File No. S360-31
GC28-2005-4

TSS

IBM Systems Reference Library

Version 8.1

IBM System/360 Time Sharing System

Linkage Editor

Tells how to use the Time Sharing System's linkage editor. This optional program can be used to join two or more related object modules into one object module, saving dynamic loader processing time. A linkage editor user can also, without having to reassemble or recompile his program, combine control sections (possibly saving external storage and reducing program execution time); rename entry points, control sections, and external references; and change control section attributes.

An introduction explains: how a TSS/360 object module is created, its structure, how control sections are named and attributes assigned, external definitions and references, the difference between explicit and implicit linkage, and the difference between static and dynamic linking.

The reader is shown how to invoke the linkage editor and how to use linkage editor control statements. Examples are provided for both conversational and non-conversational users.

Before using this publication, be familiar with the contents of:

IBM System/360 Time Sharing System: Concepts and Facilities, GC28-2003
IBM System/360 Time Sharing System: Command System User's Guide, GC28-2001

Fifth Edition (September 1971)

This is a major revision of and makes obsolete Form GC28-2005-3 and Technical Newsletter GN28-3117. This revision reflects recent changes in TSS/360 and corrects errors in the previous edition. Addition of the DXD and CXD instructions and the Q-type address constant to the TSS/360 assembler has made linkage possible between object modules produced by the assembler and those produced by the PL/I compiler. Restrictions for linkage editing PL/I compiler-produced modules have been removed. Figure 15, Format of PMD Listing, now reflects additions to this listing for DXD and CXD references. A table has been added, summarizing how to use the linkage editor. A glossary has been added. Significant changes will be indicated by a vertical line beside the changed text.

This edition is current with Version 8, Modification 1, of the IBM System/360 Time Sharing System, and remains in effect for all subsequent versions or modifications unless otherwise noted. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication, refer to the latest edition of IBM System/360 Time Sharing System: Addendum, GC28-2043, which may contain information pertinent to the topics covered in this edition. The Addendum also lists the editions of all TSS/360 publications that are applicable and current.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing System/360 Programming Publications, Department 643, Neighborhood Road, Kingston, New York 12401.

PURPOSE

This book tells how to use the linkage editor, an optional TSS/360 program called into a user's task when he issues the LNK command.

ADVANTAGES OF THE LINKAGE EDITOR

The linkage editor can be used to:

- Link two or more existing object modules (the output of the TSS/360 assembler, FORTRAN compiler, PL/I compiler, or a previous linkage editor process) into one module that requires less dynamic loader processing time.
- Combine control sections in one or more existing modules, reducing external storage requirements and paging activity during program execution.
- Rename control sections, entry points, and external references, and change control section attributes ... without having to reassemble or recompile.

TO USE THIS BOOK, YOU NEED...

A general understanding of TSS/360 concepts and terminology. This information is available in:

IBM System/360 Time Sharing System: Concepts and Facilities, GC28-2003.

A familiarity with the TSS/360 command system and, specifically, the DDEF command. This information is available in:

IBM System/360 Time Sharing System: Command System User's Guide, GC28-2001.

The reader is presumed to be familiar with at least one TSS/360 programming language.

HOW THIS BOOK IS ORGANIZED

Conceptual information is provided in "Section 1: Introduction"; it explains what an object module is, its structure, how control section names and attributes are assigned, how external definitions and references are generated, and relates linkage editor processing to dynamic loader processing. Experienced users may wish to skip Section 1.

Practical, how-to information is provided in Sections 2, 3, and 4. "Section 2: Format of Linkage Editor Control Statements" gives rules for entering statements at the terminal or on cards. Experienced users may skip Section 2. "Section 3: Linkage Editor Control Statements" tells what the statements do and how to specify them. "Section 4: How to Specify Linkage Editor Processing" describes the LNK command parameters and has LOGON-to-LOGOFF examples of using the linkage editor.

Reference information, including the PMD listing format, how errors are handled, a summary of the control statements, additional examples, and a glossary of terms, is provided in the Appendixes.

A QUICK, HOW-TO SUMMARY...

... is provided in Table 2, "Controlling the Linkage Editor", in Section 4.

CONTENTS

ILLUSTRATIONS

The linkage editor is an optional TSS/360 service program that can be used to further develop and edit programs that have been assembled or compiled. Execution of the linkage editor is started by the LNK command.

The primary linkage editor function is to put together the output of separate assemblies or compilations into one object program. Usually, this pre-execution linkage will reduce the time it takes to load and run the program. The linkage editor can be used to "edit" a program (that is, combine, rename, delete, and assign new attributes to control sections, rename and delete entry points, and rename external references) without having to reassemble, recompile, or execute the program. The linkage editor can also be used simply to move object modules from one library to another. These functions are summarized later in this section, under "Summary of Linkage Editor Functions."

The following paragraphs review the TSS/360 program preparation process and define the terms needed to understand linkage editor processing.

PREPARING A PROGRAM

A program is the collection of instructions and data that the user specifies for the solution of a problem. A program can exist in the system in source or object form. A source program exists in the symbolic form in which it was written by a user and consists of a series of statements coded in one of the programming languages available in TSS/360 (PL/I, FORTRAN IV, or assembler language). An object program exists in machine-language form; it can be loaded and executed by the system.

CREATING OBJECT MODULES

An object module (the terms "program module" and "object program module" are synonymous) can be created by language processor programs or the linkage editor.

A language processor creates an object module by converting source language statements to machine language. The input to the language processor, called the source data set, includes both source language statements to be included in an object module and statements intended only for the language processor (for example, the DC, CSECT, and END statements in assembler language).

The linkage editor generates an object module by "linking" object modules that already exist in various user and system libraries. The input to the linkage editor consists of previously assembled or compiled programs and a source data set containing only linkage editor control statements. The output of the linkage editor is a new object module (the input modules still exist to be retained or erased as the programmer desires).

Any new object module, whether created by a language processor or the linkage editor, is stored in the appropriate library. Figure 1 shows the data flow during object module creation for both language processing and linkage editing.

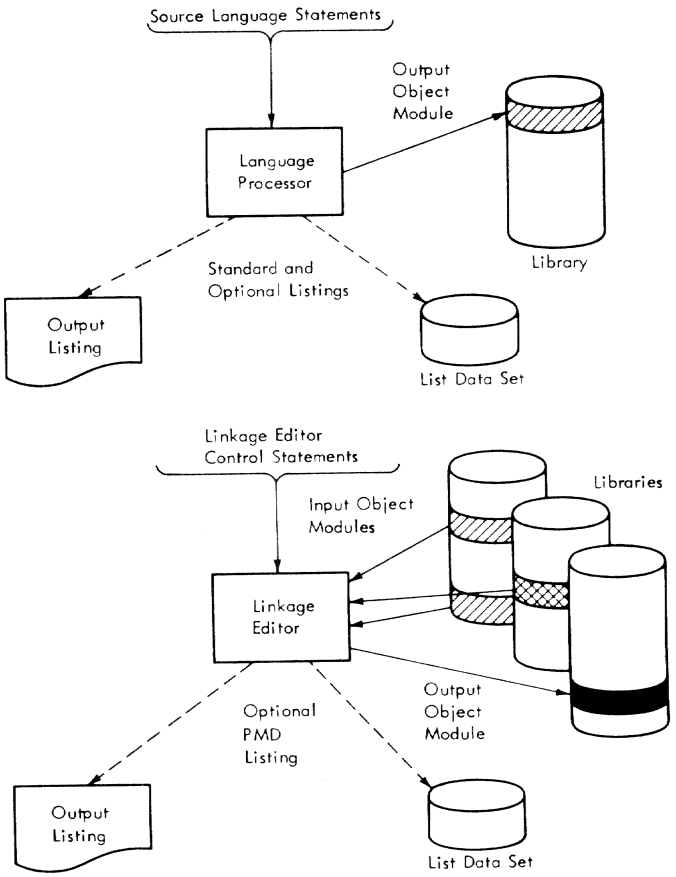


Figure 1. Creation of an Object Module

STRUCTURE OF AN OBJECT MODULE

A TSS/360 object module has three components (see Figure 2):

- Program module dictionary (PMD)
- Text
- Internal symbol dictionary (ISD)

The program module dictionary and text are always produced; the internal symbol dictionary is produced only if specified by the assembler or FORTRAN language user in defining the language processor or linkage editor run. The PL/I user cannot obtain an ISD.

Program Module Dictionary

The program module dictionary contains information the system needs to process the object module and consists of a header and a series of control section dictionaries:

- The header contains the name of the standard entry point to the module and other information common to the entire module.
- Each control section dictionary (CSD) describes its associated control section (described below). The control section dictionary contains information which describes the external definitions and references and address constants contained in the control section.

Text

The text portion of the module contains the instructions and constants generated by the assembler or compiler; it is the executable portion of the module. The text is organized by control sections, the basic unit of all TSS/360 programs.

Internal Symbol Dictionary

The internal symbol dictionary (ISD) is a list of all symbols that name control sections, instructions, and data areas within a module. The ISD enables the user to analyze the module using TSS/360 program control system (PCS) commands. A module must contain an ISD for the user to be able to take full advantage of PCS. The module produced by the assembler, the FORTRAN compiler, and the linkage editor contains an ISD, unless the user has specified otherwise; the module produced by the PL/I compiler in TSS/360 does not contain an ISD. The linkage editor produces a composite ISD, consisting of ISDs retained from input modules.

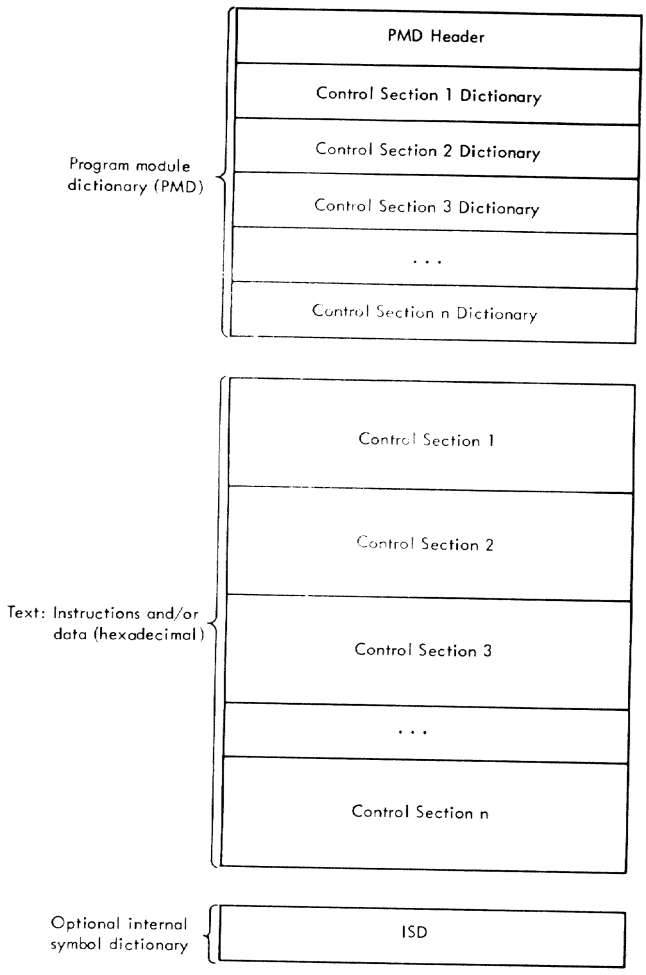


Figure 2. Format of an Object Module

CONTROL SECTIONS

A control section is a block of coding whose virtual storage location assignments can be adjusted independently of other coding during linkage editor or dynamic loader processing. At least one page (4096 bytes) of external storage is assigned to each control section; however, a control section may require more than one page.

When an object module is loaded, virtual storage is allocated for all its control sections. The space allocated to each control section begins on a page boundary unless the user, in his LOGON command, has specified that the dynamic loader, where possible, is to put more than one control section on a page. The contents of each control section occupy contiguous virtual storage addresses; however, the individual control sections may be scattered throughout virtual storage. When object modules are placed in main storage for execution, they are brought in page-by-page. The contents of each page occupy contiguous main

storage locations; however, individual pages may be scattered throughout main storage. Only the pages required for execution during a user's time slice are brought into main storage for that period.

The object module code contains virtual storage addresses. During execution, these virtual addresses are translated into actual main storage addresses, based on relationships established between each page's virtual storage base address and its main storage base address at the time it is placed in main storage. If a page that is executing had been previously paged out, and then relocated in main storage, it may have been assigned a new location in main storage. However, because a new relationship has been established between the page's virtual storage base address and its new main storage base address, the system can execute the page in its new main storage location.

FORTRAN and PL/I users do not need to concern themselves directly with organizing their source programs into control sections because the FORTRAN and PL/I compilers always generate text in a standard way.

Assembler users can control the organization of text into control sections; they also control the naming and the assignment of attributes to control sections. Names and attributes are normally defined in the source code; however, by using the linkage editor, control sections can be renamed and their attributes can be changed without changing and reassembling the source code.

Control Section Names

The FORTRAN compiler assigns control section names to each FORTRAN object module in this way: The characters #C, for a CSECT, and #P, for a PSECT (both described below), are appended to the module name if there are six or fewer characters in the name; otherwise, to the first six characters of the module name. For example, a FORTRAN object module named SORTFILE may have a CSECT with the name SORTFI#C and a PSECT with the name SORTFI#P.

The PL/I compiler assigns the PROCEDURE name to the control section containing the text (executable code). Any name longer than seven characters is truncated, using the first four characters and the last three characters. It also creates additional control sections for special purposes; the names these control sections are assigned by the compiler is based on the PROCEDURE name. The convention used to name these control sections and additional information on linkage editing PL/I modules can be found in Section 6 of the PL/I Programmer's Guide, GC28-2049.

Assembler users name control sections to assist the assembler in assigning consecutive storage addresses to them during assembly. The consecutive assignment of storage addresses within a control section, once begun, is continued throughout an assembly. A control section may be interspersed with other control sections in a source program. Each group of language statements that is part of a particular control section must be preceded by a control section statement containing the control section name. The first such statement in a program identifies the beginning of the control section; other statements with the same name indicate resumption of the control section.

There can be only one unnamed control section (other than common) in a source program. As with named control sections, the unnamed control section is provided with a location counter and its contents are assigned consecutive storage addresses throughout assembly.

In addition to controlling assembler address assignments, the identification of control sections serves three other purposes:

- It enables symbolic linkages based on control section names to be made between control sections.
- It allows the dynamic loader to allocate noncontiguous storage for different control sections of an object module during loading.
- It allows control sections to be rejected dynamically during loading or linkage editing.

The ways in which control sections can be named and assigned attributes in source code by assembler users are summarized in Table 1.

Control Section Attributes

The attributes of control sections are assigned by the assembler on the basis of specified (or defaulted) operand values. These attributes, which describe the control section to the dynamic loader and the linkage editor, are inserted in the control section dictionary for the control section. The optional attributes, named in the operand field of Table 1, may be changed by the user during linkage editing; the PSECT and COM attributes, assigned by the assembler to those control sections, may not be changed.

The attributes assigned to control sections produced by the FORTRAN and PL/I compilers are determined by those processors

Table 1. Assembler Statements Used to Name and Describe Attributes of Control Sections

Assembler Statement			Use	Remarks
Name Field	Operation Field	Operand Field		
Symbol or blank	START	Self-defining value or blank	Identifies first (or only) control section of object module; a self-defining value in operand field may specify initial virtual storage location counter value for first control section.	Assembler assigns standard control attributes: fixed-length, read/write.
Symbol or blank	CSECT	READONLY PUBLIC PRVLGD VARIABLE SYSTEM blank (none of above)	Identifies control section containing executable instructions.	Assembler assigns attributes based on specification in operand field; standard control attributes if operand field is blank.
Symbol	PSECT	READONLY PUBLIC PRVLGD VARIABLE SYSTEM blank (none of above)	Identifies control section containing address constants and save area, and/or working area.	Assembler assigns PSECT attribute to control section; also standard control attributes unless other attributes are specified in operand field.
Symbol or blank	COM	READONLY PUBLIC,PRVLGD VARIABLE SYSTEM blank (none of above)	Identifies control section serving as common storage area.	Assembler assigns COM attribute to control section; also standard control attributes unless other attributes are specified in operand field.
Symbol	DSECT	blank	Identifies control section describing layout of storage area; does not actually reserve storage; storage area reserved by another statement.	
Symbol	DXD	blank	Identifies external dummy section.	Allows assembler modules to interface with PL/I modules.

rather than by the programmer (with the exception that FORTRAN programmers in the FTN command may specify the PUBLIC attribute be assigned to CSECTs produced by the compiler and may create COMMON control sections as a result of the COMMON FORTRAN language statement). The FORTRAN compiler produces a module containing a prototype control section and one or more standard control sections with the fixed-length, read/write attributes; if the programmer specifies PUBLIC, the control sections will have that attribute as well. The PL/I compiler produces standard control sections with fixed-length, read/write attributes. It does not generate a prototype control section.

FIXED-LENGTH OR VARIABLE-LENGTH: A fixed-length control section is allocated an integral number of pages of virtual storage by the dynamic loader. The number is the minimum that will contain the bounds of the control section. For example: If, during an assembly of CSECT A, its location counter reached a maximum of 11,000, the dynamic loader will allocate three pages of virtual storage.

A variable-length control section is allocated a number of pages in addition to the minimum required. This additional number is an installation parameter. Fixed-length and variable-length are alternatives -- control sections not declared

variable-length are, by default, fixed-length.

