1000, this would happen by giving an improper PCAL.

Correct your source code, recompile, and link again.

Unsupported PCAL to an RPL
Last module relocated: <*module name*>
Fatal Error 155 – Link terminated

A PCAL to an RPL specified an illegal call sequence. In Macro/1000, this could happen by giving an improper PCAL. The only legal calling sequences in combination with RPL's are unspecified, .ENTR, .ENTN, and PCAL (that is. 0. 1. 2. and 3. respectively).

Correct your source code, recompile, and link again.

Error 156 – Not Used

PCALS only legal from code
Last module relocated: <*module name***>**
Fatal Error 157 – Link terminated

A PCAL has been specified in the data segment. This can happen from Macro/1000 by forgetting to use the RELOC code pseudo op.

Correct source code, recompile, and link again.

Cannot mix CDS code and overlays
Last module relocated: <*module name***>**
Fatal Error 158 – Link terminated

An overlay (type 5 program) was specified in a CDS program. Overlays are not allowed in CDS programs.

Warning 159: Not an RTE-A program file

The file specified for relinking is not a valid program file. This error may occur if you attempted to relink a program that was linked originally with a previous revision of LINK.

Error 161 – Not Used

Attempt to relocate a JSB in code space
Last module relocated: <*module name***> Address : <***location***>**
Fatal Error 162 – Link terminated

This can happen in Macro/1000 by forgetting to use the RELOC DATA pseudo operation before generating data. In this case, correct your source, recompile, and link again.

Another case in which this happens is when the language processor has used a JSB to microcode, but a software equivalent has been relocated with the program. In this case, you must remove the software equivalent and reinstall.

Module will not fit in one code block
Last module relocated: <*module name***>**
Fatal Error 163 – Link terminated

This module is so large it will not fit into one code block.

You can solve this by either using the NS command to make your code blocks larger or reducing the size of your module.

Error 164 – Not used

This system not set up for CDS programs
Last module relocated: <*module name***>**
Fatal Error 165 – Link terminated

You have attempted to link a CDS program on a system that does not have CDS capabilities.

Check the snapshot file to ensure that it was generated for a CDS system.

Illegal reference to code
Last module relocated: <*module name***> Address : <***location***>**
Fatal Error 166 – Link terminated

A reference to code was specified which cannot be executed.
Macro/1000 example (program fragment):

```
        reloc code
foo1    dec 8
          .
          .
          .
foo     lda foo1
        sta foo1 <------ illegal statement
```

In this example, the program attempts to use code space as data space, which is not possible.

The source code must be corrected, recompiled, and linked again.

Warning 167: Too late for DB command

The DB command must be given before any overlays are relocated.

Warning 168: Unable to create debug file

The DE command was specified, but the debug file associated with your program could not be created. You are not able to use Symbolic Debug with your program because of this.

Warning 169: Force loading

You have specified the FO command in your command file. Any references to symbols that have been forced have unpredictable results.

Warning 170: Too late for LK command

The LK command cannot be used with any relocation commands.

Warning 171: Too late for LK command

The LK command cannot be used with any relocation commands.

Not enough BREAK records in relocatable
Last module relocated: <*module name***>**
Fatal Error 172 – Link terminated

The language processor has not supplied enough BREAK records in the named module. In Macro/1000, the programmer has responsibility for placing BREAK records in his code. Higher level languages take care of this for the user.

The source code must be corrected by placing BREAK records no more than 512 words apart. Recompile and link again.

Warning 173: EMA size reduced to 1022 pages

Your program has specified more than 1022 pages of EMA, which is the maximum for Normal and Large model programs. If you need more than 1022 pages, you must use VMA or the Extended model of EMA. See the VM and WS commands for VMA usage.

Error 174 – Not Used

Illegal CDS and Non-CDS combination
Last module relocated: <*module name***>**
Fatal Error 175 – Link terminated

You have used commands or relocatables which imply that your program is both CDS and non-CDS. For example, you may have specified the PS command and then relocated a CDS module, or you may have relocated a non-CDS main with CDS subroutines.

Too many code segments. Maximum is 128
Last module relocated: <*module name***>**
Fatal Error 176 – Link terminated

The number of code segments sometimes can be reduced by rearranging the order in which modules are relocated, so that there is less space wasted in each segment, or by reducing the number of NS commands used.

Too many TR files. Maximum is 128.
Last module relocated: <*module name***>**
Fatal Error 177 – Link terminated

This error can be caused by recursion (the command file transferred to itself).

Program name not usable.
Last module relocated: <*module name***>**
Fatal Error 178 – Link terminated

A program name may not be usable for any of the following reasons:

- The program file exists and the OU command was used
- The program name is illegal
- The program file exists and is in use

Warning 179: Illegal relocatable record – record ignored

LINK has encountered a relocatable record that has an illegal format. The record is ignored.

Warning 180: Illegal backing store file specified

The backing store file or cartridge specified is invalid. The default backing store file is used.

Warning 181: Illegal snap file specified – trying defaults

The snapshot file specified was not found. The snapshot file defaults are tried.

Unsupported PCAL to non-CDS code
Last module relocated: <*module name***>**
Fatal Error 182 – Link terminated

A PCAL to non-CDS code used an illegal call sequence. In Macro, this occurs when an improper PCAL is specified. The only legal calling sequences in combination with non-CDS code are unspecified, .ENTR, .ENTN, and PCAL (that is. 0. 1. 2. and 3. respectively).

Correct your source code, recompile, and link again.

Warning 183: EMA area specified does not exists: <*EMA area***>**

The EMA area specified in the last OR command has already been ordered. The current OR command is ignored.

Warning 184: EMA area specified does not exist: <*EMA area***>**

The EMA area specified in an OR command was not found when the program was relocated. Make sure all EMA area names are correct.

Warning 185: No system libraries searched

Either there are no system libraries, or the program being relocated is non-CDS and the non-CDS system libraries were not specified when the system was generated.

Warning 186: Conflict between named common and library routine

A library routine is not relocated because a named common of the same name already exists. This does not necessarily indicate an error. This warning is useful in case a user has declared a common block, giving it a name that is also a system library name. LINK does not relocate the system routine. When the routine is called, the data in the common block is executed, usually resulting in an MP or UI.

Warning 187: Conflict between entry point and named common

An entry point name conflicts with a named common; that is, the entry point name is the same as the label of the named common. LINK uses the entry point name and ignores the named common. While not necessarily an error, this warning is useful in case a user relocates a subroutine with the same name as the named common. Because the named common does not exist, the subroutine is overwritten when the user intends to write to his common block.

Warning 188: <*command***> command is now obsolete**

The given command is no longer supported (LINK simply ignores the command).

Warning 189: Insufficient capability to execute <*command***> command**

When Security/1000 is turned on, this warning may be issued indicating that the user's capability is not high enough to issue the command. LINK ignores the command.

Illegal reference to a non-data space address
Last module relocated: <*module name*> Address: <*addr*>
Fatal Error 190 – Link terminated

An attempt was made to relocate memory reference instructions which accessed neither code nor data space. <*module name*> is the name of the module where it happened and <*addr*> is the relative module address of the instruction. For example, consider the following macro fragment:

```
          nam testeme
          ent testema
 emalabel alloc ema,100  ;allocate 100 words of EMA.

 testema  nop
          lda            ;this instruction generates a 190 error
                         ;becuase EMA is being referenced.
           .
           .
           .
          end
```

Illegal DEF to a non-data, non-code space address
Last module relocated: <*module name*> address: <*addr*>
Fatal Error 191 – Link terminated

An attempt to relocate a DEF to an illegal memory type was made, <*module name*> is the module containing the bogus DEF and <*addr*> is the module relative address containing the offending DEF.

Warning 192: "de" found after modules already relocated

A "DE" command, which causes Symbolic Debug information to be produced, was found after modules had already been relocated. Thus, debug information will be missing for the modules relocated before the "DE" command was found.

Code space JMP instruction to a non-code space address
Last module relocated: <*module name*> address: <*addr*>
Fatal Error 193 – Link terminated

A JMP was attempted to a non-code space address. Code space JMP's may only go to other code space addresses. <*module name*> is the name of the module where the JMP instruction was seen, and <*addr*> is the module relative address of the offending JMP.

Byte pointer to a non-data, non-code space address
Last module relocated: <*module name*> address: <*addr*>
Fatal Error 194 – Link terminated

A byte pointer referenced a non-code, non-data address. All memory references using a byte pointer must be either to code or data space. <*module name*> is the name of the module containing the bad reference and <*addr*> is the module relative address of the offending instruction.

Warning 195: No CDS system libraries found in snap file

The snap file used to load a CDS program does not specify any CDS system libraries to be searched. Only non-CDS system libraries will be searched, possibly causing the data partition to become too large. This warning may indicate a problem with your system generation answer file.