PUBLIC: The control section contains instructions and/or data that can be shared by other tasks if (1) the owner of the library containing the object module that includes this control section issues an appropriate PERMIT command authorizing its sharing, (2) each sharer issues a SHARE command updating the system catalog so that the system can locate that library by each sharer's name, and (3) the owner and sharers define the library by a DDEF command in their respective tasks prior to attempting to use the object module involved. The first task actually to reference a module will cause the loader to allocate each control section with the public attribute in that module to public storage; references from other tasks, then, will be tied to the copy already allocated to public storage. The use of public storage techniques is the method by which TSS/360 implements reenterable programming. The dynamic loader attempts to place all public control sections of a given module within the same segment. Public control sections must not contain any relocatable address constants (adcons).

PRIVILEGED: Privileged control sections are allocated storage with a protection key that allows reference to the control section only by a privileged system service routine. An attempt by a nonprivileged program to write or read a privileged control section will result in a storage protection error. Entry points in privileged control sections must have names that begin with CHB or CZ. The normal user may not declare privileged control sections; the dynamic loader will remove such an attribute from control sections in any loaded module except those that come from SYSLIB.

READONLY: A control section whose contents cannot be modified during access by other modules or control sections possesses the READONLY attribute. The CSECTs to be associated with shared object modules will normally be designated READONLY as well as PUBLIC. If READONLY is not specified as an operand, a control section will have the standard read/write attribute and may be both read and written (when writing occurs, the new data overlays and thus destroys previous data).

PROTOTYPE: Prototype control sections (PSECTs) are normally used for the modifiable storage associated with read-only public control sections. This modifiable storage consists of save areas, working storage, and variable program data. The normal reenterable module consists of a nonmodifiable public control section, which contains the executable instructions, and a prototype control section, which contains all the adcons and other modifiable data.

COMMON: In FORTRAN and assembler language, the programmer can create control sections that reserve virtual storage areas and that may be common to two or more separate programs. Referred to as common, these control sections are produced in the object modules by the language processors. These common areas are used either for communication -- facilitating the sharing or transferring of data -- among different parts of a program, or for reserving virtual storage areas for control sections that may be provided by other modules. These common areas are either named or unnamed (blank).

During linkage editor processing, the blank common areas are examined to determine the size of the largest area encountered. If more than one blank common area is found in the input, the linkage editor will reserve, in its output, an area equivalent to the largest of those blank common areas, and will ignore the rest. All references in the output module to a blank common area refer to the one area retained.

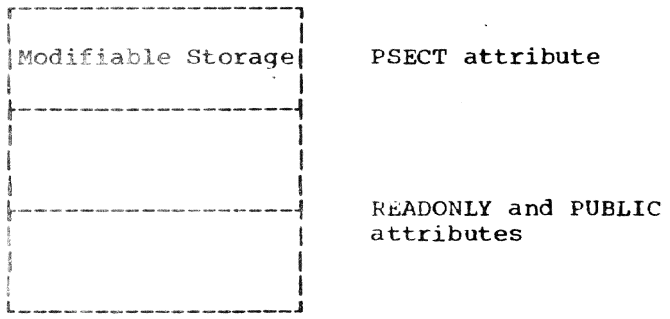
Generally, blank common control sections do not contain data constants; in any case, whether data constants are present or not, the storage content of the first (not the largest) blank common control section encountered is retained in the linkage editor output.

Named common sections are treated in the same manner as noncommon sections, except that common control section names are not placed in the partitioned organization directory (POD) of a library, and hence may not be loaded by external reference to their name alone. Reference to the module name or some other external symbol in the module containing the desired common control section is necessary to ensure that an external reference to a non-locally-declared common will be resolved.

SYSTEM: A system attribute can also be specified by system programmers, if the control section is to be part of a system (rather than a user's) object module. This attribute is never specified for application programs. Control sections marked SYSTEM are maintained by the dynamic loader in a manner that prohibits user reference to them, except through SYS symbols. The loader will not allow the user to declare system control sections; the system attribute will be unconditionally erased from control sections in any module except those loaded from the system library (SYSLIB). Furthermore, only system control sections may contain SYS symbols; therefore, the user is prevented from defining symbols beginning with SYS. This is the only

symbol-naming restriction imposed upon the user. SYS symbols are used to label entry points to nonprivileged system routines to which the user may transfer control by a standard CALL linkage. (Examples of such routines are GETBUF and FREEBUF of the access methods.)

Shared Object Modules: In TSS/360, shared object modules normally contain one or more control sections with read-only and public attributes and a prototype (PSECT) control section for the modifiable storage required by the read-only portions. (Modules produced by the PL/I compiler, however, do not contain a PSECT, and, since address constants are present in control sections, may not effectively be shared by different users.) A simplified format of a shared-program object module is illustrated below:



COMBINING OBJECT MODULES

A source program and its corresponding object module may represent all or part of the actual program required to solve a user's problem. A user may design his program in parts and separately assemble or compile each part. As a result of each assembly or compilation, he will produce an object module. If he has supplied the proper linkages, he can then use the linkage editor to permanently combine the various modules prior to program execution.

STANDARD ENTRY POINT OF A LINKAGE-EDITED MODULE

As for any object module, the module that results from linkage editing will contain a standard entry point, the point at which a program will begin execution if invoked by its name. The standard entry point names of all input modules will be saved if the original effect of calling each such name can be preserved.

Assembler and PL/I Programs

The standard entry point of the new linked module will be assigned to a location in the first module that is linkage edited. In output produced from assembler language input modules, this location will be the operand of the assembler END statement, if the operand is nonblank. If the operand is blank, the standard entry point will be assigned to the first byte in the control section.

FORTRAN Programs

The standard entry point is the existing entry point of the main program, which is its first executable instruction.

CONTROL SECTION ATTRIBUTES OF A LINKAGE-EDITED MODULE

The TRAITS control statement, which is used to assign attributes to a linkage edited control section, is described in Section 3. If no TRAITS control statement is specified for a control section, the linkage-edited control sections retain the same attributes as on entry to a linkage editor run.

If control sections with different attributes are specified to be combined by use of the COMBINE control statement (described in Section 3), the statement will not be processed, and a diagnostic message will be issued. Identical attributes may be obtained by preceding the COMBINE control statement with a TRAITS control statement, which would assign new attributes to one of the non-identical control statements.

LINKING SYMBOLS

A symbol may be referred to (used as an operand in a statement) in one control section and be defined (that is, used as the name of a statement) in another section, to establish linkages between parts of a program.

External Definitions

An external definition (the term "external symbol definition" is synonymous) is a symbol within a module that may be referred to by name from other modules, or from control sections in the same module. Referability is effected when the language processor creates external definition (DEF) tables within the control section dictionary (CSD) that name the symbols that can be referred to from outside the control section. DEFs are created by:

1. Creating a module, which causes a DEF entry to be created for the module name. This DEF entry specifies the standard entry point of the module.

2. Declaring a control section (including common control sections), which causes a DEF entry to be created with the same name as the control section. (If a blank common control section is declared, the name consists of blanks. If an unnamed control section is declared, its name consists of zeros.)

3. Declaring an ENTRY statement for a symbol, which causes a DEF entry to be created for the name appearing in the operand field of the ENTRY statement.

The three types of DEFs defined for the PMD are absolute, relocatable, and complex.

An absolute DEF name is defined by an EQU statement whose operand is an absolute expression. For example:

```
        ENTRY  A101
A101    EQU    100
```

will produce an absolute DEF entry for the symbol A101 whose value will be 100. The dynamic loader does not process the value field of absolute DEFs; the definition, as produced by the assembler, becomes the value of the symbol during program execution.

A relocatable DEF is one whose value is storage-location dependent. For each of these DEFs, the language processors always produce a value that is relative to the base of the defining control section. For example, if a statement at byte location 1000, relative to the origin of its control section, is named CHXAAA, then

```
ENTRY CHXAAA
```

will produce a relocatable DEF entry for the symbol CHXAAA whose value will be 1000. The dynamic loader processes relocatable DEFs by adding, to the value assigned by the language processor, the base virtual address of the defining control section allocated by the dynamic loader.

Type-1 complex DEFs are simply relocatable DEFs whose ENTRY statements appear in a control section other than the one in which the symbol itself is defined. The DEF entry for such a symbol always appears in the CSD of the control section containing the ENTRY statement. Clearly, then, a means must be provided to denote the control section whose base must be added to the symbol value output by the language processor. This is effected by an external reference (REF), which names the control section containing the definition.

Type-2 complex DEFs are defined by EQU statements whose operand fields contain one or more external symbols.

The standard entry point defines the execution starting point associated with a module's name. For this entry point, the language processor prepares a complex DEF entry containing the module name. This treatment allows external references to modules by name (naming a module with a V-con or an R-con is a valid coding practice). The linkage-editor-produced module will retain V-cons and R-cons of original modules, where possible. This is not possible for input modules whose names are identical to the output (linkage-editor-produced) module or for those containing PSECTs or CSECTs previously deleted during linkage editing.

External Symbol Values

TSS/360 employs a convention for linkage to reenterable routines that requires any external symbol to have two values associated with it.

1. The V-value, which is the virtual storage location that the external symbol labels.

2. The R-value, which is the virtual storage location of the origin of the control section in which the ENTRY statement for that symbol appeared in the source code.

The V-value is used to identify the first executable instruction of a public (reenterable) CSECT; the R-value is used to identify the base of the related private PSECT. The user places, within the PSECT code, an ENTRY statement denoting a label within his public CSECT. By convention then, reenterable linkage is effected in the CALL macro instruction by branching to the location that is the V-value of the named symbol and, at the same time, making available to the called routine the R-value

of that same symbol (which is normally the address of the called routine's PSECT).

The following coding example demonstrates how V- and R-values are assigned:

```
A  PSECT
   ENTRY  B
   ENTRY  E
   ORG    A+X'500'
B  EQU    *
   ...
C  CSECT
   ENTRY  D
E  EQU    *
D  EQU    2
   ...
```

Assume that the dynamic loader assigned PSECT A to virtual storage location X'80000' and CSECT C to virtual storage location X'205000'. The table shows the V-value and R-value for all external symbols.

Symbol	V-Value	R-Value	DEF Type
A	00080000	00080000	Relocatable*
B	00080500	00080000	Relocatable
C	00205000	00205000	Relocatable*
D	00000002	00205000	Absolute
E	00205000	00080000	Complex
*CSECT name DEF			

Note that E in this example is the only entry point that is properly coded to permit standard linkage; that is, the R-value of E is the origin of PSECT A.

The DEF entry for a CSECT name is generated by the CSECT statement, not by an ENTRY statement. Therefore, the DEF entry for a CSECT name always has an R-value equal to the V-value (note A and C above). From these definitions, a reentrant CALL by public CSECT name alone is not possible.

The object module name, or standard entry point DEF, is treated somewhat differently. Its V-value is the value of the expression contained on the END statement of an assembly. As stated earlier, the language processor prepares a complex DEF entry for the standard entry point from the END statement. For example, an assembly might look like:

```
X  CSECT
Y  EQU    X+4
Z  PSECT
   END    Y
```

In this example, the assembler will construct a complex DEF for the standard entry point that references CSECT X. If X is assigned virtual storage location 212000 by the loader, then the V-value for the standard entry point will be 212004. The R-value for the standard entry point is computed by the dynamic loader when the module is loaded into virtual storage. This computation results in an R-value equal to the origin of the first declared PSECT in the module (or the first CSECT if no PSECT is declared).

If, in this example, the module name is M, then the net result of the standard entry point computation would be as if the code had been written

```
X  CSECT
M  EQU    X+4
Z  PSECT
   ENTRY  M
   END
```

External References

An external reference (the term "external symbol reference" is synonymous), or REF, is a symbol referred to within a module and defined in the same or another module. The user can create an external reference by means of the assembly EXTRN statement; symbols appearing in the operand field of such a statement appear as REF entries in the CSD of the CSECT containing the statement. The symbols named in V-con and R-con statements will also generate REF entries.

REFs can also arise from complex DEFs. For example, if CSECT AA has symbol X and PSECT BB has the statement:

```
ENTRY X
```

then there will be a REF to AA within BB's CSD.

There may also be REFs for the module name complex DEF. For example:

```
END Z
```

where Z is contained in CSECT CC, will result in a REF that names CC.

External reference to other modules provides the means by which both the linkage editor and dynamic loader can link modules. The occurrence of a REF in some module that names a symbol not defined in the module will provoke a search to locate the module that contains some DEF entry whose name matches the REF name. External references that are not satisfied by a DEF entry in the output module created by the linkage

editor are called unresolved external references.

Explicit vs. Implicit Linkage

Object modules can be linked implicitly or explicitly.

- If two modules, A and B, are implicitly linked, the loading of A implies the loading of B. Also, such modules (if they do not reside in SYSLIB, described later in this section) can be combined into a single object module by linkage editing.

- If two modules, C and D, are explicitly linked, the loading of C does not imply the loading of D; D is not loaded until it is actually (that is, explicitly) referred to during the execution of C. The explicit linkage between two modules is not known to the linkage editor; the user therefore cannot combine such modules during linkage editing on the basis of their explicit linkage.

All FORTRAN subprograms (user-written or system-supplied) are implicitly linked either to the main program or to each other. Loading the main module of a FORTRAN program thus loads the entire program, if the libraries have been defined for the task. Only user-written subprograms can be combined with each other and/or the main program during linkage editing. System-supplied subprograms reside in SYSLIB and cannot be combined in the output module of a linkage editor run. Unresolved references to these modules are noted at the conclusion of the run (for the user's information), but they are not satisfied until the modules are loaded.

Assembler users have the option of writing programs that use a combination of implicit and explicit linkages. (Refer to the descriptions of the LOAD and CALL macro instructions in Assembler User Macro Instructions, GC28-2004.)

OBJECT MODULE COMBINATION

If a program consists of one or more object modules, the user has two options (see Figure 3):

- He can statically combine two or more object modules into one object module, prior to program execution, by using the linkage editor.

- He can let the dynamic loader automatically link the separate object modules as part of the loading process.

Static linking can be used for object module build-up during program development, and it can be used to improve program running time, if there are many implicitly linked object modules in a program. (The improved running time is achieved because only a single module need be located and loaded if the modules are previously combined into one module. If they are not so combined, each module has to be individually located and loaded.) In most cases, however, dynamic linking during execution is satisfactory, and the user does not noticeably improve running time by linkage editor processing.

Program Libraries

Module linking and editing by the linkage editor is a library-to-library process (Figure 4). Program libraries in TSS/360 are partitioned data sets whose members include object modules. Each entry point and non-common control section name in the module is an alias for the member (module) name.

TSS/360 has four library categories:

System Library (SYSLIB): This is the source of all standard, normally available, system routines that are accessible to all users.

User Library: Each user is automatically assigned a private library each time he issues a LOGON command. He is not required to define his user library. The user library is often used to contain checked-out programs (object modules) that the user frequently runs.

Job Libraries: The user may, by use of DDEF or CDD commands and macro instructions (with the JOBLIB operand), define a number of additional libraries during a given task. To facilitate orderly maintenance of programs within various job libraries, the POD? command is available. POD? enables the user to obtain on SYSOUT a list of member names (and optionally the alias names and other member-oriented data) of individual members of cataloged VPAM data sets.

The three categories above form a program library list that is searched automatically (unless otherwise specified) in the following order, to resolve external references:

1. Job libraries, in the reverse order of definitions (that is, the last one defined is searched first)
2. User library
3. System library

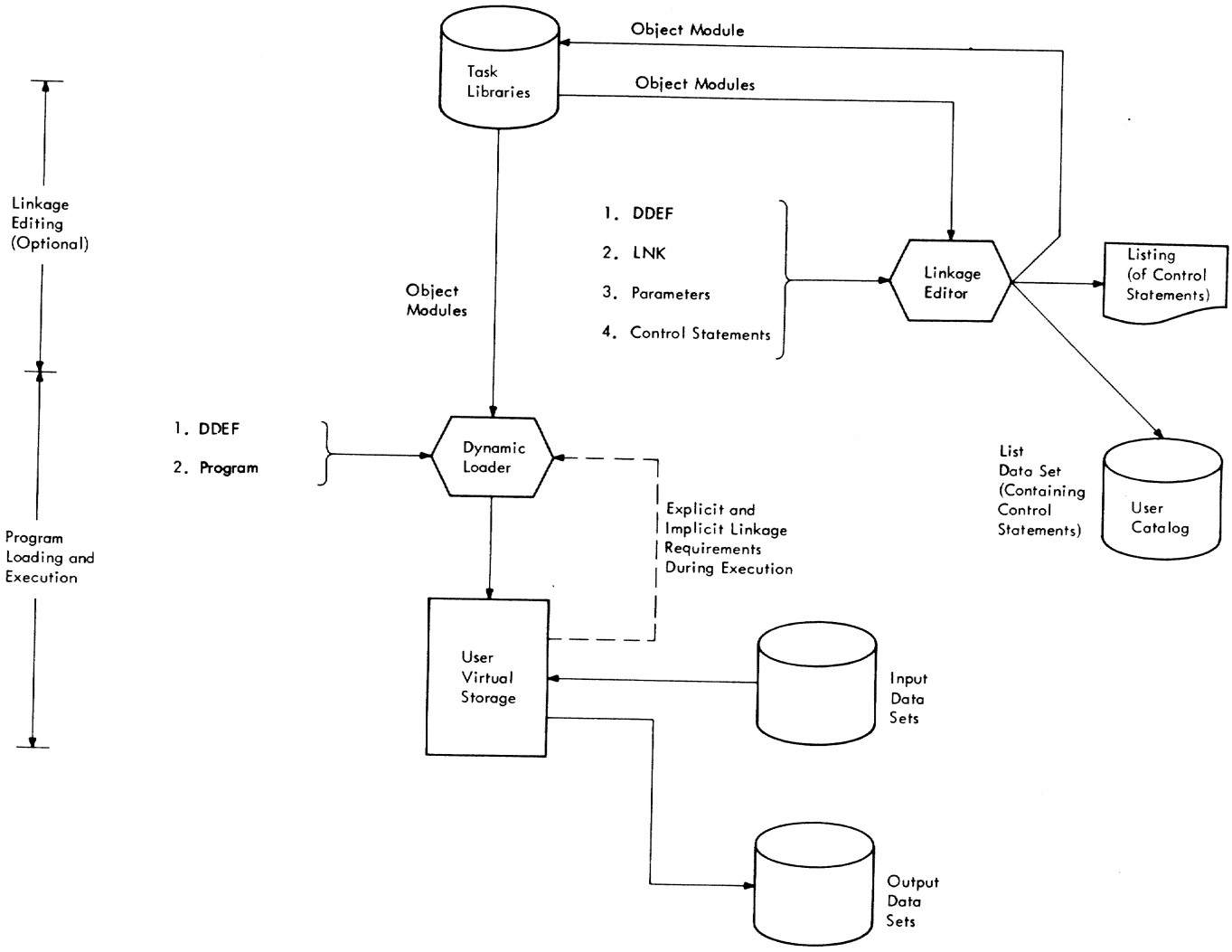


Figure 3. Object Module Combination in TSS/360

Other User-Defined Libraries: The user may also define new or existing libraries that are not job libraries (he simply omits the JOBLIB operand when he issues the DDEF command or macro). Such libraries may be designated as the source of input modules for the linkage editor, or as the destination of the output. An advantage of using this kind of library is that, unlike a job library, it is not put on the user's program library list and is not automatically searched for modules that might resolve external references; the linkage editor user can be sure that only the modules he explicitly specifies from a library will be included in the output. Another characteristic of this kind of library is that its member modules cannot be loaded for execution. When including a module from this kind of library, the linkage editor user must ensure that none of the input module's aliases (entry point and control section names) duplicate aliases in other input modules.