Unknown relocation space in XDBL record
Last module relocated: <*module name***> address: <***addr***>**
Fatal Error 196 – Link terminated

An attempt was made to relocate into an illegal memory space. <*module name*> is the name of the module containing the bad code and <*addr*> is the module relative address of the offending instruction.

Unsupported EMA address in symbolic debug record
Last module relocated: <*module name***>**
Fatal Error 197 – Link terminated

There is an illegal reference to EMA in a debug record, probably due to a problem with the debugger.

Double load immediate supported only in CDS code
Last module relocated: <*module name***> address: <***addr***>**
Fatal Error 198 – Link terminated

A double load immediate instruction was placed in non-CDS code. These instructions are only legal from CDS code.

Ran out of space in VMA, relink LINK and size up VMA
Last module relocated: <*module name***>**
Fatal Error 199 – Link terminated

The combined size of the internal data structures of LINK and the size of the relocatables have exceeded the current VMA size. Re-link LINK and size up VMA.

Symbol relative code relocation not allowed
Last module relocated: <*module name***>**
Fatal Error 200 – Link terminated

An attempt was made to relocate a module relative to an external symbol. This is illegal and is probably due to the language processor.

Module is too big
Last module relocated: <*module name***>**
Fatal Error 201 – Link terminated

CDS module <*module name*> is too big to be relocated. It must be reduced in size.

Module contains too may entry points
Last module relocated: <*module name***>**
Fatal Error 202 – Link terminated

CDS module contains more than 255 entry points; a segment can have no more than 255 entry points. Reduce the number of entry points in the module.

Illegal relocatable; CDS module size conflicts with header
Last module relocated: <*module name*>
Fatal Error 203 – Link terminated

The size of the module given in the NAM record does not match the size actually seen; this is a problem with the compiler.

Illegal literal code reference outside of module
Last module relocated: <*module name*>
Fatal Error 204 – Link terminated

A literal code reference outside of a CDS module was made. For example, assume you have the following macro fragment:

```
        Nam tmac
        cds on
        ent tmac

        reloc code
tmac    nop
        lda =Ltmac-1 ;this is illegal since an address is being
         .           ;referenced before the start of the code
         .           ;for this module
```

Illegal Pcal to external+offset, no STT entry
Last module relocated: <*module name*>
Fatal Error 205 – Link terminated

A Pcal to an external+offset was made where no entry point was declared for the offset, and the destination module is in another segment. For example, consider the following Macro main and subroutine:

```
          name main
          cds on
          ent main

          reloc code
 main     pcal sub1+3,0,0
           .
           .
          end main

          nam sub1   ;this subroutine sits in a different segment
          cds on
          maclib '$cdslb::libraries'
          ent sub1

          reloc code
 sub1     entry
          cax
          jmp finis
 dest     entry
           .
           .
 finis    exit
          end
```

The Pcal in the main would generate error 205. To fix it, you would declare 'dest' as an entry point. Also note that the 'entry' macro must be used at each entry point.

Error 206 – Initial stack frame does not fit by <*size*> words

The initial CDS stack frame does not fit in the program's data segment, either because the stack frame is too large or because the data segment is full. LINK must be able to fit at least the main program routine's stack frame into the data segment to create a legal program. The size reported is the number of words by which the frame overflows the data segment. Loading will eventually abort with fatal error 130 (data segment is too large). You may be able to correct this error by decreasing the stack requirements of the program.

Error 207 – Program/data segment is too large by <*size*> words

The non-CDS program, including the largest segment, or the CDS data segment is too large for a 32-page partition, minus the minimum 2 pages needed for the MSEG, if used. The size reported indicates the number of words by which the program or data segment is too large. This value may not be the full amount by which the address space is overflowed if LINK has previously truncated modules and other relocated objects to fit within the maximum address of 77777b. Other error messages and the partial load map will provide more information on what caused the overflow. Loading will eventually abort with fatal error 130 (program or data segment is too large).

Error 208 – Segment <*name*> does not fit with main in 32K by <*size*> words

The named non-CDS segment and the main program cannot fit together in a 32-page partition by the displayed number of words. Error 207 will also be generated, possibly indicating an even larger number of words due to other pages needed, such as for the MSEG area.

B

Using LINK with FMGR

This appendix describes how to run LINK from FMGR. Before running FMGR, set the working directory to zero so that the file searches follow the disk mounting order as shown in the FMGR cartridge list.

To set the working directory to zero, enter the following CI command:

```
CI> wd 0
```

Running LINK Interactively

To run LINK interactively from the keyboard, enter the following:

```
FMGR : ru,link
link Rev.6.1      Use ? for help
link:
```

The LINK commands can be entered one at a time after the LINK prompt. The order in which you enter the commands may make a difference in the linking process.

A sample LINK interactive session is shown in Chapter 2.

LINK Runstring

Files and runstring options can be included in the LINK runstring. Including the delimiters, 72 characters are allowed. In the runstring, LINK enforces the following file descriptor prefix conventions:

- % Relocatable file (%LKST::MC)
- $ Library file ($LKST::MC)
- # Command file (#LKST::MC)

Only one each of the following is allowed in the runstring:

' or "	List file ("LKST::MC or 'LKST::MC)
a	Output file (a can be any letter, FLKST)
^	Snap file (^LKST::MC)

In the following example, %JOB is relocated using $JOBLB and $JOBL2 as libraries. The program file is JOB1 and the list file is 'JOB. The commas shown are required.

```
FMGR : ru,link,%job $joblb job1::cr 'job $job12
```

Parameters after "link," can be in any order. Uppercase or lowercase letters are acceptable in the runstring. Blanks or commas can be used as delimiters for file descriptors. Colons must be used to delimit parameters as shown, including any that are omitted before the last parameter used.

The options that can be used in the LINK runstring are described in Chapter 2.

The following example illustrates the use of the LINK runstring. The runstring relocates the three modules of LKST from cartridge L1, listing to file 'LKST on cartridge DL. The EC option echoes the LINK commands to the list file. The SZ option sizes the program to 32 pages.

```
FMGR  :  ru,link,'lkst::dl,+ec,+sz:32,%lksta::l1,%lkstb::l1,%lkstc::l1
```

The following example relocates LCDE, overwriting the program file on T6, if it exists. The LC option accesses labeled system common. LINK uses library $LCDE for searches.

```
FMGR  :  ru,link,+lc,%lcde::gr,$lcde::gr,lcde::t6
```

LINK Command Files

To run LINK using command files, the command file names must be specified in the runstring. All runstring options are processed before the command files are read, regardless of their order.

```
FMGR : ru,link,#<command file>
```

Command file names must begin with the # sign. The following examples show the use of LINK command files.

The file #EMATS loads program EMATS, overlaying the EMATS file on cartridge DL, if it exists, by specifying the program file name as a parameter of the END command. File ^SNP.W is the snapshot file, and the library to search for external references is $VMALB. The VMA size is 1400 pages, and the working set size is 100 pages:

```
li $vmalb
ws 100
vm 1400
re %emats
sn ^snp.w
en mats::dl
```

The following command file loads the three overlays of LKST. The result is identical to that of the runstring given in the previous section. The equivalent LI, EC, SZ, and RE commands are included:

```
ll,'lkst::dl
ec
sz,32
re,%lksta::l1
re,%lkstb::l1
re,%lkstc::l1
en
```

The following command file is equivalent to the LCDE runstring given in the previous section, except that the command file does not overwrite the program file if it exists, because the name of the program file is given in an OU command. Other specifications are the same: labeled common is specified, the library is $LCDE on cartridge GR, and the program file is LCDE on cartridge T6:

```
lc
re %lcde::gr
ou,lcde::t6
se,$lcde::gr
en
```

The following command file uses the RO command. The only advantage to the RO command is that it reduces the number of base page links required. It is used here because without the RO command this program does not link due to base page overflow (fatal error 125). The command file defaults the program file descriptor, allowing LINK to construct it from the NAM record of the first main module with a transfer address:

```
ro
re,%ovbse
en
```

Using FMGR Transfer Files to Run LINK

During program development, you can use a transfer file to link and run the application program after each change to the program. By using FMGR global parameters, you can write one transfer file for all occasions.

When LINK finishes, it places information into the global parameter area and indicates if an error occurred. At completion, the global parameter area contains the following:

1P: characters 1 and 2 of the program name

2P: characters 3 and 4 of the program name

3P: characters 5 and 6 of the program name

4P:

0	if the link was successful
−1000	if the user aborted the link
nnn	if the link was unsuccessful, where *nnn* is the fatal error code as summarized in Appendix A

5P: unused (blanks)

Using the global parameters, you can link and run a program from a transfer file after determining if the link was successful. The following is a sample transfer file for linking and, if the link is successful, running the program:

```
:RU,LINK,1G
:IF,4P,EQ,0,1
:TR
:RU,10G
:TR
```

If the name of the transfer file is TRANS, the following command executes TRANS:

```
FMGR : tr,trans,#examp,examp
```

Index

L

M

N

O