Managing Libraries

The linkage editor INCLUDE statement allows the user to specify the use of any library, except SYSLIB, in the program library list; or he may specify any other library for which a DDEF command (with no JOBLIB) or macro instruction has been issued. If the second option is selected, only the specified library is used as input until either another INCLUDE statement or an END statement is given. However, the linkage editor END statement always causes the entire program library list to be searched to satisfy any remaining unresolved references (except those references specifically excluded). References resolved from the system library will not cause inclusion of modules in the output

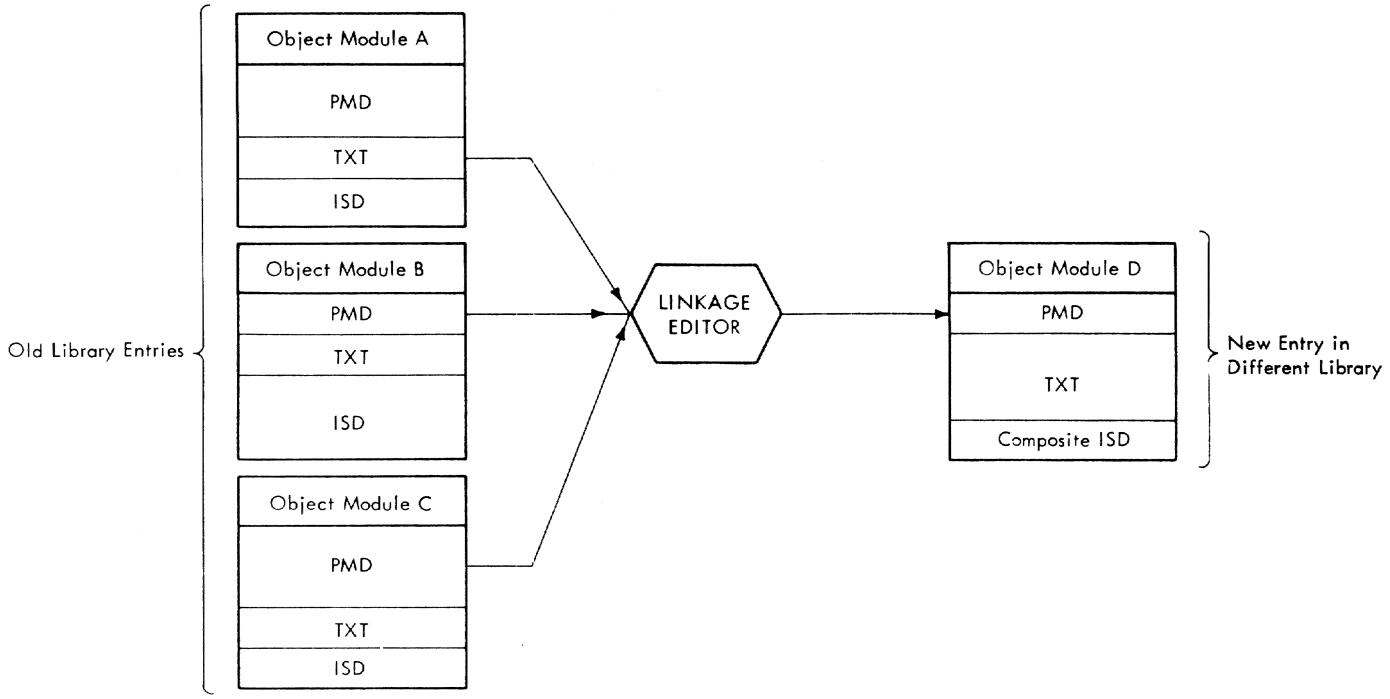


Figure 4. Linkage Editor Processing of Object Modules

module, but will be left for resolution by the dynamic loader.

The user has an option of placing the output from the linkage editor (program module D in Figure 4) in the library currently at the top of the program library list, or in a library he specifies in the LNK command (refer to Section 4, "How to Specify Linkage Editor Processing"). A program library may not contain more than one occurrence of any external symbol; hence the output module may not be placed into any of the libraries from which its parts (the input modules) were obtained, unless during the linkage editing those external names were renamed by a RENAME linkage editor control statement. Similarly, if the user specifies an alternate library, he must ensure (possibly by renaming) that none of the aliases in the output module are identical to aliases in the existing alternate library.

STATIC LINKING: THE LINKAGE EDITOR

The linkage editor INCLUDE statement, described in Section 3, enables the user to link one or more separately generated object modules into one module. A linkage editor run always produces one output object module. The user therefore combines modules by including them in the output object module.

The INCLUDE statement has three forms, allowing the user to either include a particular module (form-1), or include whatever module satisfies unresolved external references (forms-2 and -3).

Form-1: Include a specified module, optionally from a specified library; otherwise from a search of the libraries on the program library list.

Form-2: Search a specified library and include modules found in it that satisfy any currently unresolved external references in the output module.

Form-3: Search a specified library and include modules found in it that satisfy any unresolved external references, except for a specified list of unresolved external references that are not to be resolved.

During execution of form-2 and -3 INCLUDE statements, any unresolved external references in newly included modules will cause the same library to be searched again for modules containing external definitions that satisfy the unresolved references. New modules included by this search may in turn have unresolved external symbols, causing another search. Searches continue until no further unresolved external references can be resolved.

Notes:

1. During a search to satisfy unresolved references, if a module contains a name that matches a reference not to be resolved, the entire module is rejected. This rejection occurs whether or not the module could satisfy other unresolved references. However, such a rejection has no effect on the actions of the dynamic loader in resolving references.

2. The linkage editor will not link two FORTRAN main programs. An attempt to include more than one FORTRAN main program in the output module will result in a major-error condition, even where such an attempt is merely the by-product of an automatic search.

3. If a FORTRAN program requires a BLOCK DATA subprogram, this subprogram must be included prior to any module containing a common block for the same areas defined in the BLOCK DATA subprograms.

Appearance of the END statement in linkage editor processing triggers a final search of the entire program library list, to satisfy any unresolved external references within the limitations of the above notes. Those resolvable from SYSLIB are not included, as they will be linked dynamically by the loader. They are listed, as are any still unresolved external references.

During module linking, those control sections within modules to be linked whose names duplicate entry point names already in the output object module are rejected. Unnamed CSECTs and unnamed common control sections are not automatically rejected. Figure 5 summarizes the linkage editor's rejection actions for control sections.

Input control sections are not rejected if they contain entry point names that duplicate entry point names already in the output module; however, the DEFs for the duplicate entry point names are removed.

The linkage editor checks certain attributes in cases where automatic control section rejection has occurred. Automatic rejection of a control section that may lead to trouble at execution time evokes a diagnostic message, warning the user of a possible problem. Figure 6 summarizes the conditions that result in such a message.

Control Section	Linkage Editor Rejection Action
Named CSECTs, PSECTs, or Common	Subject to automatic control section rejection if name duplicates a control section name or any other entry point name that is already present in the output module.
Unnamed CSECTs	Given a unique numeric identification when processed by the linkage editor; not subject to automatic rejection.
Unnamed (blank) Common	Telescoped into one blank COMMON whose size is equal to the largest blank COMMON input, and whose content (if any) is that of the first blank COMMON input; not subject to automatic rejection.

Figure 5. Automatic Control Section Rejection by the Linkage Editor

Attribute of Input Control Section That Was Rejected	Message Issued If an Accepted Control Section Had Attribute of
Common	Non-common
Non-common	Common
Read-only	Non-read-only
Non-read-only	Read-only
Privileged	Nonprivileged

Figure 6. Conditions for Diagnostic Messages on Control Section Rejection

DYNAMIC LINKING: THE DYNAMIC LOADER

The dynamic loader, a privileged system program, loads and unloads object modules and performs the related functions of linking and unlinking the modules of a program. The user controls the activities of the dynamic loader with these commands and macro instructions:

- LOAD command (implicit linking)
- LOAD macro instruction (implicit or explicit linking)
- RUN command (implicit linking for load-and-run form)
- CALL command (implicit linking)

- CALL macro instruction (implicit or explicit linking)
- UNLOAD command (implicit or explicit unlinking)
- DELETE macro instruction (explicit unlinking)

Refer to Command System Users' Guide for details on the LOAD, CALL, RUN and UNLOAD commands; refer to Assembler User Macro Instructions for explanations of the LOAD, CALL and DELETE macro instructions.

The LOAD command, the CALL command, and the load-and-run form of the RUN command are normally used in connection with the initial or main module of a program. These commands request initiation of the loading process; also, the CALL and RUN commands initiate execution of the program. The module named in these commands is explicitly loaded, and hence can be explicitly unloaded by the UNLOAD command or the DELETE macro instruction. All modules implicitly linked to the module being explicitly unloaded are also unloaded, if they are no longer needed in the user's task.

The LOAD and CALL macro instructions can be used for either implicit or explicit linkage. Both request initiation of the loading process; the CALL macro instruction also initiates execution of the called module. Modules explicitly linked by LOAD and CALL can be explicitly unlinked by the DELETE macro instruction or the UNLOAD command. The CALL macro instruction also may be used for implicit linkage. Modules implicitly linked are unloaded when higher-order explicitly linked modules are unloaded.

The loading process is divided into two distinct phases: the virtual storage allocation phase and the text page address constant (adcon) relocation phase. The allocation phase is executed in response to a request for loading; text page adcon relocation is effected as the page to be relocated is referenced by a page executing in main storage.

Virtual Storage Allocation

The dynamic loader first checks to see if the module identified by the symbol in the LOAD, CALL, etc., command has already been loaded. If it has, no allocation is necessary, since the module defining the symbol must already be a part of the task. In this case, the loader merely fills in the V-value and R-value in the adcon group. If the symbol is not found, a library search is initiated. If the symbol is still unresolved, it cannot be defined for the task, and an error condition exists.

If the symbol is found in a library, the defining module's PMD is obtained and each control section name within the module is checked. Those control sections whose names either duplicate entry point names already known in the current task, or whose names are illegal, are rejected. Control section rejection by the loader differs from the linkage editor's rejection in the treatment of common control sections. The loader will accept the first unnamed common control section it encounters and reject all subsequent unnamed common control sections. Figure 7 summarizes the dynamic loader's rejection action.

Page entries are set up for each of the nonrejected control section's text pages. The external library storage address is recorded with each page table entry, and each page is marked "unavailable." During execution, a reference to any byte on the page will cause an interrupt. This interrupt causes the page to be transferred from the external library into the user's virtual storage at the allocated location.

At the time the page tables are set up, the dynamic loader notes each page that contains adcons. Referencing during execution of the noted pages causes the system action described below, under "Relocation."

Control Section	Loader Rejection Action
Named CSECTs, PSECTs, or named Common	Subject to automatic control section rejection if name duplicates a control section name or any other entry point name already present in the task.
Unnamed CSECTs	Given a unique internal numeric identification when processed by the dynamic loader; it is not subject to automatic rejection.
Unnamed (blank) Common	Given a non-unique internal identification when processed by the dynamic loader, it is subject to automatic rejection; after one unnamed common control section has been processed, any subsequently loaded will be assigned the same name and therefore rejected.

Figure 7. Automatic Control Section Rejection by the Dynamic Loader

The values of all entry points in the nonrejected control sections of the module are computed. Relocatable DEFs are computed by adding to the DEF value the base address assigned to the containing control section. Absolute DEFs require no computation. Complex DEFs are computed last. REFs are processed by searching for a DEF of the same name. If the DEF name to which a REF refers is not found, the entire loading process is initiated to load a module that will define the REF. Such a module-loading cascade will proceed until all REFs in all modules have either been satisfied or been marked undefinable.

Note that it is quite possible for the dynamic loader to satisfy some REF by locating an entry point in some external library, only to have that entry point lost in the allocation process because a control section is rejected.

Relocation

Whenever a page-unavailable interruption occurs, the page is moved into real storage. If the page was marked as having adcons, the dynamic loader computes the value of each adcon on the page. The processing of adcons will always involve the application of an REF value to some portion of the text on the page. There are three possible applications:

1. Add the V-value of the REF to the text value.
2. Subtract the V-value of the REF from the text value.
3. Store the R-value of the REF into the text.

At the time this relocation occurs, all REF values will have been computed during the allocation phase of the loader. When all adcons have been relocated, the dynamic loader returns control to the task monitor and eventually to the point in virtual storage where the relocated page was originally referenced.

Sequencing of Modules: Linkage Editor and Dynamic Loader

The user should be aware that the linkage editor and the dynamic loader use different methods of sequencing the modules to be included in an object program, with the result that, where control sections or entry points are duplicated (and, hence, only the first is retained), the object programs may be different.

This difference is best explained by examining a sample module group and the various module sequences that would result from linkage editor processing and dynamic loader processing. This group is illustrated in Figure 8.

Linkage Editor: The linkage editor, in processing a module specified in an INCLUDE statement, resolves all of the module's primary references first, then its secondary references, then its tertiary references, and so on, until all possible references (except, of course, those from SYSLIB) are resolved. Thus, the order of module inclusion produced by the linkage editor in processing an INCLUDE A statement for the module group shown in Figure 8 would be:

A B C D E F G

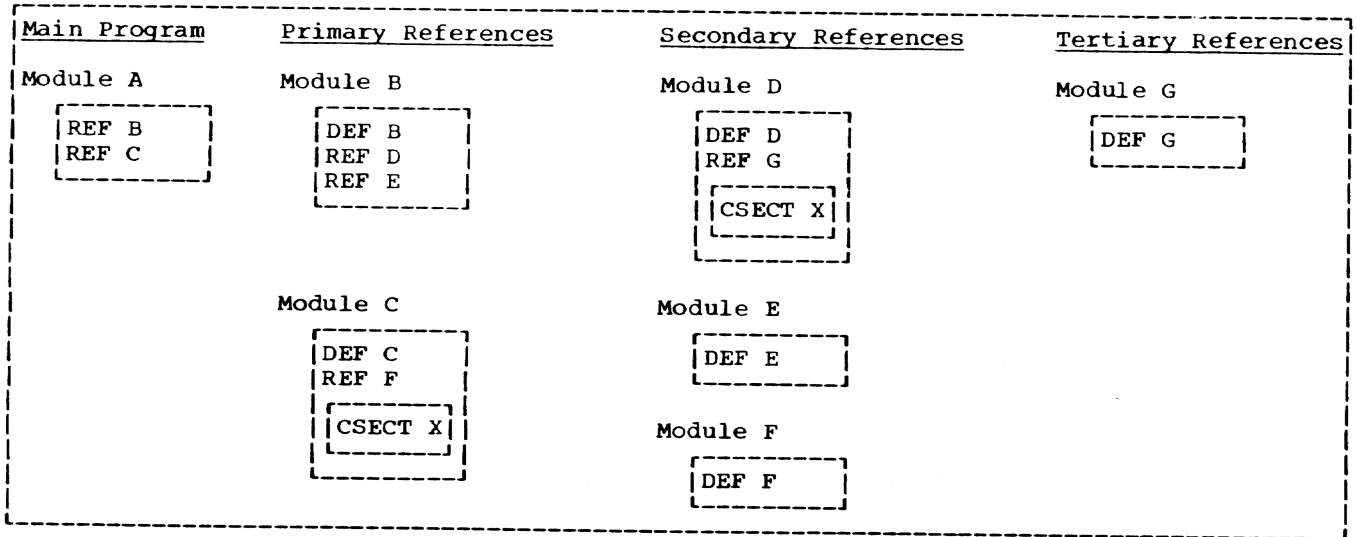


Figure 8. Sample Module Group

Note that CSECT X in Module D would be rejected because of the prior occurrence of CSECT X in Module C.

Dynamic Loader: The dynamic loader, in processing a module, will completely resolve a reference before attempting to resolve the next reference within the same module. Complete resolution consists of resolving all references encountered as a result of including a module to satisfy a previous reference. Thus, in processing a LOAD A statement for the module group shown in Figure 8, the dynamic loader would first encounter the REF to Module B. In processing B, the REF to D would be encountered and resolved by including D. The REF to G in Module D in turn causes G to be included. Since G contains no REFs and D contains no unresolved REFs, the loader returns to processing B, where it encounters the REF for E, etc. Thus, the order of the modules produced by the dynamic loader would be: A B D G E C F

Note that CSECT X in Module C would be rejected because of the prior occurrence of CSECT X in Module D.

SUMMARY OF LINKAGE EDITOR FUNCTIONS

The linkage editor will perform these functions:

1. Two or more object modules, which may have been created by different language processors, can be retrieved from defined libraries and statically linked together to form one object module (see Figure 9). (See the INCLUDE statement.) If desired, this module may be retained and used repeatedly. A statically linked program requires less dynamic loader processing time, at the cost of linkage editor processing time.

2. A module can be moved from one library to another, being retrieved from the library indicated in an INCLUDE statement and placed in the library indicated in the LNK command parameters.

3. Control sections within a module may be replaced, deleted, or renamed. Automatic rejection of control sections occurs when more than one control section has the same name (the INCLUDE, RENAME statements).

4. Entry point names and external references within a module may be renamed (the RENAME statement).

5. Entry point names may be deleted from a module (the RENAME statement).

6. The attributes of control sections may be changed (the TRAITS statement).

7. Two or more control sections of a object module may be combined into a single control section, thus reducing the number of external and virtual storage pages required and reducing paging activity during execution (the COMBINE statement).

8. A composite ISD may be prepared for the output module, as a user option, to facilitate use of PCS.

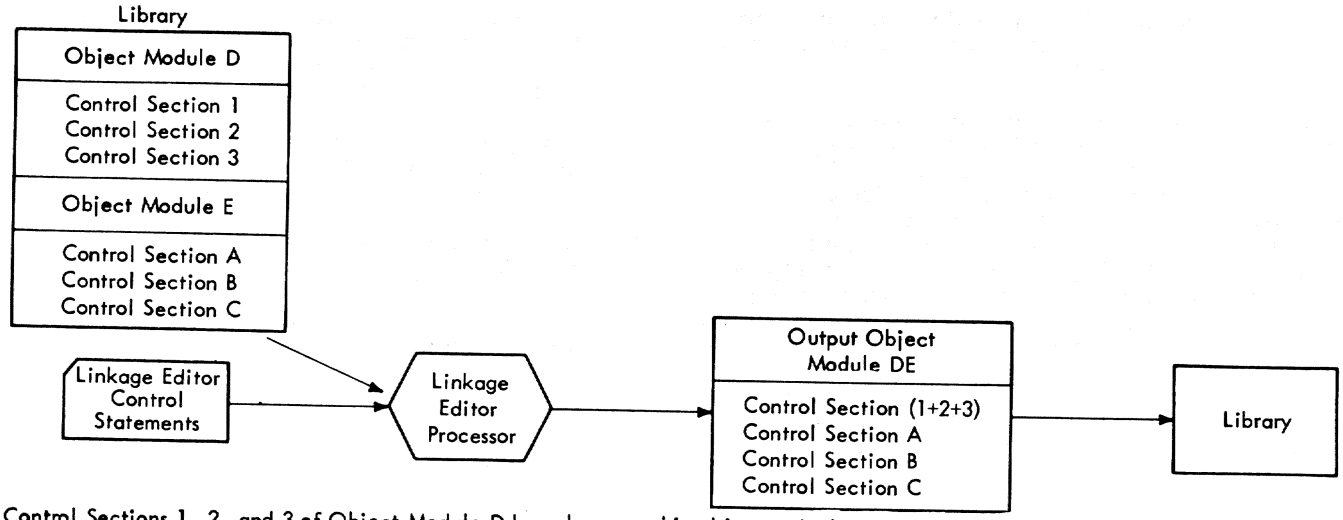
9. Blank (unnamed) common control sections generated by other language processors (FORTRAN, assembler, etc.) are automatically collected, and a virtual storage area equivalent to the largest blank common area is provided within the output module.

10. Unresolved external references, not specifically excluded, are satisfied by an automatic search of the job libraries (the END statement).

11. An optional PMD listing is prepared (parameters of the LNK command).

12. A list of all outstanding unresolved external references is prepared, distinguishing those that will be resolved from SYSLIB from those that need resolution from the JOBLIBs and USERLIB in use when the module is loaded.

13. The standard entry point for each component input module will be retained in the linked output module as an auxiliary entry point. (The user will be warned when this is not possible.)



Control Sections 1, 2, and 3 of Object Module D have been combined into a single Control Section (1+2+3) and Object Modules D and E are linked together to form the Output Object Module DE.

Figure 9. Linkage Editor Processing, Linking Modules, and Combining Control Sections

This section describes the general format for linkage editor control statements, explains the notation used in their description, and summarizes the record format information applicable for both terminal and batch initiation of linkage editor processing.

GENERAL CONTROL STATEMENT FORMAT

All linkage editor control statements have the format:

Operation	Operand
statement name	one or more operands; field may be blank

OPERATION FIELD

The operation field contains one of the following linkage editor control statement names: INCLUDE, RENAME, TRAITS, COMBINE, or END.

OPERAND FIELD

The operand field names the elements to be affected by the control statement and, where appropriate, supplies values that further specify the action to be taken. All linkage editor control statement operands are positional; keywords are not used.

Operand Name

The abbreviations used for the operand names permissible in linkage editor control statements are:

csname -- control section name

module -- name by which module can be retrieved

ddname -- symbolic name of a DDEF command

extref -- external reference

epname -- entry point name

Coded Operand Values

Operand values are entered as a string of from one to eight alphameric characters, the first of which must be alphabetic. The values to be entered are identified in the format illustration provided in Section 3 for each control statement.

Commas, Parentheses, Minus Sign

Commas are used to separate operands. Parentheses are used to enclose sublists within operands. Commas and parentheses (unlike the metasymbols described below) must be written where shown on the format illustrations. Similarly, the minus sign (see INCLUDE statement, form-3) must be written by the user.

Blanks

The operand field of the linkage editor END statement is always blank. For other control statements, if the operand field is blank, the linkage editor will not process the control statement. Blanks may not appear within the operand field except to terminate the control statement. The keyboard tab is treated as one blank.

FORMAT ILLUSTRATIONS

Linkage editor control statement formats are illustrated using the following conventions:

Upper and Lower Case Conventions

Words appearing in upper case are values to be entered exactly as shown. Words appearing in lower case are names of values that the user will supply.

Metasymbols Conventions

The symbols listed below are used in the control statement format illustrations to show how and when to enter a particular operand.

[ ] Brackets are used to denote options; anything enclosed within brackets may be entered or defaulted. For example,

[epname$_2$]

means that the user may enter a second entry point name or not.

{ } Braces are used to group the alternative selections available in the operand fields. The alternatives are stacked (to show they are alternatives) and the group is enclosed in

braces to indicate that one selection is to be made from the group. For example,

means that the user can enter A or B or C.

... Three dots indicate that the preceding may be entered one or more times in succession. For example, in

COMBINE csname,...

the three dots indicate that a series of control sections that are to be combined can be specified and that the control section names are separated by commas. No trailing comma is written after the last control section name:

COMBINE READA,READB,READC

KEYBOARD FORMAT

CHARACTER SETS

KA and KB can be used to specify the character set to be used during keyboard input. With KA, the user indicates he wishes to use the full EBCDIC character set. With KB, the user specifies that the lowercase characters (a-z and $ # @) be translated into their upper-case equivalents (A-Z and ! " ¢).

CONTROL STATEMENT BOUNDARIES

The operation field must start at the first nonblank, nontab character of a line entered on the keyboard. It must be separated from the operand field by at least one blank position (the tab may be used, if desired). The operand field may begin at any position following the operation field, as long as at least one blank (or tab) separates the fields.

The total length of a control statement, including the operation field, separating blanks, and operand field, must not exceed 256 characters. The RETURN key denotes the end of the statement.

CONTINUATION LINES

Each control statement entered from the terminal keyboard must start on a new line. A one-line control statement is terminated by pressing the RETURN key. If a control statement requires more than one line, the hyphen (-) key must be pressed once at the end of the line, just before the RETURN key is pressed, to signal that the next line is a continuation. The statement is then assumed to continue at the first nonblank, nontab character of the statement on the next line.

CARD FORMAT

CHARACTER SETS

CA and CB can be used to specify the character set used during 1056 card reader input. With CA the user indicates he wishes to convert card input from 1057 card punch code to EBCDIC. With CB the user specifies conversion from 029 keypunch code to EBCDIC.

CONTROL STATEMENT BOUNDARIES

The operation field may begin in any column. The operation field must be separated from the operand field by at least one blank position. The operand field may begin anywhere following the operation field, as long as at least one blank separates the fields; it may extend to column 71. The total length of a control statement, including the operation field, separating blanks, and operand fields, must not exceed 256 characters, even if continued.

CONTINUATION CARDS

When it is necessary to use continuation cards, these rules apply:

1. Enter a character (not blank, and not part of the statement coding) in column 72 of the statement line. All characters through column 71 are considered part of the statement.

2. Continue the statement on the next line, starting in any column.

3. When more than one additional line is needed, each line to be continued must have a character (not blank, and not part of the statement coding) entered in column 72. The number of continuation lines is limited only by the 256-character statement-length restriction.

In this section, the purpose, format, placement, and cautions are described for each linkage editor control statement. Examples of each control statement are also given.

The LNK command and its operands, a summary table (Table 2, "Controlling the Linkage Editor"), as well as examples of linkage editor use, are in the next section, "How to Specify Linkage Editor Processing."

INCLUDE

The INCLUDE statement is fundamental to linkage editor processing. Object modules, each of which may contain several control sections and be the products of different language processors, can be linked by the INCLUDE statement to form a single output module.

Three forms of the INCLUDE statement are available. These forms are discussed below. Regarding INCLUDE statement usage, see also "Static Linking: The Linkage Editor," in Section 1.

INCLUDE (FORM-1)

The form-1 INCLUDE statement is used to obtain object modules from a library and place them in the output module, linking them to any modules that have previously been included.

Operation	Operand
INCLUDE	[ddname](module name[,...])

ddname
: specifies the data definition name (not the data set name) of the library to be searched. The library must be on the current program library list, or be otherwise defined by a DDEF or CDD command or macro instruction in the current task. If no library ddname is specified, the libraries on the program library list are searched, in order, for the named modules.

module name
: specifies the object module to be included. The module can be identified by its member name, by the name of any non-common control section it contains, or by the name of a symbol defined in an ENTRY statement in the module. More than one module name may be specified.

Placement: A form-1 INCLUDE statement can be placed before, between, or after other control statements, except END (it may not be placed after END).

CAUTIONS: The input modules explicitly named will be placed in the output module in the order specified in the INCLUDE statement. As each control section of each input module is processed, its name is checked. A control section (other than blank common) will be rejected if it has the same name as a control section name or any other entry point name previously included. Internal references to a particular byte of the rejected section will then refer to the same byte of the retained section. This rejection process does not apply to blank common sections.

INCLUDE (FORM-2)

The form-2 INCLUDE statement instructs the linkage editor to scan a specified library, except SYSLIB, and to include from it other object modules whose external definitions satisfy unresolved external references in the (as yet incomplete) output module. If one of the newly included modules contains unresolved external references, the specified library is again scanned for modules that resolve these names. The scanning continues until no new modules can be included.

Operation	Operand
INCLUDE	ddname

ddname
: specifies the data definition name (not the data set name) of the library to be searched. The library must be on the current program library list, or be otherwise described by a DDEF or CDD command or macro instruction in the current task.

Placement: Since a module must already be included in the output module, at least one form-1 INCLUDE statement must have preceded any form-2 INCLUDE statement.

Table 2. Controlling the Linkage Editor

TSS/360 user issues LNK command specifying:	Name of Output Object Module	Whether Control Statements Prestored	Program Library in Which to Place Output	Version ID	Whether to produce ISD	Whether to produce PMD Listing	Where to Put PMD Listing	Line Number, Increment
Default:	Must specify	Not prestored	Latest library created	System-provided	Produce ISD	Don't produce PMD listing	Conv: Store None: Print	100,100

Then issues any of the control statements below:	Also specifying:	Placing the control statement:*	As a result, the linkage editor:
COMBINE	Names of control sections to be combined.	Ahead of a form-1 INCLUDE which will specify the object module containing the control sections.	At the next form-1 INCLUDE, combines all control sections named into the first control section named in the COMBINE statement.
RENAME	Old and new external references, entry point names, control section.	Ahead of a form-1 INCLUDE which will specify object module containing names.	Changes or deletes specified names in first object module named in next form-1 INCLUDE.
TRAITS	Name of control section and new attributes.	Ahead of a form-1 INCLUDE which will specify object containing control section.	Changes attributes of specified control section in first module named in next form+1 INCLUDE.
INCLUDE (form-1)	The ddname of a program library and object module in it.	Before END.	Causes the specified object module to be included in the output object module and any stacked COMBINE, RENAME, and TRAITS statements to be processed.
INCLUDE (form-2)	The ddname of a program library to be searched.	After at least one previous form-1 INCLUDE.	Searches library, includes in output module all modules referenced by previously included modules.
INCLUDE (form-3)	The ddname of a program library followed by a minus sign; names of external references <u>not</u> to be resolved.	After at least one previous form-1 INCLUDE.	Searches library, includes in output module all modules referenced by previously included modules, <u>except those containing the external references specified</u>.
END	Blank.	Last.	Searches all libraries on the current program library list for modules satisfying unresolved references, includes modules in output module. Furnishes message listing unresolved references and those resolvable in SYSLIB.

*<u>Placement Rules</u>:

1. A COMBINE, TRAITS, or RENAME statement may precede any other COMBINE, TRAITS, or RENAME statement, a form-1 INCLUDE, or END (it will not be processed if it is stacked when the END statement occurs). It cannot be immediately followed by a form-2 or -3 INCLUDE.
2. A form-1 INCLUDE can be placed before, between, or after any other statement except END (it may not be placed after END).
3. A form-2 or -3 INCLUDE must have been immediately preceded by a form-1, -2, or -3 INCLUDE, and at least one previous form-1 INCLUDE must have occurred.
4. The END statement must be last.

CAUTIONS: If a control section is encountered that has the same name as a control section name or any other entry point name previously included in the output module, it is rejected. Internal references to a particular byte of the rejected section will then refer to the same byte of the retained section. This does not apply to unnamed CSECTs.

The order in which external references are resolved by the linkage editor is not the same as that of the dynamic loader. It is therefore possible that the modules implicitly linked by the linkage editor will differ from those that the loader would link.

INCLUDE (FORM-3)

The form-3 INCLUDE statement is used to direct the linkage editor to scan a specified library, other than SYSLIB, and resolve certain external references, but not others. The user specifies, in the INCLUDE statement, the external references he does not want resolved; all others are resolved if possible. If one of the newly included modules contains unresolved external references whose resolution is not excluded by the user, the specified library is again scanned for modules that resolve those names. This process continues until no new modules can be included.

The excluded names are presumably to be resolved by subsequent INCLUDE statements. If not, they will remain unresolved in the output object module, and be resolved by the dynamic loader when the output program module is loaded.

Note: Although the automatic search at the end of linkage editor processing will not resolve an excluded name, the search caused by a form-1 INCLUDE with no library specified will resolve it, if the name is in the program library list and the form-1 INCLUDE follows the form-3 INCLUDE.

Operation	Operand
INCLUDE	ddname-(extref[,...])

ddname
: specifies the data definition name (not the data set name) of the library to be searched. The library must be on the current program library list, or be otherwise described by a DDEF or CDD command or macro instruction in the current task.

minus sign
: specifies that this is a form-3 INCLUDE statement, not a form-1 INCLUDE.

extref
: specifies the external reference that is not to be resolved in the search. More than one external reference may be specified.

Placement: Since a module must already be included in the output module, at least one form-1 INCLUDE statement must precede any form-3 INCLUDE statement.

CAUTION: If a control section is encountered that has the same name as a control section name or any other entry point name previously included, it is rejected. Internal references to a particular byte of the rejected section will then refer to the same byte of the retained section. Similarly, all but the largest blank common section will be rejected. The internal references will refer to the first blank common encountered (even if it was rejected).

EXAMPLE OF INCLUDE STATEMENTS

This example describes the action of the linkage editor in response to these control statements:

```
INCLUDE LIBA(MA,MB)          (form-1)
INCLUDE LIBF                 (form-2)
INCLUDE MATH-(COS,SQRT)      (form-3)
END
```

Figure 10 illustrates the contents of three libraries with ddnames LIBA (object modules MA and MB), LIBF (object modules MF and MG), and MATH (object modules TRIG and ARITH).

INCLUDE LIBA(MA,MB) causes modules MA and MB to be placed in the output module. These modules have five external references that must be resolved by other modules: ALPHA, BETA, GAMMA, SIN, and COS.

INCLUDE LIBF causes the linkage editor to search library LIBF for modules containing the unresolved references. Although the module MF has entry point names ALPHA and BETA, only ALPHA can be resolved (by adding control section N4 to the output module). Control section N1 is rejected because it has the same name as a control section already in the output module. (BETA is still unresolved, and SQRT has been added to the list of unresolved references.)

CAUTION: Any internal references in this library to the rejected control section now refer to the same byte in the retained control section -- the reference is not to a

LIBA

MA		
Control Section Name	Entry Point Name	External Reference Name
N1	E1	
N2	E2	ALPHA
		BETA
		GAMMA
N3	E3	SIN

MB		
Control Section Name	Entry Point Name	External Reference Name
N6	E6	COS

LIBF

MF		
Control Section Name	Entry Point Name	External Reference Name
N1	BETA	
N4	E4	SQRT
	ALPHA	

MG		
Control Section Name	Entry Point Name	External Reference Name
N5	E5	
	GAMMA	
N8	E8	

MATH

TRIG		
Control Section Name	Entry Point Name	External Reference Name
SINE	SIN	
	COS	

ARITH		
Control Section Name	Entry Point Name	External Reference Name
SQRT 1	SQRT	
CUBE 1	CUBE	

Figure 10. Contents of Libraries LIBA, LIBF, and MATH

symbol but simply to an address relative to the beginning of the control section. For example, a reference in control section N4 to BYTE12 of N1, where BYTE12 is the internal name of the 12th byte of that control section, becomes a reference to the 12th byte of control section N1 from object module MA.

Since object module MG has the entry point name GAMMA, the module, consisting of control sections N5 and N8, is added to the output module, and GAMMA is removed from the list of unresolved references.

INCLUDE MATH-(COS,SQRT) causes the linkage editor to search the library MATH for all unresolved references except COS and SQRT. Only one other unresolved reference is in the MATH library (SIN in the TRIG program module), but control section SINE is rejected because COS, an entry point name within it, has been explicitly excluded.

At this point, the output object module contains these control sections:

```
From LIBA:      N1, N2, N3, N6
From LIBF:      N4, N5, N8
From MATH:      none
```

The unresolved references are:

BETA, SIN, COS, SQRT

Following the END statement, the linkage editor searches the program library list for BETA and SIN, the remaining unresolved references that have not been explicitly excluded. (No search is made in the other job libraries or the user library to resolve COS and SQRT.) At the top of the program library list are the job libraries defined by the INCLUDE statements in the example. If any other job libraries have been defined within this task, they will also be searched for the unresolved references (the last-defined is searched first). If a module containing BETA or SIN as an entry point name is found in any job library, that module is added to the output module. Next, the user library is searched. Assuming, in this example, that BETA was found in the user library; that the module containing it had only one control section; that the name of the control section is J4; the output module will be as described in Figure 11.

The linkage editor will determine which of the still unresolved references can be resolved from the standard system library (SYSLIB) and will list all those external names resolvable from SYSLIB in a diagnostic message. These references will be resolved before the program is executed; the dynamic loader will load the modules from SYSLIB during the loading of the program.

The linkage editor will also print a list of any unresolved references which are not in the system library and which, therefore, will not be resolved by the loader.

Control Section Name	Entry Point Name	Unresolved References
N1	E1	
N2	E2	
N3	E3	SIN
N6	E6	COS
N4	E4	SQRT
	ALPHA	
N5	E5	
	GAMMA	
N8	E8	
J4	BETA	

Figure 11. Resultant Output Object Module

RENAME

The RENAME statement provides a means of deleting or renaming control sections and entry point names, and of renaming external references that are to be included in the output object module.

Operation	Operand
RENAME	$extref_1$($extref_2$) $epname_1$[($epname_2$)] [,...] $csname_1$[($csname_2$)]

$extref_1$
specifies an external reference that is to be renamed.

$extref_2$
specifies the new name for $extref_1$.

$epname_1$
specifies an entry point that is to be renamed or deleted.

$epname_2$
specifies the new name for $epname_1$. If $epname_1$ is written, but $epname_2$ is not, the entry point name specified by $epname_1$ will not appear in the output module.

$csname_1$
specifies a control section that is to be renamed or deleted.

$csname_2$
specifies the new name for $csname_1$. If $csname_1$ is written, but $csname_2$ is not, the control section specified by $csname_1$ will not appear in the output module.

Placement: RENAME statements must be placed ahead of form-1 INCLUDE statements. RENAME acts only upon the first module in the operand of the next form-1 INCLUDE statement. The operations specified by the RENAME statement will be executed before the next input module is considered for inclusion in the output module. This offers the user a means of overriding the automatic replacement of control sections, which occurs when more than one section has the same name. A control section may be explicitly deleted so that another control section with the same name will not be rejected, or it can be renamed, so that both control sections will be retained.

RENAME statements that are to act upon the same module as one or more COMBINE statements must appear after all such COMBINE statements have been issued. Therefore, a control section name mentioned in the RENAME operand should refer to the name that identifies a combined section.

CAUTION: Deletion of entry point names and control sections causes any external references to these names to be unresolved until they are satisfied by new entry point names or control sections of the same name. In requesting RENAME operations, four conventions should be followed, to ensure that the specified change or deletion is processed correctly.

1. External symbol references from other object modules to a changed entry point name or control section name must be changed by separate RENAME control statements. (External references and address constants within the same module automatically refer to the new symbol.)

2. For assembler language programmers only: When control sections that were or are part of a separately assembled module are to be replaced, A-type address constants that refer to a deleted symbol will be incorrectly resolved unless the entry point name is in the same position relative to the origin of both the replaced control section and the new control section. If all control sections of a separately assembled module are replaced, no such restrictions apply.

3. For assembler language programmers only: Literals normally should be pooled by LTORG instructions if control sections are to be replaced, since defaulted literal pools fall at the end of the first control section. If a control section containing pooled literals is replaced or deleted, the linkage editor will replace or delete the entire control section, including pooled literals. The linkage editor does not pool literals.

4. For a given input module, any name mentioned in a RENAME statement should not be used again in the same or a subsequent RENAME statement.

Note: The names of external dummy sections become external references. They may be renamed but not deleted.

EXAMPLES OF RENAME STATEMENTS

```
RENAME ALPHA1(BETA1)
RENAME ALPHA3(BETA3),N1(N9)
INCLUDE LIBA(MA)
RENAME N1(N9)
INCLUDE LIBA(MB)
RENAME N1(N9)
INCLUDE JOBLIB1(MC)
```

Figure 12 shows a library with ddname LIBA, which contains object module MA with two control sections, and object module MB with two control sections. The library with ddname JOBLIB1 contains object module MC with one control section.

The first three control statements in the example, above, include MA in the output module while making these changes: Entry name ALPHA1 is changed to BETA1, control section name N1 is changed to N9, and the external reference ALPHA3 is changed to BETA3. The next two control statements include MB in the output module and change the external reference N1 of MB to N9. This reference is satisifed by module MA. The external reference BETA3 (old ALPHA3) of module MA is also resolved at this point by MB. The last two control statements add MC to the output module and change the external reference N1 of MC to N9. The two external references N9 and BETA1 of MC are resolved by MA.

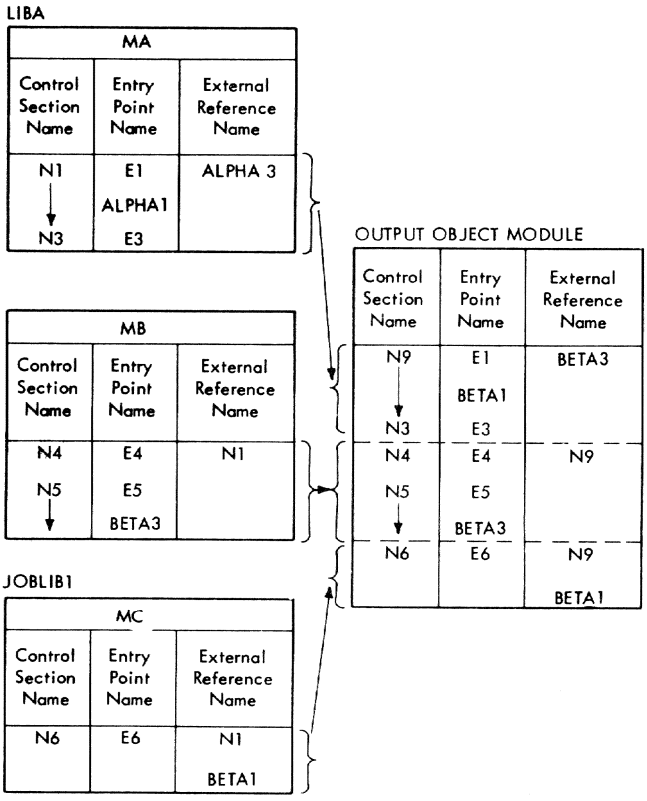


Figure 12. Contents of Libraries LIBA and JOBLIB1 and Resultant Output Object Module

TRAITS

Control sections within modules that are produced by the language processors are assigned certain implicit or explicit traits or attributes. The TRAITS statement provides the user with the facility to redefine attributes of a control section, causing all attributes previously assigned to be removed.

Operation	Operand
TRAITS	csname[([,VARIABLE][,READONLY] [,PUBLIC][,PROTO][,COMMON] [,PRVLGD][,SYSTEM])]

csname
the name of a control section in the first module mentioned in the next form-1 INCLUDE statement.

The attributes are described in Section 1, "Control Section Attributes."

If no attributes are present in the operand, fixed-length is the only attribute assigned to the specified control section.

The traits may be written in any order. They are enclosed in parentheses and separated by commas. No trailing comma is written.

Placement: The TRAITS statement must be placed ahead of a form-1 INCLUDE statement; it acts only on the first module in the operand of the next form-1 INCLUDE statement.

CAUTION: Note that only traits specifically included in the TRAITS statement are assigned to the named control section; all those assigned prior to the linkage editor run are dropped by the TRAITS statement.

The linkage editor does not check on the validity of attributes if the user assigns them to a control section with a linkage editor TRAITS statement. Such assignments should be made with care, since conflicts may arise during loading if improper attributes have been assigned.

EXAMPLE OF TRAITS STATEMENT

The following statements assign the VARIABLE attribute to control section KSECT of module MODK of library LIBK and include the module in the output module.

```
TRAITS    KSECT(VARIABLE)
INCLUDE   LIBK(MODK)
```

COMBINE

The COMBINE statement provides the ability to combine two or more control sections of a module into a single control section. Since each control section must normally occupy at least one page (4096 bytes) of virtual storage, an object module containing a number of short control sections is inefficient in its use of virtual storage and may require unnecessary paging activity when the module is executed. Thus the COMBINE statement may be used to reduce the number of pages of external and virtual storage required for a module.

Control sections appearing in separate modules may be combined by first link editing their modules, thus placing the control sections in a single module, and then combining the control sections in a subsequent linkage editor run.

Operation	Operand
COMBINE	csname,...

csname
specifies a control section name in the first module mentioned in the next form-1 INCLUDE statement.

Unless changed by a subsequent RENAME statement, the first control section name in the operand field is used as the name assigned to the new control section. The names of the control sections that follow the first become external definitions in the combined control section. The RENAME statement, as described previously, must be issued after the control sections have been combined.

Placement: The COMBINE statement must be placed ahead of a form-1 INCLUDE statement.

CAUTIONS:

1. The COMBINE statement acts only on the first module mentioned in the next form-1 INCLUDE statement.

2. The control sections that are to be combined must have identical attributes. If they do not, a diagnostic message will be issued. New attributes may be assigned to one of the nonidentical control sections in the usual manner, employing a TRAITS statement ahead of the COMBINE statement.

3. For a given input module, any name mentioned in a COMBINE statement should not be used again in the same or subsequent COMBINE statements.

EXAMPLES OF COMBINE STATEMENT

```
COMBINE   N1,N3
INCLUDE   JOBLIB1(MA)
COMBINE   N4,N5
RENAME    N4(N40)
TRAITS    N40(VARIABLE)
INCLUDE   LIBA(MB)
```

The first two statements direct the linkage editor to combine control sections N1 and N3 of object module MA in library JOBLIB1, and to include this module in the output module. The name of the newly formed control section is N1.

The last four statements result in control sections N4 and N5 of module MB in library LIBA being combined, and module MB is included in the output module. The new control section bears the name N40 and is assigned a VARIABLE attribute.

END

The END statement is used to indicate the end of the linkage editor control statements for the current run.

Operation	Operand
END	Always blank

The END statement directs the linkage editor to search all the libraries on the current program library list, to satisfy any remaining unresolved references not specifically excluded by form-3 INCLUDE statements. References resolvable from the system library modules do not cause those modules to be included in the output module, since they will be resolved by the dynamic loader during execution. All other references that are resolvable cause the defining module to be linked to the output module, assuming that the defining module contains no excluded names, and that the linkage will not violate FORTRAN standards. (See "Static Linking: The Linkage Editor," in Section 1.) Any remaining unresolved references will be listed in a diagnostic message. All references resolvable from SYSLIB will be listed in a separate diagnostic message.

Placement: END must be the last linkage editor control statement.

SECTION 4: HOW TO SPECIFY LINKAGE EDITOR PROCESSING

The preceding section showed how to use the linkage editor control statements; this section shows how to initiate linkage editor processing (with the LNK command) and illustrates various ways linkage editor processing may be approached.

Table 2, "Controlling the Linkage Editor," summarizes the linkage editor process from the user's point of view.

Figure 13, "Methods for Specifying Linkage Editor Processing," summarizes various ways the user may approach the system.

THE LNK COMMAND

Whether the user is operating from a terminal, or his job is entered nonconversationally by an operator, the process of linkage editing must begin with an LNK command. The parameters that can be entered as operands of the LNK command are shown below. Only one operand is required -- the name of the new object module. Operands may be either keyword or positional. Underlines show defaults.

NAME
: specifies the name of the object module to be created. The name consists of one to eight alphameric characters, the first of which must be alphabetic. The source data set, consisting of the control statements that direct the linkage editor, will be named SOURCE.name. The list data set (the PMD listing), if requested, will be named LIST.name. The module name must be unique to the library that will include it. (It must not be the same name as any external definition in any module already in the library.)

  Notes: If the module name supplied by the user matches that of an existing list data set (that is, LIST.module), the action taken depends on the task mode. If nonconversational, the existing list data set will be erased; if conversational, the user will be given an opportunity to save the old list data set.

STORED
: specifies whether the source data set containing the linkage editor control statements is prestored. If it is, it must have been named SOURCE.name, where name is the module name assigned in the first parameter of this command. The allowable values are Y or N. If the choice is N, the user may also enter line and increment values.

  Default: N

LIB
: specifies the ddname of the library in which the new object module is to be stored. This library must be different from any of the input libraries and it must be defined by a DDEF or CDD command or macro instruction in the current task.

  Default: The library currently at the top of the user's library list is assumed.

VERID
: specifies the version identification to be assigned; it consists of one to eight alphameric characters, the first of which must be alphabetic.

  Default: The listing and the created modules are time-stamped by the system.

ISD
: specifies whether an ISD (internal

```
LNK | NAME=module name[,STORED={Y|N}][,LIB=data definition of library]
    | [,VERID=version identification][,ISD={Y|N}][,PMDLIST={Y|N}]
    | [,LISTDS={Y|N}][,LINCR=(first line number,increment)]
```

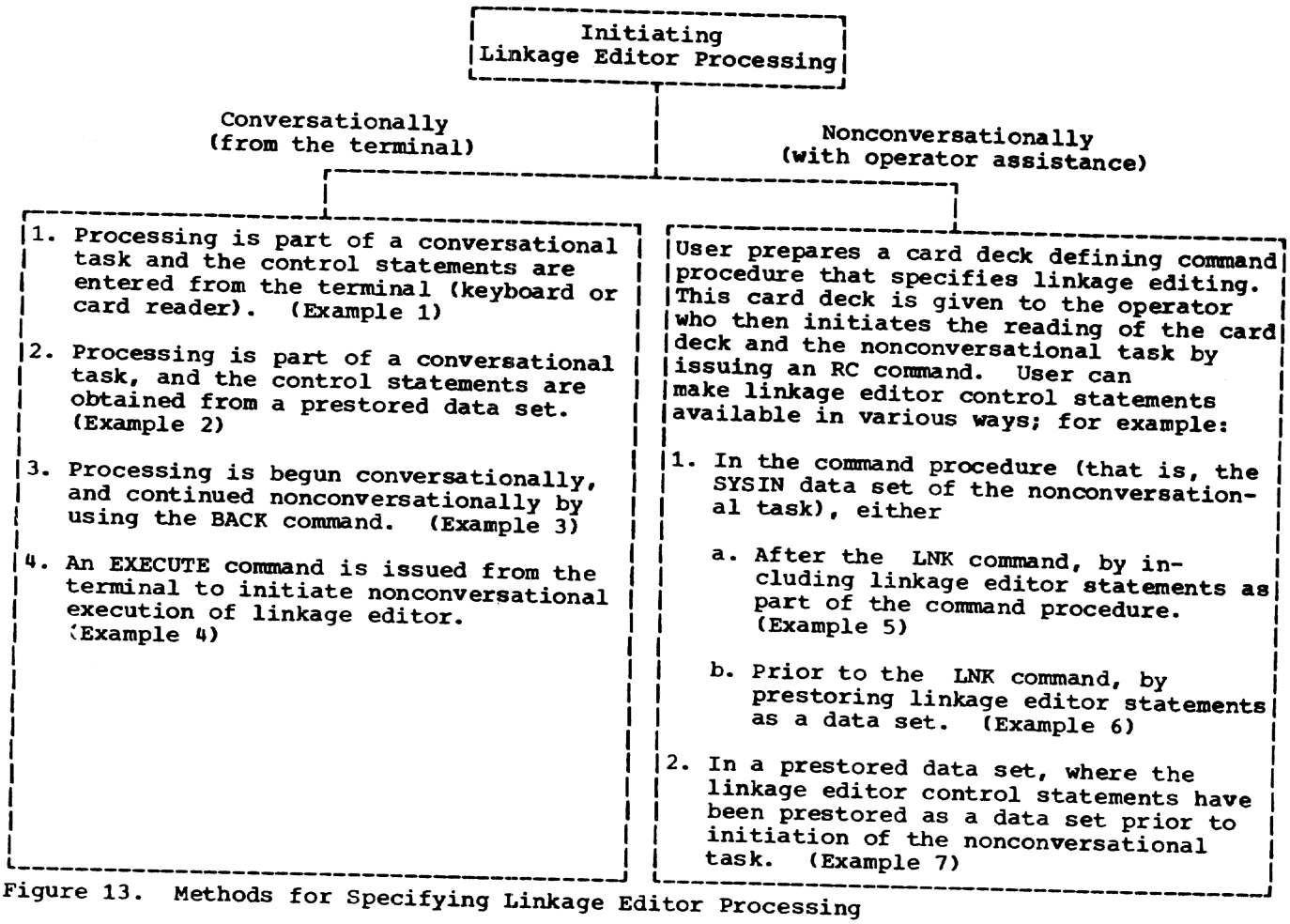


Figure 13. Methods for Specifying Linkage Editor Processing

symbol dictionary) is to be retained. The allowable values are Y or N.

Default: Y

PMDLIST
specifies whether a PMD (program module dictionary) listing is to be produced. The allowable values are Y or N.

Default: N

LISTDS
specifies whether the PMD listing, if requested, is to be put into a list data set for later access or is to be immediately printed on SYSOUT.

Specified as:

Y -- listing goes into a list data set.
N -- listing goes to SYSOUT.

Default: Y (conversational)
N (nonconversational)

LINCR
line
specifies the line number to be assigned to the first line of the source data set created. The line number may contain three to seven digits, of which the last two must be 00.

Default: The starting line number will be 100.

increment
specifies the increment to be applied to develop successive line numbers. The increment may contain three to seven digits, of which the last two must be 00.

Default: The increment will be 100.

USING THE LINKAGE EDITOR CONVERSATIONALLY

In the conversational mode of operation, the user communicates with the system through a terminal (for example, an IBM 2741 Communications Terminal or an IBM 1052 Printer Keyboard). The conversational user can communicate either through the terminal keyboard or, on a 1052 equipped with an IBM 1056 Card Reader, by entering cards.

Assume that the user has already logged on, and that he has previously stored all the object modules he intends to use in libraries (via previous compilations or assemblies, or previous linkage editor processing). The steps below are required to execute the linkage editor; data flow is illustrated in Figure 14.

1. The user must ensure that each library to be referred to in a particular session of linkage editor processing is defined within that session. These libraries can be defined by a DDEF command or macro instruction, or by a CDD command or macro instruction. (See Command System User's Guide for an explanation of the log-on process, for details concerning commands in this publication, and for means of prestoring and cataloging data sets in the system.)

2. The user issues an LNK command to initiate execution of the linkage editor, followed by a series of linkage editor parameters (described earlier in this section).

3. When the linkage editor is ready for a line of input, the next sequence of events depends on whether the linkage editor control statements are entered from the terminal or are prestored.

   a. If the control statements are entered from the terminal, the system invites input by writing a line number at the terminal. The user then types the contents of the line, which is stored in the control statement data set the system now creates.

   b. If the control statements are prestored, the system reads the control statements from the SOURCE. name data set.

4. As each control statement is received by the linkage editor, it is analyzed for correctness, and processed according to its particular function.

5. Errors that are discovered by the linkage editor will result in a diagnostic message being typed at the terminal to prompt the user to correct the statement in error. There may be one or several messages for each control statement. (These messages are described in Appendix B.) The procedure for correcting statements in error is described below.

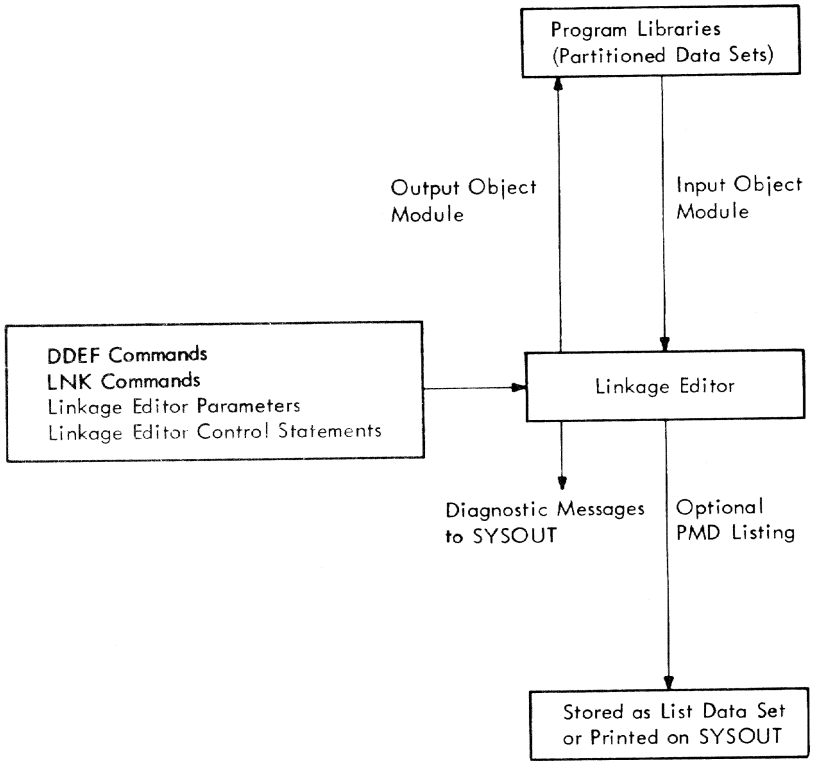


Figure 14. Data Flow During Conversational Linkage Editor Processing

The system asks the user if there are any modifications. If the user response is a yes, the system invites the user to enter modifications by writing a pound sign (#) on the terminal and unlocking the keyboard. The user may then enter a correction line[1], the first part of which is a line number identifying the line involved, followed by a comma and the content of the line. For example,

```
#500, INCLUDE JOBLIB2(MB)
```

This line is stored in the data set, either as an insertion line or as a replacement line, and the next correction line is requested by issuance of the #. To delete one or more lines, the user types, following the #,

D,lineno	(where lineno is a line number and llineno is the final line number of a range of lines to be deleted.)
or	
D,lineno,llineno	

The indicated line or, in the second case, all lines within the specified range of line numbers, are deleted. Such corrections change the source data set permanently. To end corrections, the user merely presses the RETURN key in response to the #. The correction lines are then processed by the linkage editor. If no corrections were entered, or after correction line processing, the next line is taken from the source data set for processing.

Note: Normally, during conversational execution, modifications made to statements previously entered cause the linkage editor to start again at the first control statement. An exception to this occurs when the user modifies one or more statements of a statement group in response to a linkage editor diagnostic message. (Statement groups are described in Appendix B.) In this case, the linkage editor continues operation from the beginning of the affected statement group.

---

[1]Or he may enter: i (This will inhibit display of all messages and error prompts until the final request for modifications.)
Or he may enter: c (This will inhibit the prompts for correction; the diagnostic messages are issued. The user will not be prompted until the final request for modifications.)

6. When the entire control statement data set is collected and processed, the linkage editor ends its first phase and produces a list of any unresolved external references and a list of external references resolvable from SYSLIB. In conversational mode, the linkage editor asks the user if he wants to make modifications and restart, continue processing as-is, or terminate processing.

7. If the user indicates that he wishes to continue, the linkage editor retains control until it has finished, and then informs the user of the level of error found during processing. Four severity levels are possible: 0 (no errors; ordinarily, the user will not be informed of this level), 1 (minor errors), 2 (major errors, indicating questionable validity of the output module), and 3 (fatal errors, meaning an output module could not be produced). The fatal error condition typically occurs if no form-1 INCLUDE statement appears in the set of control statements for a linkage editor run.

8. Finally, the object module is stored in the library specified by the user. If the PMD listing response was Y, either a list data set is produced as a new generation of a generation data group named LIST.name or the listing is sent to SYSOUT (the printer if the task is nonconversational; the terminal, if conversational). The LISTDS operand specifies the disposition of the PMD listing. If the user desires a printed listing from his list data set, he must issue the PRINT command. (See Command System User's Guide for a full explanation of the listing data set maintenance process for the language processors.) This concludes linkage editor processing.

DEFINING LIBRARIES

To make the following examples simpler the CDD (Copy DDEF) command is used. CDD is a useful command which allows users to prestore one or more DDEF commands in a line data set and to invoke them by issuing CDD with the name of the line data set as an operand. Details on using this command can be found in Command System User's Guide, GC28-2001.

In the CDD command used in the following examples, LNKED.DD is a line data set containing:

```
DDEF LKLIB1,VP,LIB1DS
DDEF DDJLIB1,VP,JLIB1,OPTION=JOBLIB
  .
  .
  .
DDEF DDJLIBN,VP,JLIBN,OPTION=JOBLIB
```

The first DDEF above defines a library with the ddname LKLIB1, used in the examples below as the library in which the linkage editor output module is to be placed. The remaining DDEFs define job libraries to which the linkage editor will have access in the examples. These job libraries become part of the program library hierarchy which may be searched to locate modules to be included, to locate modules which may satisfy unresolved references, and which could be used to stow the module produced by the linkage editor.

These DDEFs could also be submitted separately by the user or may have been placed in a PROCDEF. (Using PROCDEF, however, the DDEFs encountered would not be reproduced on the SYSOUT as are those taken from the data set named on a CDD.)

EXAMPLE 1

Conversational Linkage Editing: Control Statements Entered From Terminal Keyboard

The user first wishes to link three modules (LEMOD1, LEMOD2 and LEMOD3) that are in a library on the program library list; he then wishes to combine two control sections (CSECT1 and CSECT2) and add a fourth module (EYLE1), also available from the program library list.

Log on: The user first logs on:

User: (Hits attention button or dials up system)
System: (Releases keyboard)

User: logon mnjdoe,mnj733
System: B001 LOGON TASKID=F207 11/19/71
(The system then provides an underscore on the next line, indicating readiness to accept new command)

Define Libraries: The user then either enters the DDEF commands that are required to identify the libraries to be used during this linkage editor run, or, as shown below, he retrieves DDEF commands from a previously cataloged line data set, LNKED.DD.

User: cdd lnked.dd
System: (Retrieves the DDEF commands cataloged under the dsname of LNKED.DD, and prints each on SYSOUT preceded by four zeros and a comma.)

INITIATE LINKAGE EDITOR PROCESSING: To load and begin execution of the linkage editor, the user enters the LNK command, followed by the parameters that are to control this run. In the example shown, the parameters are entered in positional format; however, if he wishes, the user may instead enter these parameters with keywords.

User: lnk ts2lnk,n,lklib1,,n,y,,(500,100)

(TS2LNK)
the member name by which the linkage-editor object module is to be stored.

(,N)
tells the system that the linkage-editor control statements are not prestored, but are to be obtained from SYSIN.

(,LKLIB1)
tells the system to put the object module, TS2LNK, in the library having the ddname LKLIB1.

(,)
tells the system to time-stamp the listing and created module.

(,N)
tells the system not to produce an internal symbol dictionary (ISD).

(,Y)
tells the system to produce a program module dictionary listing.

(,)
tells the system the listing is to be stored in a list data set.

(,(500,100))
the line number to be applied to the first line of the control-statement source data set, and the increment to be applied to develop successive line numbers.

If the user chose to enter his parameters with keywords, the example would appear:

User: lnk name=ts2lnk,lib=lklib1,isd=n,-
pmdlist=y,lincr=(500,100)

ENTER LINKAGE EDITOR CONTROL STATEMENTS: When the linkage editor is ready for a control statement, the system prints out a line number at the terminal. The user types the line, presses the RETURN key, and the line is made available to the linkage editor.

Sys,User: 0000500 include (lemod1,lemod2, lemod3)

Sys,User: 0000600 combine csect1,csect2

As each control statement is received by the linkage editor, it is analyzed for correctness and processed according to the particular functions it specifies. If errors are discovered by the linkage editor, a diagnostic message is typed at the terminal prompting the user to correct the statement in error. A typical message is illustrated by this example.

Sys,User: 0000700 include,(eyle1)

System: 0000700 E***ILLEGAL DELIMITER,
0000700 INCLUDE,(EYLE1)

Sys,User: #700, include (eyle1)
(the user corrects the statement in error)

The user signifies that all his linkage editor control statements have been inserted by entering an END control statement.

Sys,User: 0000800 end

At this time, the linkage editor attempts to resolve any unresolved external references by an automatic search of the libraries on the program library list. It then provides a list, at the terminal, of all finally outstanding unresolved external references, distinguishing those that can be resolved from SYSLIB from those that need resolution from the user library or job libraries at execution time.

If any errors have been diagnosed, the linkage editor then asks the user if he wishes to make modifications:

System: (the linkage editor asks the user if there are any modifications to the control statement data set)
User: n

The linkage editor then informs the user of the success (no errors) or failure (one or more errors) of the run. Assuming there were no errors, the system would print out the following at the terminal:

System: (The system informs user that no errors were found.)

The linkage editor's output is then automatically stored in the library with ddname LKLIB1. If the user wants a printout of the PMD listing requested above in the LNK command, he types:

User: print list.ts2lnk(0),,,edit

The PMD listing printed will be the current generation of LIST.TS2LNK. The printing of the list data set is not automatic; it must be requested using the PRINT command as above.

System: (system accepts the PRINT command and assigns a batch sequence number for a separate background task for printing of the list data set.)

If the user wishes to log off at this point, he issues a LOGOFF command.

User: logoff

System: (system accepts LOGOFF)

EXAMPLE 2

Conversational Linkage Editing: Control Statements From a Prestored Data Set

This example is identical to Example 1, except that the linkage editor control statements are obtained from a prestored data set SOURCE.TS2LNK; thus, correction lines are treated in a slightly different manner. Note that in this example, the first two operands are entered as keyword operands, the remaining operands positionally.

.
.
(Statements as shown in Example 1)
.
.
.

User: lnk name=ts2lnk,
stored=y,lklib1,,n,y

.
.
.

System: 0000700 E***ILLEGAL DELIMITER,
0000700 INCLUDE,(EYLE1)

(keyboard is unlocked for the user to enter a correction line)

Sys,User: #700,include (eyle1)

Sys,User: # (presses return key)

(Same as shown in Example 1)
.
.
.

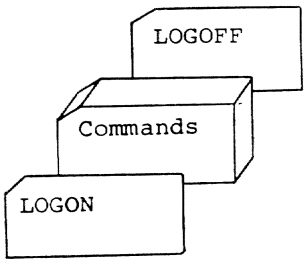
USING THE LINKAGE EDITOR IN NONCONVERSATIONAL MODE

The ways in which nonconversational linkage editing can be specified are described below.

1. The user can start his task conversationally and then switch to nonconversational mode by using the BACK command. He may switch at any time he wishes (and the system will allow him to do so). He must have created a data set containing a command procedure prior to issuing the BACK command. This data set will then be used as SYSIN to control the nonconversational continuation of his task.

2. In a conversational task, the user can issue an EXECUTE command that refers to a cataloged data set containing a command procedure that will function as a SYSIN to specify and control the linkage editor run. The command procedure must contain a LOGON and a LOGOFF command, and all other commands and parameters required for linkage editor processing. The linkage editor control statements can be included in the command procedure, or they can be in a prestored data set. Once the user has issued the EXECUTE command and received the system's reply identifying his nonconversational task, he can perform other tasks at the terminal, or he can LOGOFF.

3. The user may prepare his command procedure as a punched card deck and submit it to the system operator for batch processing. The commands are punched, each starting on a new card, in the format used to enter commands from a terminal. (Linkage editor control statement card-input formats are described in Section 2.) The card deck setup is simple (note that LOGON and LOGOFF commands must start in card column 3):



The first card must be a LOGON command; the last card of the data set must be a LOGOFF. Other commands are as required for the particular linkage editor run.

When the command procedure data set is read in, it becomes the SYSIN data set of a nonconversational task and is executed as soon as space is available. After execution, the SYSIN data set is eliminated. It does not remain cataloged nor does it remain in system storage unless the user makes arrangements within the command procedure itself.

An important factor in all nonconversational processing is that no prompting facilities are available. The user must specify exactly what the system is to do in the command procedure he defines to control the nonconversational task; any error may terminate the task. Another factor is that the system defines and prints out the SYSOUT of nonconversational processing. Results of nonconversational linkage editing are therefore available through printouts from the central computing facility.

EXAMPLE 3

Linkage Editor Processing Initiated Conversationally; Continued Nonconversationally With BACK Command

The user wants first to correct one of the linkage editor control statements in the conversational mode, then to switch to nonconversational operation to process the statements. His source statements are prestored in data set SOURCE.TS2LNK. He also has prestored the command procedure named LNK2, below, which contains commands defining his nonconversational linkage editor run and which will logoff the task involved.

Source statements are to be taken from SOURCE.TS2LNK, a prestored data set; no special output library is desired, nor is a special versionid. The ISD and a listing of the PMD are requested. The PMD listing, by default, will be printed on SYSOUT at the central facility.

```
LNK2   CDD LNKED.DD
       LNK TS2LNK,Y,,,Y,Y
       LOGOFF
```

(the user employs a MODIFY command to make a correction)

```
User:     modify source.ts2lnk
System:   (asks user to enter modifications)

          (the user enters a correction
          line below; EYLE2 is in the same
          library as EYLE1, previously processed in Example 1. The user
          indicates the end of his
          modifications)

Sys,User: #700,include (eyle2)
          #%e
```

(the user now enters the BACK command to change the task to nonconversational mode)

User: back lnk2

System: (accepts task and assigns a batch sequence number)

Further work on other tasks requires another LOGON.

EXAMPLE 4

Nonconversational Execution Requested From Terminal by EXECUTE Command

The user wants to initiate a nonconversational task that does not require a prior conversational phase. Thus, an independent nonconversational task is created, rather than changing the user's conversational task to nonconversational (as was done in Example 3). His source statements are prestored in data set SOURCE.TS2LNK. The command procedure for controlling his nonconversational task (and for completely defining his nonconversational linkage editor run) is contained in a data set named LNK3.

In this example, the user wants to save the requested PMD listing in a list data set and then have it printed on SYSOUT. He must explicitly include a PRINT command in his prestored data set.

```
LNK3  |LOGON JOHNDOE,JOH733
      |CDD LNKED.DD
      |LNK TS2LNK,Y,,,Y,Y,Y
      |PRINT LIST.TS2LNK(0),,,EDIT
      |LOGOFF
```

(the user requests nonconversational execution)

User: execute lnk3

System: (accepts task and assigns a batch sequence number)

User: (may proceed in conversational mode)

In each of the following examples, a card deck is given to the operator who then initiates the nonconversational task. The linkage editor processing that is illustrated is the same as in Example 1.

EXAMPLE 5

Operator-Assisted Initiation: Linkage Editor Control Statements Included in Command Processing

The following command procedure is prepared in punched-card form. The linkage editor control statements form part of this procedure, and thus will follow the LNK command, as shown below.

```
(the user logs on; LOGON must begin in
 column 3)

LOGON JOHNDOE,JOH33

(DDEF commands are retrieved)

CDD LINKED.DD

(linkage editor is loaded for execution
 and given its parameters for this run)

LNK TS2LNK,N,LKLIB1,,N,Y,,(500,100)

(the linkage editor control statements
 are processed.  The PMD listing
 requested is automatically printed on
 SYSOUT)

INCLUDE (LEMOD1,LEMOD2,LEMOD3)
COMBINE CSECT1,CSECT2
INCLUDE (EYELE1)
END

(the user logs off; LOGOFF must begin
 in column 3)

LOGOFF
```

EXAMPLE 6

Operator-Assisted Initiation: Linkage Editor Control Statements Prestored Prior to LNK Command

The linkage editor control statements are first placed in a data set via a DATA command, after which the LNK command is issued.

```
(the user logs on; LOGON must begin in
 column 3)

LOGON JOHNDOE,JOH33

(creates a line data set from the
 statements that follow the DATA
 command)

DATA SOURCE.TS2LNK,I,(500,100)

(linkage editor control statements to
 be placed in the data set)

INCLUDE (LEMOD1,LEMOD2,LEMOD3)
COMBINE CSECT1,CSECT2
INCLUDE (EYLE1)
END
```

```
(the following "%E" completes the data
set creation process begun with DATA
command)

%E

(retrieve DDEF commands)

CDD LNKED.DD

(linkage editor loaded for execution
and passed parameters; note that second
parameter (Y) indicates a prestored
data set)

LNK TS2LNK,Y,LKLIB1,,N,Y,Y
PRINT LIST.TS2LNK(0),,,EDIT

(the user logs off; LOGOFF must begin
in column 3)

LOGOFF
```

EXAMPLE 7

Operator-Assisted Initiation Using Prestored Linkage Editor Control Statements

The linkage editor control statement data set has been prestored prior to initiation of the current nonconversational task; only in this respect does it differ from Example 6.

```
LOGON JOHNDOE,JOH733
CDD    LINKED.DD
LNK TS2LNK,Y,LKLIB1,,N,Y,Y
PRINT LIST.TS2LNK(0),,,EDIT
LOGOFF
```

Figure 15 shows the format of the optional PMD listing for an object module. The format components are described below in the order in which they appear in the listing.

MODULE

NAME
is the alphameric name of the object module, assigned by the user in his compiler, assembler, or linkage editor command parameters.

VERSION
is either the eight-character version ID for the module, or a time and date stamp.

LENGTH
is the length of the PMD in bytes.

DIAG SEVERITY
is a value from 0 through 2 that indicates the highest level of diagnostic encountered while the module was being generated by the originating processor. The greater the value, the more critical the diagnostic level encountered.

The following applies to entries for each control section dictionary.

SECTION
is a two-digit hexadecimal control section number within this PMD.

NAME
is the alphameric name of the control section.

TYPE

CONTROL
indicates a standard control section, as described in Section 1.

COMMON and PROTOTYPE
are as described in Section 1.

VERSION
is a time and date stamp for the control section.

ATTRIBUTES

FIXED, VARIABLE, READONLY, PUBLIC, PRVLGD, SYSTEM
are as described in Section 1.

CSD LENGTH
is the length of this control section dictionary, in bytes.

SECT LENGTH
is the length of the control section in virtual storage, in bytes.

RELOCATABLE DEFINITION
is a section name, or an entry point whose ENTRY statement appeared within an unnamed section.

ABSOLUTE DEFINITION
is an EQU item with absolute value whose name has been declared an entry point.

COMPLEX DEFINITION
is either an EQU item with a complex relocatable value, or a simple relocatable definition whose ENTRY statement appeared within a named section other than the control section in which it is defined.

For each of the above three types of definitions, these entries appear.

NAME
is the alphameric name of the symbol defined.

VALUE
is the hexadecimal displacement value of the symbol relative to the base of its control section.

REFERENCE
is an external symbol referenced within the control section.

REF #
is the ordinal hexadecimal number of the reference.

NAME
is the alphameric name of the reference.

```
MODULE
  NAME              aaaaaaaa
  VERSION           aaaaaaaa (or MO/DD/YYbHH:MM:SS)
  LENGTH            xxxxxxxx
  DIAG SEVERITY     xxx
SECTION  xx
  NAME              aaaaaaaa
  TYPE              CONTROL(or COMMON or PROTOTYPE)
  VERSION           MO/DD/YYbHH:MM:SS
  ATTRIBUTES        FIXED or VARIABLE, READONLY, PUBLIC, PRVLGD.
  CSD LENGTH        xxxxxxxx
  SECT LENGTH       xxxxxxxx
  RELOCATABLE DEFINITIONS
    NAME   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa      aaaaaaaa
    VALUE  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx      xxxxxxxx

    NAME   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa      aaaaaaaa
    VALUE  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx      xxxxxxxx

  ABSOLUTE DEFINITIONS
    NAME   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa      aaaaaaaa
    VALUE  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx      xxxxxxxx

  COMPLEX DEFINITIONS
    NAME   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa      aaaaaaaa
    VALUE  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx      xxxxxxxx

  REFERENCES
    REF #  xxxx      xxxx      xxxx      xxxx      xxxx      xxxx      xxxx          xxxx
    NAME   aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa  aaaaaaaa      aaaaaaaa

  MODIFIERS FOR COMPLEX DEFS
    PAGE xx   # MODIFIERS xxxx
      LENGTH  x         x         x         x         x         x         x             x
      REF #   xxxx      xxxx      xxxx      xxxx      xxxx      xxxx      xxxx          xxxx
      TYPE    y         y         y         y         y         y         y             y
      BYTE    xxx       xxx       xxx       xxx       xxx       xxx       xxx           xxx

  MODIFIERS FOR TEXT (EXTERNAL REFS)
    TEXT PAGE xx   VIRTUAL PAGE xx # MODIFIERS xxxx
      LENGTH  x         x         x         x         x         x         x             x
      REF #   xxxx      xxxx      xxxx      xxxx      xxxx      xxxx      xxxx          xxxx
      TYPE    y         y         y         y         y         y         y             y
      BYTE    xxx       xxx       xxx       xxx       xxx       xxx       xxx           xxx

  MODIFIERS FOR TEXT (INTERNAL REFS)
    TEXT PAGE xx   VIRTUAL PAGE xx # MODIFIERS xxxx
      LENGTH  x         x         x         x         x         x         x             x
      REF #   xxxx      xxxx      xxxx      xxxx      xxxx      xxxx      xxxx          xxxx
      TYPE    y         y         y         y         y         y         y             y
      BYTE    xxx       xxx       xxx       xxx       xxx       xxx       xxx           xxx


SECTION xxx
  NAME
  TYPE
  ATTRIBUTES
  CSD LENGTH
  TEXT LENGTH

    (etc.)
```

a = alphameric character
x = hexadecimal digits
y = +, -, or R

Figure 15. Format of PMD Listing

CXD AND DXD REFERENCE
is a DXD symbol or CXD value referenced within a control section.

REF #
is the ordinal hexadecimal number of the reference; the first REF number is zero.

NAME
is either the name of the DXD symbol or '(CXD)' for a CXD reference.

LENGTH
is the byte length of a DXD symbol, in hexadecimal bytes, or, for a CXD reference, omitted.

ALIGN
is the alignment of a DXD symbol, in alphameric characters ('BYTE', 'HALF', 'FULL', or 'DOUBLE'), or, for a CXD reference, omitted.

MODIFIERS FOR COMPLEX DEFS

PAGE
is the hexadecimal page number of PMD that contains the complex definition.

MODIFIERS
is the total hexadecimal number of modifiers for complex definitions.

LENGTH
is the length of the adcon to be modified, in hexadecimal bytes.

REF #
is the ordinal hexadecimal number, in the appropriate reference list, of the reference whose definition value is to be used in modifying the adcon.

TYPE
indicates one of three operations (+,-, or R) to be performed in modifying the adcon by the reference value.

(+) The definition value of the reference at REF # is added to the field of LENGTH bytes, starting at the indicated byte of the page to which the modifier applies.

(-) The definition value of the reference at REF # is subtracted from the field of LENGTH bytes, starting at the indicated byte of the page to which the modifier applies.

(R) The definition value of the name of the control section in which the reference at REF # is defined, plus the R-value displacement associated with the definition, is stored in the field of LENGTH bytes, beginning with the indicated byte of the page to which the modifier applies.

BYTE
is the displacement in bytes (from the origin of its containing page) of the adcon to be modified.

MODIFIERS FOR TEXT (EXTERNAL REFs, Q-CONs, and CXDs) and
MODIFIERS FOR TEXT (INTERNAL REFs)
employ the same entries, with the same contents, as modifiers for complex DEFs, with these additional entries:

TEXT PAGE
indicates the number assigned by the linkage editor to the text page to which the related modifiers pertain. Text page numbers begin at zero. Pages reserved but not containing data constants or instructions (as the result of DS instructions) are not assigned text page numbers. Although all other pages are assigned numbers, only those containing address constants require modifiers and are shown in the PMD listing.

VIRTUAL PAGE
indicates the number assigned to a page after processing by the dynamic loader. Virtual page numbers also begin at zero. Although all pages, whether containing reserved space, data, or instructions, are assigned virtual page numbers, only those virtual page numbers for which there is a correlated text page containing address constants (and therefore requiring modifiers) are shown in the listing.

And under:

TYPE
two additional operations may be indicated (Q or C) to be performed in modifying the adcon by the reference value.

(Q) The definition value of the reference at REF # is added to the field at LENGTH bytes, starting at the indicated byte of the page to which the modifier applies, and the reference is to be treated as a DXD symbol.

(C) Same operation as for (Q); and the reference is a CXD, containing the cumulative length of all DXD symbols encountered.

APPENDIX B: ERROR DIAGNOSTICS AND MESSAGES

LEVELS OF LANGUAGE PROCESSOR DIAGNOSTICS

Four levels of error detection are provided for the TSS/360 language processors, to associate different severity levels with errors or special conditions encountered during processing. With each occurrence of an error or special condition, a severity level of 1, 2, or 3 is noted by the language processor. The more serious the error or condition encountered, the greater the severity level assigned. At termination of language processor execution, the highest severity level noted during processing is printed.

NO ERRORS. No errors were encountered during creation of the object module (severity level 0). This message normally does not appear.

MINOR ERRORS. At least one minor error was encountered during creation of the object module. The language processor takes automatic corrective action, which probably will not affect the validity of the module (severity level 1).

MAJOR ERRORS. At least one major error was encountered during creation of the object module. If the user is in nonconversational mode, the processor either recognizes a default condition or makes an assumption as to what the user actually desired. The validity of the module is questionable. If the user is in conversational mode, the processor prompts the user to correct his program (severity level 2).

TERMINATED: NO OBJECT MODULE CREATED. A fatal error was encountered; the processor was unable to create the object module (severity level 3).

LINKAGE EDITOR ERROR DETECTION

The linkage editor sets two of the four levels of diagnostics provided for language processors for logging errors and possible error conditions. The appropriate level of either 0 (no errors) or 2 (major errors) is assigned to the output module. (The linkage editor does not set severity level 1; however, if any of the input modules contain a level 1 or 2 code, it is passed on to the output module.) As error conditions are detected, the linkage editor, governed by the operation mode in use, responds as follows:

1. Conversational Mode: The linkage editor sends diagnostic messages to the terminal, prompting the user to correct the error condition in his program with the appropriate recovery procedure. Instead of actually correcting the error at the terminal, the user may choose not to correct the error, which causes the linkage editor to continue processing according to a default condition or terminate, if unable to continue.

2. Nonconversational Mode: When an error condition is detected by the linkage editor, the appropriate default conditions are undertaken following delivery of the diagnostic, and processing continues. Messages are sent to the system output data set (SYSOUT).

LINKAGE EDITOR DIAGNOSTIC MESSAGES

The diagnostic messages produced by the linkage editor at the terminal, with the associated causes, suggested conversational mode recovery procedures, and default conditions, are shown in IBM System/360 Time Sharing System: System Messages, GC28-2037, Part IV, Section 2. Linkage editor error messages are under the examples prefixed by "line E ***" and linkage editor warning messages are under the examples prefixed by "line W ***".

The term "statement group", used in the diagnostic messages, refers to the set of linkage editor control statements which relate to a given form-1 INCLUDE statement, and include the INCLUDE statement itself. For example, in the following sequence, the first two statements form a statement group, and the last four form a second statement group.

```
COMBINE    N1,N3
INCLUDE    JOBLIB1(MA)
COMBINE    N4,N5
RENAME     N4(N40)
TRAITS     N40(VARIABLE)
INCLUDE    LIBA(MB)
```

Statement groups are affected by the default action taken by the linkage editor in many error situations. In these instances, the entire statement group is ignored and processing continues with the first control statement following the statement group.

Each diagnostic message is prefixed by

nnnnnnnbsb***

where nnnnnnn is the line number of the linkage editor control statement, b is blank, and s is either E or W; E denotes a major error (severity level 2), and W denotes a warning message (severity level 1).

Examples of conversational recovery procedures are in Section 4, "How to Specify Linkage Editor Processing."

APPENDIX C: SUMMARY OF LINKAGE EDITOR CONTROL STATEMENTS

Operation	Operand
INCLUDE (Form 1)	[ddname,](module name[,...])

Operation	Operand
INCLUDE (Form 2)	ddname

Operation	Operand
INCLUDE (Form 3)	ddname-(extref[,...])

Operation	Operand
RENAME	$extref_1$ ($extref_2$) $epname_1$[($epname_2$)] [,...] $csname_1$ [($csname_2$)]

Operation	Operand
COMBINE	csname,...

Operation	Operand
TRAITS	csname[([,VARIABLE][,READONLY] [,PUBLIC][,PROTO][,COMMON] [,PRVLGD][,SYSTEM])]

Operation	Operand
END	Always blank

APPENDIX D: LINKAGE EDITOR STATEMENT EXAMPLES AND SIZE LIMITATIONS

INCLUDE STATEMENT EXAMPLES

1. Include modules named MODA and MODB, from the library with ddname LIBA.

```
INCLUDE  LIBA(MODA,MODB)
```

2. Include the module having external name EXTERN. Assume that the name is defined in the JOBLIB hierarchy but the library name is not known.

```
INCLUDE  (EXTERN)
```

3. Resolve unresolved references from the library with ddname LIBC. Assume that one or more modules have already been linked to the output module.

```
INCLUDE  LIBC
```

4. Include modules named MODF and MODG from the library with ddname LIBJ, and module named TRIG from the library hierarchy. Assume that modules MODF and TRIG both contain a blank (unnamed) common control section.

```
INCLUDE  LIBJ(MODF,MODG)
INCLUDE  (TRIG)
```

The output module will contain one blank common control section; its size will be the larger of the two input sections.

RENAME STATEMENT EXAMPLES

1. Rename external reference names XTREF1 and XTREF2 to XR1 and XR2, respectively, in module MODA of the library with ddname LIBA,and include the module in the output module.

```
RENAME   XTREF1(XR1),XTREF2(XR2)
INCLUDE  LIBA(MODA)
```

2. Include modules MODF and MODG from the library with ddname LIBB. Assume that both modules contain a control section named CSECT1, and that it is desired to override the automatic deletion of duplicate-named control sections by renaming the one in module MODF to CSCT1.

```
RENAME   CSECT1(CSCT1)
INCLUDE  LIBB(MODF,MODG)
```

TRAITS STATEMENT EXAMPLE

Assign the PUBLIC attribute to control section JSECT of module MODJ of the library with ddname LIBJ, and include the module in the output module. Assume that the control section previously had a READONLY attribute, which is to be retained.

```
TRAITS   JSECT(READONLY,PUBLIC)
INCLUDE  LIBJ(MODJ)
```

COMBINE STATEMENT EXAMPLE

1. Combine control sections CSECT1, CSECT2, and CSECT3 of module MODA of the library with ddname LIBA, and include the module in the output module.

```
COMBINE  CSECT1,CSECT2,CSECT3
INCLUDE  LIBA(MODA)
```

The name assigned to the combined control section is CSECT1.

2. Combine control sections JSECT and KSECT of module MODK of the library with ddname LIBK, and include the module in the output module. Rename the combined control section to LSECT.

```
COMBINE  JSECT,KSECT
RENAME   JSECT(LSECT)
INCLUDE  LIBK(MODK)
```

EXAMPLE OF LINKING MODULES

1. Link modules MA, MF, and MG of Figure 10.

2. Assign the variable attribute to control section N2 of module MA. Combine control sections N1 and N4 of module MF, and rename the combined section to N10.

The following set of statements will accomplish the task.

```
TRAITS   N2(VARIABLE)          (1)
INCLUDE  LIBA(MA)              (2)
COMBINE  N1,N4                 (3)
RENAME   N1(N10),E1(E10)       (4)
INCLUDE  LIBF(MF,MG)           (5)
END                            (6)
```

(1) Assigns the variable attribute to the first module in the following INCLUDE (Form 1) statement.

(2) Includes module MA in the output module, and leaves external references ALPHA, BETA, GAMMA and SIN unresolved.

(3) Combines control sections N1 and N4, to form one section with the label N1.

(4) Renames control section N1, and renames entry point E1 of this section. Control section N1 of module MA, already included, also contains an entry point named E1. Thus, renaming of the second E1 to E10 is necessary to avoid removal of the DEF for the second E1.

(5) Specifies that the preceding RENAME and COMBINE statements are to act on input module MF of library LIBF. After execution of those statements, includes modules MF and MG in the output module. Resolves external references ALPHA and GAMMA. Leaves external references BETA, SIN, and SQRT unresolved.

(6) Indicates end of control statement list. Directs linkage editor to search program library list for unresolved external references. (See Section 3, END statement.)

SIZE LIMITATIONS

Figure 16 shows the size limitations currently imposed by the linkage editor upon each input module and the output module; Figure 17 shows the size limitations imposed upon the control statements submitted to the linkage editor.

	Input Module	Output Module
PMD	256 pages	256 pages
Text	256 pages	256 pages
ISD	512 pages minus length of output module ISD*	256 pages

*That is, 512 pages is the limit for the total length of all input module ISDs plus the output module ISD.

Figure 16. The Linkage Editor Imposes Size Limitations on its Input and Output Modules

Statement	Number of Statements Allowed	Number of Characters Allowed per Statement*
INCLUDE	No limit.	256
RENAME TRAITS COMBINE	64 permitted if each contains the maximum of 256 characters. The limit increases as the number of characters per statement decreases. The total number of characters cannot exceed 16,384.	
END	One permitted. Any prestored source statements following the first END statement are ignored.	256

*Excluding continuation characters and extraneous blanks.

Figure 17. The Linkage Editor Imposes Size Limitations on its Control Statements

The meanings of the words defined in this glossary apply only to their use in this book; these words may have slightly different meanings in other TSS/360 publications. General TSS/360 definitions are provided in IBM System/360 Time Sharing System: Concepts and Facilities, GC28-2003.

absolute DEF: A DEF (external definition) established by an assembler EQU statement whose operand is an absolute value. For instance, this example would produce in the control section dictionary an absolute DEF entry for symbol A101 whose value would be 100:

```
        ENTRY  A101
A101    EQU    100
```

adcon: See address constant.

address constant: Space reserved in a program for the address of a symbol; program text that changes as the result of relocating the program in storage. The address constant reserves storage in a program for an address that cannot be known when the program is written and ensures that the address value will be filled in before the code containing the address constant is brought into main storage. In the following assembler statement, NAME1 contains an address constant and SUBPROG is the symbol whose address is furnished:

```
NAME1  DC  A(SUBPROG)
```

In processing address constants, the language processors and linkage editor create external reference (REF) entries in the control section dictionary. These entries enable the dynamic loader to resolve the address constant (that is, compute the virtual storage address and insert it in the reserved text word) when the page containing the address constant is referred to during program execution.

alias: 1. An alternate name that may be used to refer to a member of a partitioned data set; 2. an alternate entry point by which a program (that is, a stored member of a partitioned data set) may be called. The linkage editor and language processors all produce an external name list which is used by the VPAM STOW system routine to compile a list of aliases by which a program (that is, object module) may be called.

COMBINE: A linkage editor control statement that combines two or more control sections from an input object module into one control section in the object module being built by the linkage editor. Since each control section must start on a page boundary, combining several short control sections may reduce the total number of pages required. Page compaction in terms of virtual or main storage may also be achieved in TSS/360 through CSECT packing (specified as a LOGON command parameter); an advantage of combining with the linkage editor is that space is saved on external storage as well.

common control section: A type of control section (created with the COM assembler language instruction or the FORTRAN COMMON statement) which may contain areas and constants referred to by independent assemblies or compilations (separate object modules) that are to be loaded for execution as one overall program. (See also control section.)

complex DEF: Either of two types of external definition (DEF): A type-1 complex DEF results from a symbol being named as the operand of an ENTRY statement in a control section other than the one in which the symbol occurs as the name of a statement. This DEF is an entry in the CSD of the control section containing the ENTRY statement. A related REF (external reference) is created in that CSD to refer to the control section in which the symbol names a statement.

A type-2 complex DEF results from an EQU statement whose name is the operand of an ENTRY statement and whose operands are one or more symbols defined as external in an EXTRN statement.

composite ISD: The ISD (internal symbol dictionary) produced by the linkage editor. The linkage editor does not recompile a list of internal symbols, but simply includes in its output module each ISD existing in input modules. The composite ISD thus consists of: 1. each ISD just as it appeared in its input module, and 2. a directory which heads the composite ISD and relates the external definitions and references of each input module to those in the output module.

control section: The smallest unit of a program that is relocatable to virtual storage; that portion of text specified by the programmer to be an entity, all ele-

ments of which are to be allocated contiguous virtual storage locations. A control section begins on a page boundary and consists of an integral number of pages; the page (4096 bytes) is the smallest unit of a program that can be placed in main storage. Control section may refer to any section created by the assembler language START, CSECT, COM, or PSECT instructions (whether directly by an assembler programmer or indirectly by the FORTRAN or PL/I compilers or the linkage editor) or to the type of section created by the START or CSECT instruction as distinguished from the other instructions. The DSECT and DXD instructions create dummy sections, not control sections; this type of section is used to map storage, not occupy it, and does not result in a CSD.

control section dictionary (CSD): A table within the program module dictionary (PMD) which contains information on the external definitions and external references within a particular control section. This table makes possible communication between control sections in the same or different object modules. There is one CSD for each control section; the program module dictionary is essentially a collection of control section dictionaries.

The CSD is divided into: a heading, a definition table, a reference table, a relocation dictionary containing modification values, and a virtual memory page table which relates virtual storage assigned to the object module to the text pages it contains.

control statement: A source statement for the linkage editor. Control statements in the TSS/360 linkage editor are: INCLUDE (three forms), RENAME, COMBINE, TRAITS, and END.

CSD: See control section dictionary.

CSECT: 1. An assembler language instruction which creates and names a control section; 2. the type of control section which is created by a START or CSECT instruction.

CXD REF: A REF (external reference) entry created in the reference table of the control section dictionary by a compiler or as the result of a CXD assembler instruction. The value of the CXD REF (which is the length of combined external dummy sections) is calculated and filled in by the dynamic loader. There can be no more than one CXD REF in any CSD.

CXD-type reference: See CXD REF.

DEF: See external definition.

definition: See external definition.

definition table: A component of the control section dictionary which contains an entry for each external definition appearing in the control section. (See also control section dictionary.)

dynamic loader: A TSS/360 system component which has two main functions: 1. as the result of some demand (such as a CALL command), to allocate virtual storage to object modules residing in external storage, and 2. to resolve address constants when a page of text within a module is actually referred to during program execution. The dynamic loader does not load anything into main storage (the resident supervisor does this); it merely relates an object module's external location on an I/O device to a logical (virtual) address within a user's task by changing relative addresses within a module to virtual addresses within a task. The second function, resolution of address constants, is dynamic in that it does not occur until a page containing address constants is referred to by a page executing in main storage. Resolution consists of computing the virtual storage address value and inserting it into the space reserved for it in the text.

END: The linkage editor control statement which terminates control statement processing.

entry point: Generally, any location in a program or routine to which control can be passed by another program or routine. (See also standard entry point.)

entry point name: 1. A symbol whose value locates an entry point; 2. an operand in the RENAME control statement which must be an external entry point (one defined by an ENTRY assembler instruction or the name of a control section statement such as CSECT), not an internal entry point (accessible only from some other place within the same control section).

exclude: Pertaining to linkage editor output, not to include in the output module those object modules containing definitions which would resolve specified external references. The user specifies external references he does not want resolved in a form-3 INCLUDE statement. Presumably the unresolved references will be resolved by subsequent INCLUDE statements or by the dynamic loader.

external definition (DEF): Synonymous with external symbol definition. A type of entry in a control section dictionary for an external symbol that names a statement. A DEF resolves a corresponding REF. A DEF

is created in the control section dictionary as the result of: 1. an object module being created (its name is made the standard entry point DEF and placed in the PMD header), 2. a control section being declared (its name is made a DEF), or 3. a symbol occurring as the operand of an ENTRY instruction. (See also absolute DEF, relocatable DEF, and complex DEF.)

external dummy section: A dummy section (displacement map) known externally to the module in which it is defined. Each of different object modules forming a common program may contain one or more external dummy sections; the storage may be secured for all of them as one block. Each module will be able to refer to any displacement represented by a dummy section within that block. The external dummy section is created as the result of an assembler DXD instruction or a DSECT instruction in association with a Q-type address constant. (The total length of all external dummy sections defined in object modules loaded together must be provided for by a CXD instruction in one of the modules.) The external dummy section is used mainly by the PL/I compiler and assembler language programs that interface with PL/I programs.

external reference (REF): Synonymous with external symbol reference. A type of entry in the control section dictionary for each external symbol referred to but not necessarily defined (by an ENTRY statement) in the control section. The assembler user creates a REF as the result of an EXTRN instruction or by setting up a V-type, R-type, or Q-type address constant. A REF may also be created as the result of a complex DEF (external definition). If no corresponding external definition (DEF) exists or is found, the REF is unresolved.

external symbol: A symbol used by more than one control section within the same or different object modules. (See also external definition and external reference.)

external symbol definition: See external definition.

external symbol reference: See external reference.

INCLUDE: A linkage editor control statement which has three forms:

Form-1 - includes into the object module being developed by the linkage editor one or more input object modules from a specified library, and defines the input module to which any preceding TRAITS, COMBINE, or RENAME statements apply.

Form-2 - includes from a specified library all object modules whose external definitions resolve external references in the module being developed by the linkage editor.

Form-3 - includes from a specified library all object modules whose external definitions resolve external references in the module being developed by the linkage editor, except those external references specified.

internal reference: A type of external reference (REF) for a symbol which is internal to the object module (that is, it is resolvable by an external definition in some control section within the same object module).

internal symbol dictionary (ISD): A table containing the location, length, and type of all symbols that name program elements (the module, control sections, instruction labels, and data areas) within an object module. The assembler, FORTRAN compiler, and linkage editor produce an ISD unless the user suppresses it; the TSS/360 PL/I compiler does not produce an ISD. The linkage editor produces a composite ISD, containing all ISDs present in input modules and an initial directory pointing to these retained ISDs. The ISD makes possible program analysis using the TSS/360 program control system (PCS) commands.

ISD: See internal symbol dictionary.

linkage editor: A system-provided program, in some respects similar to a language processor, which may be optionally used to 1. join, or link two or more object modules into a new, comprehensive object module, and 2. change, or edit, control section attributes or names, entry point names, or external references in an object module by producing a new module that includes the desired changes. Using the linkage editor eliminates the need to reassemble or recompile, may save external storage and dynamic loader processing time, and may reduce paging activity when the program is executed.

load: 1. Generally, to place data into main storage or registers; 2. also, in TSS/360, to place programs (one or more related object modules) into virtual storage. The dynamic loader loads an object module (that is, allocates virtual storage addresses to it within a task) as a consequence of some user or system invocation; the program, or module(s), is not yet moved into main storage. Physical transfer of the program, or module(s), into main storage is performed in page units by the resident supervisor. When a page is physically loaded into main storage, hardware-implemented dynamic address translation

converts the virtual address of the page into a real main storage address.

loader: See dynamic loader.

object module: Also called a program module or an object program module, an object module in TSS/360 is the primary output of a language processor or the linkage editor. The object module is made up of a program module dictionary (PMD) containing control information, the text (that is, the program itself), and, at the user's option, an internal symbol dictionary (ISD), used for program analysis. When invoked by a user, an object module becomes input to the dynamic loader (unless it is already loaded).

object program module: Synonymous with object module.

PMD: See program module dictionary.

program module: Synonymous with object module.

program module dictionary (PMD): A table at the logical beginning of an object module containing control and descriptive information required by routines that must process the module. A PMD consists of a header and one or more control section dictionaries (CSDs).

pseudo-register: Synonymous with external dummy section.

Q REF: A reference to an external symbol that defines an external dummy section (for instance the name of a DXD statement).

Q-type address constant: A constant that reserves storage for the value of the displacement of an external dummy section into an area described by the dynamic loader. The symbol in the Q-type address constant must have been previously used as the name of a DXD or DSECT instruction. (See also address constant.)

Q-type reference: See Q REF.

Q-value: A value that represents the displacement of an external dummy section into the storage area reserved for external dummy sections. The dynamic loader supplies the Q-value. A program using the Q-value must get or reserve the storage required for the combined external dummy sections.

REF: See external reference.

reference: See external reference.

relocatable DEF: A DEF (external definition) whose value during execution is storage-location dependent. The value of a relocatable DEF as the result of language processing or linkage editing will be some displacement from the beginning of the control section in which the definition occurs. For example, if some statement at byte location 1000, relative to the origin of its control section, is named CHXAAA, then

```
ENTRY   CHXAAA
```

will produce a relocatable DEF entry for the symbol CHXAAA whose value will be 1000. The dynamic loader processes relocatable DEFs by adding, to the value assigned by the language processor or linkage editor, the virtual storage address of the defining control section.

relocation dictionary (RLD): A table within each control section dictionary which contains modifier pointers and modifiers for address constants (adcons). Each modifier pointer indicates a text page within the control section that contains address constants; each modifier contains information which the dynamic loader uses to determine the final value of the address constant. There are three RLDs in each control section dictionary: one for complex DEFs, one for external references, and one for internal references.

RLD: See relocation dictionary.

RENAME: A linkage editor control statement that changes entry point names, control section names, or external references, or deletes entry point or control section names.

resolved: Applied to an external reference for which the linkage editor or dynamic loader is able to find a corresponding external definition.

R-type address constant: An address constant whose value is the address of the control section in which a specified symbol was defined. For example, in

```
A   DC   R(ENTRY1)
```

the value inserted in location A by the dynamic loader will be the address of the control section in which ENTRY1 was defined (in which an ENTRY statement occurred with ENTRY1 as the operand). (See also address constant.)

R-value: The virtual storage location of the origin of the control section in which an ENTRY statement for a symbol appeared. Conventionally, when linking to reenterable (nonmodifiable) code in TSS/360, the V-

value locates an executable instruction to which control is passed; the R-value of a symbol locates the beginning of a control section (usually a PSECT) which may be used for modifiable storage. An R-value is also assigned to symbols that are the names of object modules and control sections. The R-value of the control section is simply the value of the control section name. The R-value of a module is either the address of the first PSECT in the module, or of the first CSECT if no PSECT exists.

standard entry point: The location in an object module at which program execution will begin if the module is invoked by its name. A user may call a program to run (via the CALL command or by direct call) by specifying the object module name; execution will begin at the standard entry point. An object module may have several entry points to which other programs can pass control; it can have only one standard entry point. The FORTRAN and PL/I compilers generate a value which is the location of the beginning of the main procedure in the module. The assembler uses the address of the first control section (CSECT) in the module as the standard entry point unless the user has specified another location as the operand of an END statement. The standard entry point name is contained as a DEF in the header of the PMD. The linkage editor produces a module whose standard entry point is that of the first input module; it also retains the standard entry point of each input module, enabling the user to run by name not only the linkage-edited module but any of its component input modules.

symbol: A character or combination of characters that represents addresses or specified absolute values. Through their use as names and in operands, symbols provide the programmer with a way to name and refer to elements (control sections, instructions, and data areas) of a program.

text: The instructions, constants, and reserved data areas of an object module; the program itself.

TRAITS: A linkage editor control statement that specifies new attributes for a designated control section.

type-1 complex DEF: See complex DEF.

type-2 complex DEF: See complex DEF.

unresolved: Applied to external references for which the linkage editor or dynamic loader is unable to find a corresponding external definition in another object module or control section. The linkage editor provides a list of unresolved references at the termination of its processing (as well as those unresolved but resolvable by definitions in programs in SYSLIB).

version identifier: A character string that identifies a particular assembly, compilation, or linkage editor run. The character string can be one-to-eight alphameric characters specified by the user in his command, or, if defaulted, will be the date and time of the run, supplied by the system as the number of microseconds elapsed since March 1, 1900. The version identifier is placed in the program module dictionary and appears in the PMD listing section of the list data set.

virtual address: Also called logical address, an address generated by a program that references virtual storage and must, therefore, be translated into a main storage address as it is used.

virtual memory page table (VMPT): A table in each control section dictionary which relates pages of text within the control section to virtual storage assigned the control section. A control section may occupy more space in virtual storage than its text pages require; ORG instructions will cause virtual storage to be allocated which does not contain text pages. The VMPT tells whether a page is empty (reserved) or, if it contains text, which page it is relative to the first page in the control section containing text.

VMPT: See virtual memory page table.

V-type address constant: A type of address constant that reserves storage for and whose value during program execution is the address of an external symbol. By specifying a symbol in a V-type address constant, the assembler language EXTRN instruction need not be used. Conventionally, when linking to a reenterable (nonmodifiable) program in TSS/360, the V-type address constant loaded into a register provides the address to which control is to be passed; the R-type address constant loaded in another register provides the location of a modifiable control section. For each V-type address constant, an external reference (REF) is created in the control section dictionary. (See also address constant.)

V-value: A virtual storage location that an external symbol labels. By convention in TSS/360, when linking to reenterable (nonmodifiable) code, the V-value of a symbol locates the symbol itself (provides its address); the R-value of a symbol locates the beginning of a PSECT which the executable code may use to obtain and modify data. V-values are provided by the dynamic loader.

INDEX

Where more than one page reference is given the major reference is first.

Linkage Editor Printed in U.S.A. GC28-2005-4

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM Technical Newsletter

File Number	S360-31
Base Publication No.	GC28-2005-4
This Newsletter No.	GN28-3206
Date	February 1, 1972
Previous Newsletters	None

IBM System/360 Time Sharing System
Linkage Editor

This Technical Newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

5-6
41-42

A change to the text is indicated by a vertical line to the left of the change.

Summary of Amendments

- The linkage editor now saves (in the output module) the names of all input modules.
- Two figures have been added showing size limitations imposed by the linkage editor.

Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Dept. 643, Neighborhood Road, Kingston, N. Y. 12401

PRINTED IN U.S.A.